ISBN 0-315-56108-4

Realtime Tracking using Wrist-mounted Range Profile Scanners

Suresh Venkatesan

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

April 1990

CONCORDIA UNIVERSITY
Division of Graduate Studies

This is to certify that the thesis prepared

By: Suresh Venkatesan

Entitled: Realtime Tracking using Wrist-mounted Range Profile Scanners

and submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

[illegible] Chair

Colin Archibald

[illegible]

[illegible] Supervisor

Approved by [illegible]
Chair of department or Graduate Program Director

May 7 1990

[illegible]
Dean of Faculty

Abstract

The results of experiments in realtime tracking of a moving object using wrist-mounted laser range finders are presented. The objective is to have the robot end-effector maintain a constant position and orientation (pose) with respect to a flat object moving in three-dimensional space in the field of view of the range finders. Sensory feedback from the range finders is used to servo the robot in order to meet this objective. The resulting trajectory of the end-effector is similar to that of the moving object in five degrees of freedom. Two laser range finders are mounted orthogonally on the wrist of a six-degree-of-freedom PUMA 560 robot. The algorithms for this application are written in C on a Macintosh. The executable code is downloaded to a target system that consists of four Motorola MC68020 processors, connected by an industry standard backplane VME bus, running under the Harmony operating system. Tracking is done with maximum linear and angular speeds of the target object being 25 cm/s and 0.5 rad/s, respectively. The robot is capable of maintaining the required pose with respect to the object with a small lag, in the order of tens of milliseconds, mostly due to data collection time. The software is designed to ensure that the robot has a smooth motion, without any sharp discontinuities. Maximum speed and acceleration parameters, incorporated in the software, facilitate such a trajectory. A performance study of the tracking experiment is presented.

Acknowledgements

The research described in this thesis was carried out in the Computing Technology Section (CTS) of the Division of Electrical Engineering at the National Research Council of Canada (NRCC) in Ottawa. This thesis would not have come to fruition without the assistance of the CTS staff. First, and foremost, I wish to thank Colin Archibald, who, by way of supervision at NRCC, familiarized me with the previous research done in the area of sensor-based robotics at NRCC and assisted me in clearly defining the problem to be addressed. His willingness and patience in answering some of the questions that I had, at different stages of the thesis, is sincerely appreciated. Comments provided by him in the preparation of my publications as well as my thesis is gratefully acknowledged.

The assistance of Cathy Merritt, Mike Duggan, Doug Taylor, Grant Bechtold and Art Binch is greatly appreciated. The comments provided by Mr. N. Burtnyk, Marceli Wein, Shadia Elgazzar, Gerhard Roth, Ramiro Liscano and Dave Green at various stages of my thesis are also appreciated. I wish to thank the Harmony team for their assistance and Jeff Kelley, in particular, for his help during the final stages of my thesis.

Many thanks go to Evelyn Kidd of the Editorial Office and Leslie Cameron of the Drafting Office for their assistance in the preparation of the above-mentioned publications. My sincere thanks to Dan Gamache of the Photography Section and Conrad Melanson of the Public Relations and Information Services (PRIS) for developing the photographs and preparing the video that demonstrate realtime tracking.

Dr. T. Kasvand, who co-supervised this thesis from Concordia University, also assisted in the formulation of the problem. My thanks to him for this and also for his comments on the general progress of the work as well as on the preparation of the thesis.

I thank NRCC and Concordia University for providing financial assistance during the course of this research.

I would like to thank Unimation Inc. for permitting me to reproduce figures 6, 7 and 8. I also wish to thank Colin Archibald for allowing me to reprint figures 9 and 10.

Table of Contents Page

List of Figures

List of Figures Page

List of Tables Page

Chapter 1

Introduction

The range of applications in robotics is significantly increased when sensors are used. The role of sensors in the field of robotics is to provide feedback for controlling the motion of the robot in order to achieve the desired application. Robots responding in real-time to sensory feedback are used in many areas. For example, in a quality control system it is desirable to pick out defective parts moving on a conveyor belt, while in an assembly unit parts have to be picked for assembly from oscillating baskets moving on an overhead conveyor line. Another example is in the field of aerospace where it is necessary to grasp moving satellite grapple fixtures or capture defective satellites.

All of the above applications, which are just a few of the many that exist, make use of robots. Moreover, in order to accomplish these objectives, the robot must be capable of tracking these objects in real time. Since time is of the essence, resources with a very fast response time, typically in the order of milliseconds, are needed to solve this problem. A very high rate of sensor feedback is thus essential for servoing the robot through an unstructured environment. The computational requirements of a realtime trajectory generator based on sensor feedback are not met by commercially available robot controllers. If this feedback loop is to be used optimally, the application should be able to deal with differences in the sampling rates of the sensor and the motion request rates of the robot controller. Working in a sensor-based robotics environment requires that a high degree of concurrency be incorporated in the software [1]. Processing time should be kept to a minimum if the application is to be done in real time.

1.1 Sensors in Robotics

Various types of sensors have been developed that yield different kinds of sensory information to close the feedback loop in the control of a robot [2]. These have been classified broadly as non-contact and contact sensors. This work is concerned with vision sensors that fall under the category of non-contact sensors. The choice of a vision system depends largely on the application. In some cases, special purpose sensors must be developed to suit the application. Everett [3] outlines some of the design considerations for such a sensor as required in the field of mobile robotics. However, in general, there is a large shift in the vision community towards systems that use range finding techniques. The reason for the shift is that most of the vision problems are concerned with 3-D objects and a knowledge of the third dimension, viz. the depth, is important. A survey of the various range finding techniques can be found in [3, 4].

Of the various range finding techniques, triangulation is the most widely used. Here, the range of an object of interest is determined using the principles of trigonometry. Vision systems that employ triangulation for determining the range are further classified as passive or active, on the basis of the physical arrangements. In passive systems, two cameras are focussed on a target point lit by the ambient light of the scene, while in active systems a controlled light source is directed at the target point and the reflected light is detected by an imaging sensor. Stereo disparity is an example of passive triangulation while structured light ranging is an example of active triangulation.

1.2 Realtime Systems

Realtime systems, also referred to as embedded systems, are those in which time is a critical resource. The system interacts with events in the outside world, which are usually asynchronous in nature, and responds within a specified and finite interval. The system must meet response requirements that are stipulated by these external events. It must be noted that there is no compromise in the correctness of the execution in an effort to meet these deadlines. Predictability, not speed, is the underlying principle of realtime systems. The response time of a realtime system, based on a worst case analysis, must be predictable and guaranteed. While there is an upper limit on the execution times of such systems, speed of execution is not of concern as long as it does not exceed this limit.

Several processors combined can yield performances comparable to that of a main-frame at a much lower cost. Moreover, the system performance can be gradually increased as the work load grows by adding more processors. However, Fathi and Krieger [5] show that the throughput of a multiprocessor system is directly proportional to the number of processors only up to a certain point. Beyond that point, any addition of processors results in the degradation of the system throughput as the amount of contention on the shared resources increases and so does the overhead on the interprocess communication. As long as this point is not crossed, such systems are a viable alternative for high speed computing. Moreover, the system can be better utilized by properly sharing the work load amongst the various processors.

In order to provide fast and bounded response times, needed in a realtime environment, computing systems are developed based on multiprocessor configurations. Flynn [6] classifies such systems into four categories based on the concept of instruction and data streams. These are:

1. Single Instruction Stream - Single Data Stream (SISD)
2. Single Instruction Stream - Multiple Data Stream (SIMD)
3. Multiple Instruction Stream - Single Data Stream (MISD)
4. Multiple Instruction Stream - Multiple Data Stream (MIMD)

An Instruction stream is defined as a sequence of instructions performed by the machine while a data stream is a sequence of data upon which the instruction stream acts. This taxonomy is still very widely used despite some shortcomings. For example, in a survey of the different special computer architectures in the field of robotics, Graham [7] points out that Flynn's classification does not clearly differentiate between tightly and loosely coupled (ability to share resources) systems or between overlapped and pipelined types of parallelism.

A realtime application deals with events in the external world that occur simultaneously and in an asynchronous mode. The software developed in such an environment is designed as a collection of concurrently executing processes that model these events. The complexity of realtime applications requires that modular techniques be used. In general, this is done by designing the application as a number of tasks which are scheduled for execution on the basis of their priority. Schedulers also support preemption in order to meet the time constraints [8]. This implies that it is left to the designer to assign priority values to the tasks depending on their *urgency*. In a complex system, this is very difficult due to the number of tasks that make up the application. Lee *et al.* [9] present a realtime kernel that provides services with bounded worst-case execution times. Scheduling is based on time, rather than priority, using the timing constraints specified by the programmer.

Realtime systems are classified as hard and soft. In hard realtime systems, speed of execution is not of concern as long as processing conforms to the timing constraints rigorously. This is not so in soft realtime systems, where processing is done as fast as possible without any specific deadlines for completion. Task scheduling in hard realtime systems can be static or dynamic. In a static approach schedules for tasks are done off-line and require *a priori* knowledge of the tasks. In a dynamic approach, schedules for tasks are decided on-line as tasks are dynamically created. Although static approaches have low run-time cost, they are inflexible and cannot adapt to a changing or unpredictable environment. In contrast, dynamic approaches involve higher run-time costs, but are flexible and easily adapt to changes in the environment. Cheng *et al.* [10] survey some of the scheduling algorithms in static and dynamic hard realtime systems.

Stankovic [11] describes some of the issues involved in designing a realtime system. Tasks executing in such an environment are aptly referred to as time-critical tasks. The kernel maintains a realtime clock so as to meet the timing requirements of these tasks. It must have a fast context switch and be able to respond quickly to external events, if the system is to be fast. The system should also be predictable in that it must be able to determine with certainty the completion time of an invoked task. It should support integrated CPU scheduling and resource allocation in order to satisfy, in a timely manner, the resource requirements of the cooperating tasks. Access to shared resources must be mutually exclusive. Resources needed for highly critical tasks must be preallocated so that the tasks can execute without delay. Allocation policies must be time-driven so that realtime scheduling requirements are met and resources available in time for subsequent utilization. Reliability is essential for realtime systems. Incorporating flexibility in such a system will help in dealing with situations where it is anticipated that certain deadlines cannot be met.

1.3 Robot Programming

The notion that robots are designed to do certain tasks in a repetitive and mundane manner is fading as advanced robot control techniques are developed. Teach-by-showing techniques, although simple to use and implement, are not suitable for unstructured environments. Working in such environments requires the use of robot programming languages that are capable of fetching data from sensors and determining the motion to be performed based on the sensory data.

Several robot programming languages have been developed, over the years, that have been used with different industrial robots. A detailed survey of these languages can be found in [12, 13]. Another approach is to extend some of the currently available high level languages through the use of function libraries, in order to serve as a robot programming language. Hayward and Paul [14] present the RCCL system, a library of functions programmed in C that can execute either under the UNIX[1] operating system or under a realtime operating system. The ability of the C functions to handle low-level details enables robot control to be programmed in C. A servo program resident in the kernel of the supervisory host delivers setpoints at prescribed intervals to the individual joint controllers, via a buffered high-speed parallel link. Motion *requests* issued by the user program are queued for execution and then translated into joint motions. RCCL is portable, does not depend on the configuration of the robot, and supports world modelling, sensor integration, and compliant control.

Gini [15] classifies robot programming languages into four classes based on the level of detail in which the robot operations are expressed. These are:

1. Joint level – The task is described in terms of control commands required to drive the individual motors and actuators. Teach-by-showing techniques fall into this class. It is not suitable for off-line programming and has limitations on the use of sensors.
2. Manipulator level – The manipulator positions and movements in the Cartesian coordinate system are specified. It supports the integration of sensors thus enabling the modification of the robot motion based on the sensor feedback. Since every robot action is to be explicitly specified, languages in this class are termed as *explicit languages*.
3. Object level – At this level, instruction is given in terms of the action to be carried out on the objects being manipulated. Because there is no need for explicit

[1] Unix is a trademark of AT&T.

specification of the robot actions, languages in this level are referred to as *implicit languages*. The absence of this explicit specification implies that the programming system is responsible for automatically synthesizing manipulator level instructions.

4. Task level – The instructions take the form of a general description of the task to be performed. Again, the burden of creating lower level instructions resides in the domain of the programming system.

Off-line robot programming, used in this research, enables the creation and testing of programs in a high level language. These programs are downloaded to the robot controller via a communication line. This type of programming is advantageous in that there is no need to have a robot during the software development phase. Such a need arises only when the application is ready to be executed.

1.4 Thesis Organization

Chapter 2 of this thesis describes some of the previous work done in realtime tracking and the contribution of this research. The statement of this thesis is formally presented in Chapter 3. The experimental apparatus used in this work is discussed in Chapter 4. This includes a brief description of the wrist-mounted laser range finder (also referred to as a range profile scanner), the Harmony[2] operating system, and the PUMA[3] 560 robot. The software design and implementation of the tracking application is detailed in Chapter 5. Results are presented in Chapter 6. These include a performance study of the behaviour of the robot while tracking as well as a timing analysis of the tracking application. The thesis concludes in Chapter 7 with a summary of this research.

[2] Mark reserved for the exclusive use of Her Majesty the Queen in right of Canada by Canadian Patents and Development Ltd.

[3] PUMA is a registered trademark of Unimation Incorporated.

Chapter 2

Related Research

Vision systems in the past consisted of overhead cameras that were fixed above the workspace of the robot. While this yields a large field of view of the workspace, the robot arm frequently blocks portions of the field of view that are vital to the accomplishment of the robot task. Moreover, it is not possible to zero in on a desired area with higher resolution. Loughlin [16] discusses the advantages of mounting compact cameras on the wrist of the robot as opposed to overhead camera configurations. Since the camera is always at rest in relation to the manipulator, such an eye-in-hand vision system solves the above problems. Furthermore, the parallax error that very often plagues the fixed overhead camera configuration is considerably reduced.

A learning control approach to track and intercept an object moving on a conveyor belt is described in [17]. The learning phase is designed to train the system in reproducing the relationship between the sensory feedback and the system commands. Feedback from a video camera mounted on the wrist of a robot is used for tracking and intercepting the object. Initially, in the absence of training, the robot lags behind the object but with repeated attempts the lag is substantially reduced. Kabuka *et al.* [18] propose a servo system, with two servos that can track an object in the field of view of the camera. The feedback from the camera allows the two servos to move the camera in two degrees of freedom so as to center on the object. It takes about 30 s for this to be accomplished and hence is not practical for tracking moving objects despite being interesting theoretically.

Tracking an object in a plane – three degrees of freedom – is the subject of [19]. The object, a gasket in this work, is taught to the system via teach-by-showing techniques. A wrist-mounted CCD video camera is used to recognize the gasket by using the circles on the gasket as the main features. Excluding the initial time required to recognize the gasket (about 5 s), a vision cycle time of approximately 60 to 70 ms is achieved. The motion of the robot lags behind that of the gasket at 0.1 rad/s. This lag increases with increasing speed of the rotation of the gasket and at 1 rad/s the robot is unable to track the gasket.

Gennery *et al.* [20] present the work being done at the Jet Propulsion Laboratory (JPL), in California, on tracking and acquisition in space telerobotics. The machine vision system at JPL, comprised of both fixed as well as wrist-mounted video cameras, is designed to track and acquire polyhedral objects moving and rotating in space. PIFEX, a programmable pipelined image processor, is used for performing different operations on images. Tracking is done by first using previous images to predict the pose of the target object. This prediction is subsequently compared with the observed pose of the object in order to make any corrections to the pose (as well as its time derivatives) of the end-effector. No results have been reported on the performance of the tracker.

Using photogrammetric methods, the relative pose between a camera and an object is determined in all six degrees of freedom as long as the object is rigid and carries at least three targets [21]. Zaremba [22] employs this method to track an object, with four targets, suspended on an overhead conveyor in six degrees of freedom. Prediction-based control is then used to pick these objects at suitable grasping positions. Since the pose of the object is obtained by a study of the patterns formed by the targets, proper positioning of these targets on the object is essential. However, these targets cannot entirely represent the various features of the object.

Optical flow techniques have been used in the measurement of visual motion and in the recovery of 3-D structure of moving objects [23, 24, 25]. Optical flow is the apparent motion of brightness patterns when a camera moves in relation to the object being imaged. There is, however, one disadvantage with the optical flow measurements and it involves dealing with the *correspondence problem* [26]. Bandopadhay *et al.* [27] claim that the advantages of optical flow outweigh this disadvantage and have developed a mathematical framework for active navigation employing tracking. The advantage of tracking a prominent environmental feature point is underscored.

The problem of realtime tracking has been addressed by Luo and Mullen [28] using optical flow. A PUMA 560 robot tracks an object moving on a conveyor, representing the XY plane, at different speeds. A wrist-mounted CCD camera and a finger-mounted ultrasonic sensor are used. The CCD camera provides the X and Y coordinates of the target point on the object, while the ultrasonic sensor yields the range, in the Z direction. Successive positions of the chosen target point and the velocities are determined using the optical flow approach. The robot maintains a constant standoff from the object until the relative velocity between them is near zero. At this stage, with the help of the ultrasonic sensor, the robot descends to grasp the object. While different kinds of object are used in this work, the robot is incapable of tracking these objects if the speed of the conveyor exceeds 8.5 cm/s.

All of the research described above uses visual feedback obtained from CCD cameras. More recently, the use of range finders in vision systems has gained importance over video cameras because the depth information is immediately available. By incorporating a range finder in the vision system, Lougheed and Sampson [29] address the problem of robot bin picking in real time. Sorensen *et al.* [30] describe the Minnesota scanner, developed to record human locomotion in a target volume of 8 m^3. Using the fact that three intersecting planes define a point, three planes of low power laser light sweep this volume, containing target photodetectors, to record human motion.

The telerobotic capabilities of a robot used in space are detailed in [31]. Using a wrist-mounted laser proximity sensor, the robot follows the surface of a target object. The location of the end-effector is determined using a space location system (SLS). The SLS consists of a set of six laser interferometers that track retro-reflectors mounted on the target. These retro-reflectors reflect the incident light back to the laser interferometer along the direction of the incident path. Choice of control points, similar to [22], allows the pose of the target relative to the interfercmeters in six degrees of freedom to be determined.

The use of wrist-mounted range finders combines the advantages of the eye-in-hand vision systems and those of the range finders. No work has been done, to date, on realtime robot control using wrist-mounted range finders, with the exception of [32, 33, 34]. This thesis deals with the use of this new and unique vision system to track a flat object in realtime in five of the six degrees of freedom.

Chapter 3

Problem Definition

An object in space can move in six degrees of freedom. These are the displacements, in the Cartesian coordinate system, along the X, Y, and Z axes and rotations about the X, Y, and Z axes (Fig. 1). These rotations are, respectively, referred to as the yaw, pitch, and roll motions.

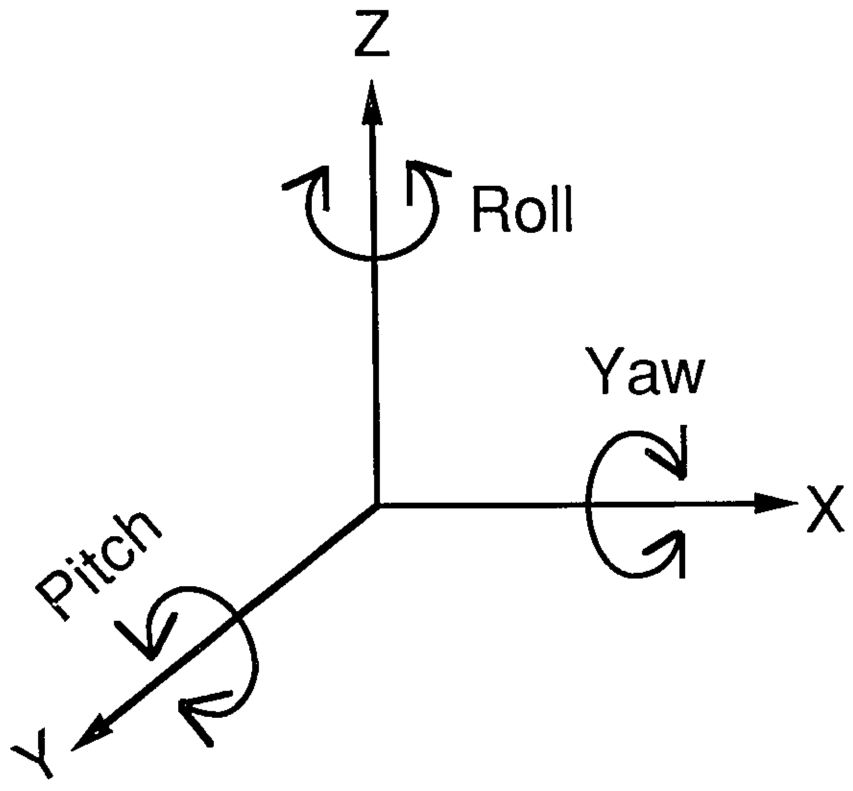

Fig. 1 Six degrees of freedom

A significant amount of research has been done in realtime tracking (Chapter 2). The scope of this thesis is to study this important problem with the help of a new and unique sensory feedback control apparatus (to be discussed in Chapter 4).

The problem is formulated as follows: Realtime tracking requires that a constant relation be maintained between a moving object and the end-effector of the robot. The objective is, at all times, to position the end-effector of the robot

a. Normal to the plane of the object, and
b. At a certain standoff from the center of the object.

Two assumptions are made in solving the above problem:

1. Objects being tracked are assumed to be flat for the sake of simplicity.
2. Both the target object as well as the background should be in the field of view of the range finder in order for it to differentiate between the two. This is guaranteed by choosing a suitable value for the standoff (35 cm was chosen in this project).

In order to accomplish this objective, the design of the software should be capable of handling different types of resources. For example, it must acquire data from sensors and provide the feedback to the robot controller. The code must also support asynchronism since this is usually the case in a realtime environment. For best performance, such an application requires a low vision cycle time, a fast reponse time by the robot, and a computing system with high speeds of execution.

Chapter 4

Apparatus

Two laser range finders, hereinafter referred to as range profile scanners (RPSs), are mounted orthogonally on the wrist of a six-degree-of-freedom PUMA 560 robot, as shown in Fig. 2. Each RPS is skewed by 15^{o} in its respective plane for two reasons:

1. To compensate for an inherent skew of 5^{o} in the optical design of the RPS, and
2. To ensure that the cross-over of the profiles from the two RPSs occurs over the entire workspace of the robot.

Data from the two RPSs are fetched in an alternate manner so as to avoid any interference between the two laser beams. Often in systems where the sensor and robot are physically disjoint, a sensor-to-robot coordinate transformation is required [29]. In this work, since the RPSs are mounted on the wrist of the robot and since the robot is controlled in its tool coordinate frame, there is no need for such a transformation.

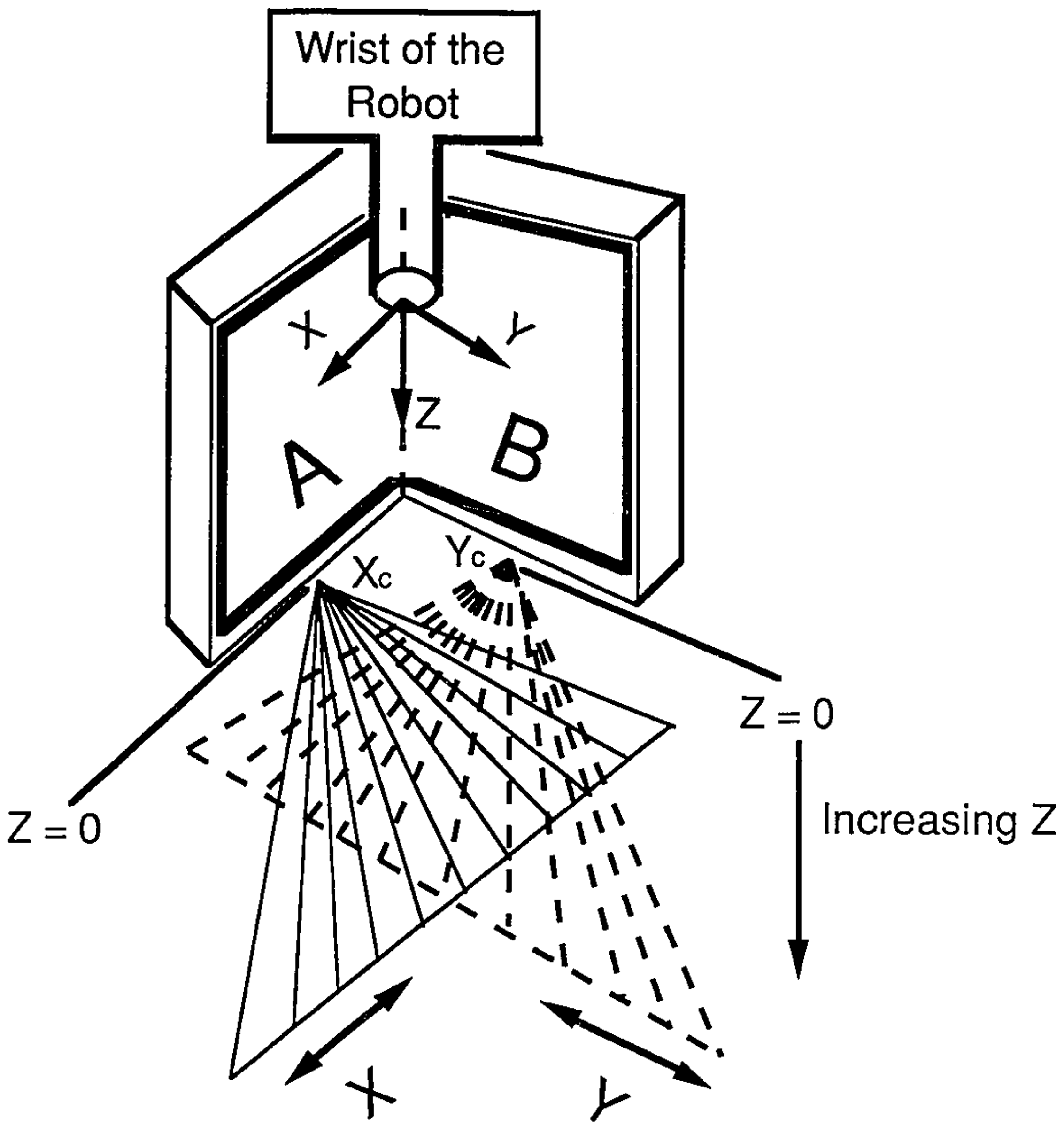

Fig. 2 Orthogonal configuration of the RPSs. Range data are measured from the perspective of the RPSs. X_c and Y_c are the centers of the range profiles collected by RPSs A and B.

The software for the tracking application is written in C and developed on a Macintosh.[4] The executable images of these application programs are downloaded onto four Motorola MC68020 processors running under the Harmony operating system. The Macintosh is not used during the execution of the tracking experiment except for any I/O. The robot can be actuated through the Unimation robot controller either by sending motion requests from the application or by issuing VAL II[5] instructions. When tracking is being carried out, the controller receives motion requests from the application software through the ALTER port. ALTER is an extension of the VAL II language that facilitates realtime path modification from an external computer [35]. It expects relative increments in position every 28 ms from the tracking application, failing which the communication is terminated and the application is halted. Software for the scanner interface, developed in the Assembly language on the VAX 11/780, enables data to be fetched from the RPSs. A schematic diagram of the experimental set-up is shown in Fig. 3.

[4] Macintosh is a trademark of Apple Computer, Inc.

[5] VAL II is a registered trademark of Unimation Incorporated.

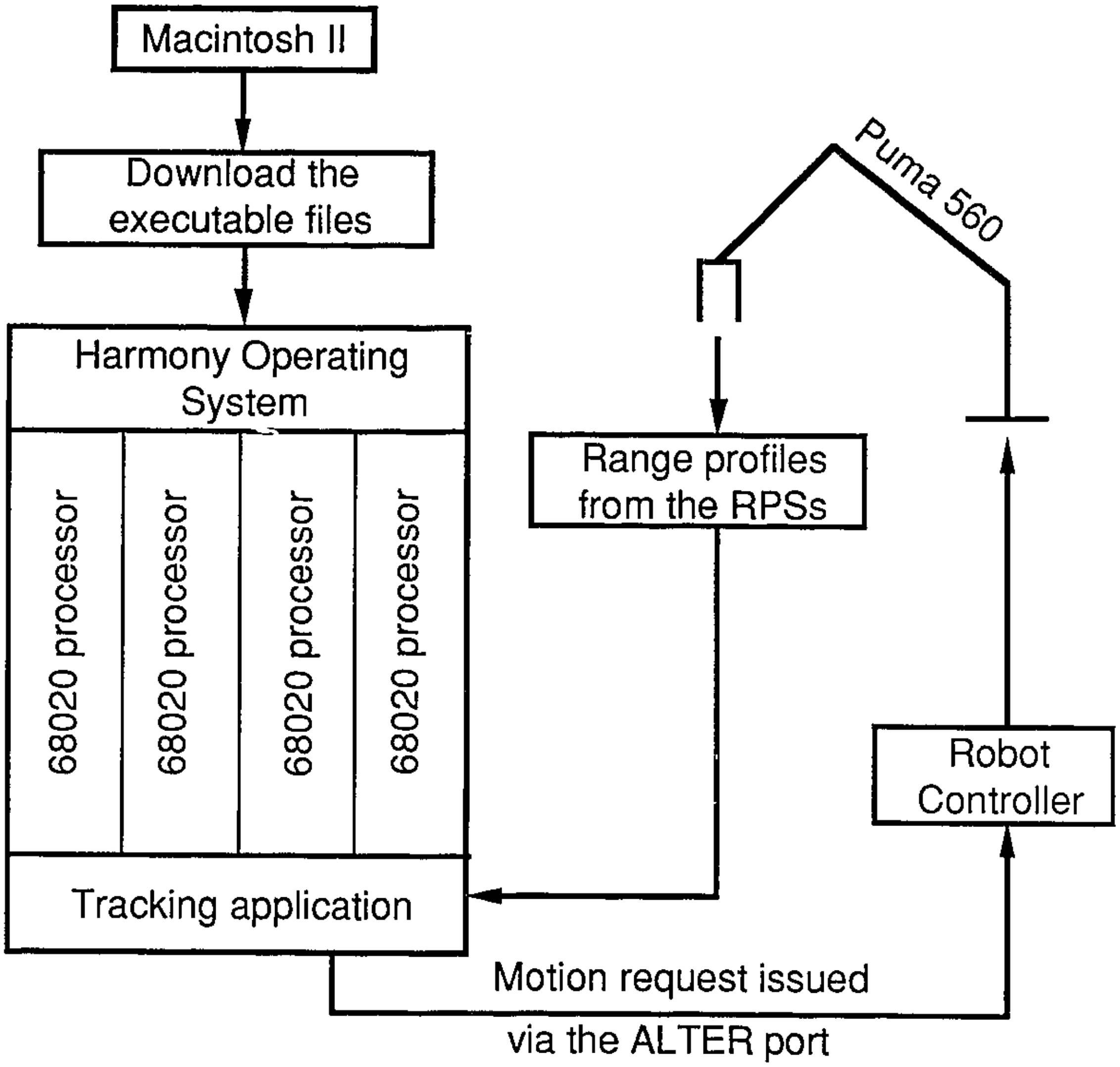

Fig. 3 Schematic diagram of the apparatus.

4.1 The Range Profile Scanner

A shadow effect, also referred to as the "missing parts" problem in [3], is present in all triangulation systems. This occurs when a source is unable to see what a detector can or vice versa (Fig. 4). As the angle between the projection and detection axes increases, the resolution is increased but so is the shadow effect and the size of the sensor. As well, a vision system with a large distance between the source and the detector has a poor depth of view. Passive triangulation has two drawbacks not found in active triangulation. It suffers from the correspondence problem and fails in a poorly lit environment. Active sensing techniques have been put to use in automatic surveillance and robot navigation [36] as well as in applications where short-range 3-D measurements are required.

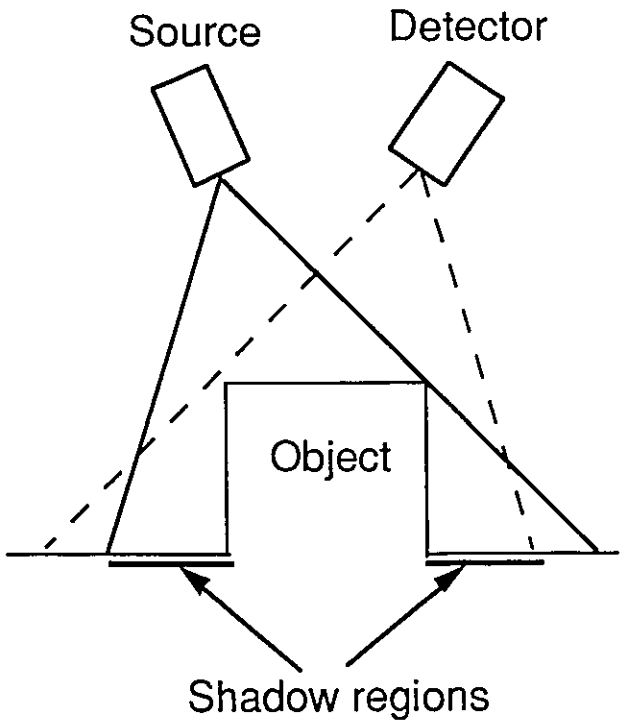

Fig. 4 Shadow effect.

In an effort to improve the performance of optical triangulation and reduce/eliminate the above pitfalls, RPSs that are instances of those described in detail in [37] were developed at the National Research Council of Canada (NRCC). The sensor was designed specifically for large depth of field of use (10 cm to 1 m), making it useful in an industrial robot environment. As the RPS is to be mounted on the wrist of a robot, it is necessary that it be compact, lightweight, and robust. The compactness of the RPS results from the source and the detector being close to each other, thereby ensuring that the angle between the projection and the detection axis is small, which minimizes the shadow effect. Each RPS has the following physical characteristics: width 90 mm, length 140 mm, depth 28 mm (this is 80 mm where the galvanometer protrudes), and the weight is 500 g including the protective case.

The optomechanical design of the RPS can be seen in Fig. 5. The design of the device provides a rigid structure with simplicity of assembly and minimal alignment. An optical fiber is used to carry the HeNe laser source. The RPS is based on the principle of synchronized scanning [38] and makes use of a double-facetted mirror on the paths of both the projected laser and the reflected light. This mirror is driven by a galvanometer that enables it to scan over a 40^{o} field of view. The projected beam is reflected off a small fixed mirror onto the front face of the scanning mirror, while the detected beam is returned to the sensor via a collecting lens using the back face of the same mirror. Two large fixed mirrors placed along the two branches of the lower half of the X frame assist in reflecting the projected and detected beams.

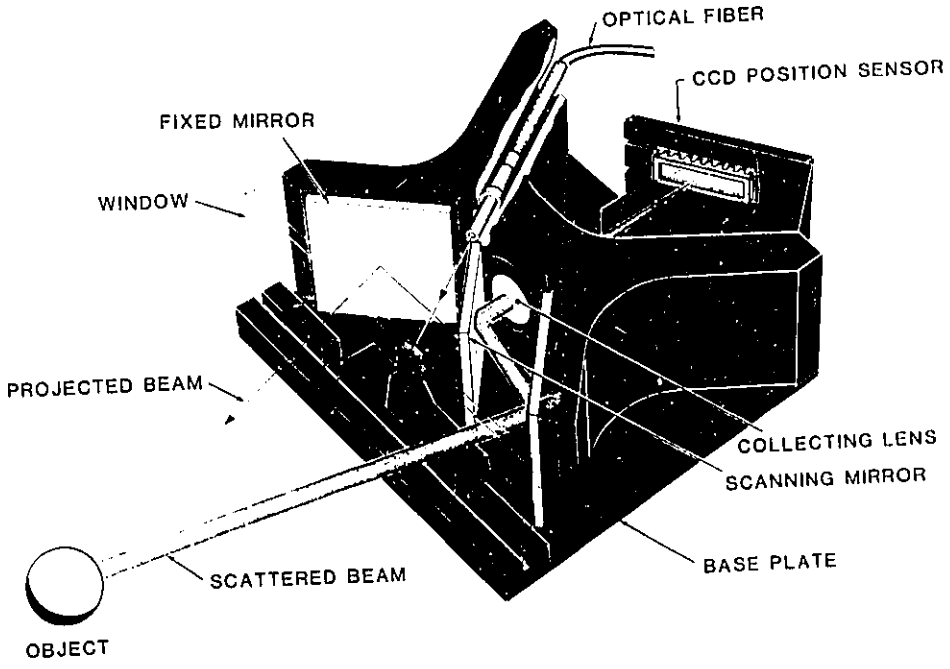

Fig. 5 Design of the range profile scanner

A 256-element linear array CCD is the position sensor and is tilted at an angle of 35^o to the horizontal, to allow for good focusing over the entire depth of view. This scanning geometry has the advantage of providing immunity to ambient light, because the viewing direction is always aligned to the direction of the scan making the instantaneous field of view very small. The signal-to-noise ratio is high, which also allows a large depth of view with a constant laser power. The field of view is approximately 10 cm in the direction of the scan (call this the X direction) at a standoff of 10 cm (the Z direction). Where Z is 100 cm, the field of view is 80 cm in X. The resolution of the RPS is an inverse function of its distance from the object and this fits into the requirements of robot vision where at large standoffs resolution is of less concern. The accuracy of the RPS is approximately ± 0.1 mm in Z at a standoff of 10 cm, decreasing to approximately ± 2 mm at a standoff of 85 cm. The field of view is not exactly an isoceles triangle because of the geometry of the optomechanical scanning mechanism.

When the scanning mirror sweeps the surface of a given object, 256 surface elements (surfels) are returned that correspond to the distances from the RPS to a row of points on the surface being scanned. The RPS is calibrated (to be discussed in Section 5.3) to provide reliable data over the entire field of view. The range data are measured from the perspective of the end-effector and are in Cartesian coordinates. This single raster scan of range data is called a *range profile*. The scanning mirror in the RPS sweeps back and forth, capturing 26 range profiles per second. Although there are 256 surfels on each scan, range data are not available from every surfel on the object. One reason for this is the shadow effect. Another reason is that the reflected light from a surfel may either be too little or too much for the sensor to determine the correct range value because of a combination of surface reflectivity and geometry. When no range value is available this fact is indicated by the sensor, ensuring that algorithms interpret only valid range data. These situations are taken care of by the Validate routine to be discussed in Section 5.4.

4.2 The Harmony Operating System

Harmony is a multitasking, multiprocessing, realtime operating system developed at the National Research Council of Canada to meet the needs of realtime applications. Gentleman *et al.* [39] provide a detailed view of this system and the services that it provides. The multitasking feature enables a Harmony-based application to be designed as a set of tasks in keeping with the modularity requirements of a complex realtime application. The multiprocessing capability can be exploited to achieve optimum system utilization and throughput by suitably distributing the application tasks on the various processors, at link time, in order to obtain a good load balance.

Harmony is an open system, that is, it can be ported to several different hardware configurations. In this research, it runs on a multiprocessor system based on an industry standard backplane VME bus. The multiprocessor system consists of four DY-4 DVME-134 single board computers based on the Motorola MC68020 processors that share a common address space. The operating system is written in C and so are the application programs. Harmony has not been designed to support development of software for the user applications and, consequently, this is done elsewhere. This partitioning results in a host-target configuration for developing an application. The host system, in this project, is an MC68020-based Macintosh that uses a Consulair Mac C compiler.

Tasks form the backbone of a Harmony environment and every instantiation of a task must be explicitly created. Once a task is created it executes independently of, and in parallel with, the parent task that created it. There is no restriction for the child task to be running on the same processor as the parent task. The processor on which a task will execute is decided at the time of linking. The multitasking feature of Harmony enables the various tasks running on a given processor to share this processor. At the time of creation, a task is associated with a task template that contains the necessary information pertinent to the task such as its id, the code to be executed, stacksize, priority, and the processor on which it will execute. A task can either be destroyed or it can commit suicide. In either case the resources used by the task during its execution are returned to the system.

Tasks running on a given processor can only compete for the resources of that processor. Harmony supports preemptive, priority-based, FIFO scheduling similar to the scheme used in VRTX (Versatile Real-Time Executive) [8]. This allows a task at the top of the highest priority queue, if ready, to be scheduled for execution. The currently executing task is preempted, if either it is blocked (discussed below) or a task with a higher priority becomes ready. Although the tasks execute in parallel, at a given time only one of them is physically utilizing the processor. The task scheduler has a fast context switching time (39 μs on a MC 68020 at 16 MHz) and interleaves the execution of the different tasks on the processor. Realtime requirements of the problem necessitate the choice of these scheduling algorithms over those found in timesharing systems.

While tasks execute synchronously and sequentially, strict parallelism and asynchronism exist between tasks. However, tasks that cooperate towards a common goal must communicate. In the thinwire model of parallel computation [40], tasks on different processors do not have access to each other's memory. In the present configuration, since the kernel is implemented in a flat address space, the memory can be shared by the various tasks. Message passing is the paradigm used in Harmony for communicating between tasks. It is independent of the actual task distribution amongst the processors and of the hardware communication channels. Tasks can have a dialog by either passing pointers to data or by sending the data itself. The latter is preferred for reasons of simplicity. There are four message passing primitives which are implemented as functions. These are:

id = _Send(*rqst*, *rply*, *id*);

id = _Receive(*rqst*, *id*);

id = _Try_Receive(*rqst*, *id*);

id = _Reply(*rply*, *id*);

When each function returns, the value "id" returned is the pointer to the task descriptor of the correspondent task. A typical communication takes place as follows: The sending task sets up the message, to be transferred, in space pointed to by the *rqst* argument. It also sets up space pointed to by the *rply* argument where the reply message from the receiver will be stored and then calls the _Send() primitive with the *id* of the receiving task. The receiving task sets up space pointed to by the *rqst* argument into which the transferred message is to be contained and then issues the _Receive or _Try_Receive primitive, as the case may be. The _Receive primitive is used if the receiving task is to wait for the message. If not, the _Try_Receive primitive is used. Only if it is receive-specific is the *id* of the sending task provided along with these primitives. After the _Receive() or _Try_Receive() functions return, the receiving task sets up in space a reply message pointed to by the rply argument and calls the _Reply primitive with the *id* of the sending task. The _Reply primitive informs the sending task that the message is transferred.

Blocking primitives enable the communicating tasks to synchronize in addition to exchanging information between them. Asynchronism between the two communicating tasks is accommodated but at the expense of parallelism. The task that executes faster has to wait for the slower task. Synchronization of tasks is dependent on the blocking behaviour of these primitives. The _Send() and _Receive() are blocking primitives, while the _Try_Receive() and _Reply() primitives are nonblocking. Implementation of this form of interprocess communication does not result in extensive message queuing in either of the communicating tasks. Some of the issues in message passing such as blocking, addressing mechanism, message format, and communication failures are discussed in detail in [41].

The methodology, in Harmony, of configuring the software is referred to as Database and Selectors Cel (DaSC) [42] and addresses two issues:

1. Managing the different versions of a program at a given time in order to minimize redundancy and thereby save storage and facilitate maintenance.
2. Coping with the different versions as they evolve over time.

Database and Selectors deal with the former issue while the latter is dealt with by Cel. A tree-structured file system is chosen as the database with the selectors being the pathnames to the various files. An application written in the Harmony environment is split across several files in accordance with this source management scheme. Each application program is divided into *src*, the source code for that program, and *inc*, the sets of selectors for that program. By confining the selectors to the inclusion directory, portability across host systems is achieved.

Evolution of the various versions of a program over time is likened to the process of film animation where different transparent layers of images are superimposed one on top of another to yield the desired image. In Harmony, the Cel is implemented as three layers of tree-structured file systems having a common directory structure. The Master layer contains the most recent release and is write-protected to preserve its integrity. The Derived layer contains the output of derivers such as compilers and linkage editors. The Working layer contains the source and inclusion files that are being developed for an application. Consolidation of the Master layer, usually done at the time of a release, involves the incorporation in the Master layer of appropriate changes which were done in the Working layer.

The source code is developed for the various tasks that form the application. These tasks may run on one or more processors depending upon the needs of the application. An inclusion file, built for the various tasks that run on a given processor, contains the source files of these tasks. These files are compiled individually and then linked to yield executable images. The *fixaddr* tool converts Macintosh (host, in this application) executable images into a Unix-type ".out" form. Once this is done, the executables are downloaded to the target system.

General debugging techniques used in sequential programs are not efficacious when applied to realtime systems, as they require the entire processing to come to a halt. *Debug* is a Harmony debugger which runs on one processor in the target system and supports debugging on all active processors. Interactive debugging is invoked when either a breakpoint is encountered or an exception or abort occurs. In such a situation the corresponding task is stopped and interactive dialog takes place between the user and that task without affecting the other tasks. The user terminates this dialog once debugging is complete and the task is restarted. During the debugging session breakpoints can be planted dynamically provided the debug control server and the clock server are created and initialized.

When the debugger is used via a terminal emulator on the Macintosh II, there is a rapid access to the disassembled binary executable file through a companion tool, *Examine*, as well as to a set of source files through the Macintosh II-based multiwindow editor. *Bound* is another Harmony tool used for determining the size of the stack frame for a specified function. A useful non-interactive debugging tool called *_Log_gossip* prevents the messages from several tasks, executing in parallel, from becoming intermingled when dumped onto the screen.

4. 3 Robot Control – ALTER

The PUMA 560 robot system used in this research consists of a controller, robot arm, system software, CRT, teach pendant, and a floppy drive [43]. The controller is based on a DEC LSI-11 processor. A six-degree-of-freedom PUMA 560 robot arm, used in this work, is shown in Fig. 6. The system software that controls the robot arm is called

VAL (Vicarm Assembly Language) II and is resident on an EPROM inside the controller. It consists of simple instructions for writing and editing robot programs. It supports on-line as well as off-line programming and has features that can support the use of sensors in the robotic environment. Robot motion is accomplished, either by issuing VAL II instructions through a CRT (connected to the controller by an RS-232C interface) or with a teach pendant. A floppy drive allows permanent storage of the VAL II programs and robot locations on floppy disks.

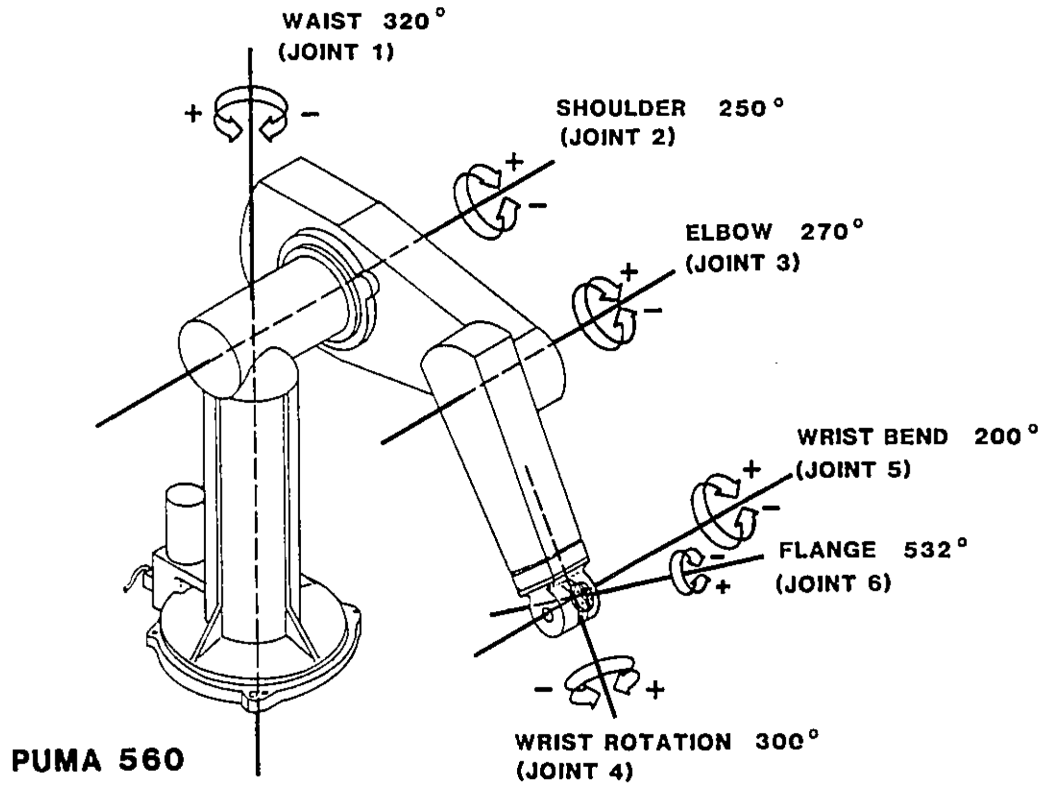

Fig. 6 The PUMA 560 robot.

Reprinted from — Equipment Manual for the 500 Series PUMA Mark II Robot: 398P1, Unimation Inc., April 1984.

To facilitate working in an unstructured environment, VAL II provides a feature that enables realtime path control while the robot is moving. When this feature is utilized, VAL II is in the alter mode. There are two types of alter mode, namely, external alter mode and internal alter mode. In the external alter mode, data for realtime path control are provided by an external computer. The controller sends messages (alter requests) once every 28 ms, requesting information to control the trajectory of the manipulator. The external computer responds by issuing motion requests which, in effect, are incremental motions that the robot should carry out. In an internal alter mode, a user-written VAL II program executing in parallel with the robot-control program provides the data for realtime path modification. The robot can be controlled in either the world coordinate system (Fig. 7) or in the tool coordinate system (Fig. 8). The data provided to the controller for realtime path modification can be in the cumulative or non-cumulative modes. In the cumulative mode, the robot moves from an initial reference point by an amount equal to the sum of all past alter data. In the non-cumulative mode, the robot moves by an amount equal to the most recent alter data with respect to its initial reference point.

In this work, the robot moves in the cumulative mode and in the tool coordinate system. The reason for the choice of this coordinate system will be given in Section 5.6. The external alter mode is employed and data produced in the external computer, based on the sensory feedback, are generated as increments in Cartesian space. These data provide online modifications to the planned path of the robot and are sent to the controller over ALTER, a high speed serial port. Further details on the functionality of ALTER may be found in [35, 44].

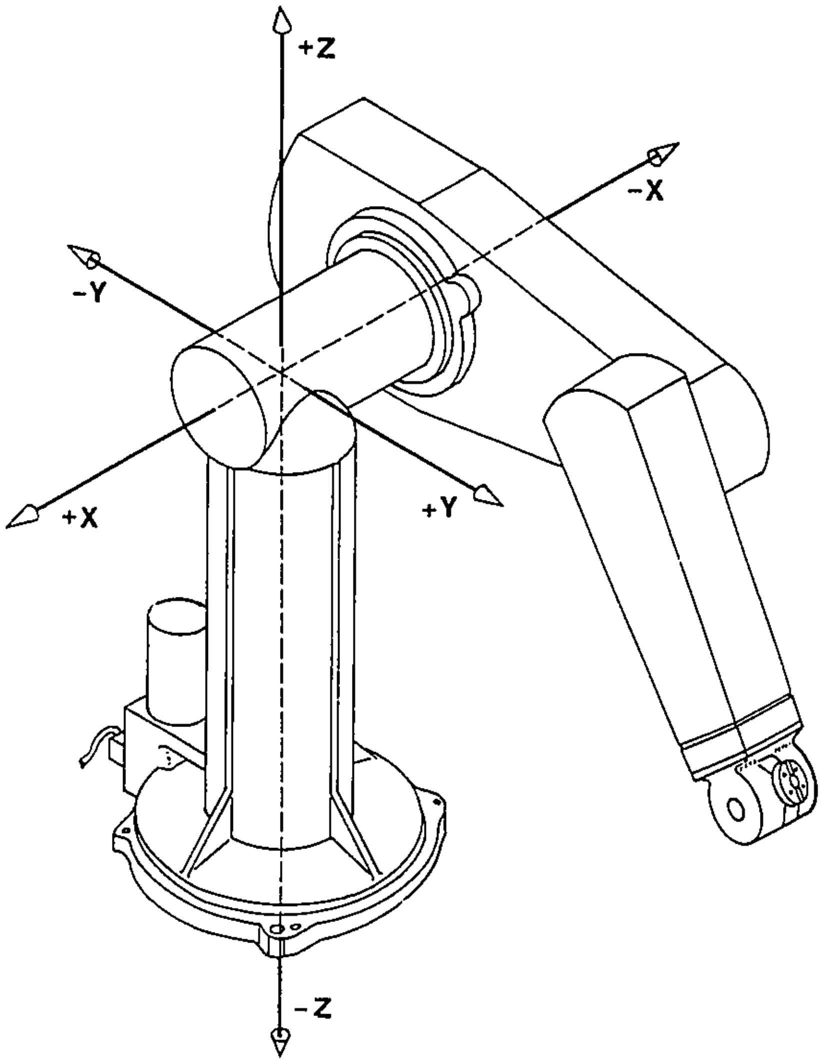

Fig. 7 World coordinate system of the robot.

Reprinted from: Equipment Manual for the 500 Series PUMA Mark II Robot, 398P1, Unimation Inc., April 1984.

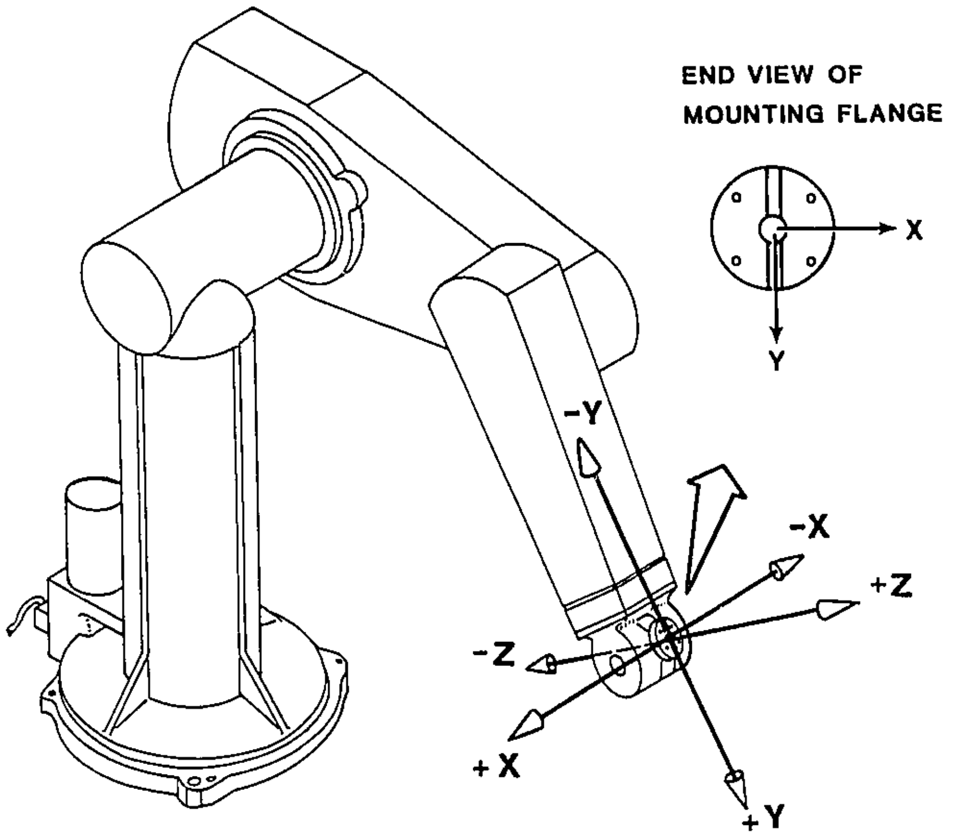

Fig. 8 Tool coordinate system of the robot.

Reprinted from: Equipment Manual for the 500 Series PUMA Mark II Robot, 398P1, Unimation Inc., April 1984.

Chapter 5

Realtime Tracking

5.1 Design Issues

The complexity of realtime sensor-based robotic applications requires that the problem be handled in stages. Albus *et al.* [45] stress the need for hierarchical control, in such applications, by modularizing the problem. The result is that such modules are not too complex to handle. A MIMD architecture, used in this work, is more suitable for realtime processing [46]. In this architecture a single job is distributed over a number of processors, each able to execute its own code on arbitrary data, independent of what the others are doing.

Baldwin [47] points out that the development in software technology has not kept pace with that of hardware technology and stresses the need for programming languages that allow a user to exploit much more parallelism than is presently possible with existing languages. The need for concurrency, through parallelism, is echoed by [1]. As a result, partitioning of an application into tasks, in an effort to achieve parallelism, is left to the software designer. Concurrency is achieved by exploiting the multitasking, multiprocessing nature of the Harmony operating system. Parallelization can be done at many different levels of granularity. A consequence of parallel programming is that much effort is spent on properly starting processes, synchronizing them, and passing data between them. As these operations are expensive, an effective use of multiprocessors can be achieved by ensuring fairly coarse-grained parallelism in programs with minimum communication among the processes.

In this application, fetching data from the RPSs and controlling the robot are time-critical tasks. Data are fetched alternately, every 38 ms, from each of the two RPSs. Motion requests are to be provided to the controller every 28 ms for modifying the trajectory of the robot. Thus, the tasks that perform these routines should be given the highest priority over other tasks. Since the application is to be comprised of several tasks, there is bound to be communication between them in order to realize the objective. The different tasks execute asynchronously and any communication between them would most likely result in these tasks being blocked. While such a behaviour restricts parallel processing, the software can be designed to minimize or eliminate the time for which a task is blocked. A good software design ensures that the interprocess communication is kept to a minimum.

While keeping the amount of interprocess communication to a bare minimum, care should be taken to avoid scenarios where two tasks issue a _Send() primitive to one another and, consequently, get blocked forever. This is tantamount to a cyclic graph, which is a potential for a deadlock. The design can be deadlock-free if the _Send() primitives are maintained in a single direction. Too much communication would only add to the complexity in the design of a deadlock-free system. If there is to be a dialog with a time-critical task, care should be taken to ensure that such a communication does not jeopardize the performance of the system. In this work, such a situation can arise if a task fetching data from the RPS or one that controls the robot is blocked.

5.2 Pseudocode

The steps involved in achieving the objective (Chapter 3) are:

1. Calibrating the range profiles from the RPSs.
2. Validating the calibrated profiles.
3. Processing these profiles:
    a. Detecting the presence of an object in the profile.
    b. Determining the center point on the object to be tracked in $X, Y,$ and Z with respect to the RPSs.
    c. Determining the orientations (α) and (β) of the object with respect to the RPSs.
    d. Computing the relative motion parameters for the end-effector.
4. Passing the motion parameters to the robot controller.
5. Repeating steps 1 to 4 indefinitely.

5.3 Calibration

The field of view of an RPS is shown in Fig. 9. The RPS yields data that are in range-azimuth pairs. Each profile is comprised of 256 azimuth positions, corresponding to the angles of projection of the laser beam from the RPS. For a given azimuth position, the set of all range values in the field of view of the RPS constitutes an azimuth contour. The different azimuth contours project radially outward from the RPS. A range contour, shown as a concentric curve, is made up of a set of azimuth positions having a common range value.

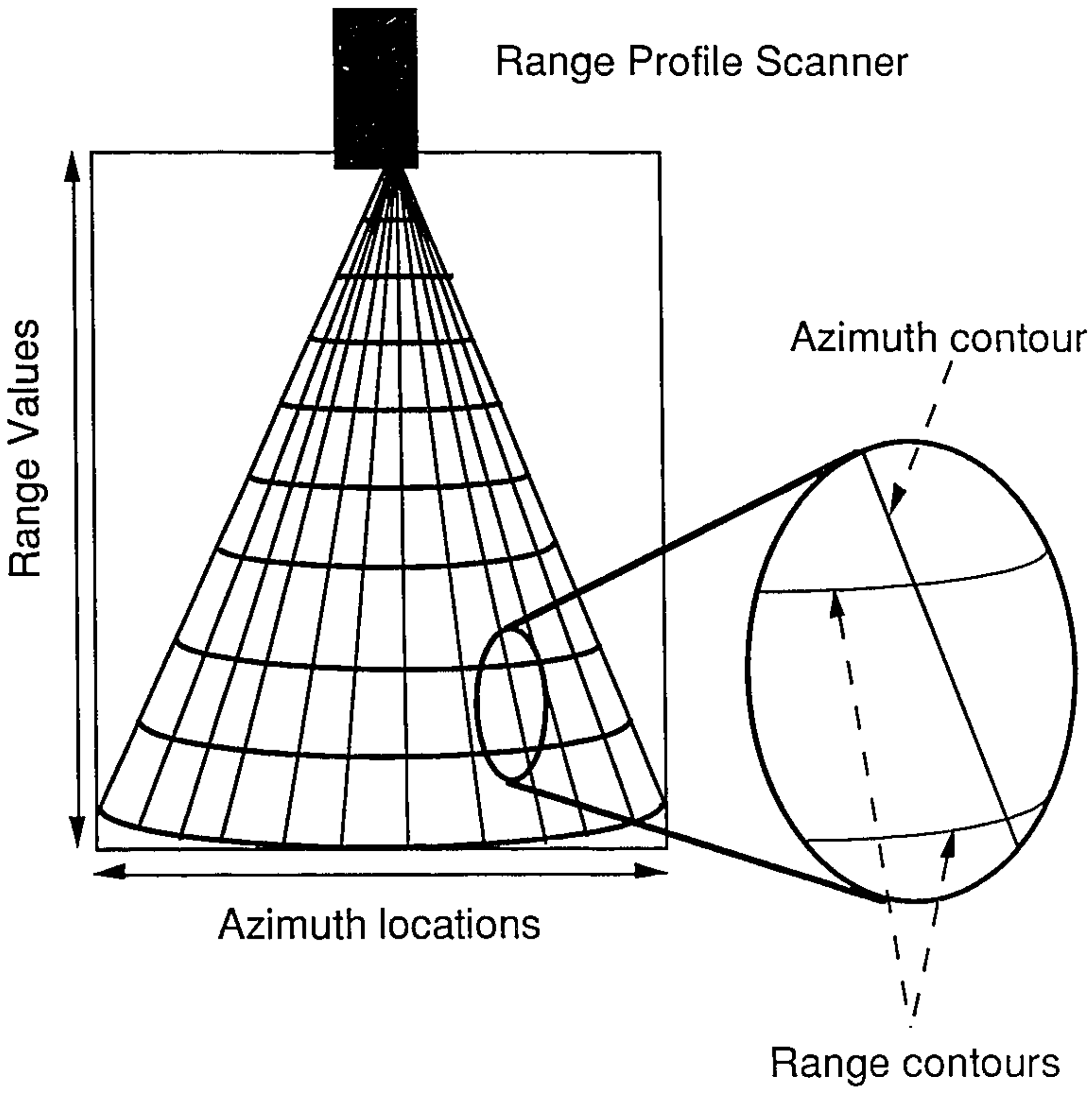

Fig. 9 Field of view of the RPS

After: C. Archibald and S. Amid, "Calibration of a Wrist-mounted Range Profile Scanner," *Proceedings of Vision Interface '89*, pp. 24-28, London, Canada, June 1989.

Calibration is required to convert these range-azimuth pairs in the distorted polar coordinate system to the more useful X-Z pairs in the Cartesian coordinate system. It is done in two stages:

1. A lookup table (LUT) is created which yields the Z values for the different range-azimuth pairs.
2. The X value of a certain range-azimuth pair is determined using the value of Z at that position (from the LUT) and the equation of the azimuth contour passing through that point.

The LUT is created with the help of a calibration bench. The RPS, mounted on this bench, is moved up or down to yield different Z values. For each of these Z values, read from the bench, the range-azimuth pairs of the various points on the profile are noted from the scanner hardware. In order to determine the range and Z values of those points on a given azimuth contour between two adjacent Z levels, linear interpolation is used. Thus, the Z value of a point in space can be obtained directly from the range-azimuth pair, by using the LUT.

Azimuth contours are radial lines extending outward from the RPS. For a given azimuth contour, knowledge of the X values at two different Z values is sufficient to yield the equation of that azimuth contour. Since there are 256 azimuth contours for every standoff, there are, correspondingly, 256 X values. This set of X values is determined for several different standoffs (although two values would suffice) to obtain the equation of the azimuth contours, shown graphically in Fig. 10. Since each azimuth contour is a straight line it is characterized by a unique *slope* and *intercept*. By knowing the Z value at the azimuth position of interest, the X value can be determined using these parameters. A detailed description of the way an X value is determined for a given Z value can be found in [48].

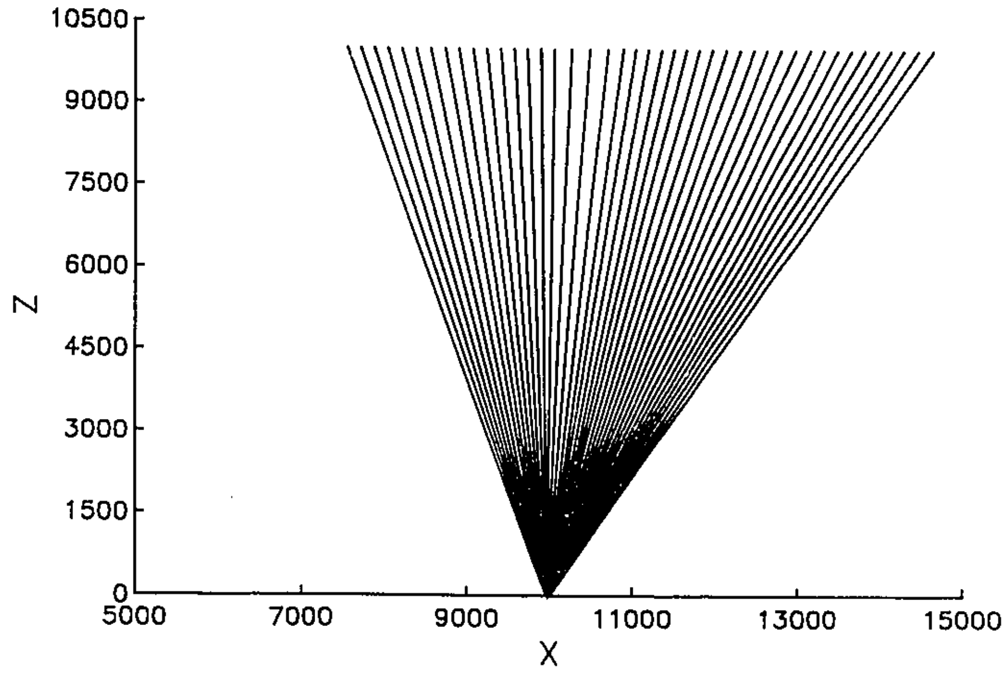

Fig. 10 Determing the X values from the azimuth contours

Reprinted from: C. Archibald and S. Amid, "Calibration of a Wrist-mounted Range Profile Scanner," *Proceedings of Vision Interface '89*, pp. 24-28, London, Canada, June 1989.

5.4 Validation

The calibrated data fetched from the RPSs occasionally contain invalid data due to shadow effect or spurious reflections. A typical range profile which has a shadow is shown in Fig. 11. Validating the data ensures that these surfels are not used in the computation.

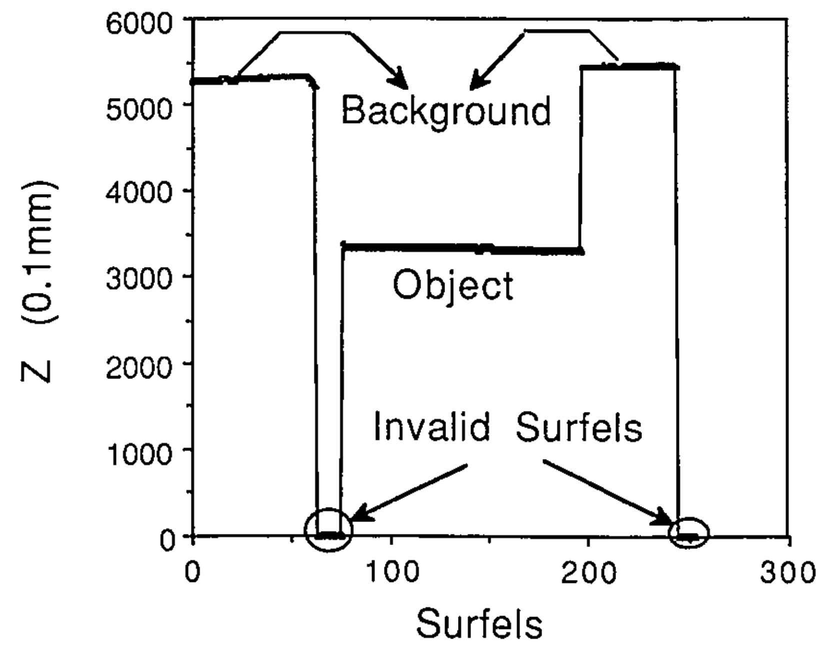

Fig. 11 Calibrated range profile

Range data obtained from the RPSs are 16 bits long. Bit 15 indicates the RPS from which data are being fetched while bit 14 is used to determine if the CCD position sensor has registered a range value. If a shadow is detected this bit is set and as a result bits 0 to 13 are cleared. The range data are logically ANDed with the number *hex 3fff* so as to strip off the flag bits and obtain the range values. These range values are checked for bounds specified by the user. If they are within these bounds, they are treated as valid. If not, the corresponding surfels are flagged as invalid. Figure 12 shows the profile from Fig. 11 after validation.

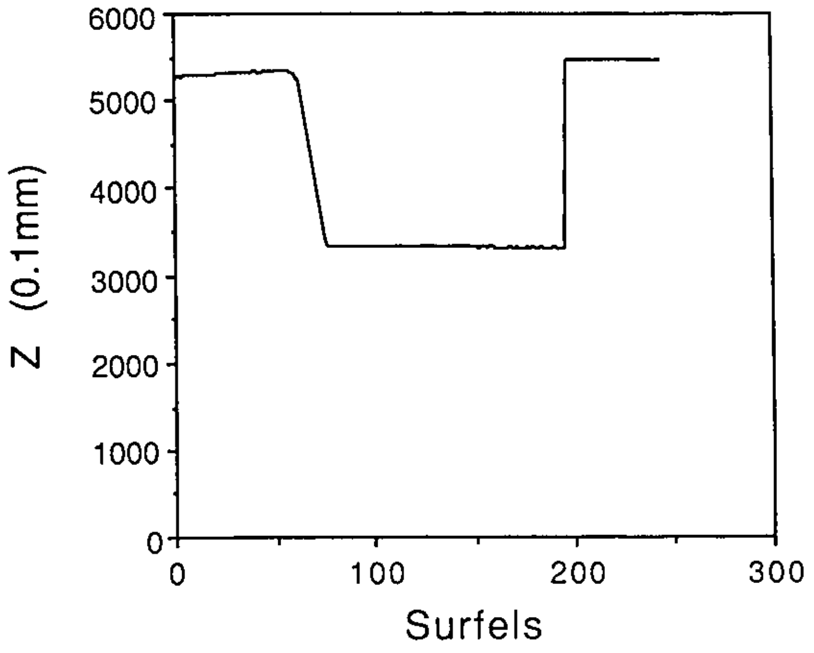

Fig. 12 Validated range profile

5.5 Software Design

This section presents the method of solving this problem in a sequential manner. In order to meet the realtime constraints present in this work, parallelism has been incorporated into the design of the tracking application. Parts of the problem that can be done in parallel are identified and implemented in a manner consistent with the constraints imposed by the computing system. The parallelized design is presented in Section 5.6.

Range data fetched from the RPSs are calibrated and validated, as discussed in Sections 5.2 and 5.3. These data are in the form of coordinate pairs (XZ or YZ) and are measured from the perspective of the RPSs (see Fig. 2). As a result, objects that are closer to the RPS are at a shorter standoff than those that are farther away. A typical profile, Fig. 12, gathered by the RPS with a flat object in its field of view illustrates this fact. The horizontal line with a smaller value of Z represents the object while the horizontal line with a larger value of Z represents the background, a bench in this case.

In determining the presence of an object, three possible scenarios exist (as shown in Fig. 13):

1. The RPS sees the entire object — jump in the range data occurs twice.
2. The RPS sees a part of the object — jump in the range data occurs once.
3. No object is seen by the RPS — no jump in the range data occurs.

In the first two cases, the part of the valid profile that lies on the object is called the *target*. In the first case, as the entire object is seen by the RPS, the target and the object are the same. The profile of the target, after the background data in Fig. 12 are removed, is shown in Fig. 14. In the second case, as only part of the object is seen by the RPS, the target is only a part of the actual object. Once tracking commences, the robot moves in order to achieve the objective described in Chapter 3. In doing so, the target and the object become the same and the RPSs on the wrist of the robot have a consistent view of the object. In the third case, where no object is detected, the tool of the robot aligns itself along its Z-axis at the required standoff from the background without moving in the other four degrees of freedom.

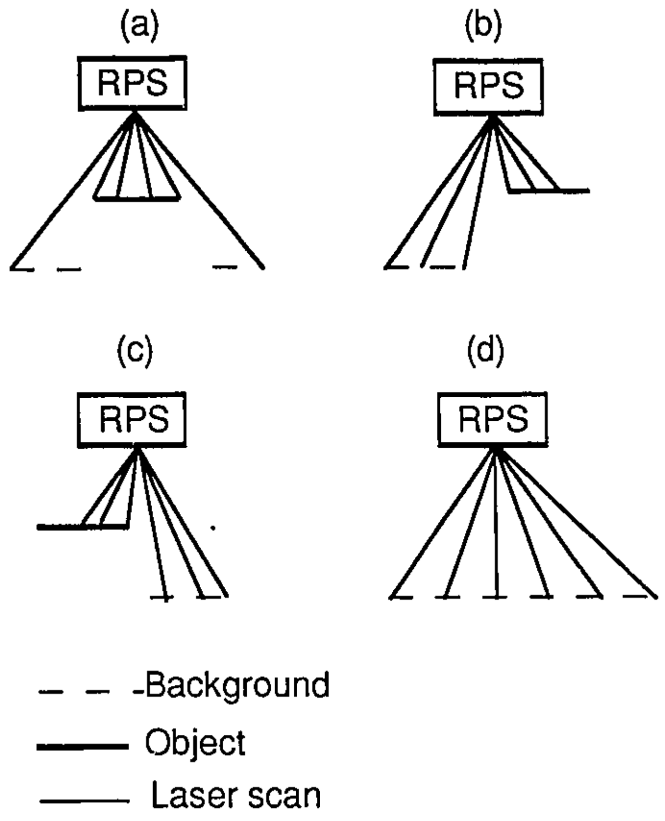

Fig. 13 Different scenarios of a flat object in the field of view of the RPS

(a) The RPS sees the whole object. (b) Only a part of the object is seen.

(c) Only a part of the object is seen. (d) Object is not visible to the RPS.

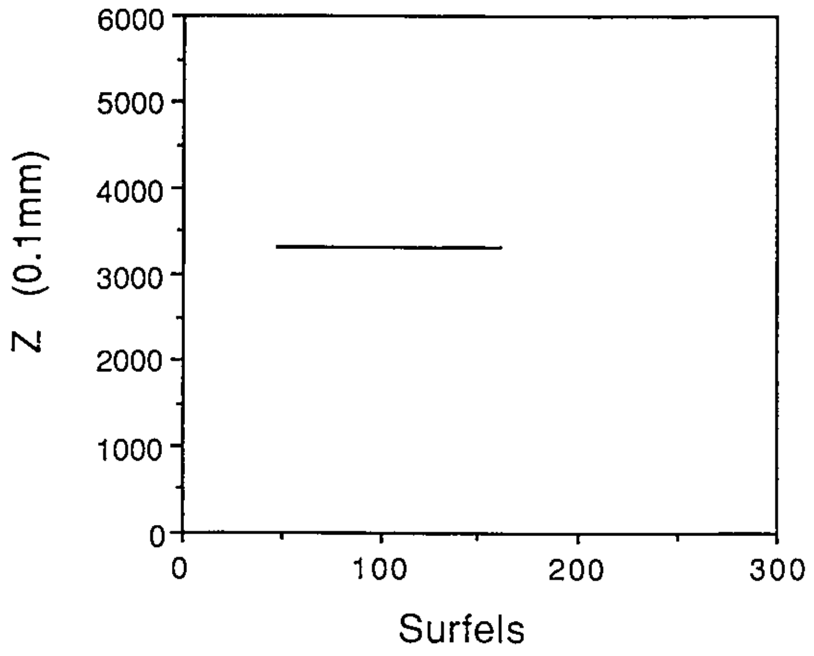

Fig. 14 Valid range profile on the target object

When an object is detected, the orientations (α) and (β) of the object in the XZ and YZ planes are determined using a gradient operator. This operator is applied every a surfels to the profile in the XZ plane and every b surfels to the profile in the YZ plane. The results of this operation on successive sets of surfels on the object in the XZ plane are averaged to yield the pitch, β, of the object. In a similar manner, the yaw of the object, α, is determined. The coordinates of the center of the target in the local coordinate system of the RPSs, (X_t, Z_t) and (Y_t, Z_t), are calculated as the average of all the XZ and YZ pairs of those surfels. These are calculated as follows:

$$\alpha = tan^{-1}[\{\sum_{i=1}^{\lfloor n/b \rfloor}([Z_{(ib+1)} - Z_{((i-1)b+1)}] / [Y_{(ib+1)} - Y_{((i-1)b+1)}])\} / (\lfloor n/b \rfloor)] \tag{1}$$

$$\beta = tan^{-1}[\{\sum_{i=1}^{\lfloor n/a \rfloor}([Z_{(ia+1)} - Z_{((i-1)a+1)}] / [X_{(ia+1)} - X_{((i-1)a+1)}])\} / (\lfloor n/a \rfloor)] \tag{2}$$

$$X_t = (\sum_{i=1}^{n} X_i) / n \tag{3}$$

$$Y_t = (\sum_{i=1}^{n} Y_i) / n \tag{4}$$

$$Z_t = (\sum_{i=1}^{n} Z_i) / n \tag{5}$$

The value of Z_t is obtained from both the RPSs and a decision is to be made as to which one to choose. As only flat objects are dealt with in this work, the presence of an object in the field of view of the RPSs should yield identical values of Z_t as seen by each of the two RPSs. However, it is possible that if the object moves faster than the robot, it is only in the field of view of one of the RPSs. The other RPS does not see any object. In this case, the RPS that sees the object yields a value of Z_t that corresponds to the distance to the object, while the RPS that does not see the object yields a value of Z_t that corresponds to the distance to the background. The lower of these two values represents the distance to the object and is the obvious choice.

The actual motion to be carried out is the difference between the present and previous poses of the object. Ideally, the amount to be traversed along the X, Y, and Z directions are, respectively; the difference between the center of the range profile in the XZ plane and X_t; the difference between the center of the range profile in the YZ plane and Y_t; and the difference between the required standoff and Z_t. The centers of the range profiles in the XZ and YZ planes are determined off-line. These values, along with the desired standoff, are fed as parameters to the tracking application. Since the robot is to be normal to the plane of the object, the amounts of rotation required about the target point (the center of the object) in the XZ and YZ planes are β and α.

The values sent to the robot controller, however, are not the actual values as computed above but a certain percentage of those values. This is due to the asynchronism between the rate at which vision updates are received and the rate at which the robot controller expects motion increments. In Chapter 6 it will be shown that the vision update cycle time is 76 ms. Since the robot takes 28 ms to servo to a particular point, there are two or three robot setpoints corresponding to every vision update. This uncertainity in the number of setpoints per vision update is due to the fact that 76 is not a multiple of 28. This implies that if the robot is to perform the actual motion of the object, as computed above, there are two options to be considered for subsequent setpoints in the absence of visual feedback. It could repeat the previous motion or it could idle. While the first option results in a continuous motion, the robot would overshoot and lose sight of the object. The second option results in a jerky motion and is not preferred.

A modified version of the first option was used. The problem to be addressed was that of the robot overshooting and losing the object. This was solved by allowing the robot to move by a fraction of the motion of the object so that the robot is still in sight of the object even after the same motion is executed subsequently. Since the number of setpoints per vision update cycle is either two or three, instructing the robot to move by either a half or a third of the motion of the object results in a very jerky motion on the part of the robot. The optimum percentage that the robot is allowed to move is determined by studying the performance (to be discussed in Chapter 6) of the tracking application.

While the target is moving in the XZ and/or YZ plane, the robot carrying the RPS is also moving, rotating and translating in order to satisfy the objective. Hence, the frame of reference of the RPS, which is in the tool frame of the robot, is also in motion. Consequently, successive profiles do not yield the same range data and the resulting motion varies. As the object is capable of accelerating or decelerating, the software has been designed to emulate the behaviour of the object, within the capabilities of the robot. Parameters are incorporated in the software that allow the robot to accelerate/decelerate and approach speeds dictated by the limits associated with these parameters. This is done for both translational as well as rotational motion. These limits are imposed to keep the system critcally damped [49, 50].

5. 6 Implementation

The complex nature of tasks in robotic applications calls for a decentralized control. Gauthier *et al.* [51] refer to these tasks as workcells in a distributed environment that use the interprocess communication facility to accomplish certain objectives. The various tasks used in this application along with the communication between these tasks are depicted in Fig. 15.

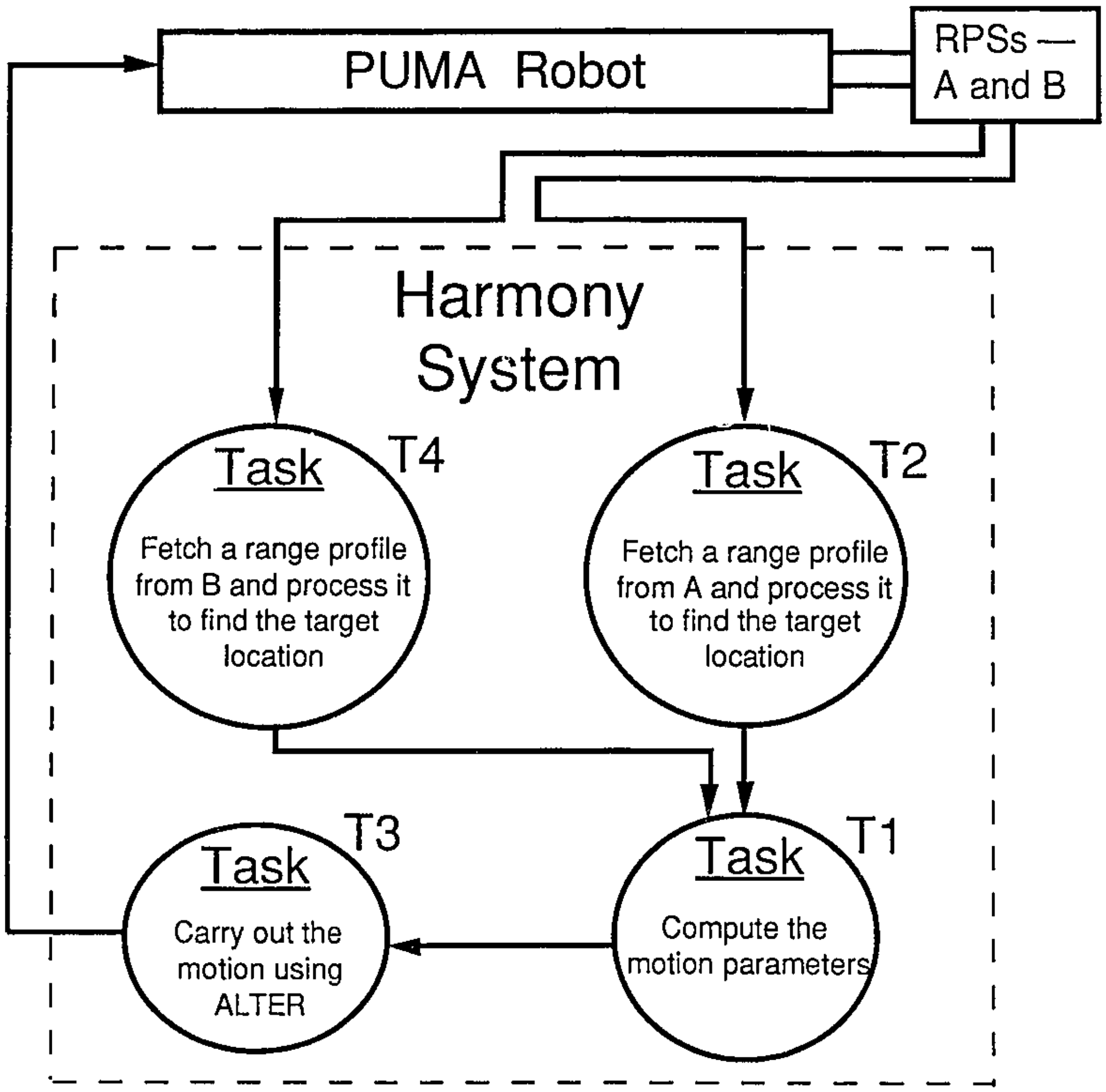
PUMA Robot
RPSs — A and B
Harmony System
Task T4
Fetch a range profile from B and process it to find the target location
Task T2
Fetch a range profile from A and process it to find the target location
Task T3
Carry out the motion using ALTER
Task T1
Compute the motion parameters

Fig. 15 Software design – Distribution of tasks.

There are two time-critical tasks involved – communication with the robot every 28 ms via the ALTER port and collection of a range profile every 38 ms. These time constraints, imposed by the robot controller and the RPS, are met using four processors and the Harmony operating system. Two tasks, T2 and T4, are dedicated to the collection of range profiles from the RPSs A and B respectively. Since the RPSs scan in an alternate manner, when A collects a profile of the surface, B does not and vice versa. Thus, task T2 collects a profile from A and idles for the next 38 ms before collecting another profile from A. The same is true of the task T4 with regard to collection of profiles from B. In order to achieve optimum efficiency, each of the tasks T2 and T4 validate the range profiles collected by them. These validated range profiles are processed to determine the values X_t, Z_t, and β, in the case of T2, and Y_t, Z_t, and α, in the case of T4. These values are then sent from T2 and T4 to T1.

Upon receipt of these values from T2 and T4, the task T1 chooses the appropriate Z_t and computes the motion to be carried out (both of which have been discussed in Section 5.5) to meet the required objective. The object being tracked may accelerate or decelerate. In order for the robot to emulate the motion of the object, acceleration/deceleration parameters have been incorporated in the tracking algorithm. If the sensory feedback indicates that the object is continuously accelerating or decelerating the robot is allowed to move by increasing or decreasing amounts every setpoint. In the case of an object that is accelerating, there is a ceiling on the amount to which the robot can accelerate – the maximum speed parameter. If the object accelerates any further, the robot tracks the object at this maximum speed, provided it does not lose sight of the object. The tracking algorithm is also capable of quickly adapting to a situation where the direction of motion of the object changes. Such a situation is detected when the motion being computed differs in sign from the previous motion.

A *handler* task is created by the task T3 to receive motion requests from different clients. In this work, only one client, task T1, sends motion requests. Motion requests received by the *handler* are pushed onto a stack. The *handler* creates the *courrier* task which, in turn, creates the *server* task. The *courrier* and *handler* tasks are designed to assist the *server* task, as will be seen below. The sequence of steps followed from the time of issuance of a motion request to the time when the motion is executed are listed below. Figure 16 gives a pictorial representation of this sequence. The numbers indicate:

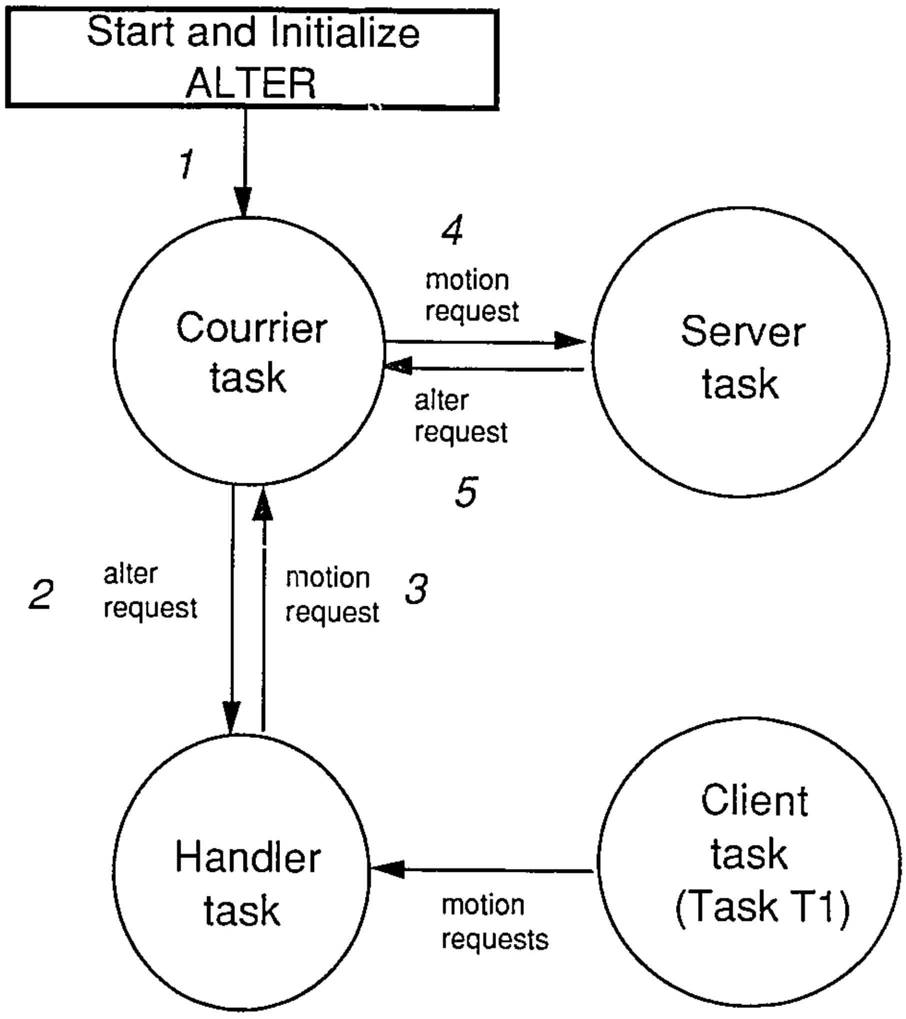

Fig. 16 Sequence in which a motion request is executed

1. The ALTER communication line is started and initialized.
2. The *courrier* task sends an alter request to the *handler* task.
3. When the *handler* receives an alter request it checks the stack for any motion request. If the stack is empty an *idle* motion is sent to the *courrier*. If not, the motion request on the top of the stack is sent to the *courrier*.
4. The *courrier*, upon receipt of the motion request from the *handler*, forwards it to the *server* task and is blocked until the actual motion is done.
5. The *server* replies to the *courrier*, after the execution of the motion, along with an alter request.
6. Steps 2 through 5 are repeated.

Figure 17 explains a significant synchronization problem that was faced in this work and how it was overcome. In the original design, a motion request from T1, in the form of a _Send(), would be unblocked only after the actual motion was performed. This meant that T1 was blocked for the entire robot setpoint interval of 28 ms. This was a serious constraint on T1 – motion requests could only be issued at robot setpoints. If this constraint was not met, T1 would be blocked till the next robot setpoint. The blocking period could be anywhere from 1 to 27 ms depending on when the motion request was issued. Tasks T2 and T4 were, in turn, blocked by T1 causing a synchronization problem with the RPSs as well.

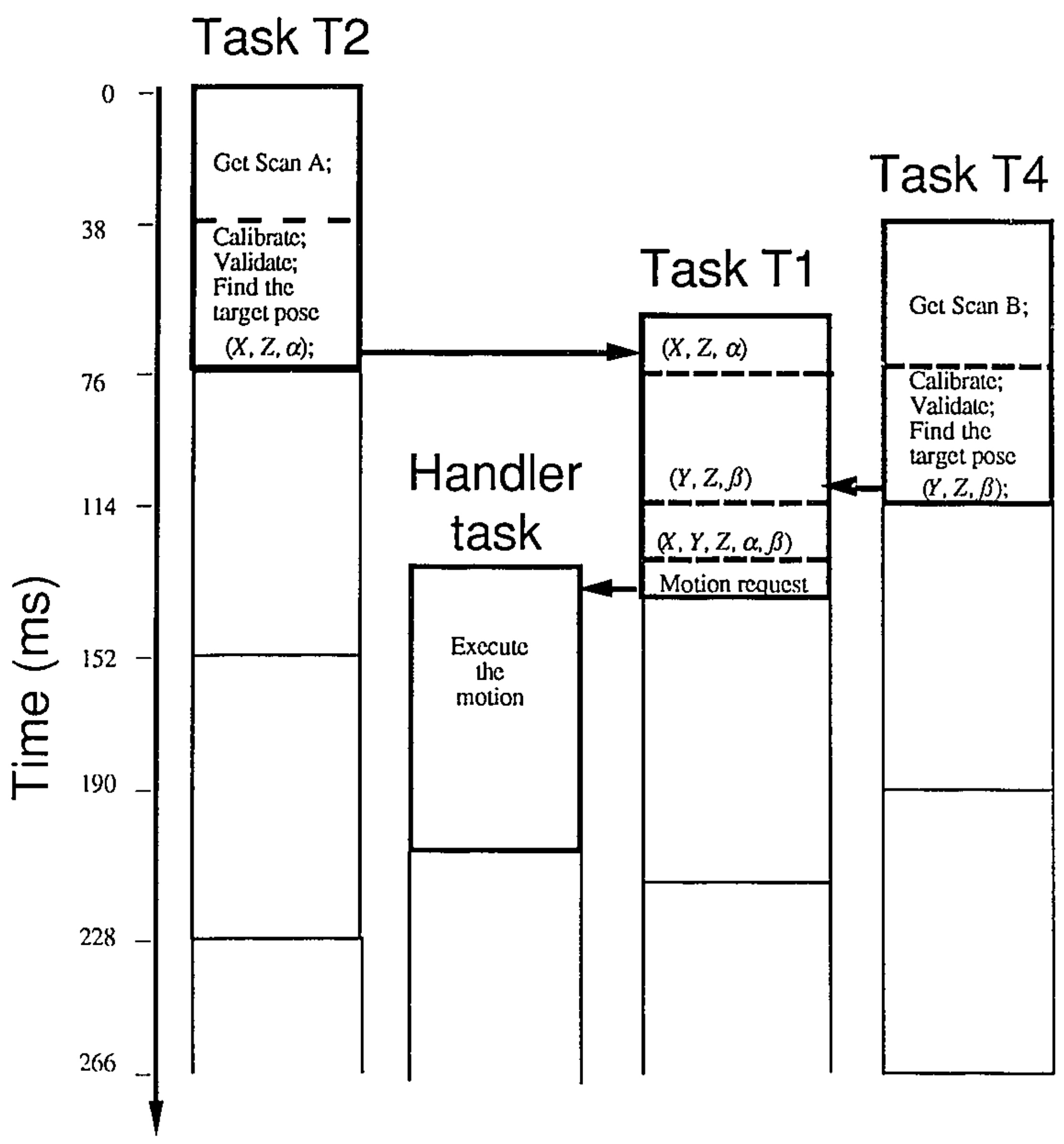

Fig. 17 Communication among tasks in the time domain

This synchronization problem is resolved by a slight modification of the software. The motion requests received from T1 are pushed into a stack by the *handler* and an immediate reply is sent to T1 thus unblocking T1 as well as tasks T2 and T4 that are send-blocked on T1. In this manner, the constraint on T1 is removed and issuance of motion requests need not correspond to the robot setpoints. The stacking of motion requests does not change the behaviour of the robot as long as the vision update cycle time is greater than 28 ms (which is the case), as will be shown in Chapter 6. This is because the rate at which the stack is filled with motion requests is slower than the time taken by the robot to carry out each of these motion requests. Thus, at any given time, there is only one motion request in the stack that is repeated until a new motion request is received.

The robot can be controlled to move either in its world coordinate system (Fig. 7) or in its tool coordinate system (Fig. 8). Since the RPSs are wrist-mounted and since all range measurements are done from the perspective of these RPSs, the tracking application is done in the tool coordinate system of the robot. To justify this choice, consider a scenario where the frame of reference of the RPSs and the world coordinate system of the robot are identical except for an angular shift (twist) about the Z-axis. Thus the XZ and YZ planes of the frame of reference of the RPSs differ from those of the world coordinate system of the robot by an amount equal to the above angular shift. If the world coordinate system of the robot was chosen for the tracking application, a displacement of the target object in the XZ plane of the frame of the RPSs will result in the robot moving in the XZ plane of the world coordinate system and since these two XZ planes are not identical the motion is erroneous. Such an error will not be found if the motion is performed in the tool coordinate system. This is because the frame of reference of the RPSs is always in the tool coordinate system of the robot, both while the robot is stationary as well as when it is either translating or rotating.

The default position of the tool coordinate system is that of the mounting flange (Fig. 8). Thus, any translational or rotational motion that is to be performed is done with respect to this point. However, tracking a flat object requires that the manipulator rotate and translate about the center of the object, rather than about the flange. Hence, a tool transformation is done, using a VAL II instruction, that results in the tool coordinate system being at the center of the object.

Chapter 6

Results

A videotape of the tracking experiment, enclosed with this thesis, gives a qualitative description of this research. The robot tracks a target object in 3-D space without any jerks. Three target objects are used in this work. They are:

1. A triangular plate attached to a pole in order to move it in a random motion in a plane (XZ, in this case).
2. A circular disk attached to a pole for random motion.
3. A circular disk that is driven by a 12 V D.C. motor. This allows the disk to be rotated in a periodic manner. The speed of rotation of the disk can be controlled by applying different voltages to the motor.

Time-sequenced photographs (Figs. 18, 19, and 20) depict the behaviour of the robot when realtime tracking is done with the three objects mentioned above.

The quantitative analysis of the tracking experiment is done in two parts:

1. The motion behaviour of the robot is studied corresponding to different motion behaviours of the object – Section 6.1.
2. The times of execution of the various tasks in the tracking application is observed. This study (to be discussed in Section 6.2) is helpful in determining the optimum software design for this application. The synchronization problem discussed in Section 5.6 was identified with the help of this study.

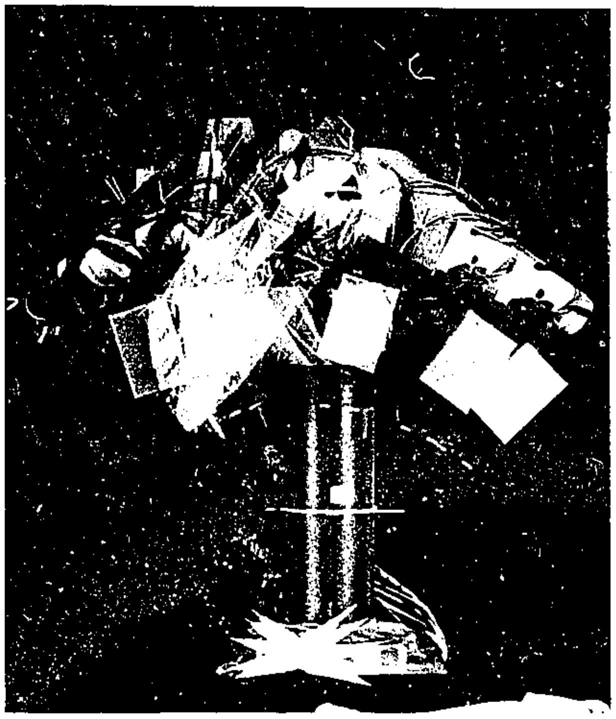

Fig. 18 Time-sequenced photograph of realtime tracking in three degrees of freedom i.e. X, Z, and β. A triangular plate is the target object.

Fig. 19 Time-sequenced photograph of realtime tracking in five degrees of freedom with random motion of a hand-held circular disk.

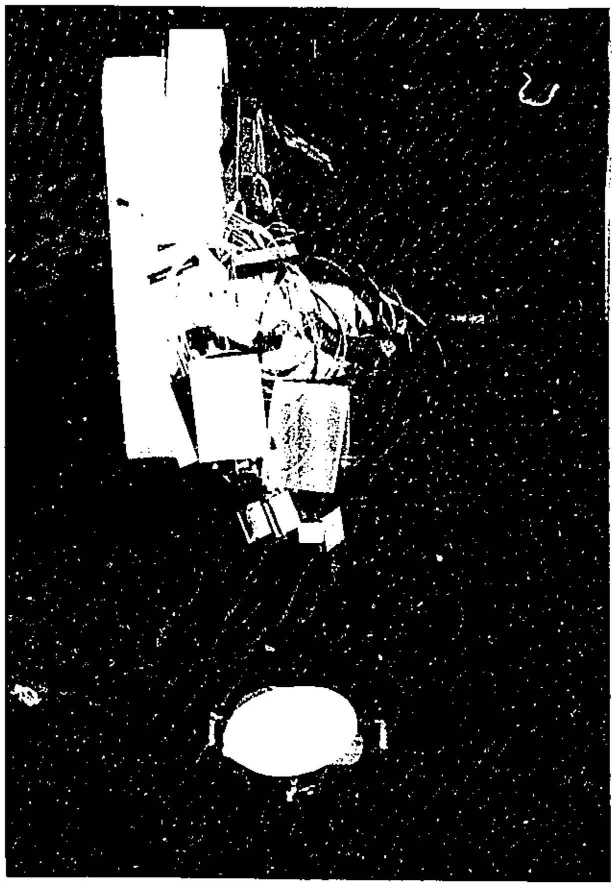

Fig. 20 Time-sequenced photograph of realtime tracking of a motor-driven circular disk.

6.1 Performance Study

Different values are assigned to the various parameters used in the tracking experiment, in order to decide on a set of values that yields a trajectory that is continuous without any compromise on the response time of the robot. After several repetitions, the following values are chosen for the different parameters:

1. Percentage of the motion of the object that is performed by the robot at each setpoint:	5%
2. Maximum linear acceleration of the robot: (It can accelerate by 6 mm, every 28 ms)	7.65 m/s^2
3. Maximum angular acceleration of the robot: (A maximum acceleration of 3^o, every 28 ms, is allowed)	66.8 rad/s^2
4. Maximum linear speed of the robot: (It can translate 15 mm, every 28 ms)	53.6 cm/s
5. Maximum angular speed of the robot: (A maximum rotation of 3^o, every 28 ms, is permitted)	1.87 rad/s
6. Value of *a* (used in Equation 2 in Section 5. 5)	23
7. Value of *b* (used in Equation 1 in Section 5. 5)	15

All of the results presented later are obtained by performing the tracking experiments with the above set of values. In the ensuing graphs, the behaviour of an ideal tracking system is represented by bold lines. Since range measurements are done in the frame of reference of the RPSs, an ideal tracking system always views the object as horizontal. Moreover, in such a system, the end-effector is always at the center of the object at the required standoff. The reponse time of the robot to meet the required objective, using the above set of values, is studied for linear as well as angular motion. An object is placed at a depth of 57 cm, in the Z direction, from the tool of the robot and is oriented in the XZ plane by 16^o from the perspective of the RPSs. The time taken by the robot to achieve a standoff of 35 cm from the object, in the Z direction, is shown in Fig. 21. Figure 22 shows the behaviour of the robot as it rotates in order to be normal to the plane of the object.

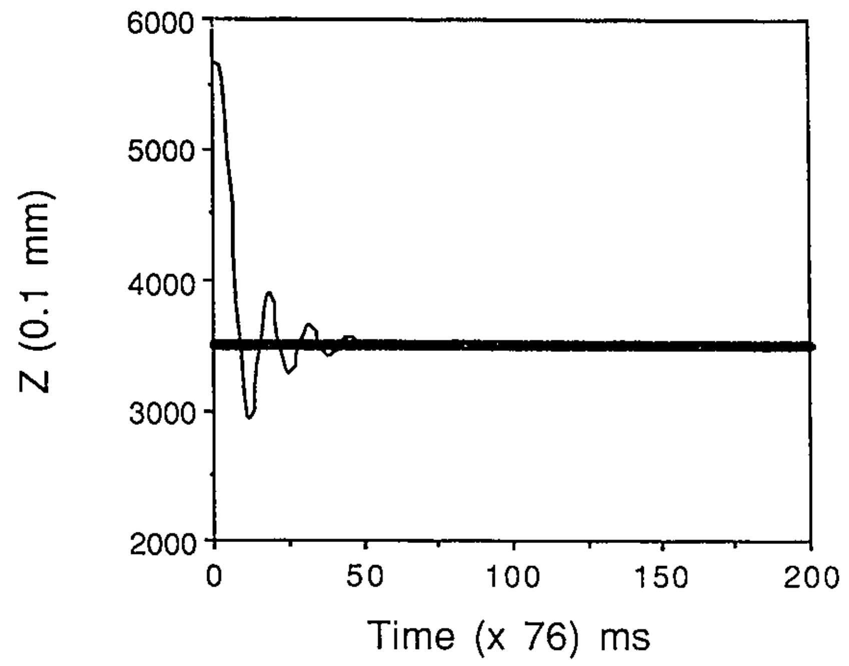

Fig. 21 Response time of the robot for translational motion

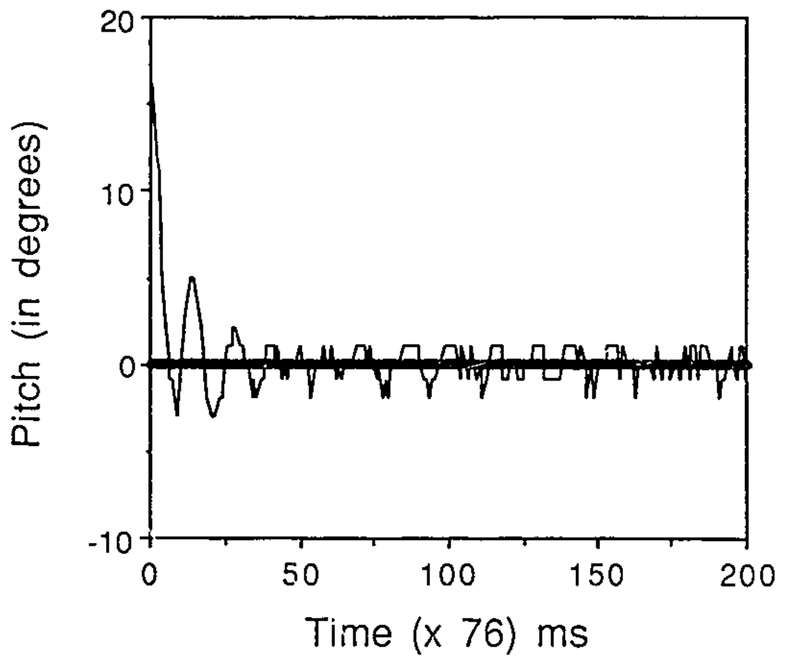

Fig. 22 Response time of the robot for rotational motion

The performance study is done in two stages:

1. When the robot tracks an object that performs a rotational motion, and
2. When the translational motion of the object is being tracked.

In the first case, the object to be tracked is moved at varying angular speeds about the X and Y axes of the tool coordinate system of the robot. The object used is a circular disk that rotates about the vertical (Z) axis of the tool frame at varying speeds. When the circular disk is rotated with its plane tilted, the angles made by the plane of the disk with the XY plane of the tool coordinate frame of the robot undergo sinusoidal variations in the XZ and the YZ planes respectively. While tracking has been achieved with speeds of rotation of up to 0.5 rad/s, for purposes of this explanation the disk is rotated at speeds of 0.05 rad/s, 0.1 rad/s, and 0.17 rad/s. As the robot moves, attempting to maintain the required pose with respect to the disk, the pitch and yaw of the disk, as seen by the RPSs, are noted every time a new profile is collected and processed. The values of the pitch of the disk from the perspective of the RPS are plotted as a function of time in Figs. 23, 24, and 25 corresponding to the speeds of rotation of 0.05, 0.1, and 0.17 rad/s of the disk .

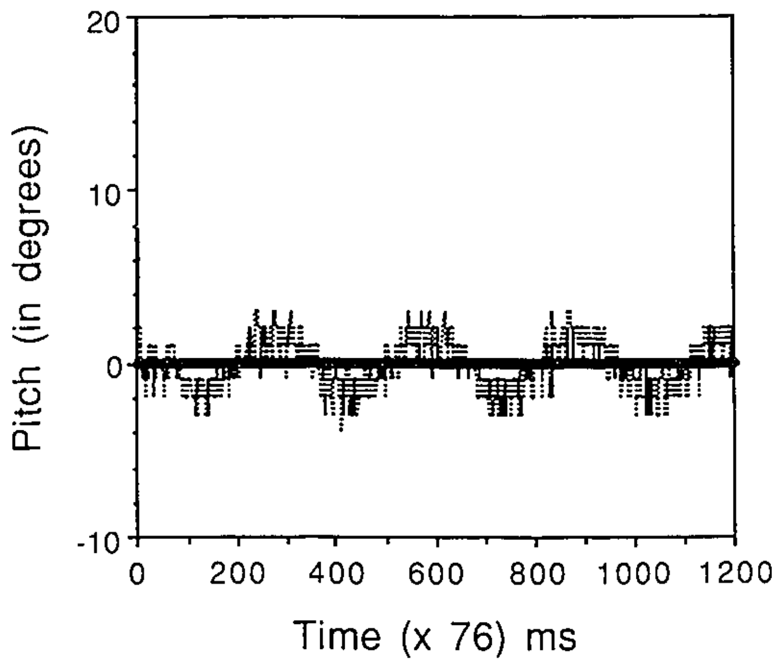

Fig. 23 Pitch of the disk from the perspective of the RPS.
Disk rotates at 0.05 rad/s.

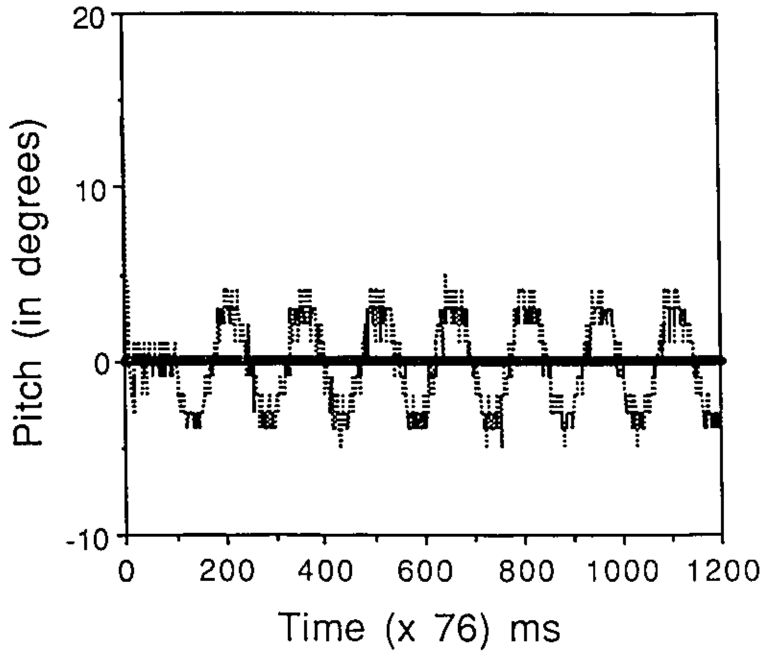

Fig. 24 Pitch of the disk from the perspective of the RPS. Disk rotates at 0.1 rad/s.

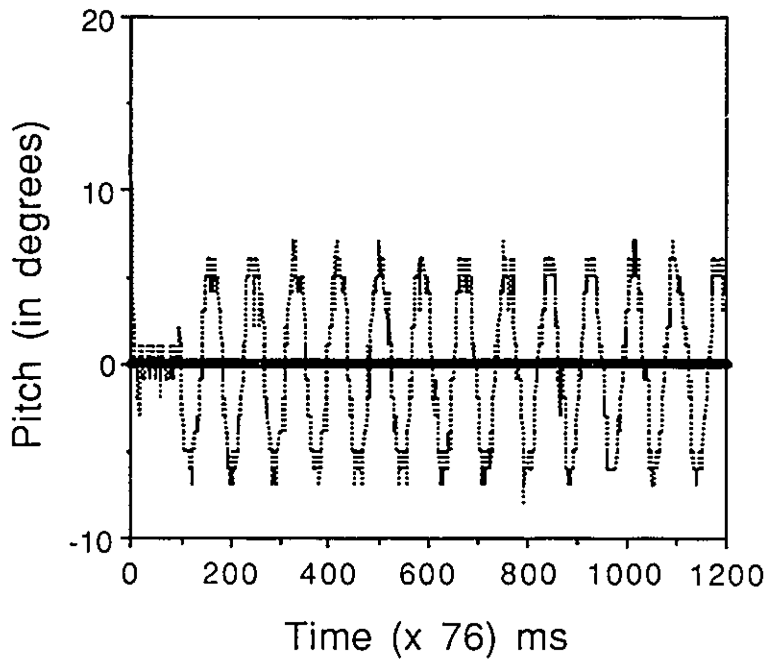

Fig. 25 Pitch of the disk from the perspective of the RPS. Disk rotates at 0.17 rad/s.

In the second part of the performance study, a flat disk attached to a pole is tracked. In order to study the behaviour of the robot in tracking the translation motion of the object, the pole is moved back and forth along the X axis of the tool coordinate system of the robot. The tracking behaviour of the robot is studied by repeating the experiment three times with changes in the manner in which the object is moved.

1. The object is accelerated from rest to 6, 10, 16 and 25 cm/s. The target point in the X axis, as seen by the RPS in the XZ plane, is plotted as a function of time. Figure 26 shows the deviation of the robot position from that of the target in the X dimension.
2. The object is subjected to acceleration and deceleration by moving it through speeds of 4, 6, 8, 12, and 25 cm/s, along the X axis, with a pause between the different speeds. The corresponding behaviour of the robot is depicted in Fig. 27.
3. The object is accelerated from rest to a speed of 25 cm/s. It is then moved back and forth, at this speed, along the X axis. See Fig. 28.

In Figs. 23 through 28, use of the bold line as a reference (since it represents an ideal system) shows that the motion of the robot lags behind that of the object. This lag is found to increase as the speed of rotation or translation of the object is increased. Two factors contribute to this lag.

1. Lag due to the sensory feedback from the RPS. Data collection time from each of the RPSs is 38 ms.
2. Lag due to the restriction on the maximum linear and angular speeds and accelerations of the robot. With increasing speeds of the object, the robot must move at increasing speeds but cannot because of a limit on the maximum allowable speed. If the maximum speed of the robot is increased, without exceeding the rate at which servoing is no longer gradual, this lag can be reduced.

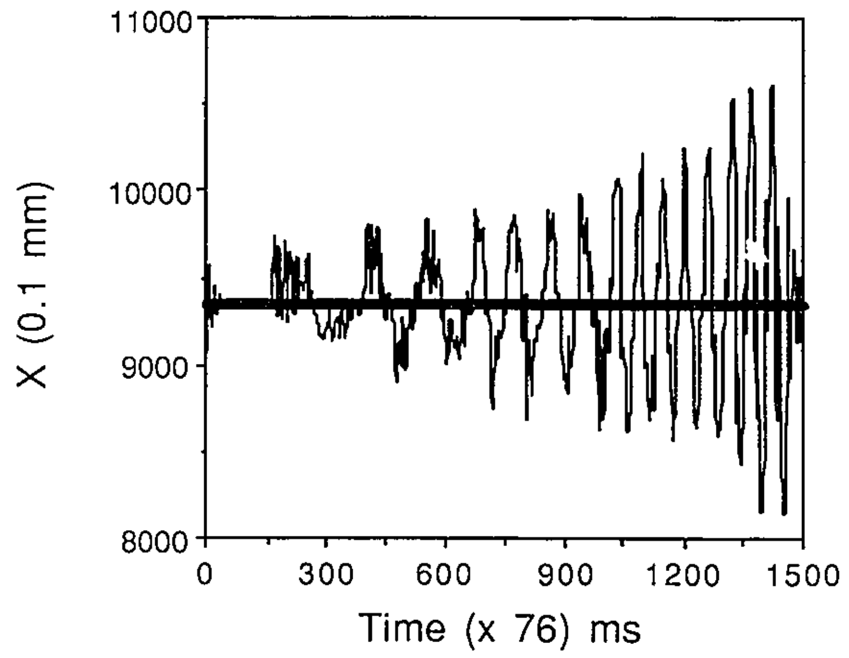

Fig. 26 Robot position in the *X* dimension as viewed by the RPS. Object translates with increasing speeds up to 25 cm/s.

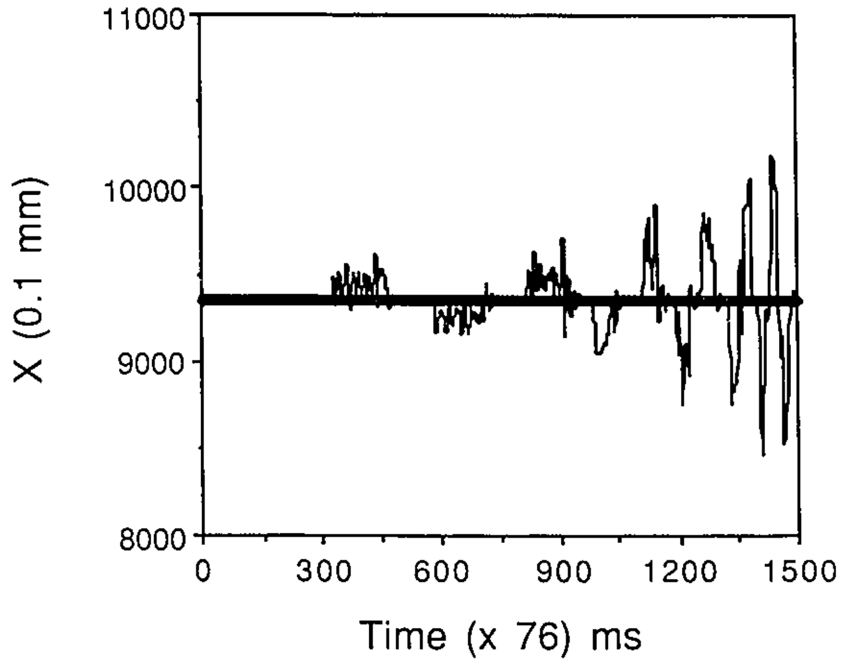

Fig. 27 Robot position in the *X* dimension as viewed by the RPS. Object translates with increasing speeds up to 25 cm/s with stops between.

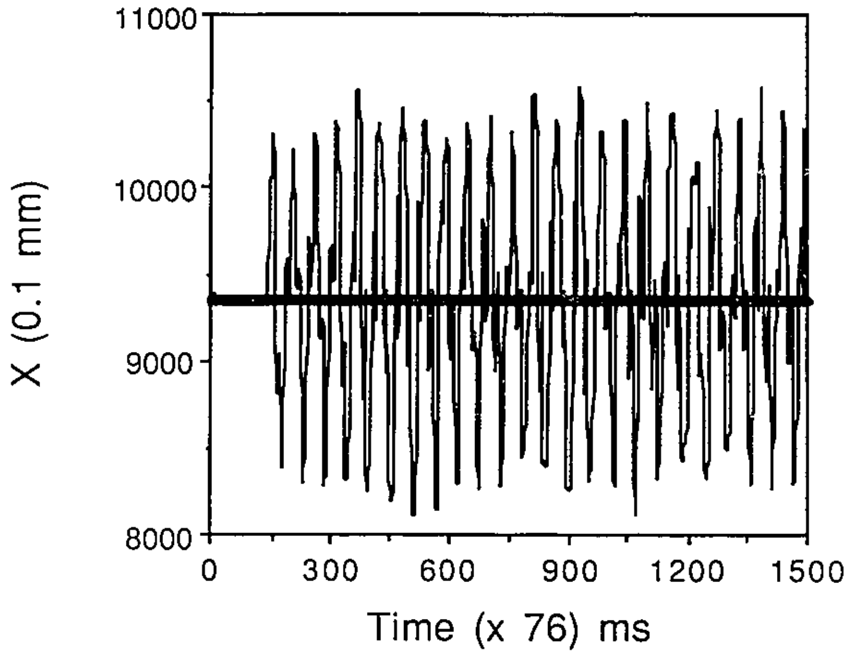

Fig. 28 Robot position in the *X* dimension as viewed by the RPS. Object translates at a uniform speed of 25 cm/s.

To show how the limits imposed on the speed and acceleration/deceleration of the robot affect its motion behaviour, a one-dimensional (say, X axis) tracking problem is considered. Arbitrary values are chosen (Table 1) that correspond to the positions of a moving object. The corresponding positions of the robot, determined by applying the algorithm, are plotted along with those of the object (Fig. 29). When the object is moved by an amount greater than the limits imposed on the robot, as is the case when the two curves of Fig. 29 diverge when viewed from left to right, the robot follows the path of the object, but in a gradual manner. The speeds of the robot and the object are determined from the above set of data (Table 2) and plotted (Fig. 30). While the data in Tables 1 and 2 are computed on the basis that the robot can move at 100 % of that of the object, the robot can only reach a speed of 15 (speed units) and can at most accelerate/decelerate by 3 units of acceleration.

If there is no motion, the graph will be a straight line along the X axis. If the object moves at a constant speed, the graph will be a straight line parallel to the X axis. In Fig. 30, the speed of the object keeps changing, implying that the object accelerates/decelerates. When this behaviour is detected the software is designed so that the robot can emulate this behaviour gradually. If the robot can reach this new speed by either accelerating or decelerating then this motion is carried out. If it cannot, it accelerates or decelerates to the allowable speed. As can be seen from Fig. 30, the speed of the robot changes, within its limits, so as to follow the object. It can also be seen from this figure that the motion of the robot lags behind that of the object. The data used here are exaggerated to demonstrate the effect of acceleration or deceleration.

Object Position	Robot Position
2	2
6	6
14	13
20	20
32	30
32	37
28	34
21	28
10	19
3	7
-1	-2
-6	-8
-16	-16
-24	-24
-36	-35
-44	-44

Table 1 Positions of the target object and the robot.

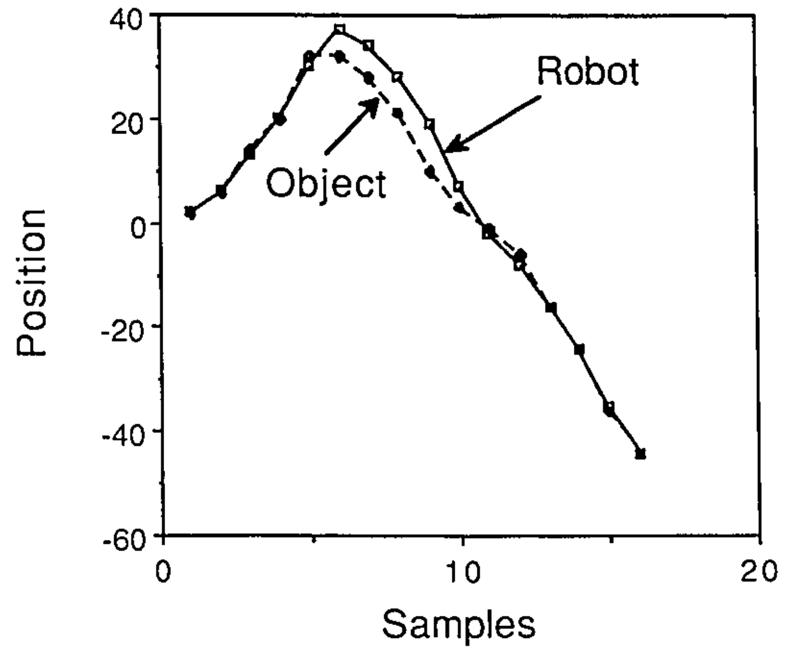

Fig. 29 Comparison of the positions of the object with those of the robot, based on the tracking algorithm.

Object Speed	Robot Speed
2	2
4	4
8	7
7	7
12	10
2	7
9	3
13	6
18	9
16	12
8	9
4	6
8	8
8	8
12	11
9	9

Table 2 Speeds of the target object and the robot.

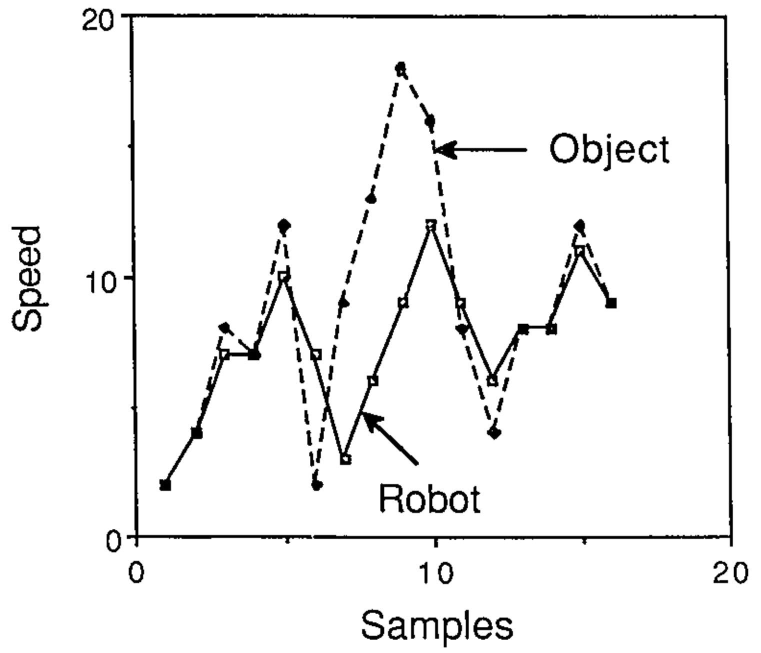

Fig. 30 Comparison of the speeds of the object with those of the robot, based on the tracking algorithm.

6.2 Timing Analysis

The various tasks involved in the tracking application execute in parallel (refer to Fig. 15). As the activities performed by the various tasks are different, their times of execution are not the same. This gives rise to asynchronism amongst these tasks which is not of concern as long as there is no communication between them. However, in order to achieve a common objective, coordination amongst tasks is essential. Consequently, the task sending the message or the one receiving the message must wait in order to establish synchronism amongst them. For a realtime application, synchronization of tasks is an important design issue which, if not properly implemented, results in either a poor performance or a total failure of the application.

The synchronization behaviour is studied by monitoring the times of execution of the various activities involved in the tracking application. A timing study is done on the tasks T1, T2, and T4. Since the task T3 is used only for creating the *handler* task, no timing is done on it. In order to do an analysis of this study, the times of execution of the various activities for two consecutive feedback loops are considered. This feedback loop begins when either T2 or T4 begins fetching a range profile and ends when T1 issues a motion request based on the sensor data collected by both T2 and T4. The absolute readings of the system clock at various stages of execution of these tasks are noted. From these readings the times of execution of the different activities in these tasks are deduced and are shown in Table 1.

The data in Table 3 are represented graphically in Fig. 31. Since this is a timing study, change occurs only in one dimension, viz. the time axis. The ordinate is chosen to denote the tasks being timed. However, to visually distinguish between the graphs for the two feedback loops, ordinates of 4 and 4.5, in the case of T4, are chosen for the successive feedback loops. In addition, the plots for the two sets of timing data are differentiated by the use of bold and dotted lines. Ordinates of 1 and 1.5 are used for the two feedback loops on task T1, while ordinates 2 and 2.5 correspond to those of task T2. The following notations, used in Table 3, represent the various activities performed by the different tasks, each of which is executing on its own.

A	Task T4 fetches a range profile, *XZ*, from one RPS.
B	This range profile is processed.
C	The processed profile is sent to T1.
D	Task T2 fetches a range profile, *YZ*, from the other RPS.
E	Range profile, *YZ*, is processed.
F	This processed profile is sent to T1.
G	T1 receives the processed profiles from T2 and T4.
H	The motion to be performed is computed.
I	Task T1 issues a motion request to the handler task.

Task T4			Task T2			Task T1		
	Feedback loop 1	Feedback loop 2		Feedback loop 1	Feedback loop 2		Feedback loop 1	Feedback loop 2
Work done	At time (ms)	At time (ms)	Work done	At time (ms)	At time (ms)	Work done	At time (ms)	At time (ms)
–	0	76	–	44	120	–	50	126
A	48	124	D	86	162	G	118	194
B	74	150	E	116	192	H	121	197
C	75	152	F	119	195	I	125	201

Table 3 Execution times of the different activities in the tracking code.

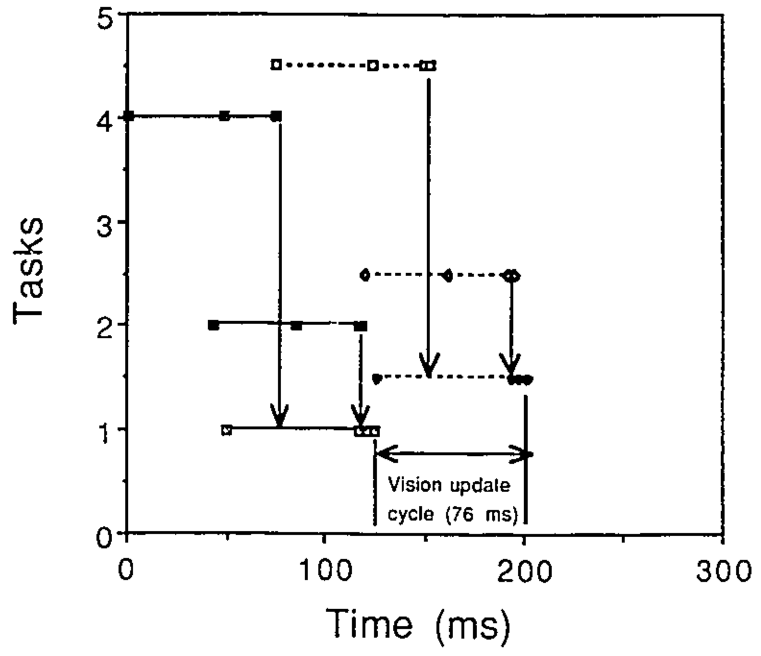

Fig. 31 Times of execution of the various tasks

It is seen from Table 3 that the tasks T2 and T4 execute in a tight loop of 76 ms. Since fetching a profile from an RPS takes 38 ms, each task must process the profile within the next 38 ms so that it is on time to collect the next profile. In the first feedback loop the task T1 issues a motion request 125 ms after the collection of a profile begins. The next motion request is sent by T1 at 201 ms, 76 ms later. This represents the time interval between two successive vision updates. The time taken for the execution of one feedback loop is 125 ms. The vision update cycle time is thus less than the time to execute one feedback loop and is attributed to the parallelism among the tasks. By repeating this study, it is found that the robot receives vision updates from the task T1, in a consistent manner, once every 76 ms.

Chapter 7

Conclusions

The work has been carried out successfully with any flat object moving in the field of view of the RPSs at speeds of up to 25 cm/s along the X, Y, and Z axes and rotating at about 0.5 rad/s in the XZ and YZ planes. The robot tracks the target object, satisfying the conditions outlined in Chapter 3, while maintaining a smooth motion. It responds to changes in the pose of the target object enabling the RPSs to have a consistent view of the target object. It starts tracking when the object is introduced in the field of view of the RPSs and stops tracking when the object is withdrawn from the field of view of the RPSs, settling down at a predefined rest position. Acceleration/deceleration has been incorporated to match the motion of the object being tracked and prevent damage to the robot from sudden starts and stops.

At present, when vision updates are not received, the robot is instructed to carry out the motion computed from the previous update. Instead, predictive algorithms can be incorporated that complement the tracking application in an effort to reduce the lag. Another extension of this work is to study realtime tracking of complex-shaped objects using model-based object recognition. Another area of research is to grasp an object instead of constantly tracking it. This not only requires a knowledge of the right time to descend and grasp but also a suitable grasping position on the object.

References

[1] I. J. Cox and N. H. Gehani, "Concurrent Programming and Robotics," *The International Journal of Robotics Research*, Vol. 8, No. 2, pp. 3-16, 1989.

[2] D. Nitzan, "Assessment of Robotic Sensors," *Proceedings of the 1st International Conference on Robot Vision and Sensory Controls*, pp. 1-11, 1981.

[3] H. R. Everett, "Survey of Collision Avoidance and Ranging Sensors for Mobile Robots," *Robotics and Autonomous Systems*, Vol. 5, No. 1, pp. 5-67, 1989.

[4] R. A. Jarvis, "A Perspective on Range Finding Techniques for Computer Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-5, No. 2, pp. 122-139, 1983.

[5] E. T. Fathi and M. Krieger, "Multiple microprocessor systems: What, Why, and When," *IEEE Computer*, pp. 23-32, March 1983.

[6] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, Vol. 54, No. 12, pp. 1901-1909, 1966.

[7] J. H. Graham, "Special computer architectures for robotics: Tutorial and Survey," *IEEE Transactions on Robotics and Automation*, Vol. 5, No. 5, pp. 543-554, 1989.

[8] J. F. Ready, "VRTX: A Real-Time Operating System for Embedded Microprocessor Applications," *Tutorial on Hard Real-Time Systems*, pp. 318-327, Computer Society of the IEEE, 1988.

[9] I. Lee, R. B. King, and R. P. Paul, "A Predictable Real-Time Kernel for Distributed Multisensor systems," *IEEE Computer*, pp. 78-83, June 1989.

[10] S.-C. Cheng, J. A. Stankovic, and K. Ramamritham, "Scheduling Algorithms for Hard Real-time Systems: A Brief Survey," *Tutorial on Hard Real-Time Systems*, pp. 150-173, Computer Society of the IEEE, 1988.

[11] J. A. Stankovic, "Real-Time Computing Systems: The Next Generation," *Tutorial on Hard Real-Time Systems*, pp. 14-37, Computer Society of the IEEE, 1988.

[12] T. Lozano-Pérez, "Robot Programming," *Proceedings of the IEEE*, Vol. 71, No. 7, pp. 821-841, 1983.

[13] P. G. Ranky, "Programming industrial robots in FMS (A survey with particular reference to off-line, high-level robot program generation using VAL, VAL-II, AML and MARTI)," *Robotica*, Vol. 2, No. 2, pp. 87-92, 1984.

[14] V. Hayward and R. P. Paul, "Robot Manipulator control under Unix RCCL: A Robot Control *C* Library," *The International Journal of Robotics Research*, Vol. 5, No. 4, pp. 94-111, 1986.

[15] M. Gini, "The future of robot programming," *Robotica*, Vol. 5, No. 3, pp. 235-246, 1987.

[16] C. Loughlin, "Eye-in-hand Robot Vision Scores over Fixed Camera," *Sensor Review*, pp. 23-26, January 1983.

[17] W. T. Miller, III., "Sensor-based Control of Robotic Manipulators using a General Learning Algorithm," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 2, pp. 157-165, 1987.

[18] M. Kabuka, J. Desoto, and J. Miranda, "Robot Vision Tracking System," *IEEE Transactions on Industrial Electronics*, Vol. 35, No. 1, pp. 40-51, 1988.

[19] J. T. Feddema and O. R. Mitchell, "Vision-Guided Servoing with Feature-Based Trajectory Generation," *IEEE Transactions on Robotics and Automation*, Vol. 5, No. 5, pp. 691-700, 1989.

[20] D. B. Gennery, T. Litwin, B. Wilcox, and B. Bon, "Sensing and Perception Research for Space Telerobotics at JPL," *Proc. of the 1987 IEEE Conference on Robotics and Automation*, pp. 311-317, 1987.

[21] V. Kratky, "Real-Time Photogrammetric Support of Dynamic Three-Dimensional Control," *Photogrammetric Engineering and Remote Sensing*, Vol. 45, No. 9, pp. 1231-1242, 1979.

[22] M. B. Zaremba, "Vision-based control of Industrial Robots and Telemanipulators," *Hardware and Software for Real Time Process Control*, pp. 527-533, North-Holland, 1989.

[23] B. K. P. Horn, "Robot Vision," The MIT Press, 1986.

[24] P. Anandan, "A Computational Framework and an Algorithm for the Measurement of Visual Motion," *International Journal of Computer Vision*, Vol. 2, pp. 283-310, 1989.

[25] P. Anandan, "Visual Motion Analysis," *Tutorial in the Vision and Graphics Interface Conference*, London, Canada, June 1989.

[26] D. Marr, "Vision: A Computational Investigation into the Human Representation and Processing of Visual Information," W. H. Freeman and Company, 1982.

[27] A. Bandopadhay, B. Chandra, and D. H. Ballard, "Active Navigation: Tracking an Environmental Point Considered Beneficial," *Proceedings of the 1986 IEEE Workshop on Motion: Representation and Analysis*, pp. 23-29, Charleston, USA, May 1986.

[28] R. C. Luo and R. E. Mullen Jr., "A Modified Optical Flow Approach for Robotic Tracking and Acquisition," *Journal of Robotic Systems*, Vol. 6, No. 5, pp. 489-508, 1989.

[29] R. M. Lougheed and R. E. Sampson, "3-D Imaging Systems and High-Speed Processing for Robot Control," *Machine Vision and Applications*, Vol. 1, pp. 41-57, 1988.

[30] B. R. Sorensen, M. Donath, G.-B. Yang, and R. C. Starr, "The Minnesota Scanner: A Prototype Sensor for Three-Dimensional Tracking of Moving Body Segments," *IEEE Transactions on Robotics and Automation*, Vol. 5, No. 4, pp. 499-509, 1989.

[31] J. E. Jones and D. R. White, "Development of a Semi-Autonomous Service Robot with Telerobotic Capabilities," *NASA Proceedings of the Workshop on Space Telerobotics (PWST)*, pp. 307-316, Jet Propulsion Laboratory, USA, July 1987.

[32] C. C. Archibald, W. M. Gentleman, and D. H. O'Hara, "Realtime Feedback Control Using a Laser Range Finder and Harmony," *Proc. of the 7th Canadian CAD/CAM and Robotics Conference*, pp. 6:56-6:62, 1988.

[33] S. Venkatesan and C. Archibald, "Three Degree of Freedom Tracking in Real Time Using a Wrist-mounted Laser Range Finder," *Proc. of the 2nd Conference on Intelligent Autonomous Systems*, pp. 386-392, 1989.

[34] S. Venkatesan and C. Archibald, "Realtime Tracking in Five Degrees of Freedom Using Two Wrist-mounted Laser Range Finders," *Proceedings of the 1990 IEEE International Conference on Robotics and Automation.* In Press.

[35] Programming Manual – User's Guide to VAL II, Version 1.1, 398T1, Unimation Inc., August 1984.

[36] F. J. Pipitone and T. G. Marshall, "A Wide-field Scanning Triangulation Rangefinder for Machine Vision," *The International Journal of Roboitcs Research*, Vol. 2, No. 1, pp. 39-49, 1983.

[37] M. Rioux, G. Bechtold, D. Taylor, and M. Duggan, "Design of a Large Depth of View Three-Dimensional Camera for Robot Vision," *Optical Engineering*, Vol. 26, No. 12, pp. 1245-1250, 1987.

[38] M. Rioux, "Laser range finder based on synchronized scanners," *Applied Optics*, Vol. 23, No.21, pp. 3837-3844, 1984.

[39] W. M. Gentleman, S. A. MacKay, D. A. Stewart, and M. Wein, "Using the Harmony Operating System: Release 3.0," ERA-377, NRCC No. 30081, National Research Council of Canada, February 1989.

[40] W. M. Gentleman, "Realtime Applications: Multiprocessors in Harmony," ERB-1011, NRCC No. 30692, National Research Council of Canada, August 1989.

[41] W. M. Gentleman, "Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept," *Software-Practice and Experience*, Vol.11, pp. 435-466, 1981.

[42] W. M. Gentleman, S. A. MacKay, D. A. Stewart, and M. Wein, "Commercial Realtime Software Needs Different Configuration Management," ERB-1025, NRCC No. 30939, National Research Council of Canada, November 1989.

[43] Equipment Manual for the 500 Series PUMA Mark II Robot, 398P1, Unimation Inc., April 1984.

[44] D. H. O'Hara, S. Elgazzar, and G. The, "ALTER-Harmony: Control of a PUMA Robot from the Chorus Multiprocessor," ERB-1002, NRCC No. 28492, National Research Council of Canada, December 1987.

[45] J. S. Albus, A. J. Barbera, and M. L. Fitzgerald, "Hierarchical control for sensory interactive robots," *Proceedings of the 11th International Symposium on Industrial Robots*, pp. 497-505, Dearborne, USA, 1981.

[46] B. Bhanu and L. A. Nuttall, "Recognition of 3-D objects in Range Images using a Butterfly Multiprocessor," *Pattern Recognition*, Vol. 22, No. 1, pp. 49-64, 1989.

[47] D. Baldwin, "Why We Can't Program Multiprocessors the Way We're Trying to Do It Now," Technical Report 224, Department of Computer Science, University of Rochester, USA, August 1987.

[48] C. Archibald and S. Amid, "Calibration of a Wrist-mounted Range Profile Scanner," *Proceedings of Vision Interface '89*, pp. 24-28, London, Canada, June 1989.

[49] M. W. Spong and M. Vidyasagar, "Robot Dynamics and Control," John Wiley & Sons, 1989.

[50] R. P. Paul, "Robot Manipulators: Mathematics, Programming, and Control," The MIT Press, 1981.

[51] D. Gauthier, P. Freedman, G. Carayannis, and A. S. Malowany, "Interprocess Communication for Distributed Robotics," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 6, pp. 493-504, 1987.