NOTICE

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

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

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

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

AVIS

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

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

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

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

Robotic Active Tactile Sensing Skills

Michael A. Greenspan, B.Sc., B.A.Sc.

A thesis submitted to the School of Graduate Studies in partial fulfilment
of the requirements for the degree of
Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute for Electrical Engineering

Department of Electrical Engineering
Faculty of Engineering
University of Ottawa

ISBN 0-315-75085-5

Canada

UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

To my wife, Mary

Abstract

This thesis examines the use of tactile sensing for robotic contact skills. Two skills were developed which successfully tracked the contact of a plane and a sphere in the presence of positional uncertainties and varying applied forces. In the plane tracking skill, the normal force of the contact as well as a rotational moment about an axis of the sensor was tracked. In the sphere tracking skill, the normal force and positional variations lateral to the sensor were tracked. This work is significant in that it represents the first attempt, to the author's knowledge, at implementing these skills using pure tactile sensing.

Acknowledgements

I would like to thank my thesis supervisor, Dr. Emil Petriu, for his guidance and enthusiasm throughout this work I would also like to thank my friend Dr. Moshe Krieger who was always generous with his time, Bill McMath and Stephen Yeung of the Canadian Space Agency for their support, Benito Carrara and Steve Symons of the University of Ottawa, and my friend and colleague Habib Khalfallah.

Most of all, I would like to thank my wife Mary, whose support and confidence were inspirational. Much thanks to uncle Micky, who kept Mary busy when I was working. Thanks also to my parents, brother, and sisters. Including Paula.

List of Figures and Graphs

Table of Contents

Chapter 1 : Introduction

1.1 Purpose and Scope of Work

The international research community has identified autonomous robotic systems as a technology whose time has come. It is well recognised that in the not so far future, a country with an advantage in advanced robotic systems will have a technological and economic edge over others. Research laboratories throughout the world have been working fervently to advance their robotic systems to the next stage of autonomous behaviour.

Exactly what is meant by robotic autonomy is not well defined. While it is agreed that it would be nice if there existed a robotic system which could perform certain functions currently carried out by a human, it is not well understood what the rudimentary sensory and reasoning capabilities are which would allow such a system to perform effectively and reliably. There are many schools of thought as to what the "real issues" are in developing robot autonomy. There are those who believe that a complex expert system can be built which could mimic human behaviour so well as to react intelligently to unforeseen situations. There are others who believe that it is unprofitable in any way to attempt to replicate human behaviour, and that we are better off building insect-like robots which individually exhibit no intelligence, but as a group communally respond in an effective way.

One common understanding among all robotics researchers is the importance of sensing in the development of any advanced robotic systems. Imparting a machine with the ability to perceive conditions and events in an unknown or partially unknown environment is a key step to the development of system autonomy.

This thesis addresses the issues of robotic tactile perception, the attempt to endow a robotic system with a capability analogous to the human sense of touch. All of the various robotic sensing modes, which include vision, stereo vision, force-torque, sonar range, laser range, proximity, and others, exhibit unique issues which must be addressed along their evolutionary development into a useful sensory system. Tactile sensing has proven to be one of the more problematic sensing modes, even at the proof of concept level.

There are a number of reasons for the slowness in the development of tactile sensing systems. The tactile data is by its nature a three dimensional data. While recent work on laser range data has contributed much to the understanding of 3D data processing, the field is very much in its infancy. Also, tactile sensing is an active sensing mode, dependent upon the motions and repositioning of the robotic manipulator. Whereas in vision sensing it is general to acquire a single global view and then apply processing/recognition routines on this static image, in tactile sensing, the image acquired is necessarily local to a small area on the sensed object. To make any sense of this minimal data set, it is necessary to have

a knowledge of the position of the sensor at the instance of image acquisition (i.e. kinesthetic information), as well as the relation of the image to other previously acquired images.

Another difficulty in tactile sensing is that it is a contact sensing. The information from the sensor is acquired through the direct contact of the sensor with the surface of an object, which requires a great degree of precision and control. It is this very contact which the tactile sensing is employed to control. The nature of this contact will affect the fidelity of the tactile data acquired, which will in turn affect the evolution of the contact.

Finally, a major reason for lethargy in the advancement of tactile sensing is the unavailability of reliable tactile transducers. Some time ago, the desired characteristics of an anthropomorphic tactile transducer were delineated by Harmon [5,6,7]. Nevertheless, the mechanical difficulties of developing an array of small robust pressure transducers have proven substantial. While development work in this thesis employed a commercially available tactile sensor (the LORD Corporation LTS210), that device has since been removed from distribution due to problems of robustness. Today there exist no commercially available tactile sensors. Furthermore, although there has been a great deal of effort and creativity in the development of sensiferous materials, there has not yet emerged a standard transduction method for tactile sensing.

Despite these difficulties, there exists today an amount of interest

and research into robotic tactile sensing. Tactile sensing constitutes not only a desirable, but also an indispensable aspect of future autonomous systems.

The purpose of this thesis is to experimentally demonstrate the use of robotic tactile sensing in contact skills. A tactile sensing test bed was developed as an integrated system incorporating robotic motion functions and tactile sensing capabilities. Two skills have been developed : two degree of freedom (d.o.f.) plane tracking, and three d.o.f. sphere tracking. In the two d.o.f. plane tracking, contact with the planar surface is maintained, within a pressure range, in the presence of changes in the applied force on the plane in the direction normal to the contact, and about an axis of the sensor. This skill is useful in a follow-the-leader mode of control for a dual arm robotic system, and has also been solved using force torque sensing.

In the second skill, three d.o.f. sphere tracking, a spherical object is tracked in the direction normal to the sensor surface, as well as the two lateral directions tangential to the sensor surface. This work represents, to the author's knowledge, one of the first attempts at implementing these skills using tactile sensing.

1.2 Plan of Thesis

The thesis is organised as follows. In the second chapter, the field of autonomous robotics is presented. A system definition of robotics as presented, and the necessity for robotic sensing is discussed in terms of the range of robotic applications.

Tactile sensing is defined and described within the robotic sensing continuum, and the issues particular to tactile sensing are presented. The uses of tactile sensing are examined, as are the particular difficulties which have limited the application of tactile sensing. At the end of the chapter research into tactile transduction technologies are reviewed, as well as tactile sensing for object recognition.

Chapter three presents the kinematic solution developed and implemented for the control of the RHINO XR3 robotic manipulator. The solution was developed to control the placement of the planar tactile sensor. As the robot was a 5 d.o.f. manipulator, to position the planar sensor generally it was necessary to modify the mechanical linkage with an end-effector bracket which attached the sensor at a 45 degree angle to the wrist. This improved the reachability of the system, and allowed the sensor to be positioned with a full 5 d.o.f. within the robot's working envelope. An iterative solution was developed to the inverse kinematics. The solution searched the space of two most distal joints to minimize an error function. Two different search phases were employed, one for a coarse estimate, and one for a fine estimate. The resulting solution could

find a gross motion solution within an acceptable time limit (less than 2 seconds), or a fine motion incremental solution in a shorter time (less than 0.25 seconds).

In chapter 4, the development of the robotic skills is presented. Two cutaneous tactile image processing functions, a sum-of-taxels and a center-of-mass function, were tested for their effectiveness at conveying information about the parameters of contact of the manipulator with objects. The sum-of-taxels was used as an estimate of the contact pressure between the manipulator and the object. The center-of-mass provided a measure of the distribution of the pressure over the planar sensor surface, and was used to estimate the angle of the contact normal in the plane tracking skill, and the lateral position of the sphere in the sphere tracking skill. The relation between the measured values and the contact parameters was determined empirically.

Three experiments were performed using the skills. In the first, two d.o.f. plane tracking was demonstrated. The second experiment used plane tracking, as well as a lateral component of motion, to move a toy automobile along an unknown, unlevel plane. In the third experiment, sphere tracking was used to maintain contact with a sphere moving along a level plane. Conclusions are drawn about the performance of the work, and implications to further research are discussed.

The contribution of this work was the development and testing of two active tactile sensing skills. These skills facilitated the tracking of

two contacted objects ; a plane in two d.o.f., and a sphere in three d.o.f. To the best of the author's knowledge, the implementation of these skills using a tactile array sensor has not been reported in the literature. As the development of sensor based skills is a current and important research area in robotics, it is believed that this work constitues a valid contribution to the field.

Chapter 2 : Tactile Sensing For Robotics

2.1 Introduction

The mention of the term 'robot' evokes in the minds of most people images of giant, lumbering mechanical men culled from 1950's science fiction movies. Present day robotics bear little physical resemblance to their movie star ancestors. There is, however, an important functional similarity between the two; in both cases, robots are superior to human beings at performing certain tasks, while inferior at others.

As the state of the technology advances, the goal is to increase the range of tasks which robots can perform more effectively than humans. The level of robotic technology, however, has reached a plateau. While earlier developments followed through at a steady rate, there exist certain fundamental issues which must now be addressed in order to proceed to the next level of functionality. At the top of this list is the need for better robotic sensing and perception.

This chapter examines the role of tactile sensing, and of robotic sensing in general. It will be shown that, rather than a frill which incrementally enhances system functionality, sensors are an integral component of any robotic system.

2.2 Robotic Sensing

2.2.1 System Definition of Robotics

The Encyclopaedia of Robotics defines the term robot as ;

> "A robot is a general purpose, programmable machine which is capable of manipulating objects in the environment to perform work"

The description as general purpose distinguishes a robot from machines such as a lathe or milling machine. There do however exist sensorized, numerically controlled machines which are certainly general purpose within their particular domain, in that they can be programmed to produce a large variety of objects. Further, the programmable requirement of the definition distinguishes the robot from other general purpose manipulation machines which require direct human control, such as a backhoe. There is nevertheless a large branch of robotics, called telerobotics, which do operate under direct human control.

Another definition offered by Brady [8] gives an insight into the philosophical imperative of the technology ;

> "Robotics is the intelligent application of perception to action"

The proposed system definition describes robotic functionality by partitioning behaviour into three constituent subsystems : mechanical, computational, and sensory.

The most evident robotic subsystem is the mechanical system, which allows the robot to interact with the physical world. The mechanical system of a conventional robotic manipulator consists of the links, motors, end-effectors, and all other physical structures associated with the machine.

The computational system is responsible for the control of the mechanical actuators, the synthesis of tasks, and the processing and interpretation of sensory data. It is the set of software routines and hardware which generate the behaviour of the robot. With the exception of specialised neural type circuits, the computational system is based upon a digital computer.

The third functional subsystem is the sensory system. It is the sum of the sensory transduction devices, and their associated low level processing, used to accumulate information about the physical condition of the robot and its environment. Those sensory devices which gather information about the state of the robot are labelled proprioceptors. Common examples of proprioceptors include limit switches and joint encoders, which measure the position of individual joints. Cameras have also been used to this end. Sensory devices which gather information about the state of the external environment are labelled exteroceptors,

and include cameras, acoustic range sensors, laser range sensors, tactile sensors, proximity sensors, etc.

Any robotic function can be classified as belonging to one or more of the three constituent subsystems. The kinematic solution of a robotic manipulator for example is a computational process which is integrally dependent upon its mechanical structure. The number of joints (d.o.f.), types of joints (prismatic or rotary), number of links, length of links, and other mechanical considerations are all factors which must be considered when developing the kinematic solution for a particular manipulator. Further, the mechanical design of manipulators can greatly affect the computational complexity of the kinematic solution.

2.2.2 The Need for Robotic Sensing

To appreciate the need for robotic sensing, consider the two types of activities in which robots are currently engaged. The first type are activities which are cost justified, i.e., activities where current robotic technology can perform the work cheaper than humans. Examples of these activities are the well defined repetitive tasks encountered in large run manufacturing. In these situations, the major expense is the large setup cost of the robot. When amortized over the lifetime of the machine, and compared to the high wage costs of humans, this becomes a justifiable expense. In the majority of cases, the productivity of the robot system is higher than that of the human system which it replaces.

The second activity group in which robots are engaged are those which are dangerous for humans to perform. Examples of these are activities which take place in hazardous environments, such as nuclear facilities, underwater construction and exploration, mining applications, and space applications. In the case of space robotics, due to the large times involved in space transport, it may be simply impossible for humans to be present to perform the work. Another successful application of this type is for use in bomb retrieval in bomb squads.

The need for robotic sensing in this second activity group is evident. The environments in which the robots are functioning are unstructured. For the successful execution of robotic tasks, it is necessary for the system to have a knowledge of the environment, which includes the identity and location of objects in the environment, and other factors. In some cases such as man-in-the-loop telerobotic systems, a human is able to interpret the environmental conditions. This is particularly true in telepresence systems, where the emphasis is on as rich a transmission of the sensory data as possible, for human interpretation.

In many cases, the successful transmission of adequate information for human interpretation has not been realized. This requires a great number of sophisticated sensors, as well as advanced control mechanisms (e.g. high d.o.f. joysticks). In other situations, such as space applications where a significant time delay exists between the robot and the remote

operator, it may not be possible for the remote operator to respond adequately. This is true for a significant number of fine manipulator skills. In all of these situations, the machine must have the ability to interpret and respond to its sensory information autonomously.

Of course, the first activity involves the majority of today robotic systems, as it includes all industrial manufacturing robots. The need for robotic sensing in this environment is one of cost justification. A manufacturing system is a highly structured environment, and as such there is a great deal of knowledge on which the robots can draw without sensing. Davey [9] succinctly identifies the cost issue of the tradeoffs between environmental structuring and sensing as follows ;

> "How will the cost of a completely determinate factory, i.e., one in which the position of every part and of every tool is known at all times to the full accuracy, compare with that of a factory utilising "second generation" robotic equipment having vision, touch and perhaps other senses to establish the relation between the machines and the parts they are handling ?"

Structuring of the environment is a difficult and costly process. The use of sensing can reduce the amount of structuring, and make robotic installations justifiable.

2.3 Tactile Sensing

The desire for an artificial tactile sensing system is driven by our experiential understanding of our own biological sense of touch. This is known as the haptic sense, from the Greek haphe, or liver, as was then thought to be the source of this sensory capability. Biological touch, like any biological sense, is a complex process which involves many different levels of cognition. As artificial tactile sensing is a young technology, it is important at this stage to clearly identify the components of a tactile sensing system, and to specify the uses for and goals of tactile sensing.

2.3.1 Tactile Sensing Terminology

There exists today a great deal of variation in the terminology of tactile sensing. The following definitions will be used consistently throughout the remainder of this work.

Cutaneous : having to do with the local reference frame of a tactile transducer. For example, a cutaneous image is an image of the contact of the surface of a tactile sensor with an object, without any reference to its location in space.

Kinesthetic : having to do with the position of a manipulator

end-effector in a global reference frame.

Tactile Sensing : the continuous acquisition and processing of cutaneous contact information over an area.

Active Tactile Sensing : tactile sensing which includes a kinesthetic response.

Taction : tactile perception. Tactile sensing for the purpose of object recognition and localization.

Taxel : a tactile element, a single discrete transduction element of a tactile sensor, analogous to a pixel in image processing. Tactile transducers consist of a number of these tactile elements distributed in a spatial arrangement, such as a planar array. Usually, taxels are pressure transducers sensitive over a discrete range.

Touch Sensing : similar to tactile sensing, but with just one taxel. This is also known as simple touch.

Contact Sensing : a sensing mode where the transducer comes into direct contact with the sensed object. Tactile sensing is a form of contact sensing, as is force torque sensing.

Range Sensing : a sensing mode where the nature of the sensed data is range or distance information. Examples include laser range cameras and acoustic range sensors. Note that, while some sensing systems can extract range information, such as structured illumination vision systems, stereo vision, range through focus vision, and tactile sensors, the low level data is not fundamentally range data.

2.3.2 Problems of Tactile Sensing

By the above discussion, tactile sensing attempts to emulate the functionality of the biological sense of touch, and is therefore fundamentally anthropomorphic. Our own human sense of touch provides us with a capability which we would like to impart upon our robotic proteges, and also serves as a model to guide the development. The biological sense of touch can also be said to provide an existence proof that such a function is possible.

Of all of the robotic sensing modes, which include vision stereo vision, acoustic range, magnetic proximity, laser range, etc., tactile sensing has proved to be the most difficult to develop and implement.

There are two reasons for this. First and foremost, the tactile transduction technology has advanced very slowly. Unlike vision cameras, which were a direct spinoff of television technology proliferated by the entertainment industry, there has been no economic impetus for the development of tactile transducers. With a few exceptions, such as the LORD and Barry Wright Corporations, the development of tactile transducers has been limited to university facilities.

The very nature of tactile transduction has also proved a barrier to development. There are few materials which can naturally transduce contact pressure over an area into an electrical signal. Two materials which have generated interest to this end have been force sensitive resistive polymers (FSR) and piezoelectric film ; others include magneto-resistive, capacitive, and optoresistive devices.

The lack of materials for tactile transduction has spurred the development of sensors which use the more complex characteristics of a number of materials to achieve the desired sensiferous capacity. A typical example utilises the elastic characteristics of a rubber pad to deform under pressure and block the light reception of an array of diodes. This scheme was used in the LORD LTS 210 tactile sensor.

Each tactile transduction system has its strengths and weaknesses, and while some have shown more promise than others, no sensor technology has yet surfaced as the standard for tactile transduction.

The second factor which has slowed progress is that tactile sensing is a contact sensing, and is consequently an active sense. An active sensing mode is one in which the location of the transducer is an intrinsic aspect of the sensing act. Of course, no sensory information can be interpreted independent of the location of the sensor. In the case of active sensing, however, the successful acquisition of information can be achieved only through deliberate relocation of the sensor. As opposed to passive sensing, active sensing is a deliberate exploratory act requiring a complex combination of activities. The manipulation of the tactile sensor to contact the environment is a difficult activity which is integral to the act of tactile sensing.

This is true not only of artificial systems, but also of the biological sense of touch (see Appendix A). Recently, much of the emphasis into research of the physiology of touch has focussed on the relationship between the active acquisition of the tactile information and the interpretation of this information. It has been found that the method by which a person actively feels an object will greatly determine his ability to identify the object. It is far more difficult for a person to identify an object which is statically placed into the palm of the hand or brushed across the fingertips, than one which that person is allowed to actively explore through a series of hand motions.

2.3.3 Tactile Sensing Applications

There are four classes of sensing functions to which taction can be applied, as illustrated in Figure 2.1. The first class is error detection, where the sensing system determines the presence or absence of a certain error condition. For example, when a part is grasped, tactile feedback could be used to determine the correctness of the hold.

The second class is contact control. The tactile sensor is fixed to the jaws of a simple two fingered gripper, or to the fingertips of a multifingered hand. The contact information is then used to sense and control the grasp of a particular object. Pressure thresholds which could deform the object, or cause slippage, can be sensed and controlled. An example of this type of application is the use of pressure information to guide the grasp and manipulation of an egg. Very little work has been done on the use of tactile sensing for grasp/contact control [64], and it is towards this end that the experimental development in this thesis is directed.

Environment exploration is the third class of application. This includes both object recognition and object localization. Object recognition is the identification of an objects within an unstructured environment. An example of this type of problem occurs in the category of applications known as bin picking problems. Here, through sensing, a robotic system must identify a specific, often occluded object, from

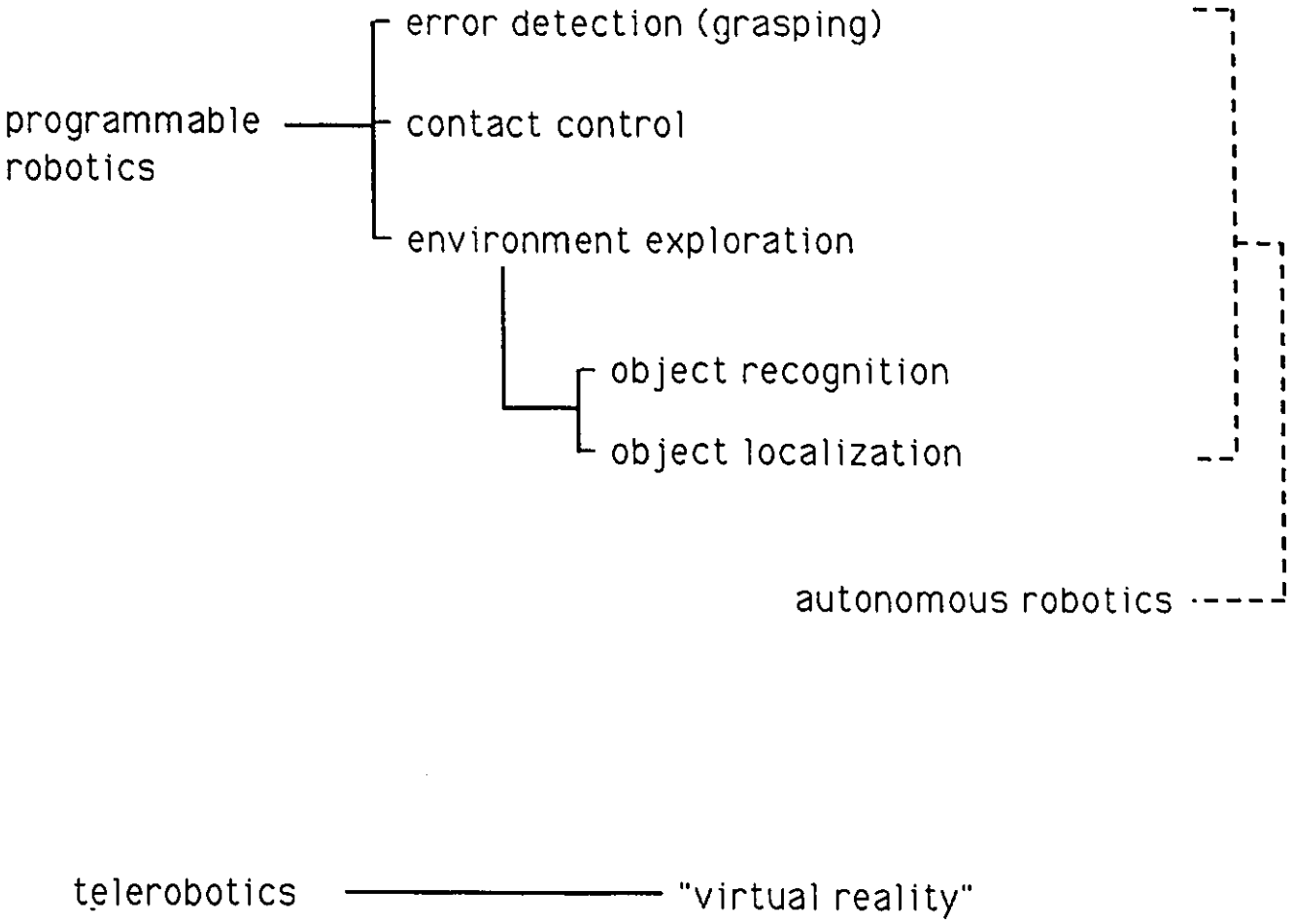


Figure 2.1 : Applications of Tactile Sensing

general assortment of objects. Object localization is the identity of an object's location through sensing. This is closely related to the function of object recognition : the techniques to perform both are closely related, and sometimes identical [49,50].

Of the above described classes of applications, only error detection and contact control are required in industrial robotics, or "hard automation". Environment exploration is further needed for autonomous robotics.

There is one final class of sensing function to which taction can be applied; the remote transmission of tactile data for telepresence. In this man-in-the-loop application, a tactile sensing system is used to transduce the tactile data to a human operator, who then interprets the data and performs the required action. An application of this class of problem is a telerobotic system, where data received from the sensorized grippers is transmitted to the operator to aid in grasp control. Another nonrobotic application is medical prosthetics. Here, the tactile data can be transmitted to a sensitive area of skin.

2.4 Review of Tactile Sensing Research

2.4.1 Tactile Transduction Technologies

The earliest tactile sensors were based upon mechanical transduction principles. The deflection of a sensing peg was measured through the use of simple binary contact switches, linear or angular potentiometers, shaft encoders, or other means. The difficulties of constructing a miniature array of these mechanisms while maintaining sensitivity and durability prompted research into sensiferous materials capable of transducing force information. Sensors constructed from conductive elastomer, piezoresistive, piezoelectric, magnetic inductive, and other materials have been developed. Sensors have also been developed based upon capacitive and optoelectronic principles which still rely upon mechanical transduction, but which use effective modulation techniques to facilitate miniaturization.

Nicholls and Lee [10] in a recent survey have proposed a categorization for the type of data which tactile sensors transduce ;

1. **simple contact** : the presence or absence of contact between sensor and stimulus.

2. **magnitude of force** : a numerical value transduced from each site represents the amount of force applied by the stimulus

at that site; note that the force sensed may be torque, shear, force normal to sensor surface, etc.

3. **three-dimensional shape** : the transduction process gives numerical values directly related to the three-dimensional shape of an object touching the sensor; for example, linear displacement of an array of probes.

4. **slip** : the data indicates the movement of a contacting stimulus relative to the sensor surface. Typically, a slip sensor is mounted on a robot gripper.

5. **thermal properties** : the thermal conductivity of a contacting stimulus, as well as its absolute temperature, can be measured through contact.

Most tactile sensors transduce data of type 2, an array of normal force values. Some groups have also managed to detect some rudimentary slip information. Thermal information can also been transduced, although heat may serve to degrade the force sensing performance. In the case of piezo/pyro electric materials, for example, heat and pressure information are both transduced to a single electrical signals, which must be decoupled. It is also not clear exactly how useful thermal information would be in present day applications.

There has been very little demonstration of type 3 sensors, which

directly measure 3D surface geometry. This is attributed to the difficulty of producing a compliant sensing surface capable of conforming to the geometry of the sensed object. Usually, this information is inferred from the contact pressure [3]. Additional types of tactile data include wetness, hardness, and texture.

Based mainly on a comparison to the biological sense of touch (see Appendix A), Harmon [5,6,7] identified the following characteristics of an ideal tactile transducer :

1. **intertaxel resolution** : the distance between taxels should be approximately 1 mm.

2. **spatial distribution** : the taxels should be arranged in an array of typically 10 X 10 elements.

3. **flexibility** : the tactile array should be flexible to conform to the sensed surface.

4. **time response** : each taxel should operate in the 1 to 10 Hz range.

5. **force response** : each taxel should operate in the 1 to 1000 gram range.

6. **linearity** : the response need not be linear, although monotonicity is essential.

7. **hysteresis** : the difference in the response during loading and unloading must be low.

8. **robustness** : the sensing material must be able to survive harsh environments.

There has yet to be a sensor developed which satisfies all of Harmon's original characteristics. The development of flexible compliant sensors has proven particularly difficult. With the exception of a few fingertip shaped tactile probes, all tactile sensors are rigid and planar. Hysteresis has also been a limitation for many sensing materials. Since Harmon's time, a number of other characteristics of tactile transduction have come to light :

9. **overload capacity** : the ability to withstand forces beyond the sensing range without damaging the transducer must be substantial.

10. **repeatability** : the number of cycles of force which the sensing material can withstand before it begins to degrade must be large.

11. **scanning electronics** : in the simplest case, the electronic interface to each taxel is accomplished through 2 wires. To facilitate miniaturization, it is desirable

to reduce the total number of wires by incorporating scanning circuitry. For example, the total number of lines needed to uniquely address all taxels in an N X M array can be reduced to N + M for row/column scanning.

12. **cross talk** : the signal from each taxel must be independent from that of neighbouring taxels. Alternately, it must be possible to decouple the effects through processing.

13. **interference** : the sensiferous material and electronic circuitry must be impervious to any environmental interference, such as electromagnetic flux or temperature. Also, the scanning and processing circuitry must be shielded from any internally generated noise.

What follows is an examination of various tactile transduction materials. The working principle underlying each technique is presented, and the practical advantages and limitations are discussed. Successful implementations of sensors using each transduction method are noted and, where applicable, industrial developments.

2.4.1.1 Capacitive

The capacitive effect has been used to measure small displacements over an array of taxels. Capacitance is a measure of the ability of a device to store a charge over an area separated by a distance, as described by the equation ;

[2.1] $$C = \varepsilon A / d \qquad [\text{Farad}]$$

where ε is the permittivity or dielectric constant [Farad/m],
A is the effective area of the charge distribution [m^2],
d is the charge separation [m].

The voltage across the capacitive element changes according to ;

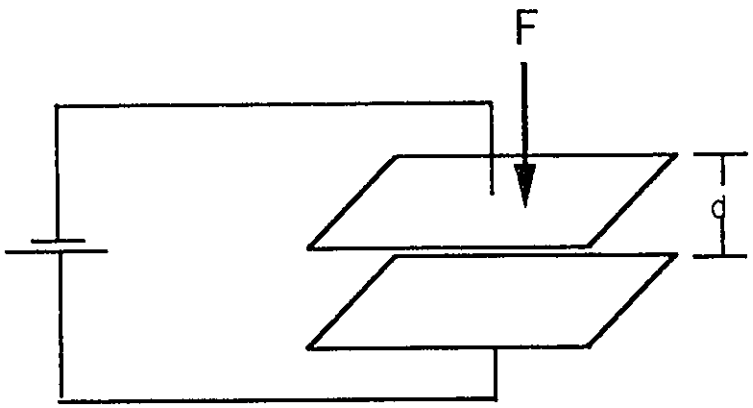
[2.2] $$V = q/C \qquad [\text{Volt}]$$

where q is the stored charge [Coulombs]. If the charge is kept constant, then a change in voltage is a measure of a change in capacitance. A change in capacitance will result from changing any of the three parameters in

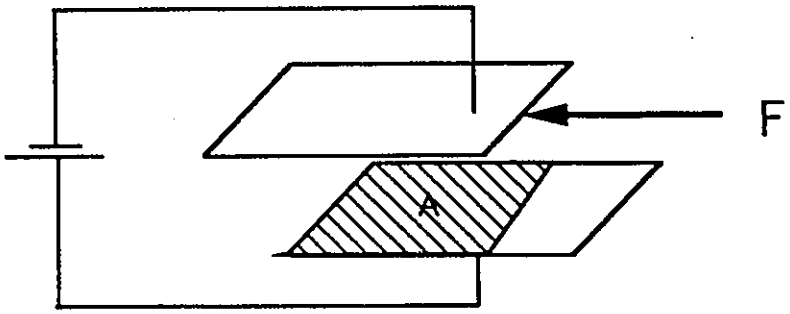
equation 2.1. For example, in Figure 2.2 a), the distance parameter is changed in a parallel plate capacitor by applying a force to the top plate. In part b), a lateral force causes a change in the effective area. Part c) introduces a dielectric between the charge, effectively changing the constant ε.

The main advantage of capacitive transduction is sensitivity. It is possible to measure very small changes in voltage resulting from small displacements, on the order of 10^{-1} mm . The circuitry is also simple and well understood. Further, in the case of semiconductor capacitors, there is the capability to measure tangential forces.

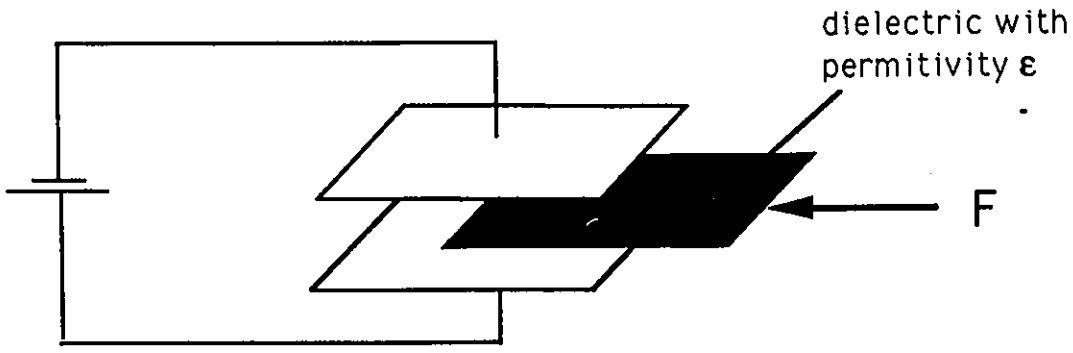
There are a number of disadvantages of capacitive transduction. It is difficult to isolate the charge effects of the adjacent measurement elements, which results in cross talk between taxels. There is also interference from the surrounding circuitry. Harmon dismisses the use of capacitive techniques for tactile transduction for this reason [5]. Since



a) change in plate separation d



b) change in effective area A



c) change in electrical permitivity ε

Figure 2.2 : Changes in Capacitance Due To Applied Force F

his time advances have been made in electromagnetic shielding, but in certain hazardous environments, such as space, there may exist electromagnetic flux. Furthermore, the measurable capacitance decreases with physical size, which limits the degree of miniaturization.

A successful capacitive sensor was constructed by Boie [16]. The sensor consisted of 2 sets of 8 conductive strips orientated orthogonally and separated by an elastic dielectric. Applying pressure to the top layer deforms the intermediate dielectric and results in a measurable change in capacitance. The intertaxel distance of his device was 2.5 mm, and each taxel could measure up to 5 kPa of pressure.

Other designs based on this principle are described in [17]. A sensor based on the principle of a moving dielectric was constructed by Jayamant et. al. [18]. Fan [19] has produced a sensor using semiconductor capacitors which has the ability to measure both normal and shear forces.

2.4.1.2 Inductive

Complimentary to the capacitive effect is induction. Induction is a statement of the interdependence between electrical and magnetic fields.

According to Amperes Law ;

[2.3] $$I = \oint H \cdot dl$$

where I = electrical current enclosed by path [Ampere]
H = magnetic filed about closed path [Henry]

The above relationship states that an electric current generates a magnetic field and, conversely, a magnetic field will induce an electric current. As an example, a current flowing through a wire generates a magnetic field perpendicular to the direction of the current, in circles surrounding the wire.

The effect of mutual induction is utilized in tactile transduction. A current flowing through a primary coil generates a magnetic field. A secondary coil within the proximity of the primary coil will be enveloped in the magnetic filed, and thus a current will be induced in the secondary coil. The strength of the current in the secondary coils is dependent upon 3 factors : the strength of the current in the primary coil, the geometric arrangement and proximity of the coils, and the magnetic permeance of volume containing the magnetic field.

There are two ways that mutual induction has been used to transduce pressure information. The first is through purely mechanical methods. The pressure changes the geometric proximity of the two coils, or advances a material between them which changes the permeance of the magnetic

filed volume. An example of a transducer which alters the proximity of the coils was the pressductor, constructed by Pewa [20]. Here, the coils were wound in a deformable rubber bracket. Applied pressure would advance the proximity of the two coils and increase the inductive coupling between the coils, thereby increasing the current in the secondary coil proportional to the pressure, as illustrated in Figure 2.3.

A successful sensor which used the technique of changing the magnetic permeance was demonstrated by Sato et.. al. [21]. The primary and secondary coil were magnetically coupled by a metal peg. When the peg advanced, the permeance of the magnetic field increased, as illustrated in Figure 2.3. 64 of these pegs were arranged in a 8X8 array, and deflections on the order of 1-2 mm where detectable.

The second method of inductive pressure transduction makes use of the phenomenon of magnetoelasticity exhibited by certain materials. Magnetoelastic materials experience a change in internal magnetic field properties when subjected to an applied pressure. Magnetoelastic materials are manufactured in the form of amorphous ribbons ($Fe_{70}CO_{10}B_{20}$) and tefernol rods ($Tb_{27}Dy_{73}Fe_2$).

An example of the working principle of a magnetoelastic pressure sensor is illustrated in Figure 2.4. In part a), no force is applied and the internal magnetic properties are regular. The magnetic field generated from the primary coil has negligible effect on the secondary coil. When a force is applied, as in part b), the internal magnetic field changes and the

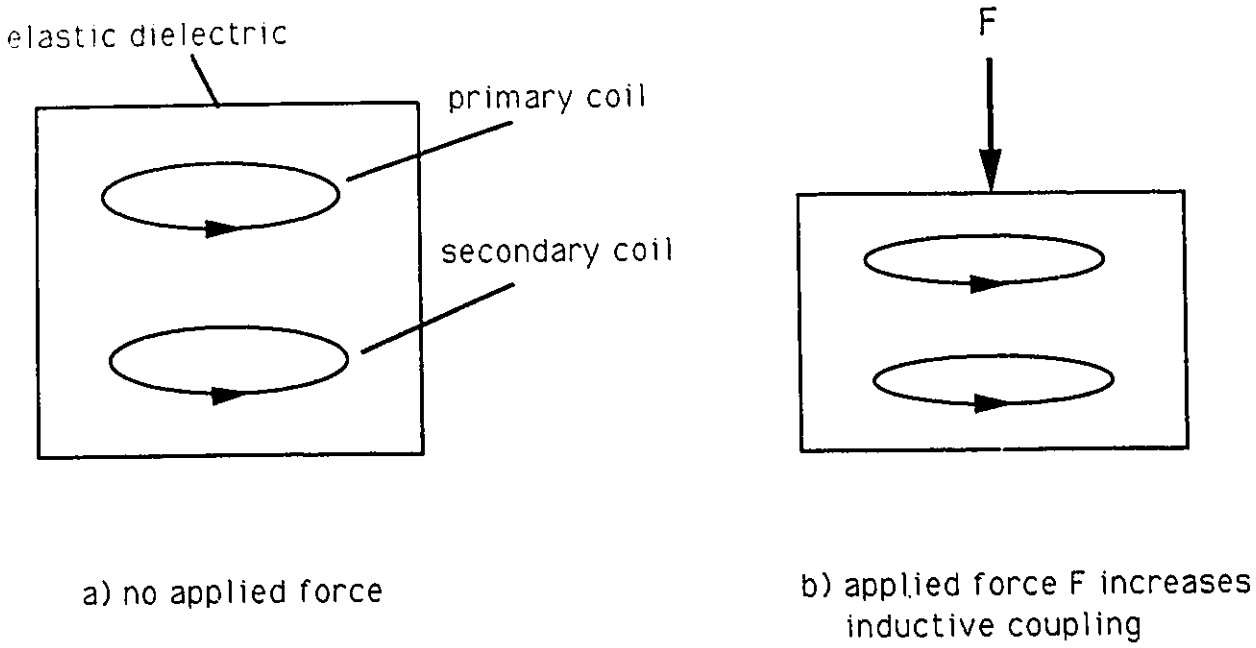


Figure 2.3 : Inductive Pressure Sensor (Pressductor)

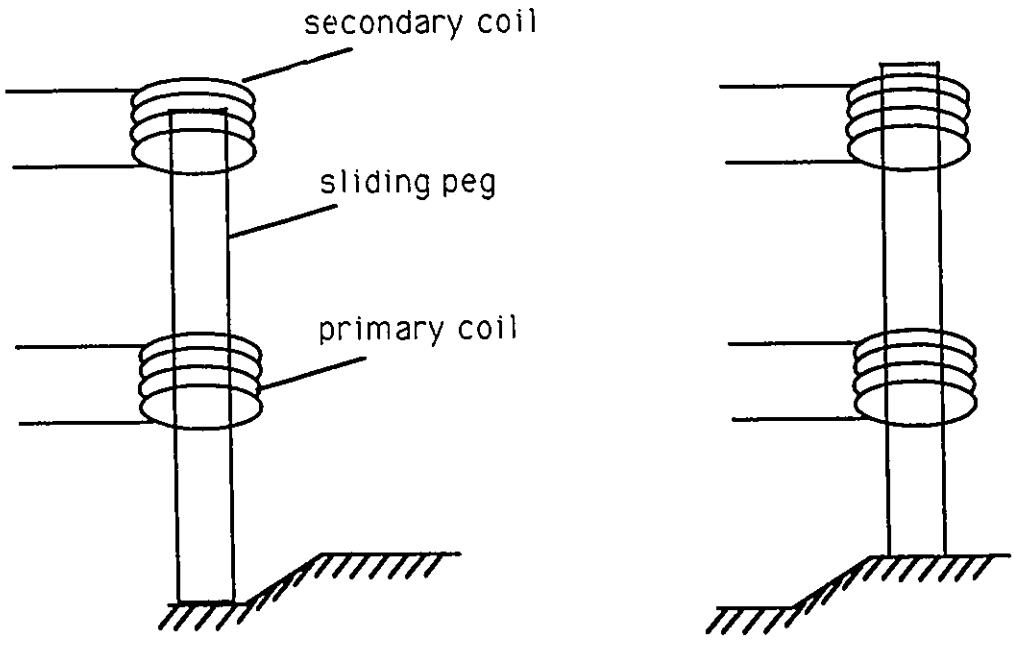
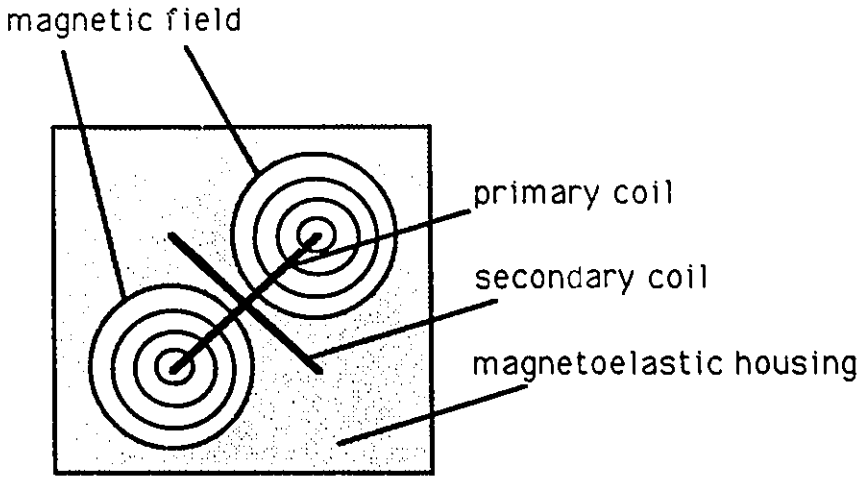


Figure 2.4 : Inductive Displacement Sensor

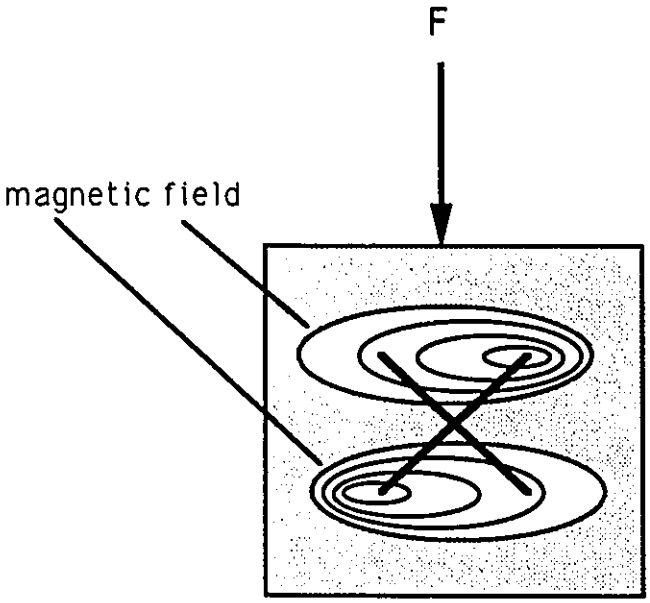
secondary coil is immersed in the generated magnetic field.

Vranish [22] implemented a 16X16 taxel sensor with 2.5 mm intertaxel spacing based on the principle of magnetoelsticity. The material used was VITROVAC 4040, an amorphous magnetoelastic material based upon Fe and Ni. This substance is extremely sensitive and the resulting sensor had a dynamic range of 30 - 300 Pa with 8 bit/taxel resolution. The scanning rate of the sensor was particularly outstanding, able to capture and digitize successful tactile images with a frequency of 1 kHz.

The first type of inductive sensor is reliant upon mechanical transduction, and is restricted by all of the previously mentioned difficulties. The magnetoelastic sensors have the distinct advantage of not having any moving parts. This results in sensors with excellent durability, particularly in their tolerance to overload. As the working principle of inductive sensors depends upon the reception of magnetic fields, they are susceptible to external electromagnetic noise. Also, as is the case with capacitive sensors, they lose sensitivity as the size is decreased.



a) magnetic field does not encircle secondary coil



b) applied force F causes magnetic field to encircle secondary coil

Figure 2.5 : Magnetoelastic Pressure Sensor

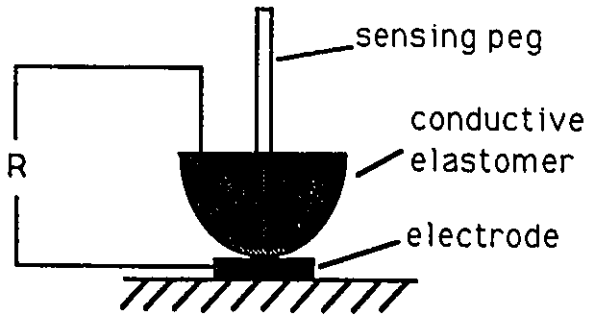
2.4.1.3 Conductive Elastomers

A highly rugged form of pressure transducer is fabricated through the implantation of metal powders or fibres into elastomers (elastic polymers). The resulting material not only retains its elasticity, but is also electrically conductive.

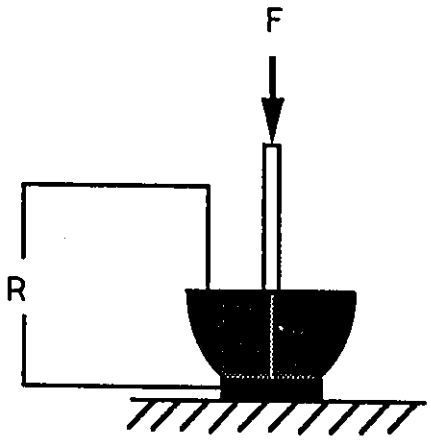
The working principle of a single pressure sensitive taxel is illustrated in Figure 2.6. Applying pressure to the taxel increases the area of contact between the electrodes, which decreases the contact resistance.

The main advantages of conductive elastomer transducers are their general ruggedness, tolerance to force overload, durability, and temperature insensitivity. The sensors are mechanically simple and can be easily miniaturized, and the transduction is measured by standard resistance techniques.

The disadvantages are the hysteresis and creep inherent in the material. The response is also highly nonlinear. Choosing the proper mix of elastomer and metal particle can reduce these effects, but the elastomer must be optimized simultaneously for mechanical and electrical properties. There is generally a great deal of industrial interest in elastomers, so it is likely that these issues will be addressed at the material level and that this transduction technology will continue to advance.



a) no applied force, contact area is small



b) with applied force F, contact area increases

Figure 2.6 : Conductive Elastomer Pressure Sensor

A successful implementation of a conductive elastomer tactile transducer was developed by Hillis [23]. The sensor had a 16 by 16 array of taxels on a 1 cm^2 area, and was sensitive over a range of 1-100 g. The elastomer used was anisotropically conductive silicon rubber (ACS) which has the additional property of being conductive in only one direction along the plane. This allowed the scanning circuit to be vastly simplified and easily miniaturized, with only 32 wires for a 256 taxel sensor. The ACS is commercially available and is fabricated by alternating 250 μ thick sheets of nonconductive Si with conductive Si. A resolution of up to 50 lines/cm can be achieved, with a conductivity of 100 ohms/cm.

The sensor consisted of a dielectric separator sandwiched between the ACS and a PCB with fine etched lines of conductivity. The separator used was a nylon stocking sprayed with nonconductive paint. This allowed contact between the ACS and the circuit and the subsequent transmission of electricity when pressure was applied, while functioning as a separator when pressure was released.

Other successful sensors based on conductive elastomer materials are those of Prubrick [24] and Raibert [25]. Prubrick's design used a grid of carbon doped foam rods. The cross section of each rod was D-shaped, and so the taxel junctions behaved as in Figure 2.6. This design was eventually commercialized as the Barry Wright Corporation Sensoflex. The only other company mass producing tactile sensors, the LORD Corportation, also has a sensor based on conductive elastomer materials.

2.4.1.4 Carbon Fibre

Carbon fibre sensors are similar in principle to the conductive elastomer type, with the exception that the change in resistance is measured internally through the thickness of the material, rather than at a contact junction. Thin strands of conductive carbon microfibrils, ranging in diameter from 7 to 30 μm, are woven into either ropes or felt-like pads. When pressure is applied to the pad, the number of carbon filaments which are in contact, as well as the total area of contact, increases. This serves to decrease the resistance through the thickness of the material.

There are a number of advantages of carbon fibre sensors. The material is very strong and can be subjected to high pressures of up to 1.5 to 2.5 GPa over many cycles before experiencing mechanical breakdown. Robertson and Walkdon [26] subjected a carbon felt pad to a 0 to 3 bar cycle for 2 X 10^6 cycles and observed only a 5 percent degradation.

Carbon fibre pads are flexible and can be construed into odd shapes. The hysteresis is low, and the thermal stability is the highest of any pressure transduction material, able to withstand temperatures above 500° C.

On the negative side, carbon fibres may be susceptible to contact noise at the scanning junctions. Several techniques have been suggested to

reduce this noise [27]. Further, while the material is durable in the transverse or thickness direction, it suffers mechanical break down when subjected to lateral forces.

Robertson and Walkden [26] constructed a 16 X 16 element tactile sensor with a scanning rate of 1 MHz/taxel. The sensor consisted of a carbon fibre felt pad sandwiched between a PCB base and a top layer of flexible electrodes. The top layer was fabricated from 16 strips of silicon doped rubber arranged parallel on an insulating rubber. The PCB had 16 parallel lines and were oriented perpendicular to the top layer. The PCB was also flexible, allowing the sensor to be moulded into non-planar shapes, one of the few sensors which demonstrates this capability.

2.4.1.5 Strain Gauge

The most common device for transducing force is the metal strain gauge, which measures the piezoresistive effect inherent in metals. Piezoresistance is the characteristic of any material which exhibits a change in resistance due to some applied pressure, and is described for a metal wire by the equation ;

[2.4] $$R = \rho L / A \qquad [\text{ohm}]$$

where ρ = resistivity [ohm m],
L = length [m],
A = cross sectional area [m^2].

The resistivity is a function of the particular metal, as well as environmental factors such as temperature, and can be considered constant. The strain gauges are constructed so that an applied force in a preferred direction will induce a strain in the wire, which will effect a change in the length and/or area of the wire. This is measured by the resultant change in resistance.

The gauges are fabricated from thin metal foils which are bonded to the rigid surface subjected to the strain. Strain gages are industrially available and very accurate. They have well known responses, little hysteresis, good repeatability and overload capability, and the processing circuitry is well developed.

All commercially available force-torque sensors use strain gauge transduction. Their application in tactile transducers has been limited, however, due to the size restrictions. The strain gauges are not only difficult to handle (they must be bonded accurately) but they also become less sensitive as they become smaller. Further, a current is passed through the wire in order to measure the resistance, which generates heat. In isolation, the heat generated is small and it's effect on the measuring accuracy can be accounted for. When there are a large number of strain gauges in a small area, such as in a tactile array, the heat generated may be substantial.

Another form of commercially available strain gage is the

semiconductor type. Semiconductors also exhibit the piezoresistive effect and are much more sensitive to pressure changes than are metal strain gages. The gage factor is a measure used to describe the sensitivity of the piezoresistivity of materials, defined as ;

[2.5] Gage Factor = $\frac{\text{change in resistance/total resistance}}{\text{change in length/total length}}$

$$= \frac{dR/R}{dL/L}$$

Metal strain gages have a gage factor of around 2, while semiconductor strain gages have a gage factor of 200 or more.

Semiconductor strain gages have an exponential response, low hysteresis, good repeatability, and are industrially available. They are constructed using the same techniques as other semiconductors, so there is an impetus towards miniaturization. It is conceivable that future semiconductor strain gages will have signal processing functions built into the same wafer as the pressure transducers.

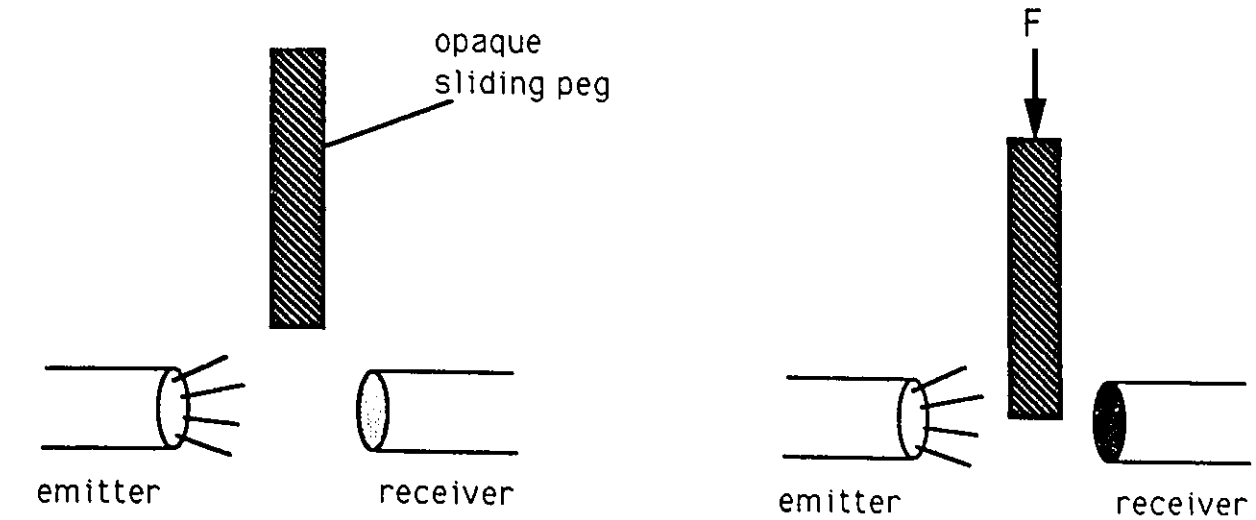
On the negative side, silicon is inherently brittle, so there is not much tolerance to overload. Also, the stain gages are subject to the same environmental hazards as all semiconductors : temperature, humidity, photoelectric sensitivity, EMI, and fatigue all affect the response.

2.4.1.6 Optoelectronic

Advances in fibre optics and optoelectronics over the last few years have generated a number of transmission/receiving devices which can be used effectively to measure mechanical deflection. In optoelectronic tactile sensors the contact pressure modulates the intensity of a light source, which is then measured.

There are two categories of optoelectronic tactile sensors : external reflectance, and internal reflectance. In external reflectance devices, the light escapes from the carrier (usually a fibre optic cable), is modulated through mechanical means, and is received and measured. There are a number of methods by which the light is modulated. In the LORD LTS210 sensor, pressure on a sensing pad advances a peg into a cavity containing a photoeimtter/receiver pair, illustrated in Figure 2.7. The greater the pressure, the further the peg is advanced into the cavity and the more light is blocked. In other schemes, a reflective surface advances towards the transmitter/receiver pair, increasing the intensity with greater pressure [28], as illustrated in Figure 2.8. Reflective elastomers have similarly been used to simplify the mechanics.

The LORD LTS100/200 sensor was the most successful commercially available tactile sensor. 160 photoemitter/receiver pairs were arranged over a 10X16 array at 1.8 mm intertaxel spacing, as illustrated in Figure 2.9. The contact surface was an elastic rubber pad which was ciliated on



a) no applied force

b) applied force F occludes light transmission

Figure 2.7 : Optoelectronic Displacement Sensor with Occlusion

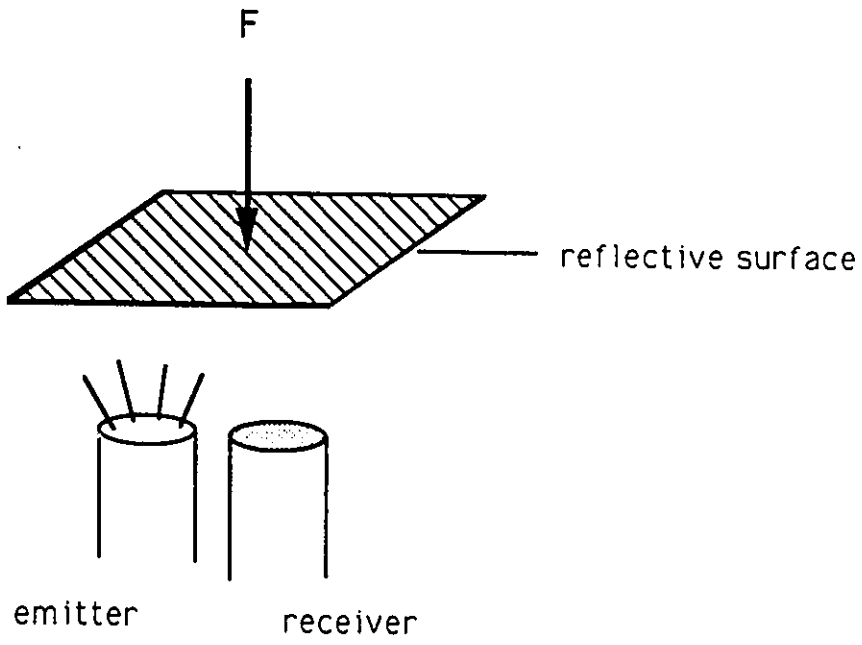
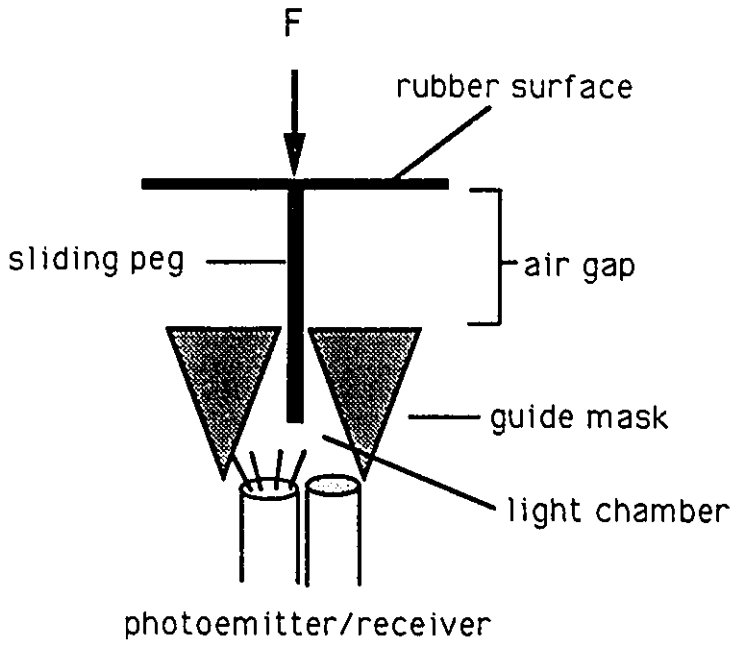
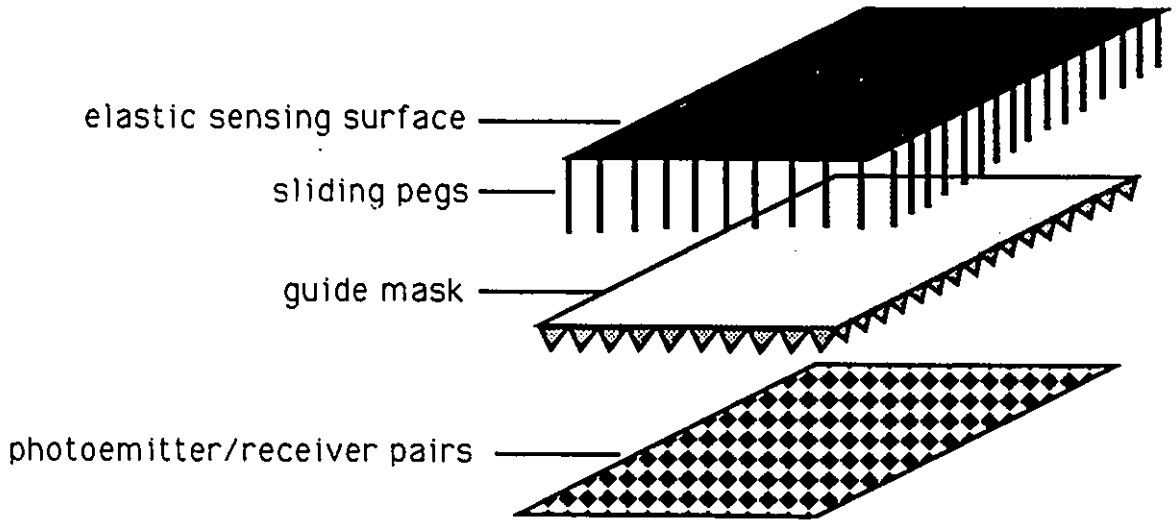


Figure 2.8 : Optoelctronic Displacement Sensor with Reflectance



a) cross section of single taxel



b) array of taxels

Figure 2.9 : Construction of LORD LTS210 Optoelectronic Sensor

its inner surface. For each taxel there was a 5 mm long peg which extended through a guide mask and into a light chamber. The sensor was capable of detecting 0.001 in depressions in the rubber pad, corresponding to force increments of 0.01 lbs. The response was highly linear, repeatable, and showed little hysteresis. Unfortunately, the sensor was rather delicate, and the pegs had a tendency to pull through the guide mask and disabling the sensing site.

In contrast, internal reflectance sensors do not allow the light source to escape the carrier. The pressure is applied directly to the carrier and alters the intensity of light reflected over an area. The simplest of these devices is constructed from a sheet of plexiglass covered with a textured flexible foam sheet, as illustrated in Figure 2.10. Light is beamed laterally through the edge of the plexiglass. Contact with the foam sheet impresses a pattern onto the surface of the plexiglass which will reflect the path of the light and project a dark spot on the bottom surface of the plexiglass. The image on the bottom surface is then captured by a video camera.

One of the great advantages of internal reflectance sensors is that the image is received by standard video equipment. The resolution of the sensor is therefore not limited by the mechanical layout of the taxels, but rather by the resolution of the video camera. Further, by acquiring and transmitting the image through fibre optic cables, non planar sensors can be constructed.

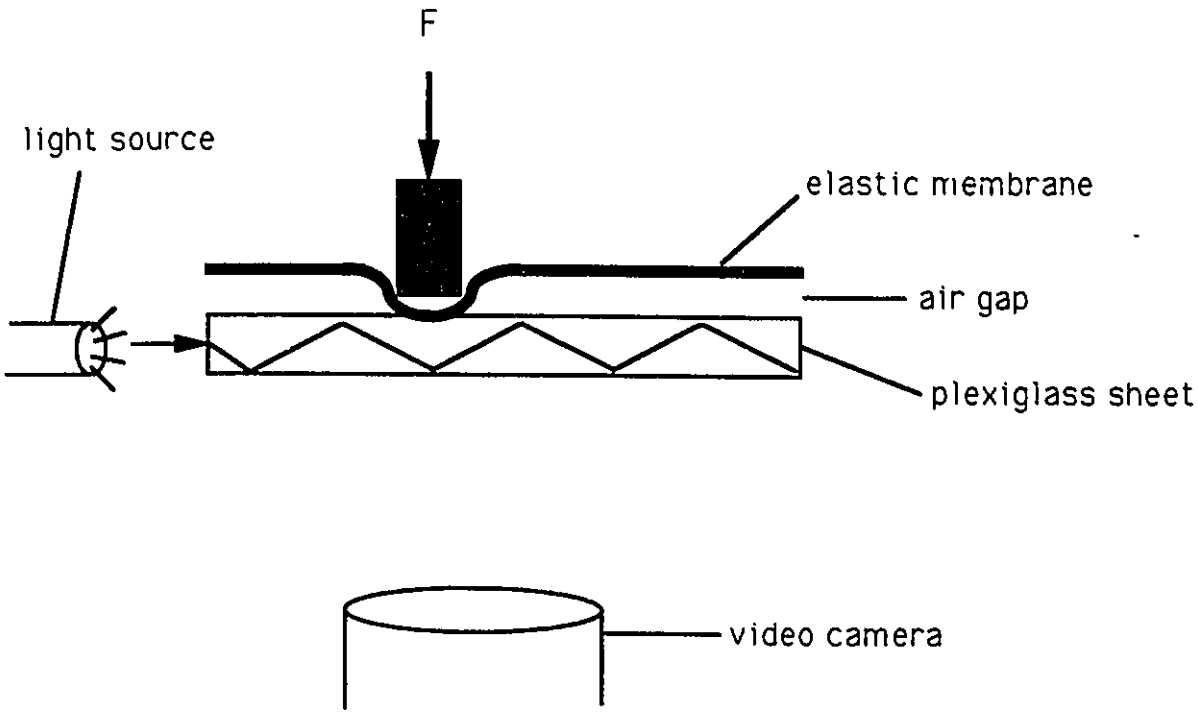


Figure 2.10 : Total Internal Reflectance Pressure Sensor

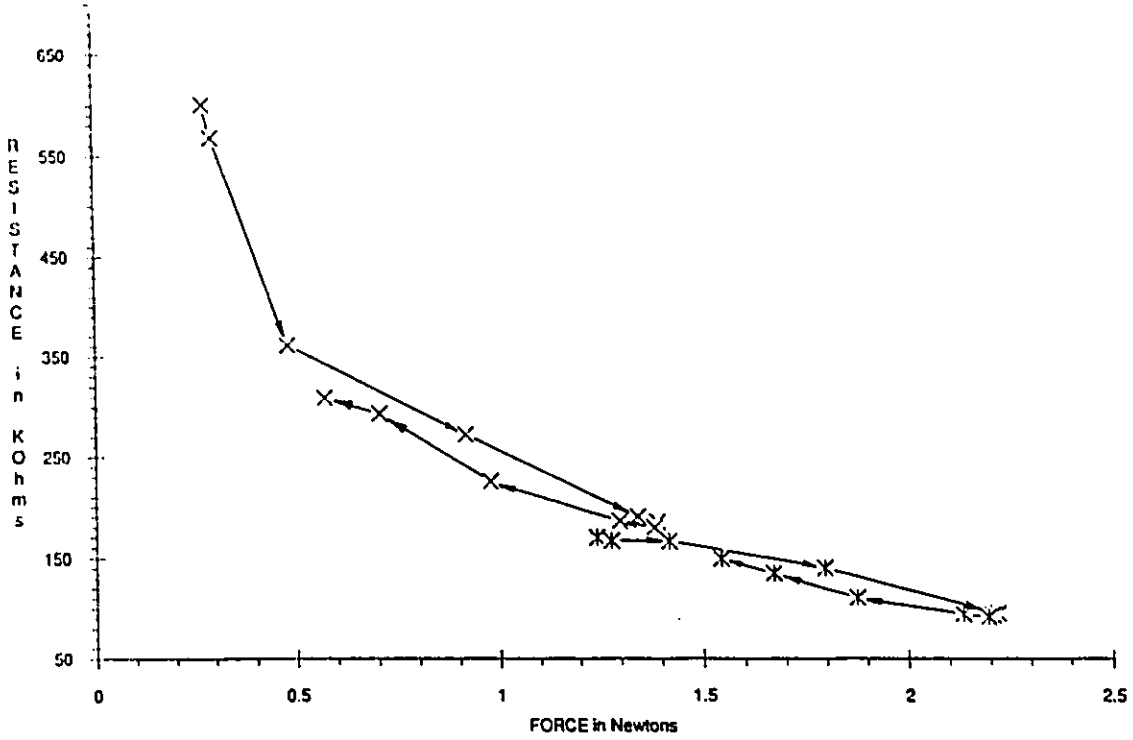
Begej [29] has developed a table top sensor based on the principle of internal reflectance. This sensor had the startling resolution of 1500 taxels/cm^2. In another design, the image is not read directly by video camera, but rather is diverted through a bundle of fibre optic cables. This resulted in a more compact version of the sensor, with the acceptable resolution of 54 taxels/cm^2.

The use of optoelectronics solves a number of problems in tactile transduction. Light is not subject to EMI, temperature, or other conditions which limit other sensors. Also, photodetection technology is mature and miniature and fibre optic cables provide a medium for the transmission of signals.

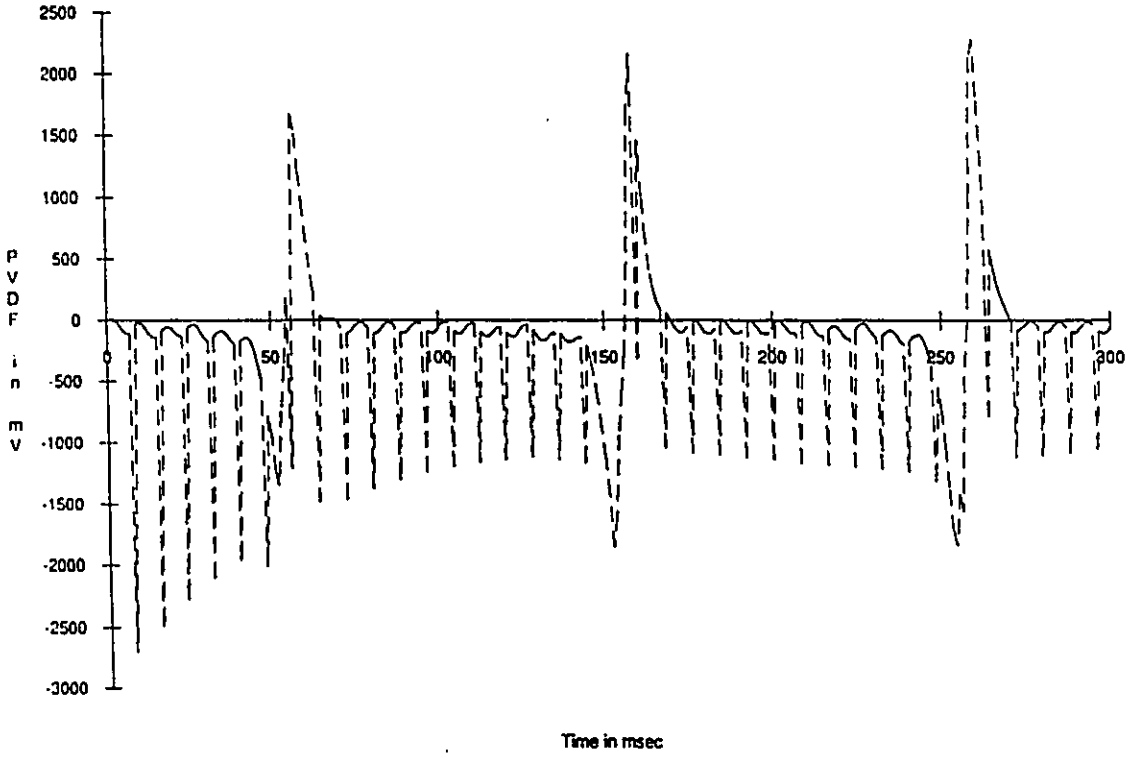
2.4.1.7 Piezoresistive (FSR)

The most promising material industrially available is a piezoresistive substance manufactured by Interlink Electronics, Inc. Piezoresistance is a change of electrical resistivity under applied force. Commonly known as a force sensitive resistor (FSR), this proprietary thick film polymer exhibits a decreasing resistance under applied pressure.

A number of samples were acquired from Interlink Electronics, Inc., and the change in resistance of the material under applied pressure was measured. The results are displayed in Graph 2.1. The initial unloaded resistance was approximately 575 kohms, and decreased nonlinearly but



Graph 2.1 : FSR Response to Applied Force



Graph 2.2 : PVDF Respnse to Applied Force

monotonically to a value of 100 kohms under a loading of 2.2 N. Following the direction of the arrows between data points, notice that the response upon unloading was biased slightly lower than that upon loading. This was the effect of the hysteresis inherent in the material.

The Interlink FSR has been utilized in a number of tactile sensing arrays. It has the qualities of large dynamic range, great tolerance to overload (we were not able to damage the substance with any amount of normal force), and the response is low in noise and easy to measure. The polymer itself can be deposited on a variety of different surfaces, and is currently marketed as a 16X16 array on a 1 in^2 area.

2.4.1.8 Piezoelectric (PVDF)

Piezoelectric materials generate an electrical charge when subjected to applied pressure. Piezoelectric materials include natural substances with a preferred polarization at the molecular level, such as quartz and other crystals, and some biological tissues, such as wood.

Piezoelectrics can also be manufactured as a ceramic or a thin film. The ceramics are produced by heating the nonpolar material to a suitable temperature (the Curie temperature) at which point they become fluid. The fluid is subjected to a strong magnetic field and cooled. The magnetic field aligns the dipoles and when the ceramic solidifies, it exhibits piezoelectric properties. The film Polyvinylidene Flouride (PVDF or PVF_2)

is produced in a similar manner and is manufactured in large sheets which resemble plastic wrapping paper. The film can be made as thin as 10 μ and has the advantage of being flexible.

As a force sensing material, peizoelectrics have had limited success [30,31,32]. They have a great sensitivity, and have been used as the sensing elements for acoustic and underwater microphones. The problem with the material for force sensing is that the response is dynamic. The electrical charge produces a spike when contacted, and quickly dissipates. This means that the signals must be integrated to interpret the static response, a process which has not been successfully implemented.

Another disadvantage is that all piezoelectric materials are also strongly pyroelectric, meaning that a change in heat is transduced as an electrical signal. In order to interpret the force information, the signal must be decoupled into its temperature and force components [32].

We performed an experiment to illustrate the transient nature of the force transduction of piezoelectrics, the result of which is charted in Graph 2.2. A periodic force was applied across a piece of 16 μ thick PVDF film, and the generated voltage was measured. The force was applied at a frequency of 10 Hz, which corresponds to the peaks at the 50, 150, and 250 ms. All other peaks are caused by line noise. The sharp negative peaks at -2000 mV are followed by a sharp positive peak of the same magnitude. This was the characteristic response to an applied constant force, and demonstrates the transient response of piezoelectric materials.

2.4.1.9 Other Materials

Other material properties which have been used in tactile transduction include :

optoresistance and photoelasticity : the optical properties change with applied pressure,

optical phase modulation : the applied pressure is used to modulate the phase of the light frequency, rather than the intensity,

Hall effect : applied pressure is used to induce the Hall effect in transistors,

ultrasonic : similar to optical techniques, except that the carrier is sound rather than light waves.

2.4.2 Object Recognition Systems

The first comprehensive survey of the state of the art, and the natural starting place for any study into tactile sensing, was presented by Harmon[5,6,7]. He reported on developments in transduction technology, analysis of the tactile data, and also relevant aspects of the physiology of human touch.

Subscribing to the anthropomorphic view of design, Harmon proposed a set of 5 criterion for the performance of an ideal tactile transducer. These criterion were based on experiments into the physiology of human taction, and are commonly used today as an outline for the development of transducers.

Less well known is Harmon's proposal for the development of a tactile language. Motivated by the observation that there is no adequate terminology to describe the intricate hand movements of human taction, he proposes the development of language which could be used to decompose taction into its primitive activities. He proposed a syntax consisting of 3 constructs ; elements, object condition, and operation.

The elements are the primitive features of tactile information, which describe the physical properties and conditions necessary to characterise an object. The elements consist of both intrinsic properties, such as shape, texture, and hardness, and extrinsic properties, such as force, moment, and displacement. The object condition is a statement

constructed from the elements, which describes an object and its status at a given time. The operations implement algorithms for active sensing, pattern recognition, and hierarchical control in order to accomplish a given task.

While the early work on tactile sensing was concerned with the development of tactile transduction technology, it was often accompanied by some attention to the analysis of tactile data. This was presented in the context of demonstrating the transducer function, and as such was restricted to analysis of the cutaneous component of the data. The tactile image was treated analogously to a visual image, for which there exist a plethora of processing techniques. The application of the data analysis routines was limited to the recognition of objects which were smaller than or roughly the same size as the transducer, and the kinesthetic and local aspect of the data was ignored. Examples of the treatment of purely cutaneous tactile images include [21,42,43].

Harmon was highly critical of the analysis of the purely cutaneous component of tactile data ;

> "It would seem that passive, static taction in automata would offer little potency and, in fact, probably draw attention away from the important problems."

As the limitations of purely cutaneous analysis became apparent, research began to focus on the coupling of tactile sensing with visual

image processing [44-48]. In an attempt to emphasise the strengths of both sensing modalities, visual data was processed to provide a rough global view of the environment, while taction was used to resolve ambiguities in the scene through active exploration.

An ambitious project along this theme was that of S.A. Stanfield [48]. Stanfield's experimental setup consisted of a LORD tactile sensor mounted on the end effecter of a 6 d.o.f. PUMA 560 manipulator, and a pair of CCD cameras. The purpose of this work was what Stanfield called the "apprehension" of unmodelled objects, i.e., the determination of the object features and their relationships.

The 3D vision system provides sparse, moderately inaccurate range information. These clues to the global structure of the object were then used by the tactile system to guide an exhaustive exploration of the local regions. The system was reported to have successfully searched and described a number of common kitchen items.

A particularly interesting aspect of this work was that the system structure was based on theories by Marr, Lederman, Klatsky, about the structure of the human perception system (see Appendix A). The tactile information was divided into 7 low level primitives and 9 features, based on observations made by Lederman about the human tactile system. Once the information from each exploratory procedure had been acquired, it was slotted into a frame database.

Another successful attempt at integrating tactile sensing and vision was that of Allen [44-46]. Allen's setup differed from that of Stansfield in that the sensor used was a finger shaped tactile probe, which had 133 sensitive sites distributed over 12 faces. The data derived from this sensor was point data, or simple touch, rather than true tactile data. This point data was used to build surface models of objects based on Coons splines, which require 4 knot points for each surface patch. Again, the visual information was used as a global reference to guide the tactile sensing.

While the integration of tactile and vision systems advanced the emphasis of the kinesthetic aspect of the data, the difficult nature of the taction problem became more apparent as the sensed objects were modelled three dimensionally. As tactile sensing is a local sensing, acquiring data from only a small fraction of the sensed object, it was necessary to consider all components, including the kinesthetic component, of the information to achieve adequate recognition.

The most illuminating work in the area of object recognition using geometric models was performed by Lozano-Perez and Gaston [49] for the two dimensional case, and Grimson [50] for the three dimensional case. The problem was formulated as follows : given a small number of sparsely sensed data points, and a database of object models, how can the data be tested against the model database to identify potential matches and poses?

The most straightforward method of testing the data against the

database was a generate-and-test technique. This method is prohibitively computationally expensive, however, as it is exponential in the number of polyhedral model faces. The developed solution compiled very simple features, such as distance ranges across faces, and normal ranges, about each model in the database. The features were also calculated over the sensed data, and this reduced set of information was used to search through the database. By further organizing the data as a tree structure, the infeasible solutions could be efficiently pruned.

It is interesting to note that a very similar approach was proposed at roughly the same time by Browse [51]. Browse again identified geometric features about the object models, and compared them to the measured data. Browse tested his system with a hand held video camera which was successively positioned over local areas of an image.

A unique approach to three dimensional object recognition was developed by Allen [45]. A number of common shapes were modelled as superquadrics, a form of three dimensional surface function. A superquadric function can take on a number of geometric shapes, while it is dependent upon only 11 parameters. By grasping a number of objects, and recording the grasp positions of the contacting surfaces, the shapes of the objects were closely reproduced as superquadrics. Of course, not all objects can be modelled as superquadrics, and the model is best reproduced if the object is sensed at a number of disparate locations, such as in grasping.

2.5 Conclusions

In this chapter, sensing has been presented as not only a desirable but a necessary component of a robotic system. In cases where the environment is unstructured, sensing provides a mechanism for a greater degree of system autonomy. In cases were the environment is very structured, such as a factory, sensing is needed to reduce the cost of automation.

There is a clear cost tradeoff between the degree of environmental structuring and the degree of advanced sensing in a robotic system. As sensing technologies advance and are proven, the cost of implementing these systems will decrease making sensing a preferable alternative to environmental structuring. Successes in the development of sensing systems will also advance robotic technology into more unstructured environments.

Tactile sensing has been presented as an emergent sensing technology, and a number of difficulties particular to this sensing mode were identified. One difficulty is that tactile sensing is an active sensing mode, which requires fine control of the kinematics of the manipulator. Another problem is that the transduction of tactile information is inherently difficult. A number of tactile transduction technologies were presented, but none has emerged as a standard. No tactile sensor has yet been developed which satisfies all of Harmon's ideal characteristics.

The final difficulty with tactile sensing is its use for object recognition. A number of representative developments were surveyed which attempt to use tactile information to identified objects. Early taction systems considered only static cutaneous images, and treated the data analogously to visual image processing. Later developments incorporated static kinesthetic information, which utilized the three dimensional component of the data.

The main conclusion of this chapter is that effective tactile sensing capabilities will develop only when the active component of the sensing process is taken into consideration. The transition has been made from the treatment of tactile data as static cutaneous images, to the inclusion of the kinesthetic component of the data. The next advancement in tactile sensing will be a treatment of the active component of the data acquisition.

Chapter 3 : Manipulator Kinematic Solution

3.1 Introduction

Robotic manipulators are controlled at the simplest level through the transmission of angular values from a host computer directly to the robot joint controller. This is known as joint space or configuration space control. A manipulator location is identified by the joint values, and no reference is made to the Cartesian position of the end-effector.

As the location of objects and sensors in the robot's workspace are specified by their Cartesian coordinates, the end-effector locations are much more naturally expressed in the world Cartesian coordinate system rather than joint space.

The kinematic solution provides the mapping between the two spaces. It is partitioned into two parts : the forward kinematic solution (FKS), and the inverse kinematic solution (IKS). Given the manipulator joint values, the FKS is used to calculate the Cartesian position and orientation of any point on the manipulator, usually the end-effector. Conversely, given and specified Cartesian location, the IKS returns the set of joint values which will position the end-effector at that location.

To clarify the concept of the kinematic solution, consider the two link planar manipulator illustrated in Figure 3.1. The links lie on the x-y plane,

and are controlled by the values of the two revolute joints Θ_1 and Θ_2, where Θ_1 is the angle between the x-axis and link L1, and Θ_2 is the angle between link L1 and link L2.

The FKS is the function which returns the location of a point $P = [x_p, y_p]$ defined at the end of the second link ;

[3.1] $$FKS(\Theta_1, \Theta_2) = [x_p, y_p]$$

Inverting the FKS provides an expression for the IKS. The IKS returns the joint values $[\Theta_1, \Theta_2]$ which position the end-effector at the desired location P ;

[3.2] $$IKS(x_p, y_p) = [\Theta_1, \Theta_2]$$

A simple geometric derivation reveals the expressions for the functions ;

[3.3] $$FKS(\Theta_1, \Theta_2) = \begin{cases} x_p = L_1 \cos(\Theta_1) - L_2 \cos(\Theta_1 + \Theta_2) \\ y_p = L_1 \sin(\Theta_1) - L_2 \sin(\Theta_1 + \Theta_2) \end{cases}$$

[3.4] $$IKS(x_p, y_p) = \begin{cases} \Theta_1 = \tan^{-1}(y_p / x_p) \\ \qquad + \sin^{-1}(L2 \ \sin(\Theta_2) / (x_p^2 + y_p^2)^{1/2}) \\ \Theta_2 = \cos^{-1}((L_1^2 + L_2^2 - x_p^2 - y_p^2) / 2 L_1 L_2) \end{cases}$$

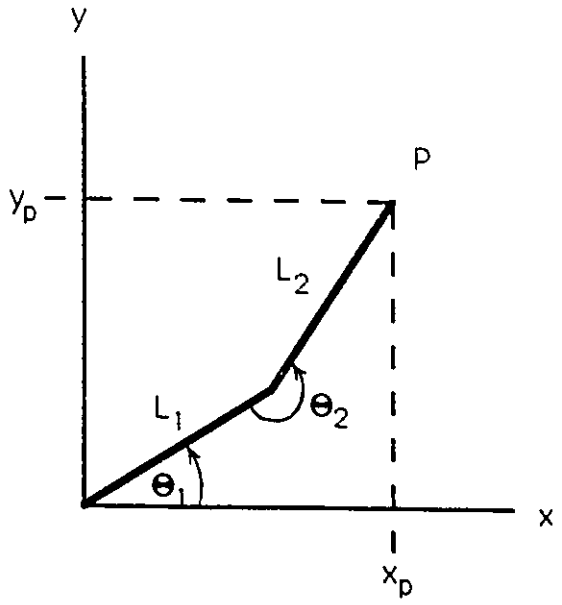


Figure 3.1 : Two Link Planar Manipulator

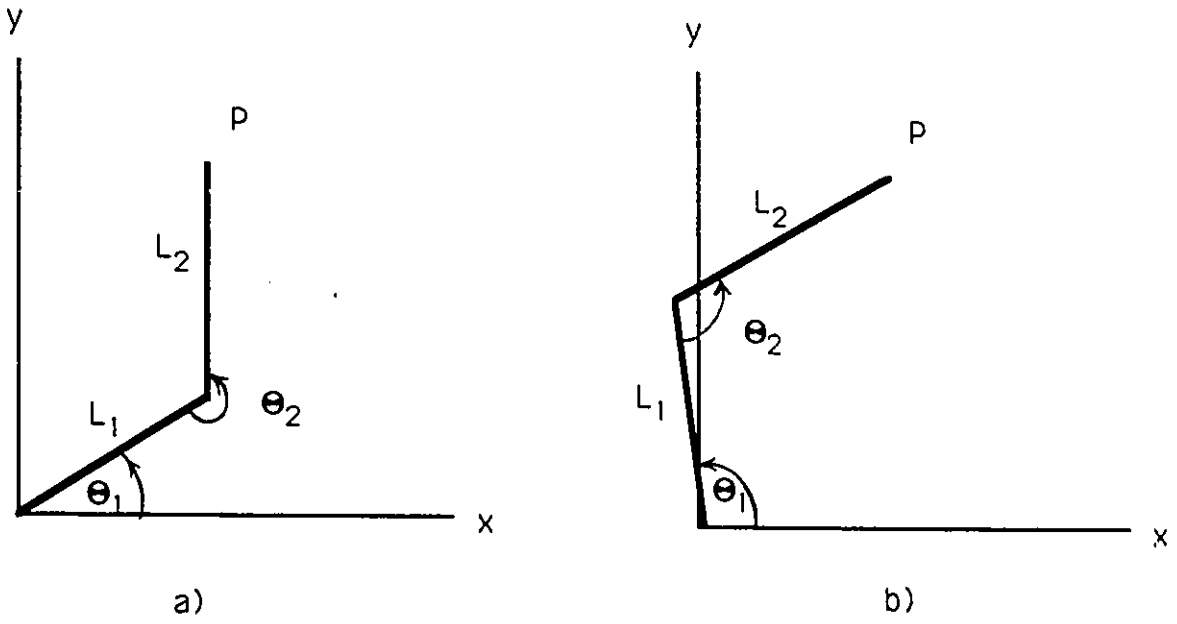


Figure 3.2 : Redundant Configurations of the
Two d.o.f. Planar Manipulator

To solve the kinematics for a mechanical linkage, it is necessary to describe the system mathematically. A matrix notation for the mathematical description of robotic manipulators was first proposed by Denavit-Hartenberg [62]. It provides a systematic convention for establishing a coordinate system for each manipulator link.

Each link is fixed within its own coordinate system. A link coordinate system is defined with respect to the previous link, and so the successive multiplication of each sequential transformation results in a relationship between the end-effector coordinate frame and the reference coordinate frame. The end-effector location can therefore be described in base or world coordinates, which is exactly the goal of the FKS.

The notation specifies rules for generating a 4X4 homogeneous matrix for each link coordinate frame. The matrix describes the relationship between a link coordinate frame and that of its most proximal neighbour. As each link frame is connected to the previous frame through exactly one joint, the transformation matrix can expressed in terms of the joint angle and a number of geometric constants.

Each homogeneous matrix is composed of four submatrices, as illustrated in Figure 3.3. The rotational submatrix is an expression of the rotational offset between neighbouring coordinate frames. If the joint is revolute, this submatrix will be a function of the joint value. The positional submatrix describes the translational offset between the two frames. If the joint is prismatic, the joint value will have an effect on

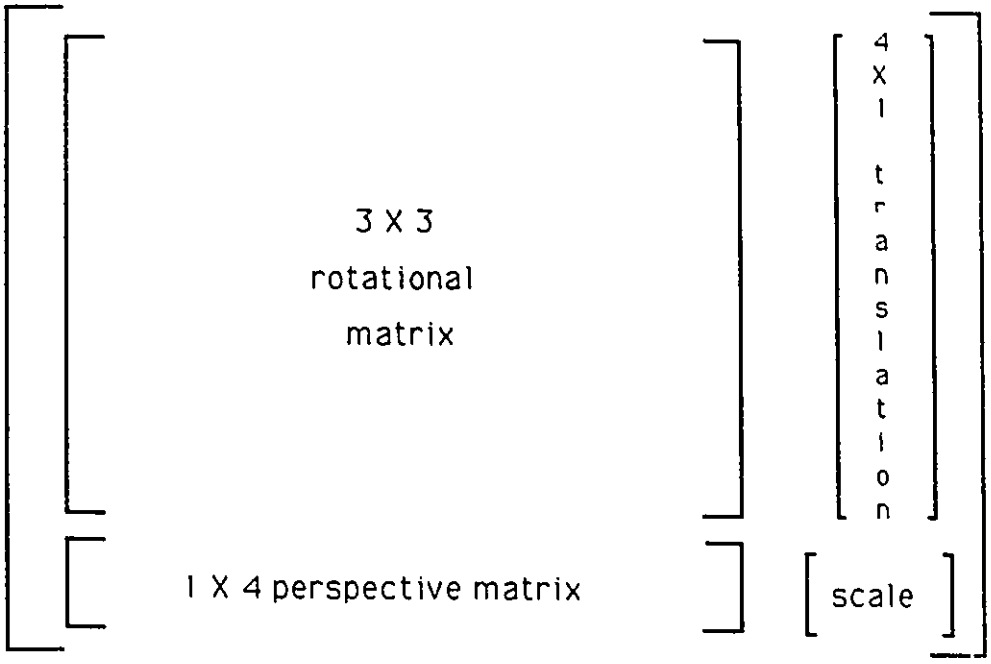


Figure 3.3 : 4X4 Homogeneous Matrix

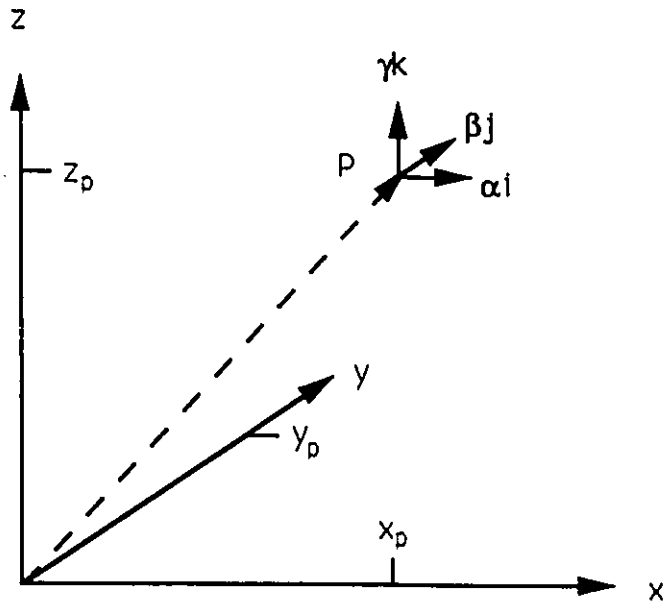


Figure 3.4 : Location Vector

this submatrix. The scaling and perspective submatrices complete the general transformation, and are used in computer graphics applications. In robotics, the scaling factor is conventionally set to 1, and the perspective matrix is zero. The rotational submatrix defines a relationship between the orthogonal bases of the two coordinate frames. The components of the [i,j,k] vectors in the transformed frame are the rows of the matrix, expressed in the reference frame.

Denavit-Hartenberg proposed a convention for generating the transformation matrices for a given manipulator geometry. The coordinate frame corresponding to the i^{th} joint value is described by the following 4 quantities ;

θ_i = angle from x_{i-1} axis to x_i axis about the z_{i-1} axis. In a revolute system, this is exactly the angle corresponding to a rotation of the i^{th} joint. In a prismatic system, this value is constant.

d_i = the distance, along the z_{i-1} axis, from the origin of the $i\text{-}1^{th}$ coordinate frame to the intersection of the z_{i-1} and x_i axis.

a_i = the offset between the intersection of the z_{i-1} axis with the x_i axis to the origin of the i^{th} frame.

α_i = the offset between the z_{i-1} and z_i axis. In a revolute system, this value is constant. In a prismatic system, this value corresponds to the extension of the joint.

From the above quantities, each coordinate frame can be identified according to the following rules ;

1. The z_i axis is the axis of rotation for the $i+1^{th}$ joint,
2. The x_i axis is normal to the z_{i-1} axis,
3. The y_i axis completes the right handed coordinate system.

C.S.G. Lee [63] has proposed an 11 step algorithm for generating the link coordinate parameters of any robotic manipulator. Following these steps for the RHINO XR3 generates the manipulator coordinate frames described in Figure 3.5.

One of the oldest problems in robotics, there is still today much research into the development of effective kinematic solutions for robotic manipulators. The FKS is direct : given a set of joint values, there exists a single closed form solution to the 6 d.o.f. end-effector position and orientation. The IKS however is equivalent to the solution of a set of non linear, possibly underconstrained, equations, and therefore poses a more difficult problem.

There are two main forms of IKS : geometric and iterative. The geometric method expresses the system equations in closed form, allowing for a direct solution. The iterative method successively iterates an error expression involving the joint values until it eventually converges upon a solution.

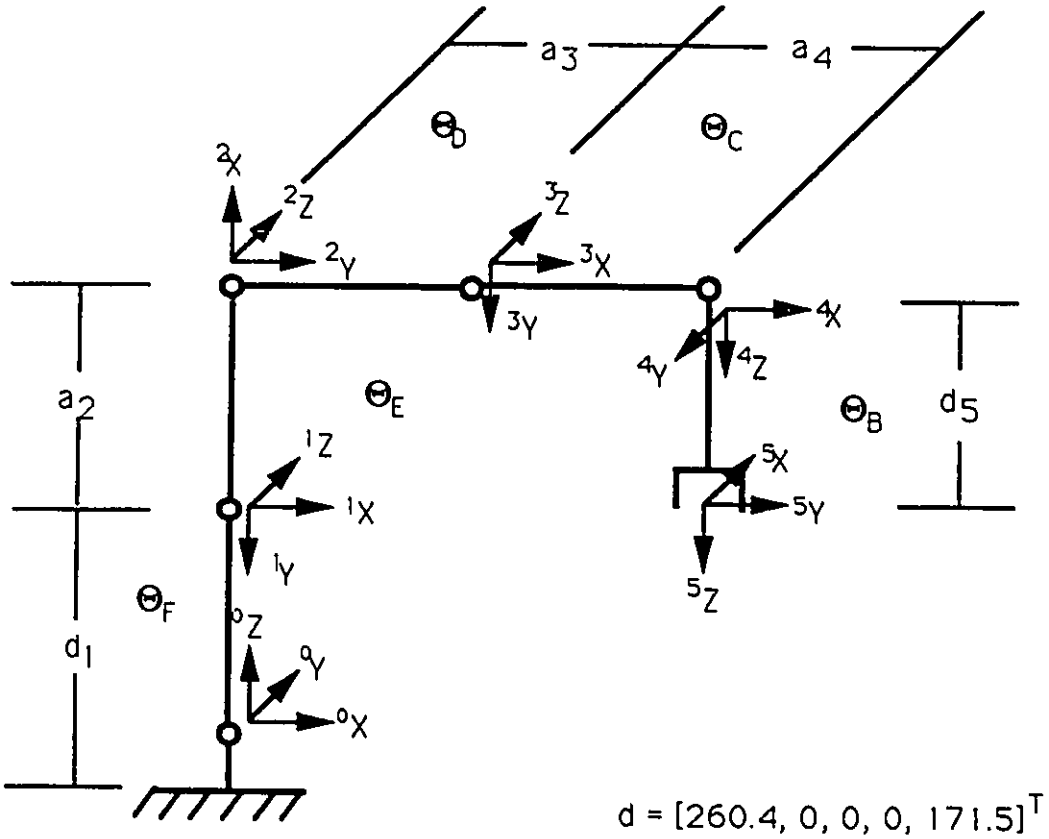


Figure 3.5 : Denavit Hartenberg Coordinate Frames for RHINO XR3

The most computationally efficient method is the geometric [65]. In the cases where there exist closed form geometric solutions, the calculation is a set of trigonometric transforms as derived through the homogeneous matrix transformations. Any configuration decisions due to link redundancy can be resolved through heuristics which relate directly to the mechanical configuration, such as elbow up/down, shoulder left/right, etc. Note that a geometric solution exists for 6 d.o.f. articulated manipulators with a coaxial 3 d.o.f. spherical wrist, such as the PUMA line.

There are manipulators where the mechanical linkage does not permit a closed form geometric solution. In redundant manipulators there may be sets of configurations which locate the end-effector equivalently. An obvious heuristic for resolving ambiguities is not always clear, and it may be necessary to resort to other considerations, such as angle minimisation. In non-redundant manipulators there may also exist degeneracies in the linkage, which must be dealt with equivalently.

An example of two redundant configurations is illustrated in Figure 3.2. The end-effector, point P, is positioned at the same Cartesian coordinate $[x_p, y_p]$ in both the configurations in part a) and b). Note that while the position of the end-effector is equivalent its orientation, i.e., the angle with which L2 is positioned in the reference frame, has changed. Adding one more link to the planar manipulator allows redundancy in both the position and the orientation of the system.

The iterative technique is widely employed in cases where there is no closed form geometric solution. Given a desired location, and an initial estimate of the joint configuration, the iterative technique will repeatedly vary the joint values according to some error metric. Eventually, the FKS of the joint configuration will converge to the desired end-effector location. Although less computationally efficient than the geometric method, the iterative technique is applicable to redundant and degenerate linkages. An example of a general iterative solution which can be applied to any mechanical linkage is presented in [66].

The solution developed here combines aspects of both the iterative and geometric methods. The manipulator is partitioned at the wrist, and only the two most proximal joints are solved iteratively. The remaining three joints are then solved in closed form through a set of geometric expressions. It is a computationally efficient solution : the error function is a single computational expression derived from an analysis of the manipulator forward kinematics, and represents a measure of the 3D Cartesian distance between the end-effector location of the intermediate iterative configuration and the desired final value.

3.2 RHINO XR3 Manipulator

Both the FKS and IKS are dependent upon the geometry of the manipulator. Prior to proceeding to the details of the kinematics, it is necessary to identify the geometric linkage of the RHINO XR3.

Following the method presented by C.S.G.Lee, the Denvit-Hartenberg coordinate frames for the Rhino XR3 were developed, as illustrated in Figure 3.5.

The Denavit-Hartenberg convention provides a standard mechanism for the identification of manipulator geometries. In general, the joint coordinate frames can be defined in any convenient manner, with the rotations/translations occurring about any of the coordinate axes. In the development of the kinematic solution, an alternate coordinate frame composition will be introduced.

3.2.1 Location Vector

The goal of the kinematic solution is the control of the location of a point on the end-effector. The location is defined in the reference coordinate frame, and in general consists of 6 parameters : 3 position and 3 orientation values. The positional parameters describe the [x,y,z] offset of the end-effector in the reference frame. The orientational parameters describe the attitude of the end-effector; the roll, pitch, and yaw. The

combination of all 6 parameters is termed a location vector.

It is not always possible to locate a manipulator generally. If the manipulator has fewer than 6 d.o.f. (i.e. 6 joints), it will not be possible to specify all 6 location parameters. As a rule of thumb, it is necessary to have at least as many d.o.f. as there are location vector parameters. Even with 6 d.o.f., however, there may exist certain locations, or sets of locations, which are unattainable, due to coupling or redundancies in the linkage. The union of all of the attainable location vectors is termed the reachability or working envelope of the manipulator.

As the RHINO is a 5 d.o.f manipulator, it is not possible to fully control the location of the end effector. There will always remain one degree of freedom which is arbitrarily set by the solution. The goal, however, is to control the placement of a planar sensor, and as the position of a plane can be described in space by only 5 parameters, a five d.o.f. manipulator is acceptable.

Let the location vector be defined by the parameters $[x_p, y_p, z_p, \alpha_p, \beta_p, \gamma_p]$, where $[x_p, y_p, z_p]$ are the positional and $[\alpha_p, \beta_p, \gamma_p]$ the orientational components in the respective **i**, **j**, and **k** directions, as illustrated in Figure 3.4.

The positional components are straightforward : they are the Cartesian coordinates of the end-effector in the reference frame. The orientation parameters describe the normal direction of the plane in unit

vector coordinates. An equivalent representation quantifies the orientation with 2 parameters in spherical coordinates.

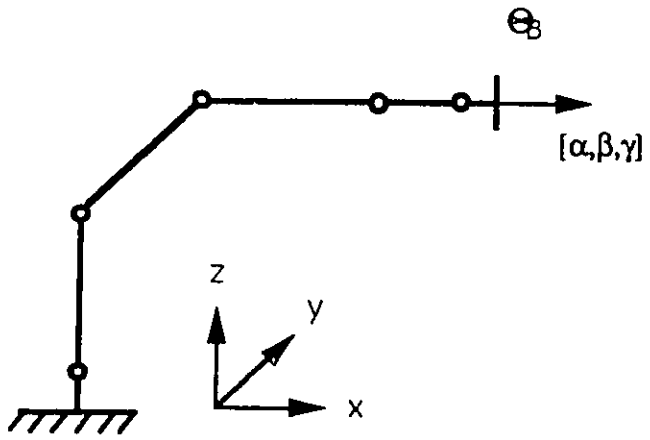
In the following kinematic solution, the location vector will describe a point P at the center of the planar end-effector bracket, known as the point-of-control.

3.2.2 End-Effector Bracket

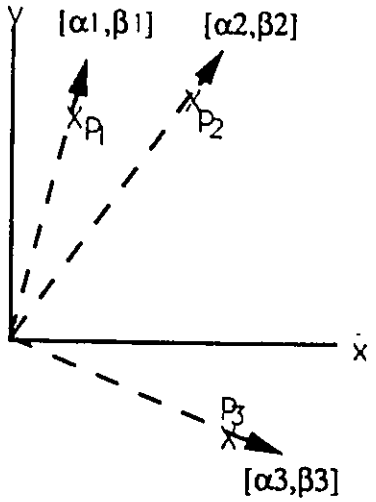
There was a sacrifice of one locational degree of freedom imposed by the geometry of the manipulator. Not all 5 d.o.f. are equivalent, however, and to achieve the desired reachability, it was necessary to modify the mechanical system by mounting an end-effector bracket, offset at an angle to the joint B.

To appreciate this point, consider the consequences of mounting the bracket flush with the joint B, as illustrated in Figure 3.6 a). A rotation of joint B is equivalent to a rotation around the plane normal, and does not affect the **i** and **j** components of the orientation. This restricts the orientation vector to lie concentric with the center of the manipulator base, as illustrated in Figure 3.6 b).

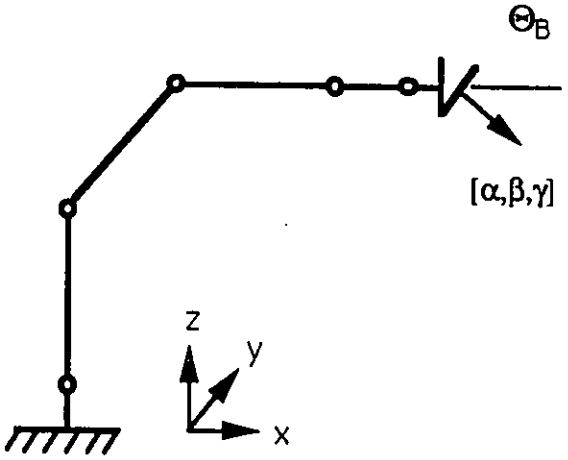
Alternately, by defining the end-effector coordinate frame to be offset from joint B at 45^{o} as in Figure 3.6 c), the possible orientation vectors associated with each point is expanded to the hemisphere



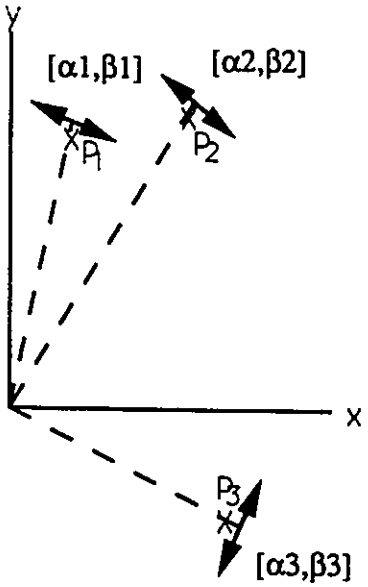
a) end-effector bracket
flush with joint B



b) orientation vectors
radial with base



c) end-effector bracket at
45 degree angle with joint B



d) orientation vector
range increased

Figure 3.6 : End-Effector Bracket Angle and Reachability

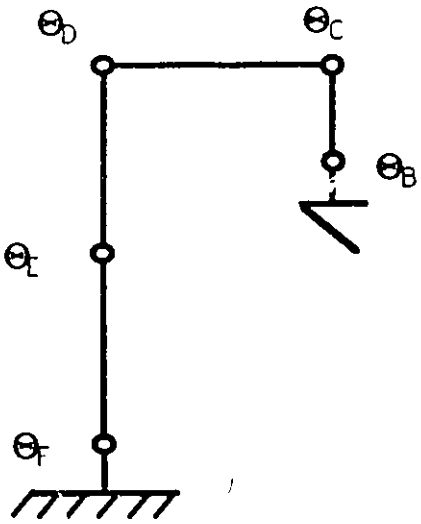
around the radial direction, as illustrated in Figure 3.6 d). In effect, the rotational d.o.f. normal to the point of control has been forfeited for two general orientational components. There still therefore exists one uncontrollable d.o.f., but this rotation is not necessary for the successful mating of the sensor with a planar surface.

Any offset angle would suffice to improve the controllability of the planar end effector. The value of 45° was believed to provide the greatest reachability from the resulting mechanical system.

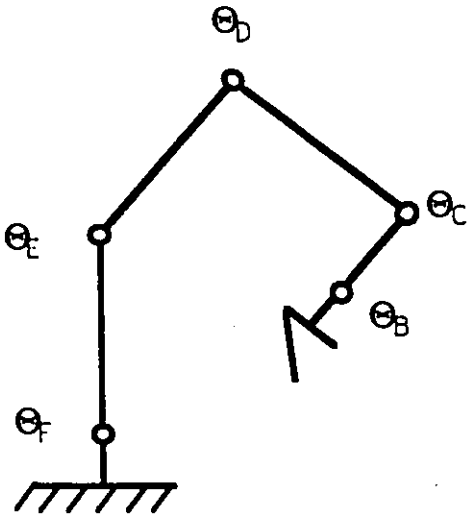
3.3 Forward Kinematic Solution

Having described the manipulator according to the Denavit-Hartenberg frames, we will now depart from that convention and solve the FKS using an alternate frame system and vector algebra. The reason for this departure is that joints C, D, and E are mechanically coupled, such that a change in the value of a proximal joint is compensated for in the remaining distal joints.

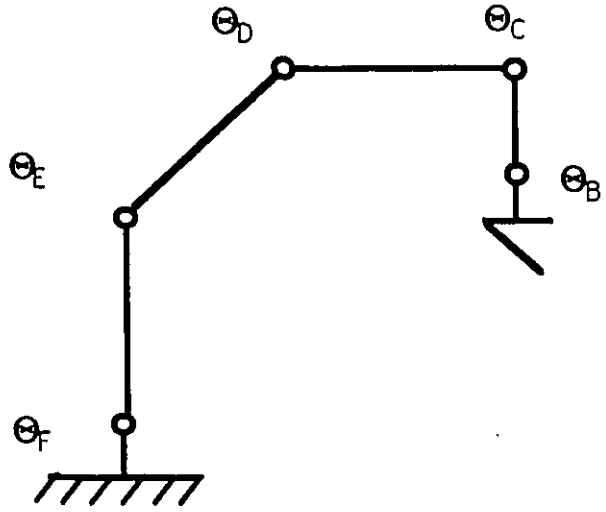
This mechanical coupling is illustrated in Figure 3.7. In part a) the manipulator is in the home position. Part b) illustrates a change in configuration due to a change in the joint E value without mechanical coupling. The attitude of the more distal links changes in the world reference frame. In part c), the same change to joint E is encountered, this time with mechanical coupling. The successive links remain parallel to fig



a) home position



b) change in Θ_E without mechanical coupling



c) change in Θ_E with mechanical coupling

Figure 3.7 : Effect of Mechanical Coupling

their attitude in the home position, and the orientation of the end-effector remains the same.

The effect of this mechanical coupling in the kinematic solution is to allow the orientational and positional aspects of the FKS to be derived separately, which simplifies the solution of the IKS.

The following notational conventions will be used through the FKS and IKS solutions ;

$[x_p, y_p, z_p, \alpha_p, \beta_p, \gamma_p]$ = location vector, the position and orientation of the point-of-control in the world reference frame,

$[\Theta_B, \Theta_C, \Theta_D, \Theta_E, \Theta_F]$ = joint location, the values of the 5 manipulator joints,

${}^{A}U$ = reference frame A, A E [W,F,E,D,C,B,P]

${}^{A}T_Z$ = homogeneous transformation from ${}^{Z}U$ to ${}^{A}U$,

The coordinate frame system is illustrated in Figure 3.8. The goal is to express the bases of the point-of-control frame ${}^{P}U$ in the world reference frame ${}^{W}U$. This is accomplished through developing the transformations in the following kinematic chain ;

[3.5] $$ {}^{W}T_{P} = {}^{W}T_{F}\,{}^{F}T_{E}\,{}^{E}T_{D}\,{}^{D}T_{C}\,{}^{C}T_{B}\,{}^{B}T_{P} $$

From the geometric relationships in Figure 3.8, the following homogeneous transformation matrices are generated;

[3.6] $$ {}^{W}T_{F} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_{UF} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

[3.7] $$ {}^{F}T_{E} = \begin{bmatrix} \cos\Theta_{F} & -\sin\Theta_{F} & 0 & 0 \\ \sin\Theta_{F} & \cos\Theta_{F} & 0 & 0 \\ 0 & 0 & 1 & L_{FE} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

[3.8] $$ {}^{E}T_{D} = \begin{bmatrix} 1 & 0 & 0 & L_{ED}\cos\Theta_{E} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_{ED}\sin\Theta_{E} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

[3.9] $$ {}^{D}T_{C} = \begin{bmatrix} 1 & 0 & 0 & L_{DC}\cos\Theta_{D} \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_{DC}\sin\Theta_{D} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

[3.10] $$ {}^{C}T_{B} = \begin{bmatrix} \cos(90-\Theta_{C}) & 0 & \sin(90-\Theta_{C}) & L_{CB}\cos\Theta_{C} \\ 0 & 1 & 0 & 0 \\ -\sin(90-\Theta_{C}) & 0 & \cos(90-\Theta_{C}) & L_{CB}\sin\Theta_{C} \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

[3.11] $$ {}^{B}T_{P} = \begin{bmatrix} 2^{-1/2}\cos\Theta_{B} & -\sin\Theta_{B} & 2^{-1/2}\cos\Theta_{B} & 0 \\ 2^{-1/2}\sin\Theta_{B} & \cos\Theta_{B} & 2^{-1/2}\sin\Theta_{B} & 0 \\ -2^{-1/2} & 0 & 2^{-1/2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} $$

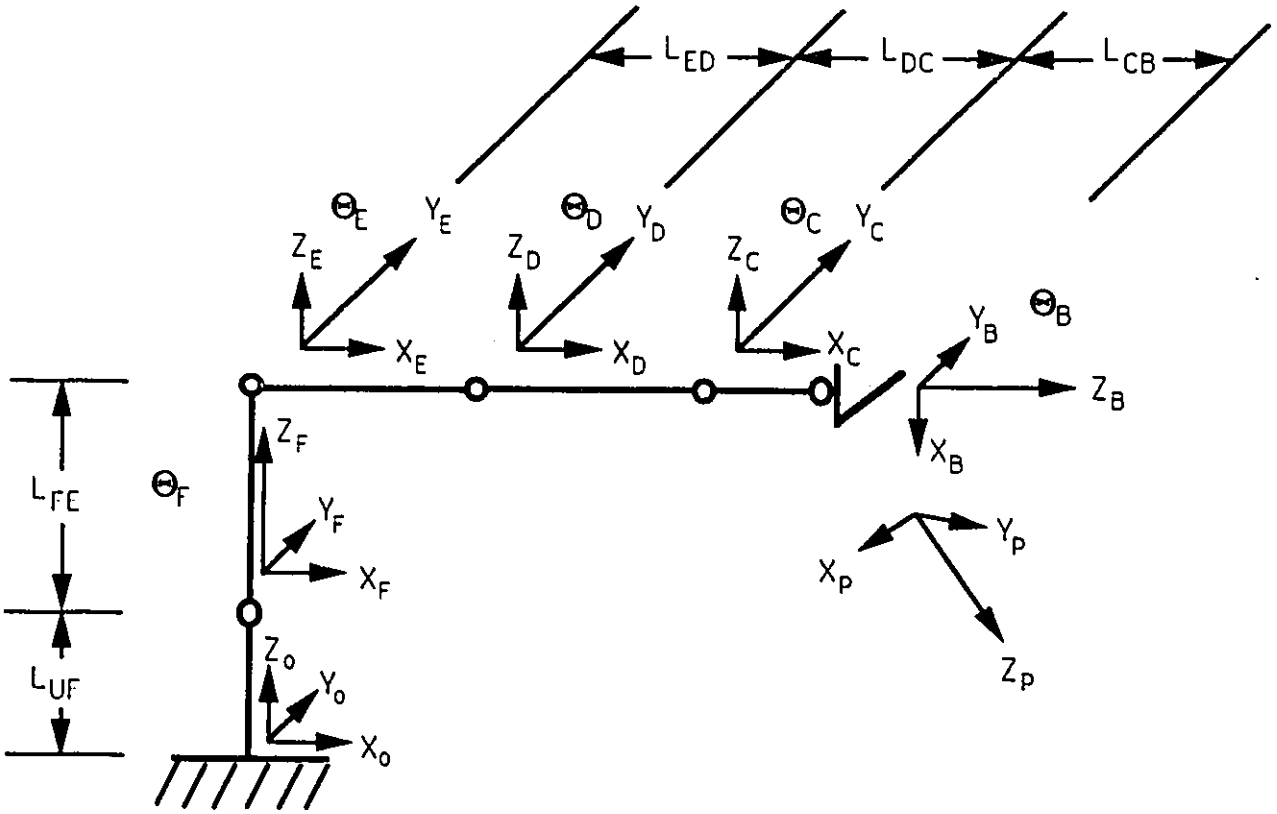


Figure 3.8 : RHINO XR3 Kinematic Coordinate Frames

Multiplying the transformations through gives the result ;

$$[3.12] \qquad {}^{W}T_{P} = \begin{bmatrix} \frac{1}{\sqrt{2}}\begin{bmatrix}\cos\Theta_F \sin\Theta_C \cos\Theta_B \\ -\sin\Theta_F \sin\Theta_B \\ -\cos\Theta_F \cos\Theta_C\end{bmatrix} & \begin{bmatrix}\cos\Theta_F \sin\Theta_C \sin\Theta_B \\ -\sin\Theta_F \cos\Theta_B\end{bmatrix} & \frac{1}{\sqrt{2}}\begin{bmatrix}\cos\Theta_F \sin\Theta_C \cos\Theta_B \\ -\sin\Theta_F \sin\Theta_B \\ +\cos\Theta_F \cos\Theta_C\end{bmatrix} & \cos\Theta_F \begin{bmatrix}L_{ED}\cos\Theta_E \\ +L_{DC}\cos\Theta_D \\ +L_{CB}\sin\Theta_C\end{bmatrix} \\ \frac{1}{\sqrt{2}}\begin{bmatrix}\sin\Theta_F \sin\Theta_C \cos\Theta_B \\ +\cos\Theta_F \sin\Theta_B \\ -\sin\Theta_F \cos\Theta_C\end{bmatrix} & \begin{bmatrix}\sin\Theta_F \sin\Theta_C \sin\Theta_B \\ +\cos\Theta_F \cos\Theta_B\end{bmatrix} & \frac{1}{\sqrt{2}}\begin{bmatrix}\sin\Theta_F \sin\Theta_C \cos\Theta_B \\ +\cos\Theta_F \sin\Theta_B \\ +\sin\Theta_F \cos\Theta_C\end{bmatrix} & \sin\Theta_F \begin{bmatrix}L_{ED}\cos\Theta_E \\ +L_{DC}\cos\Theta_C \\ +L_{CB}\sin\Theta_C\end{bmatrix} \\ \frac{1}{\sqrt{2}}\begin{bmatrix}-\cos\Theta_C \cos\Theta_B \\ -\sin\Theta_C\end{bmatrix} & \frac{1}{\sqrt{2}}\begin{bmatrix}\sin\Theta_F\end{bmatrix} & \frac{1}{\sqrt{2}}\begin{bmatrix}-\cos\Theta_C \cos\Theta_B \\ +\sin\Theta_C\end{bmatrix} & \sin\Theta_F \begin{bmatrix}L_{FE} \\ +L_{ED}\sin\Theta_E \\ +L_{DC}\sin\Theta_C \\ +L_{CB}\cos\Theta_C\end{bmatrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The third column of this matrix is the z axis of the ${}^{P}U$ frame expressed in world coordinates, and the fourth column is the position of the origin of the ${}^{P}U$ frame. Referring back to Figure 3.8, these vectors are the orientation and position vectors respectively. We therefore have derived an explicit expression for the FKS.

3.4 Inverse Kinematic Solution

The FKS models the manipulator system as set of 6 nonlinear equations in 5 variables. A cursory examination reveals that there is no simple method to invert the system and solve the IKS in closed form. The solution will therefore depend upon numerical approximation methods. The approach taken here was to iterate through a reduced joint space, solving that subsystem for the orientational parameters. A geometric analysis of the remaining systems reveals a solution for the positional parameters.

There are a few observations about the system which explicates the choice of numerical approximation method. First, the value of Θ_F can be determined directly from the positional parameters ;

[3.13] $$\Theta_F = \tan^{-1}(y_p / x_p)$$

Next, note that the orientational parameters depend only upon the 3 joint angles $[\Theta_B, \Theta_C \Theta_F]$, whereas the offset parameters are dependent upon the 4 joint angles $[\Theta_C \Theta_D, \Theta_E, \Theta_F]$. Assuming that no further simplifications are possible, the numerical solution would solve for all free variables in each subset of equations. Based on this assumption, it is desirable to first solve for the 2 free variables $[\Theta_B, \Theta_C]$ in the orientational subsystem, and then apply these values to the translational subsystem and solve for $[\Theta_D, \Theta_E]$. An analysis of the system geometry will reveal that a closed form solution for $[\Theta_D, \Theta_E]$ exists once the other angles have been solved,

relieving the need for a second numerical approximation.

A psuedocode algorithm for solving the IKS is as follows ;

```
Given [x_p, y_p, z_p, α_p, β_p, γ_p] {

      Given [x_p, y_p],
            use closed form expression to solve [θ_F]

      Given [θ_F, α_p, β_p, γ_p],
            use iterative approx. to solve [θ_B, θ_C]

      Given [θ_F, θ_B, θ_C],
            use closed form expression to solve [θ_D, θ_E]
}
```

This algorithm is illustrated diagrammatically in Figure 3.9.

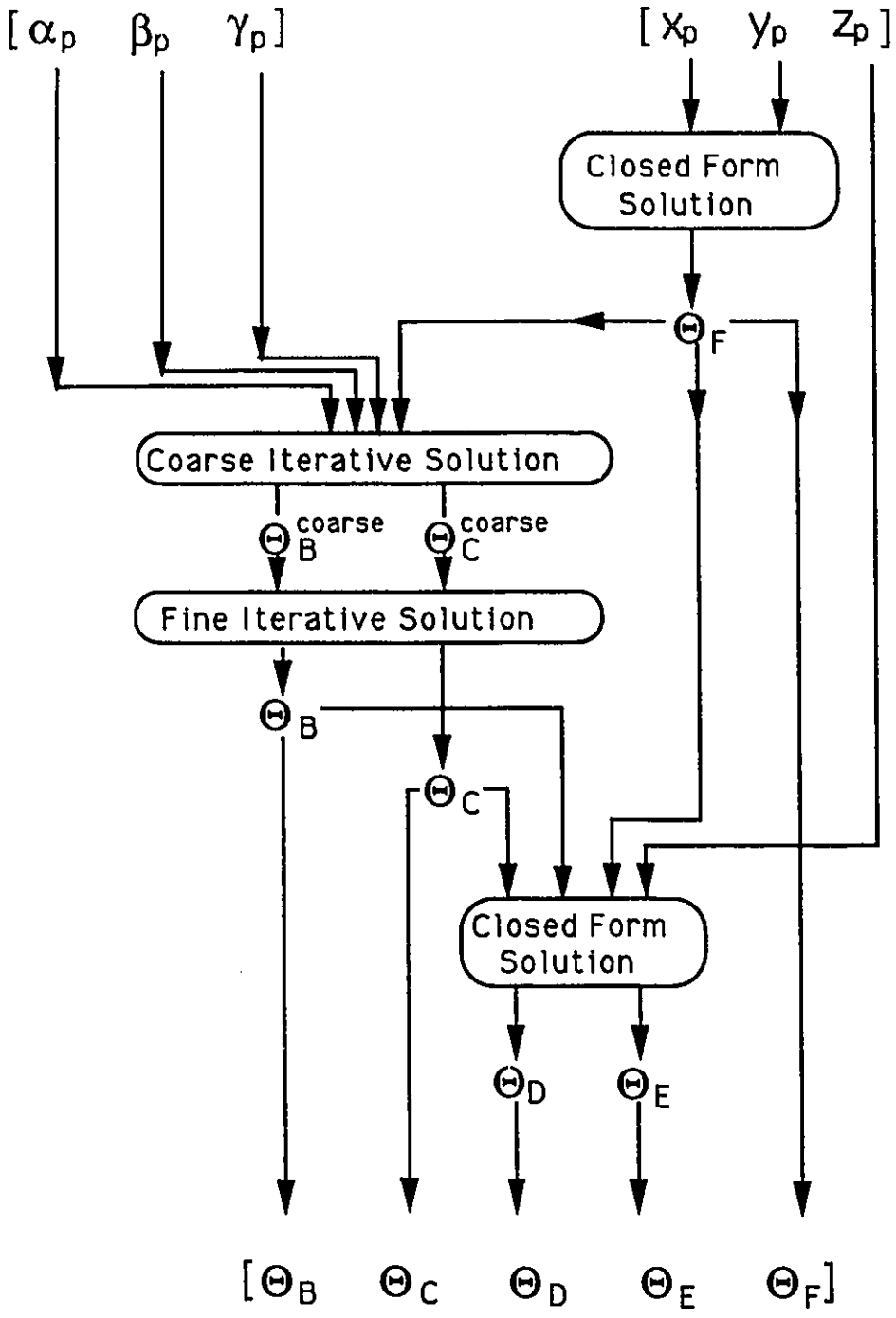


Figure 3.9 : Diagram of IKS

3.4.1 Numerical Approximation for $[\Theta_B,\Theta_C]$

A suitable error expression will return a measure of the distance between the desired orientational parameters $[\alpha_p,\beta_p,\gamma_p]$, and an estimate of these parameters $[\underline{\alpha}_p,\underline{\beta}_p,\underline{\gamma}_p]$. Let ε be the absolute error from the estimated to the actual orientational parameter values ;

[3.14] $$\varepsilon = |\alpha_p - \underline{\alpha}_p| + |\beta_p - \underline{\beta}_p| + |\gamma_p - \underline{\gamma}_p|$$

Given the value for Θ_F solved from the positional parameters $[x_p, y_p]$, and given an initial estimate of the joint angles $[\underline{\Theta}_B, \underline{\Theta}_C]$, an estimate of the orientational parameters can be calculated from the FKS as follows;

$$\underline{\alpha}_p = 2^{-1/2} (\cos\Theta_F \sin\underline{\Theta}_C \cos\underline{\Theta}_B - \sin\Theta_F \sin\underline{\Theta}_B + \cos\Theta_F \cos\underline{\Theta}_C)$$

[3.15] $$\underline{\beta}_p = 2^{-1/2} (\sin\Theta_F \sin\underline{\Theta}_C \cos\underline{\Theta}_B + \cos\Theta_F \sin\underline{\Theta}_B + \sin\Theta_F \cos\underline{\Theta}_C)$$

$$\underline{\gamma}_p = 2^{-1/2} (-\cos\underline{\Theta}_C \cos\underline{\Theta}_B + \sin\underline{\Theta}_C)$$

The solution proceeds in 2 phases. In the first phase, an initial rough estimate of the 2 joint parameter values $[\underline{\Theta}_B, \underline{\Theta}_C]$ is obtained by searching through the entire 2 dimensional joint space at a given coarse discrete granularity. The error at each point in the search space is determined and the point which minimizes the error is returned as the initial estimate. The time performance of this step is directly proportional to the number of points in the space which are tested, which is in turn inversely proportional to the granularity of the discrete quantization.

There were two considerations in choosing a discrete quantization value. It was important to select a value which was large enough to provide decent time performance of the algorithm, whereas too large a value could result in the return of local minima, which would lead to unreachable states. The experimentally chosen value for the increment was 10 degrees for both joint parameters, which resulted in a search space of 360/10 X 360/10 = 1296 iterations.

A pseudocode representation of the coarse iteration phase is as follows ;

```
error = LARGE_NUMBER ;
Δθ_B =      coarse θ_B increment ;
Δθ_C =      coarse θ_C increment ;
for (θ_B = 0; θ_B < 360; θ_B = θ_B + Δθ_B) {
    for (θ_C= 0; θ_C < 360; θ_C = θ_C + Δθ_C) {
        this_error = error_function(θ_B,θ_C,α_P,β_P,γ_P)
        if (this_error < error) {
            θ_B_coarse = θ_B ;
            θ_C_coarse = θ_C ;
            error = this_error ;
        }
    }
}
```

The second phase of the algorithm uses a method of bisection to converge upon the solution. Given an initial estimate, and an interval size equal initially to the value of the coarse quantization, the error function was evaluated at 5 values of the joint parameters $[\theta_B,\theta_C]$ in the neighbourhood of the initial estimate, as illustrated in Figure 3.10. The parameter pair which minimised the error function was used as the new estimate : the interval size was halved, and the procedure was repeated. A solution was returned when the error function was below some small value.

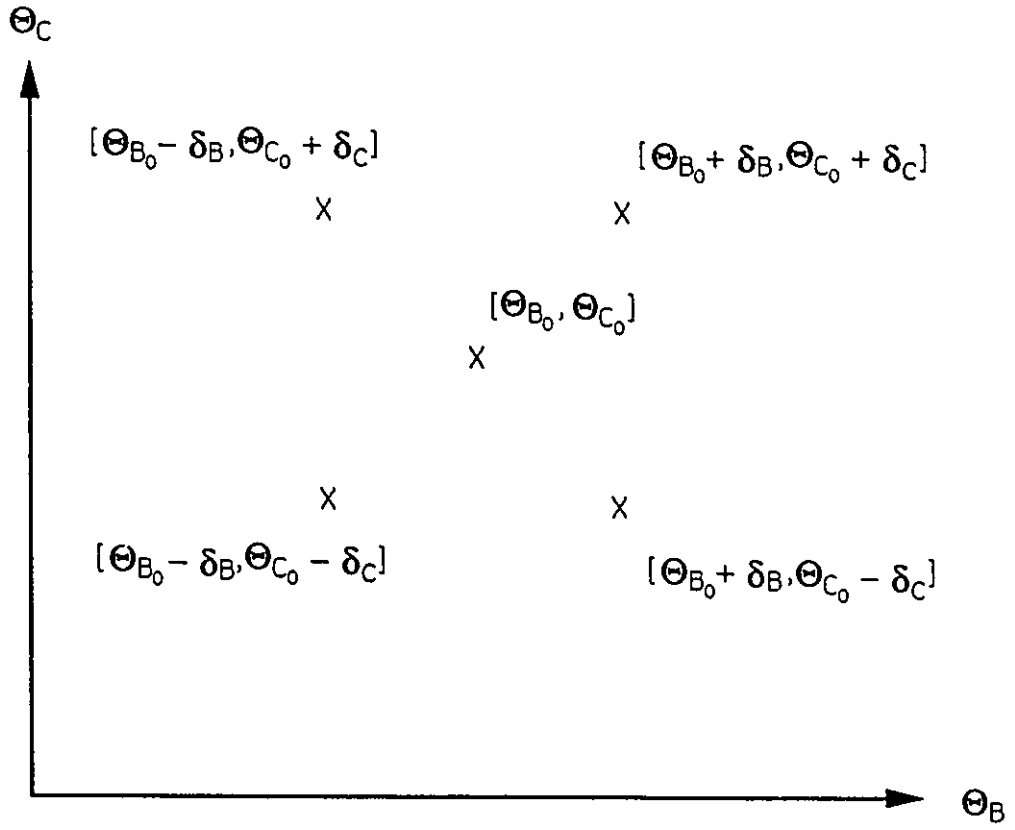


Figure 3.10 : Search Space Around Initial Estimate $[\Theta_{B_0}, \Theta_{C_0}]$

A pseudocode representation for the fine iteration algorithm is as follows ;

```
δ = coarse Θ increment ;
Θ_B_fine = Θ_B_coarse ;
Θ_C_fine = Θ_C_coarse ;
error = LARGE_NUMBER ;

do {
     for (all points [Θ_B_test, Θ_C_test] in neighbourhood of
          [Θ_B_fine, Θ_C_fine]) {
          this_error =
               error_function(Θ_B_test,Θ_C_test,orientation);
          if (this_error < error) {
               Θ_B_fine = Θ_B_test ;
               Θ_C_fine = Θ_C_test ;
               error = this_error ;
          }
          δ =  δ / 2 ;
     }
} while ((error > ACCEPTABLE_ERROR)
               and (δ > FINE_GRANULARITY)) ;
```

3.4.2 Closed Form Expression of Θ_D, Θ_E

A geometric analysis of the mechanical system reveals a closed form expression for $[\Theta_D, \Theta_E]$. We first define the following values illustrated in Figure 3.11 ;

$^{E}V_C$ == a vector to the end of link L_{DC} (i.e. the origin of the coordinate frame U_C)

$|^{E}V_C|$ == the length of $^{E}V_C$

γ == the bisecting angle between link L_{ED} and link L_{DC}

α == the angle between the world horizontal and $^{E}V_C$

β == the angle between $^{E}V_C$ and the link L_{ED}

Θ == the angle between link L_{DC} and the horizontal

ρ == the angle between $^{E}V_C$ and the vertical

The following geometric relationships hold ;

[3.16] $$|{}^{E}V_C| = ({}^{E}X_C{}^2 + {}^{E}Z_C{}^2)^{1/2}$$

[3.17] $$K = L_{ED} = L_{DC}$$

[3.18] $$K/\sin\ 90^o = {}^{E}L_{EC}/2\ \sin\ \gamma$$

[3.19] $$|{}^{E}V_C|/\sin\ 90^o = {}^{E}X_C/\sin\ \rho$$

[3.20] $$\beta = 90^o - \gamma$$

[3.21] $$\Theta = 90^o - (\beta + \rho)$$

[3.22] $$\alpha = 90^o - \rho$$

Combining [3.18] and [3.20] gives ;

[3.23] $$\beta = 90^o - \sin^{-1}({}^{F}L_{EC}/2K)$$

and from [3.19], [3.21], and [3.22] ;

[3.24] $$\Theta = 90^o - (\beta + \sin^{-1}({}^{F}X_C/{}^{F}L_{EC}))$$

[3.25] $$\alpha = 90^o - \sin^{-1}({}^{F}X_C/{}^{F}L_{EC})$$

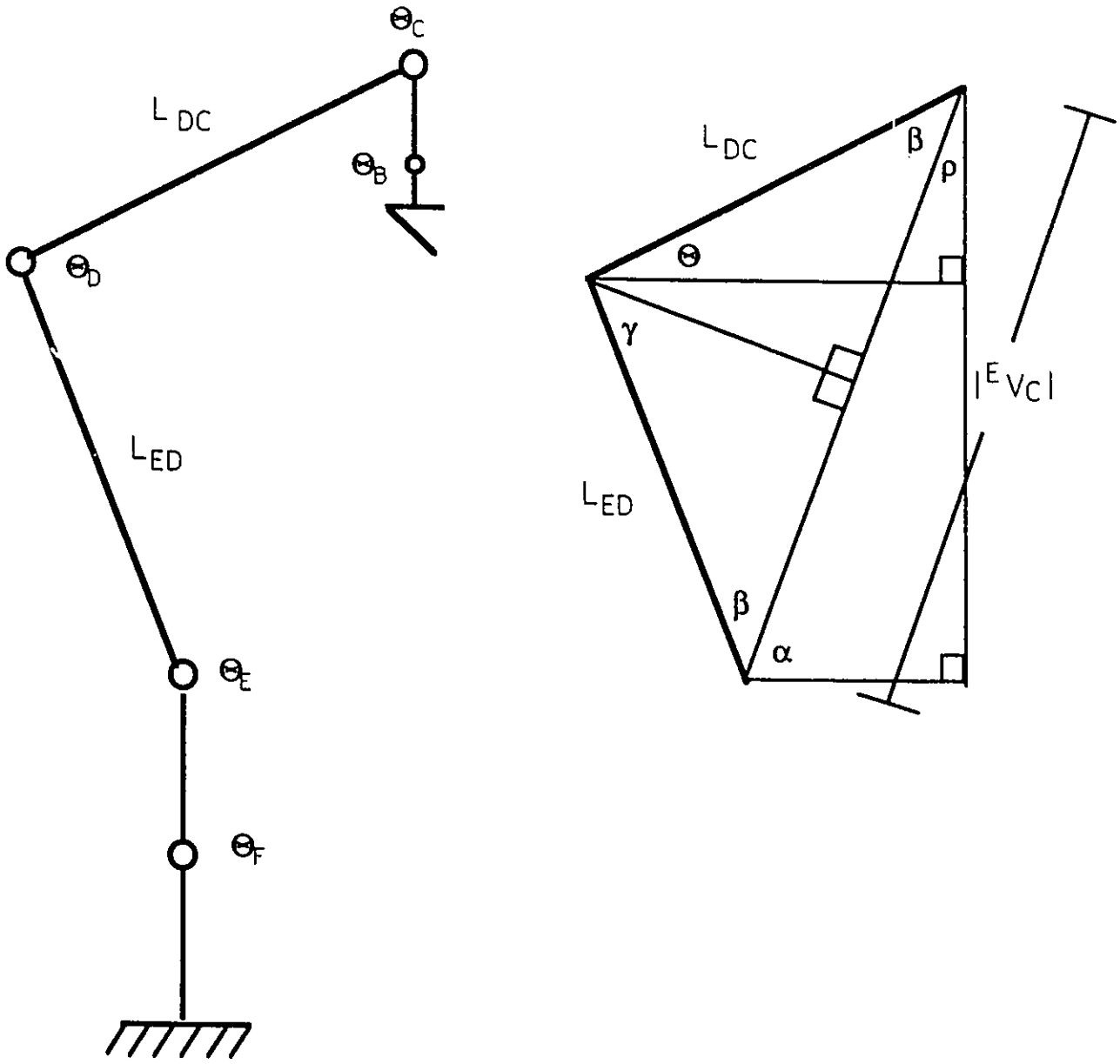


Figure 3.11 : Geometric Decomposition of Manipulator

Relating the above values to the world coordinate frame ;

[3.26] $$^{E}Z_C = z_p - L_{CP} \sin \Theta_C - L_{UE}$$

[3.27] $$^{E}x_C = (x_p^2 + y_p^2)^{1/2} - L_{CP} \cos \Theta_C$$

[3.28] $$\Theta_E = \alpha + \beta = 90^o - \rho - \beta$$

[3.29] $$\Theta_D = \Theta = 90^o - \rho + \beta$$

which solves $[\Theta_D, \Theta_E]$ in closed form.

3.4.3 Incremental Motion : The Incremental IKS

The most time consuming step of the above algorithm was the initial exhaustive search through the quantized space of $[\Theta_B, \Theta_C]$ to derive an initial coarse estimate. In the case where the motion is known to be incremental, the present configuration can be used as an initial estimate, reducing the overall solution time significantly. This solution for incremental motions, the incremental IKS, is used extensively in the development of tactile sensing skills. A block diagram for the Incremental IKS algorithm is presented below in Figure 3.12.

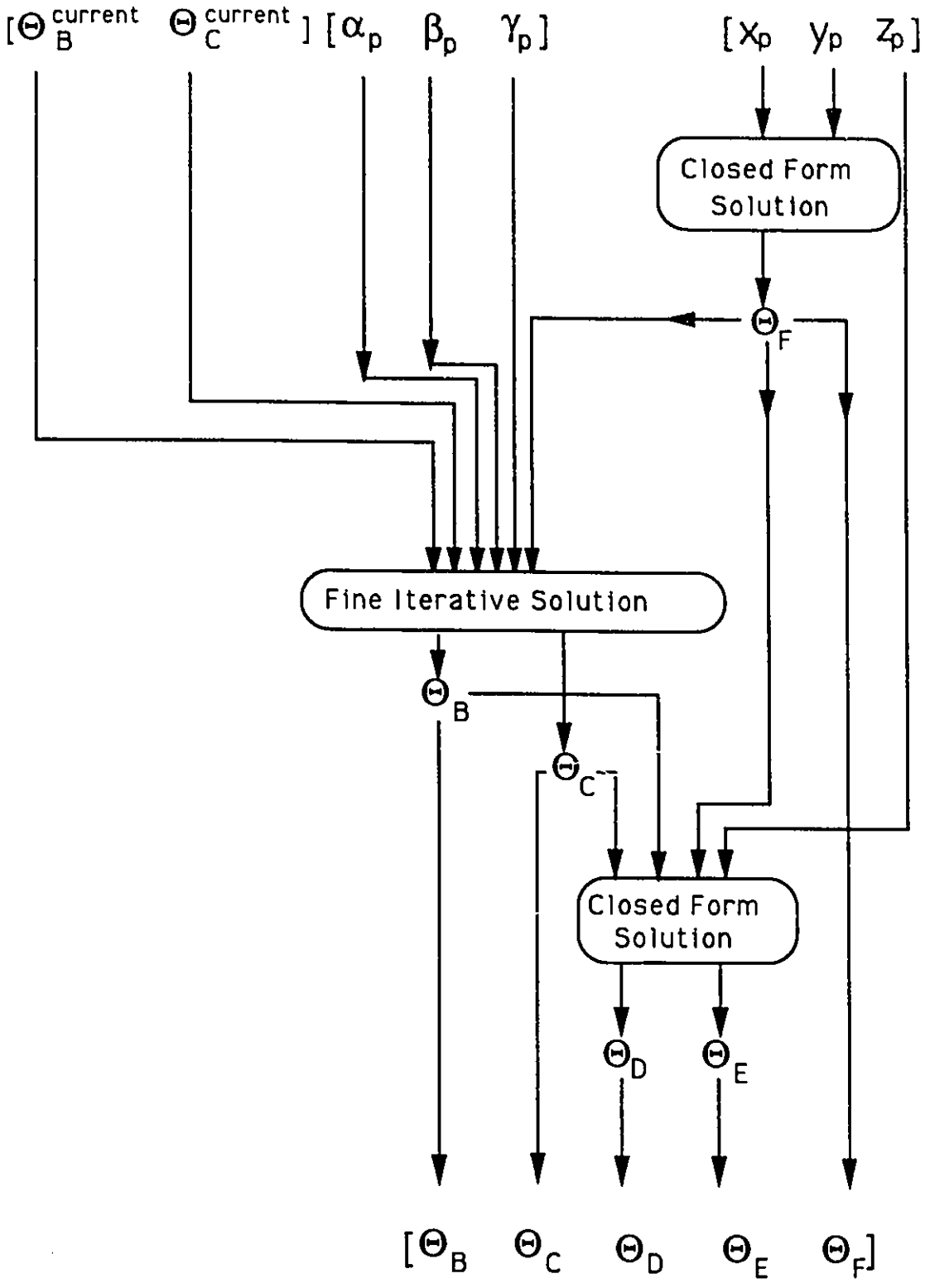


Figure 3.12 : Diagram of Incremental IKS

3.5 Conclusions

Some commercially available robotic manipulators are supplied with the software solutions to the FKS and IKS problems. These machines are also capable of interfacing directly with a host computer to drive the joint values explicitly. Dependent upon the application, situations will occur where the factory supplied kinematic solution falls short of the performance requirements, whereby the engineer is required to modify or redesign the kinematic control [34]. In any case, it is difficult to perform any significant robot programming task without a good knowledge of the kinematic mechanism.

The above development outlines the kinematic solution implemented for use in the following set of experiments. In this particular case, it was necessary to refit the end-effector with an angled bracket in order to achieve the desired reachability, i.e. the general contact of a planar sensor with a surface. The factory supplied kinematic software was insufficient for this task.

There are three ways in which this solution can be improved and enhanced. During the coarse and fine iterative solution steps, the process loops repeatedly until the evaluated error function falls below some value. The size of this value is significant only in as much as it effects the accuracy of the numerical solution. Presently, a value of 10^{-5} ensures that the accuracy of the solution is well below the positioning error of the manipulator, about 5 mm.

A value which is significant to both the success and the time efficiency of the solution is the incremental step values which are used for the initial coarse iterative approximation. The entire range of Θ_B and Θ_C values is stepped through at this discrete value, so too small a value will cause a slow down in execution. Alternately, if the value is too large, a local minima may be settled upon in place of the global minima, which in turn could result in the erroneous conclusion of an unreachable state in the fine iterative stage.

The currently implemented incremental value of 10 degrees was determined experimentally to be satisfactory. A more thorough analysis of the 2 dimensional space formed from the derivatives of Θ_B and Θ_C would reveal the minimum distance between local minima, which could then be used as a lower limit to the quantization coarseness.

In the case of gross motions, the solution arbitrarily determines between redundant configurations. It would be simple to add the capability to specify which redundant alternative should be chosen in a given situation. Note that in the incremental IKS, the redundant alternative will be chosen to be consistent with the current arm configuration, which is clearly a desirable feature.

A preliminary attempt was made to solve for straight line and other functional trajectories through inverting the Jacobian of the FKS. This line of development was very successful, and is recommended for future

research. The present implementation of incremental movement was fast enough to generate straight line trajectories dynamically through successive recalculations of the Incremental IKS.

Chapter 4 : Active Tactile Sensing Skills

4.1 Introduction

This section documents the development of two robotic skills which make use of tactile sensing. The experimental setup used to develop these skills is illustrated in Figure 4.1. It consisted of a LORD LTS210 tactile sensor mounted to the wrist of a RHINO 5 d.o.f. robotic manipulator. Both of these devices were connected through serial communication links to a PC with 8088 CPU and math coprocessor.

The first skill is two d.o.f. plane tracking, in which the contact of the sensor with a planar surface is maintained throughout repositioning and reorientation of the plane. This skill has traditionally been implemented using force-torque sensing methods, but to the author's knowledge has never been attempted using pure taction.

The second skill is three d.o.f. sphere tracking and cannot be implemented without the use of taction. A cutaneous feature, in this case the impression made through contact with a sphere, is tracked in three dimensions.

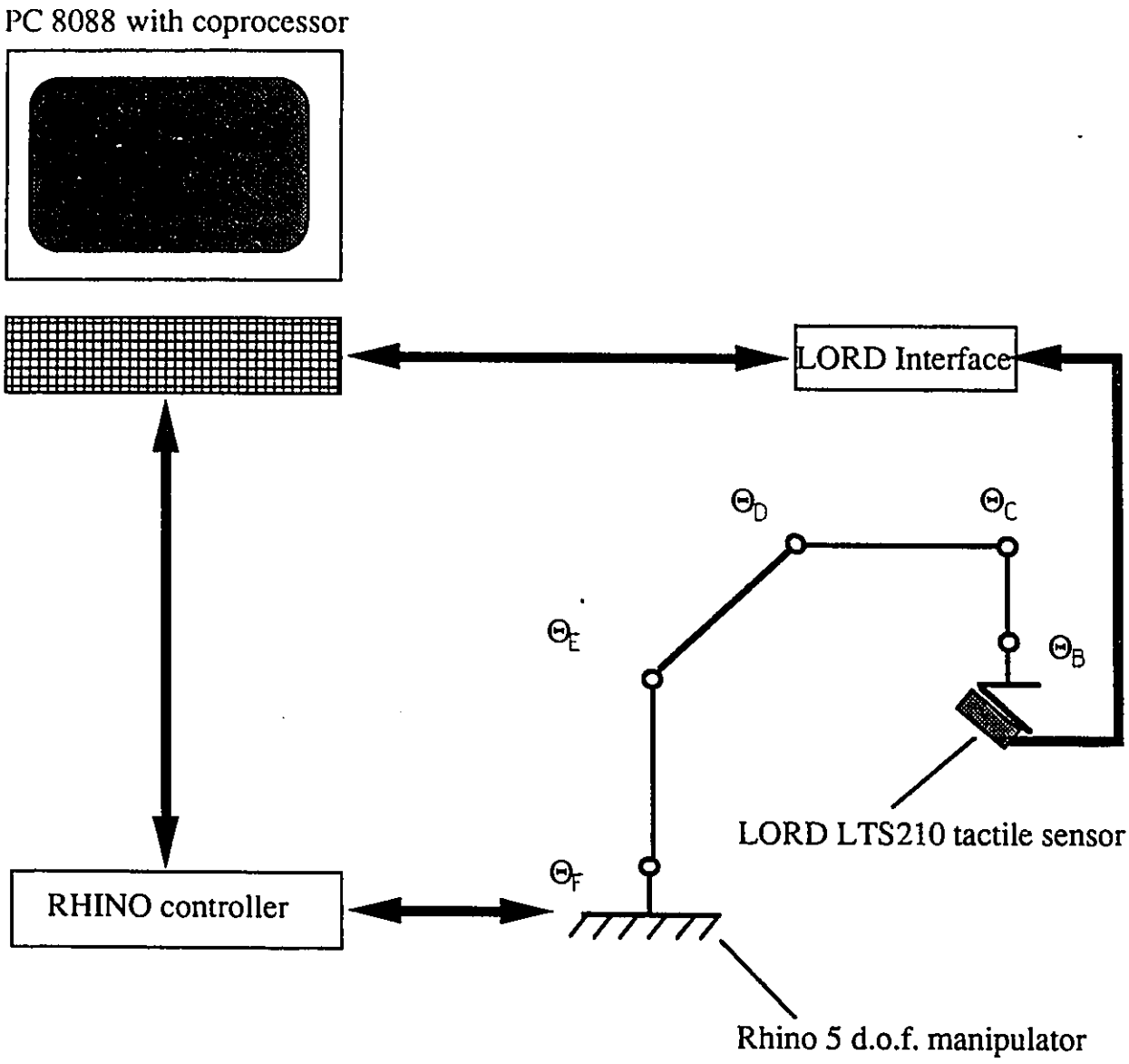


Figure 4.1 : Experimental Setup

The execution of both skills follow the general structure illustrated in Figure 4.2. A tactile image is acquired from the contact of a surface with the planar sensor. This image is processed to determine information about certain contact parameters. The manipulator is then relocated in an attempt to maintain the values of the contact parameters within certain ranges.

The success of the tracking skills is defendant upon a number of factors : the ability to process the cutaneous tactile image and estimate the contact parameters; the control method; and the communication rates and response frequencies. The system has been developed utilising a basic point-to-point kinematic control mechanism, described in Chapter 3, and a single multichannel uniprocessor personal computer. Rather than high performance, the intent is to demonstrate the applicability of the sense of touch in the performance of robotic skills. The emphasis is on the interpretation of the incoming sensory data to derive useful information for the desired function. Any enhancements in the low level control and real time processing capabilities of the system would improve system performance in a well understood way.

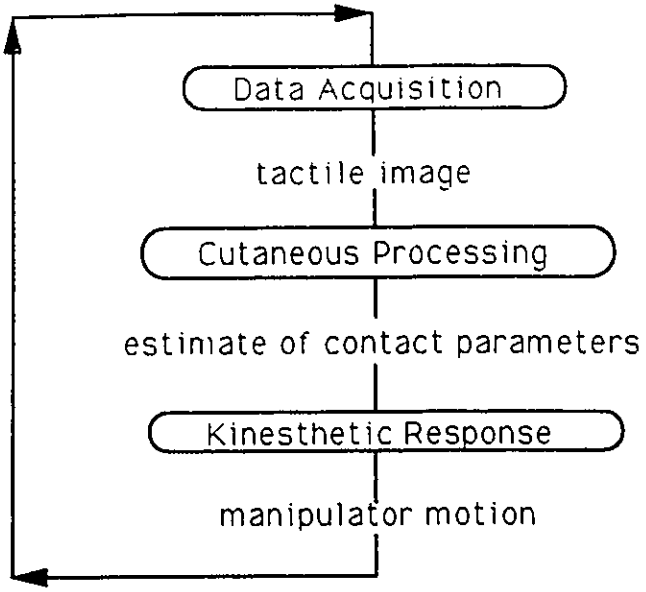


Figure 4.2 : Structure of Tactile Skills

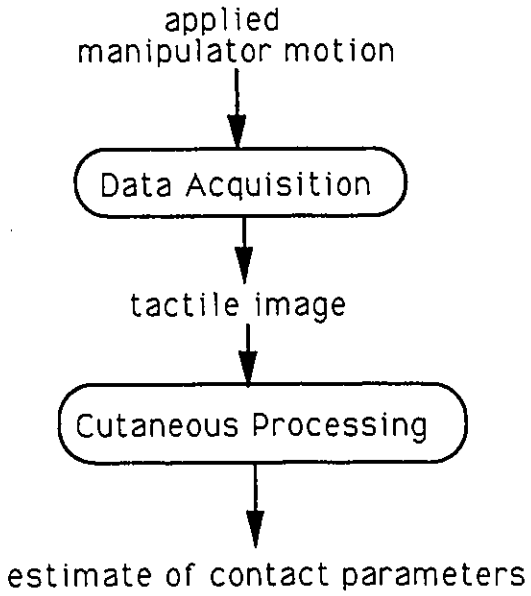


Figure 4.3 : Decision Function Calibration

4.2 Contact Parameters

The goal of the skills is to respond to relative changes between the end-effector and the contacted surface in order to maintain the contact parameters. Cutaneous data processing of the acquired tactile image and provides an estimate of the contact parameters.

Figure 4.4 illustrates the contact parameters for the plane tracking skill. The contact is described by the magnitude of the force F which the plane exerts on the sensor in the normal direction, as well as the angle Φ which resolves the y_p component of F.

In plane tracking, we wish to keep the planar surface in flush contact with the sensor throughout repositioning. This is equivalent to maintaining F and Φ centered around some values. As the responses to these parameters will result in two motions in the ^{P}U frame (a translation and a rotation about x_p), this skill is a two d.o.f. skill.

Unlike the case of contact with a planar surface, contact with a nonplanar surface reveals a discernible cutaneous pattern on the planar array sensor. The sphere tracking contact parameters measure the lateral position of the contact in the x_p- y_p directions. Figure 4.5 illustrates the contact parameters for a spherical feature. If the desired contact position of the sphere is the center of the sensor, then the contact parameters $[\Delta x_p, \Delta y_p]$ measure the distance of the contact from the desired position.

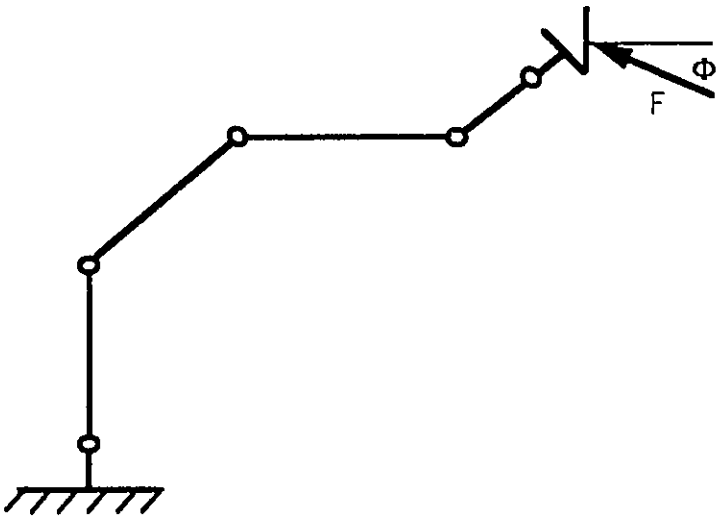
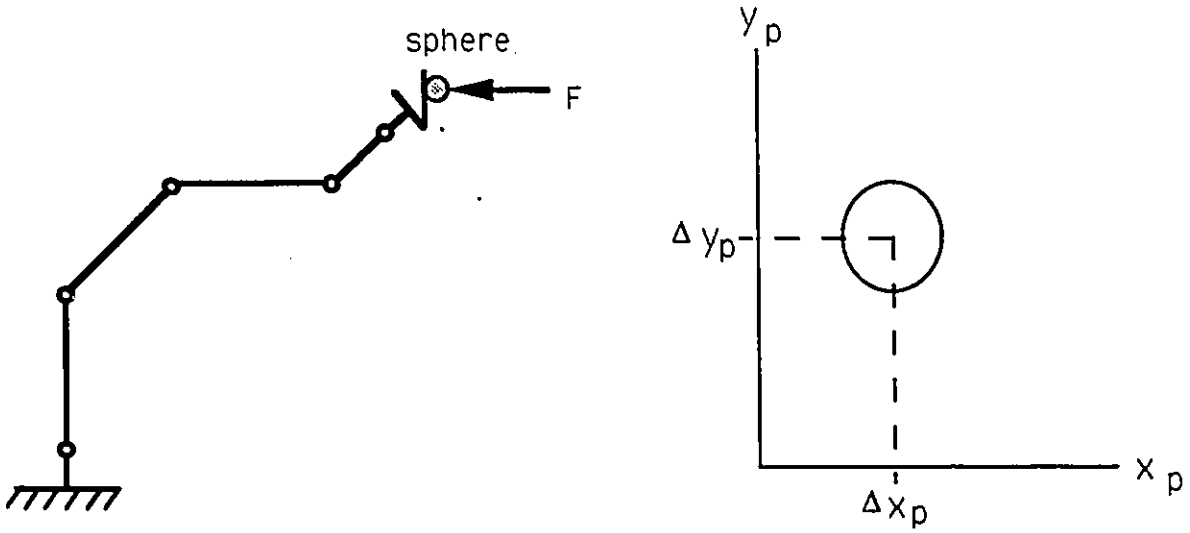


Figure 4.4 : Contact Parameters For Plane Tracking



a) force applied to nonplanar surface

b) lateral offset in end-effector Frame

Figure 4.5 : Contact Parameters For Sphere Tracking

Including the magnitude parameters of the normal contact force F (as defined in plane tracking) there are three contact parameters $[\Delta x_p, \Delta y_p, F]$ for sphere tracking.

4.3 Cutaneous Data Processing

Two cutaneous processing functions were implemented to estimate the contact parameters. The sensor used was a LORD LTS 210 optoelectronic tactile sensor, which had a rectangular array of 10X16 taxels with 1.8mm intertaxel spacing. Each taxel measured the displacement of the covering rubber skin over a range of [0,30], where each unit represents a deflection of 0.001 in., or equivalently an applied force of 0.01 lb. For all practical purposes, the value at each taxel can be interpreted as a measure of the normal pressure applied at that location.

For completeness, it should be remarked that the sensor was also outfitted with a 6 d.o.f. force-torque sensing capability. Although not used in this work, an interesting further study would be to examine the added capability which this imparts to the development of sensor based skills.

4.3.1 Sum-Of-Taxels Function

The value at each taxel can be considered a measure of the normal force applied. The sum of these normal forces is therefore a measure of the magnitude of F. This function takes on a numerical value of [0,4800]. If the cutaneous image is stored as a 10 X 16 array of integers named taxel[row][col], then the sum-of-taxels function is calculated as ;

$$[4.1] \qquad \text{sum-of-taxels} = \sum_{row=1}^{16} \sum_{col=1}^{10} \text{taxel[row][col]}$$

4.3.2 Center-Of-Mass Function

While the sum-of-taxels value gave an estimate of the pressure on the sensor surface, in order to track rotational motions about an axis of the sensor it was necessary to have some information about the spatial displacement of the cutaneous image. An effective function for estimating the angle of a contacted plane with respect to the sensor surface was the centre of mass function. The function is calculated as follows ;

$$[4.2] \qquad \text{row center\_of\_mass} = \frac{\sum_{row=1}^{16}\sum_{col=1}^{10} \text{row x taxel[row][col]}}{\sum_{row=1}^{16} \text{taxel[row][col]}}$$

$$[4.3] \qquad \text{col center\_of\_mass} = \frac{\sum_{row=1}^{16}\sum_{col=1}^{10} \text{col x taxel[row][col]}}{\sum_{row=1}^{16}\sum_{col=1}^{10} \text{taxel[row][col]}}$$

4.4 Kinesthetic Response

The kinesthetic response is determined by a decision function. A decision function must be determined for each of the contact parameters for each skill. There are two methods for determining the decision functions. The first is through an analysis of the manipulator dynamics [64]. This gives a model of the force response of the manipulator joints, which can be inverted to respond to the measured applied force. While the analysis is fairly complex, this method is necessary for any high performance system.

The second method used in this implementation derives the force response empirically through direct calibration. The system dynamics are considered as a black box, and the cutaneous response to a kinesthetic motion is measured directly. The relationship is then inverted, with the cutaneous response as input and the kinesthetic motion as output. Figure 4.3. illustrates the process of inverting the measured function.

4.4.1 Two d.o.f. Plane Tracking

The calibration of the response of the sum-of-taxels function in plane tracking was executed by advancing the kinesthetic position of the end-effector in the normal direction towards a fixed planar surface, as illustrated in Figure 4.6. Note that after contact was made, the incremental advancement of the end-effector was absorbed by the inherent compliance of the manipulator. The result was not a displacement, but rather an increase in the contact force. At each incremental advance, a tactile image was acquired and the sum-of-taxels value was calculated.

The decision function was determined through the relationship between the kinesthetic motion and the output of the cutaneous processing. The average of thirty samples are plotted in Graph 4.1. A linear regression of the data reveals the relationship ;

[4.4] approach = 0.02 mm/sum_of_taxels [mm]

The calibration of the rotational response proceeded analogously, as illustrated in Figure 4.7. Contact was made between the sensor and the fixed planar surface with a bias contact force F_b, and the orientation of the end-effector was altered by discrete rotations around the sensor y-axis.

The response to this calibration procedure averaged over 30 samples is charted in Graph 4.2, and the resulting decision function was determined as ;

[4.5] ϕ = 0.007 degrees/(row center_of_mass) [degrees]

Note that saturation occurred when the sensor had rotated to the point where there was very little contact with the surface, as indicated in the plateau region of the Graph.

4.4.2. Three d.o.f. Feature Tracking

In plane tracking, the row center-of-mass function was used to estimate the rotation of the force of contact around the end-effector y-axis. In feature tracking, the center-of-mass functions gave an estimate of the position of the feature in the end-effector x-y plane.

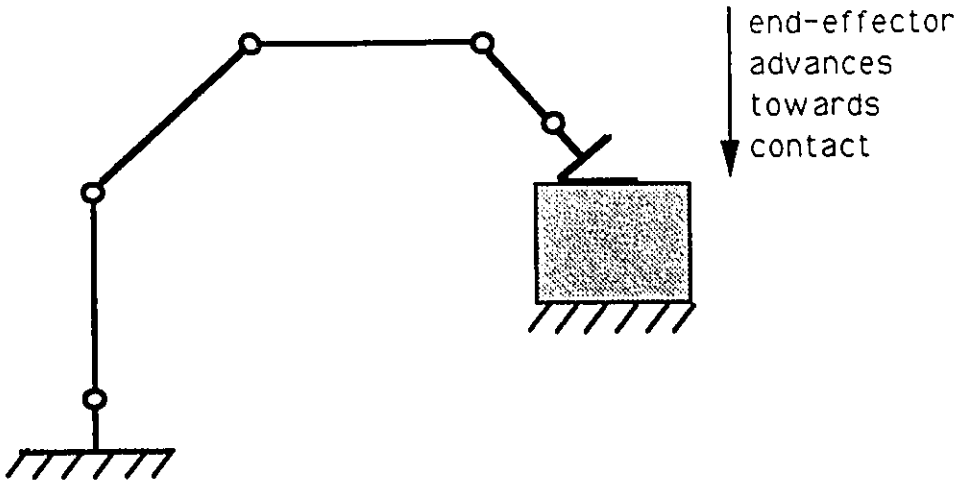


Figure 4.6 : Calibration of Sum-Of-Taxels Decision Function

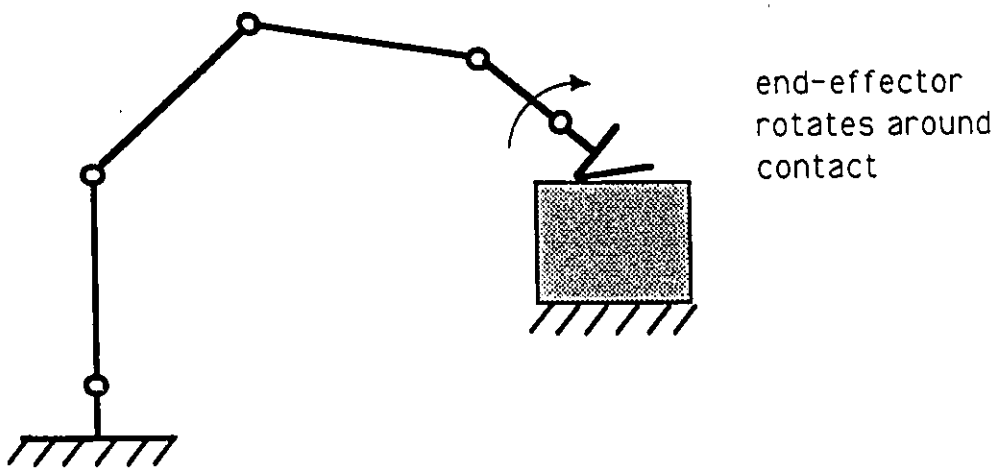
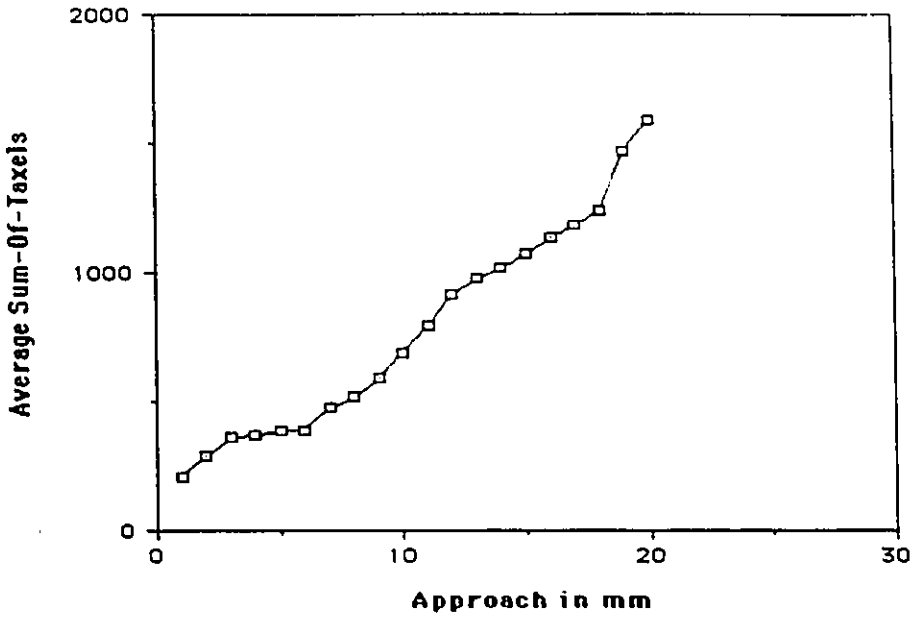
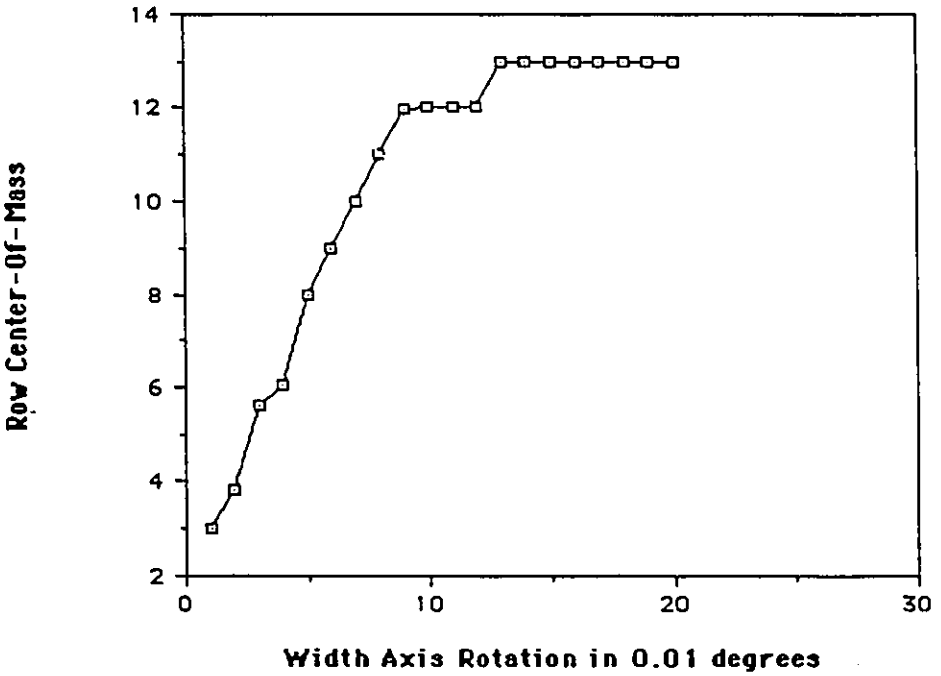


Figure 4.7 : Calibration of Center-Of-Mass Decision Function



Graph 4.1 : Approach vs. Average Sum-Of-Taxels



Graph 4.2 : Width Axis Rotation vs. Row Center-Of-Mass

The kinesthetic response was to translate the end-effector the estimated distance towards the desired center-of-mass. The direction of translation must be expressed in the world coordinate frame by converting the x_p and y_p axes to their components in the ^{W}U system. The expressions for the x_p and y_p axes is taken from the FKS, equation [3.12].

$$[4.6] \qquad x_p = \frac{1}{\sqrt{2}}\begin{bmatrix}\cos\Theta_F \sin\Theta_C \cos\Theta_B \\ -\sin\Theta_F \sin\Theta_B \\ -\cos\Theta_F \cos\Theta_C\end{bmatrix} i + \frac{1}{\sqrt{2}}\begin{bmatrix}\sin\Theta_F \sin\Theta_C \cos\Theta_B \\ +\cos\Theta_F \sin\Theta_B \\ -\sin\Theta_F \cos\Theta_C\end{bmatrix} j + \frac{1}{\sqrt{2}}\begin{bmatrix}\cos\Theta_C \cos\Theta_B \\ -\sin\Theta_C\end{bmatrix} k$$

$$[4.7] \qquad y_p = \begin{bmatrix}-\cos\Theta_F \sin\Theta_C \sin\Theta_B \\ -\sin\Theta_F \cos\Theta_B\end{bmatrix} i + \begin{bmatrix}-\sin\Theta_F \sin\Theta_C \sin\Theta_B \\ +\cos\Theta_F \cos\Theta_B\end{bmatrix} j + \frac{1}{\sqrt{2}}\begin{bmatrix}\sin\Theta_F\end{bmatrix} k$$

$$[4.8] \qquad z_p = \frac{1}{\sqrt{2}}\begin{bmatrix}\cos\Theta_F \sin\Theta_C \cos\Theta_B \\ -\sin\Theta_F \sin\Theta_B \\ +\cos\Theta_F \cos\Theta_C\end{bmatrix} i + \frac{1}{\sqrt{2}}\begin{bmatrix}\sin\Theta_F \sin\Theta_C \cos\Theta_B \\ +\cos\Theta_F \sin\Theta_B \\ +\sin\Theta_F \cos\Theta_C\end{bmatrix} j + \frac{1}{\sqrt{2}}\begin{bmatrix}-\cos\Theta_C \cos\Theta_B \\ +\sin\Theta_C\end{bmatrix} k$$

As in the case with two d.o.f. plane tracking, the estimate of the normal force is responded to with a motion in the end-effector z-direction. The area covered by the spherical feature will necessarily be less than that of the planar contact, which results in a smaller bias force. The response from this bias force is a motion which is orthogonal to the lateral response from the center-of-mass decision function. The sum of these two motions will therefore preserve the response in each component.

4.5. Performance Considerations

In this section, the performance of the tracking skills is evaluated with respect to two criterion. The first is the response rate, the frequency with which the reaction cycle is executed. The response rate for each cycle is composed of five sequential actions, which are either data processing or data communication routines. The second performance criterion is the manipulator motion speed, the speed with which the joint motors can reposition the manipulator.

It will be shown that the most time consuming aspect of the response cycle is the data communication and manipulator motion processes.

4.5.1 Response Rate

The current implementation was executed on an IBM PC 8088 personal computer with numeric coprocessor, and therefore the system performance was limited by the minimal computing power. An analysis of the time performance of the processing stages reveals the computational and communication overhead of the system.

The system response period was approximately 0.5 seconds, which was the total of the response periods of the following stages ;

1. **data input** : the LORD LTS 210 sensor was scanned by a call to the function lord_scan. This function would initiate the data input process by transmitting 5 bytes to the sensor over a serial communication line. It would then receive 160 consecutive bytes representing the taxel values. As the transmission time was set to 9600 baud, the total time for this transaction was calculated to be approximately 165/960 = 0.17 seconds. Through direct measurement via the computer's internal clock, this time was found to be 0.26 seconds. Note that there is also a certain amount of processing and synchronisation which transpires internal to the sensor, which could account for the difference between the calculated and the measured times.

2. **data processing** : the sum-of-taxels and center-of-mass metrics were designed as fast single pass functions. Both of these functions were programmed into a single subroutine. The execution time was measured at 0.04 seconds.

3. **decision function** : a very simple linear function was used to approximate the control function, which was programmed as basically a single multiplicative operation. Its processing time can be considered negligible.

4. **inverse kinematics** : the function rhino_move_increment executed the inverse kinematic process. Its execution time

can be analysed by dividing the function into 2 stages. First, there was the serial communication between the computer and the robot controller. As the motion was incremental, it can be assumed that the change in joint position were transmitted through a single numerical value (large increments required transmission through a series of smaller values). The communication overhead was approximately 8 bytes per joint : 3 bytes for ensuring that the joint had completed any previous motions (e.g. C?<CR>) and 5 bytes to transmit the new motion (e.g. B+40<CR>). With 6 joints and a transmission rate of 9600 baud, the communication overhead was calculated as 8X6/960 = 0.05 seconds, which is negligible.

The second stage to consider when evaluating the time performance of the inverse kinematic solution was the iterative search through the joint space. Once again, as the motion was incremental, there was no need to perform the costly coarse iteration : only the incremental IKS was solved. It is difficult to estimate exactly how much time this process would take. It completed when a certain error value was minimised and, depending upon the goodness of the initial approximation (the smallness of the incremental move) the convergence time would vary. A typical execution time for the incremental IKS, including the serial data communication, was measured to be 0.14 seconds.

5. **robot motion** : Once the joint motion commands had been transmitted to the robot, each joint must have completed its motion before another command could be sent. The computer-to-robot interface function was designed such that a joint was repeatedly polled until it was ready to accept another command. If the motion was sufficiently large as to consume more time than the data acquisition and processing of the next step, the system would wait until the previous motion command was completed. Empirically, it seems unlikely that this delay played any role in the current system, as the motions were small enough to be completed before any new motion was calculated.

Graphically, the time contribution of each section of the cycle is illustrated in Figure 4.8. Only the software implemented stages (data processing and inverse kinematics) could be optimised to improve the response time. As they were designed with time performance in mind, it is unlikely that any substantial decrease in execution time would result from any further optimisation. Of course, upgrading the processing hardware to an 80286 or 80386 based machine would improve the response time, but it is clear that the communication overhead and, of course, the frequency limitations imposed by the robot controller would prove to be an upper limit. Thus, even with negligible processing speeds, using the current sensor and robot hardware, the maximum response frequency would be around 2.5 Hz .

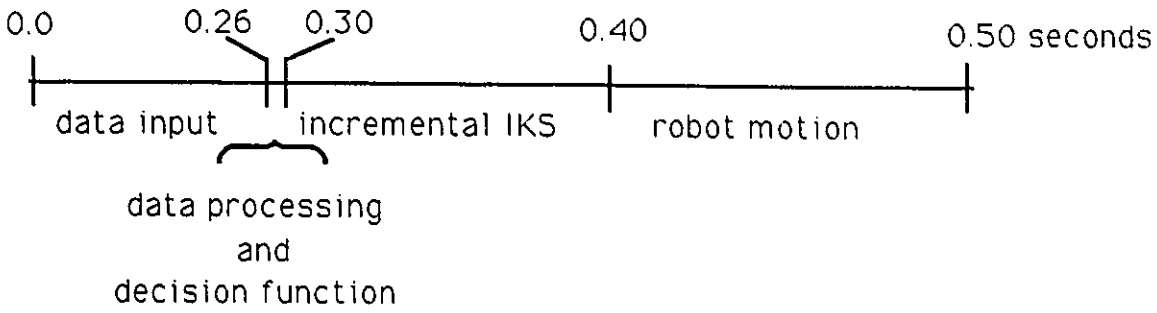


Figure 4.8 : Processing and Communication Timing For Single Cycle

4.5.2 Manipulator Motion Speed

In plane tracking it was assumed that a minimal pressure must be maintained to ensure contact. The noise level of the taxels was 2 ; therefore, letting the sum-of-taxels minimal threshold which would ensure contact be a value of 3 units/taxel by 160 taxels = 480. This was found experimentally to be a reasonable if conservative value.

The normal displacement per unit sum-of-taxel value was determined to be 1/68.69 = 0.15 mm/taxel unit. To identify the maximum speed at which the system could respond, it was assumed that the sum-of-taxels metric was initially at a maximum contact value of 30 X 160 = 4800, i.e., the contact was held with the maximum force measurable. An impulse retraction of (4800-480) X 0.15 mm must occur within 0.5 seconds to maintain contact, resulting in a speed of 1296 mm/second.

This is of course a very fast speed for a standard manipulator, even in direct joint control mode. The maximum speed of the RHINO was approximately 100 mm/sec in point-to-point mode. The response of the joint motors were therefore the main limiting factor to the speed of the manipulator motions.

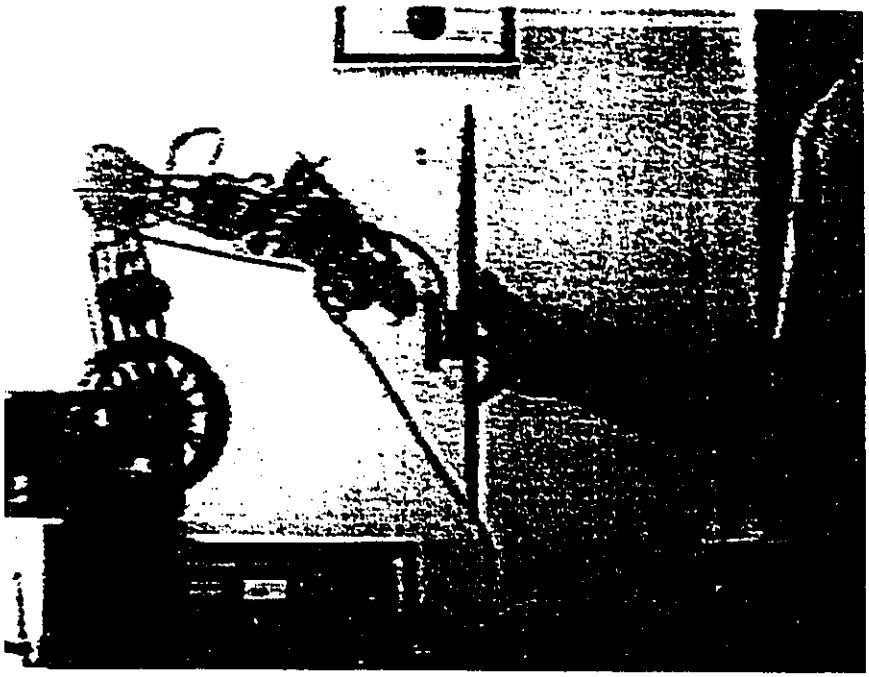
4.6 Experimental Results

The skills were implemented, and a series of three experimental tests were conducted to gauge the success of the tracking skills. As the system was not developed with high performance as a goal, it was sufficient to qualitatively judge the capability of the manipulator at detecting and responding to trends in the applied contact. Note that a fairly sophisticated setup would be required to measure the time response of the system. To qualitatively judge the tracking capabilities, a video tape was made showing the following set of experiments. The figures presented below are frames of the taped experiments.

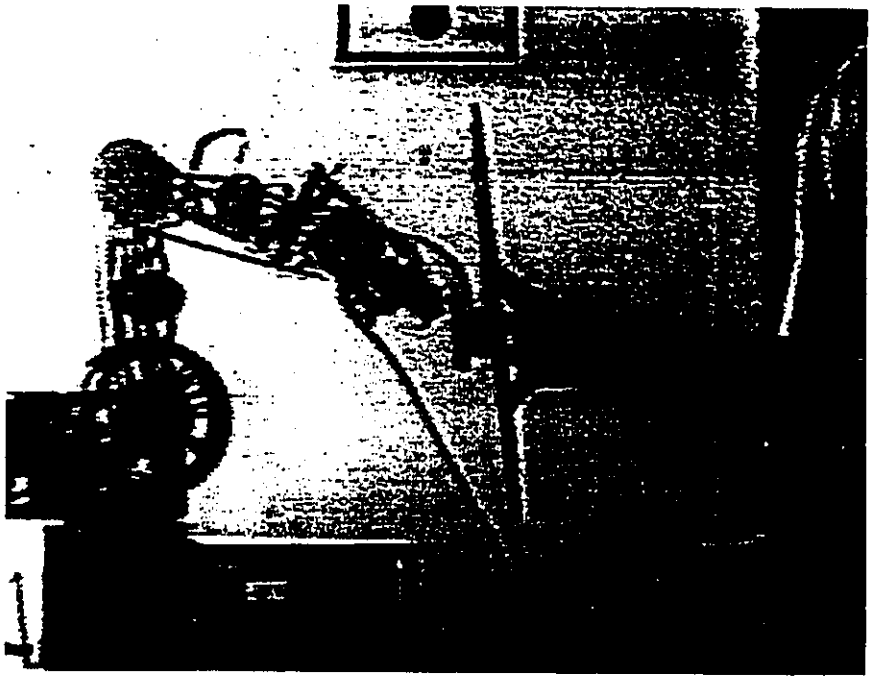
The first experiment utilized the straightforward plane tracking skill. A planar surface was pressed against the end-effector, and the contact parameters were altered by manually applying varying degrees of force. This is illustrated in Figure 4.9.

The second experiment, shown in Figure 4.10, combined the plane tracking skill with a lateral motion. The end-effector was contacted to the surface of a wheeled toy truck, and a bias force was applied. A motion strategy advanced the manipulator a certain incremental distance along the x_p axis, which in turn advanced the truck. Changes in the elevation of the surface along which the truck travelled were transmitted as changes in the normal force of contact, and the end-effector responded by reorienting the applied contact.

In the third experiment, shown in Figure 4.11, sphere tracking was used to track the motions of a sphere. A small rigid sphere was sandwiched between the end-effector and a planar surface. Lateral motions of the planar surface served to translate the sphere along the face of the sensor, which in turn caused the manipulator to respond in an effort to center the sphere. Changes in the normal forces were also tracked.



a) applied force normal to sensor



b) kinesthetic response to rotational force

Figure 4.9 : Plane Tracking Experiment

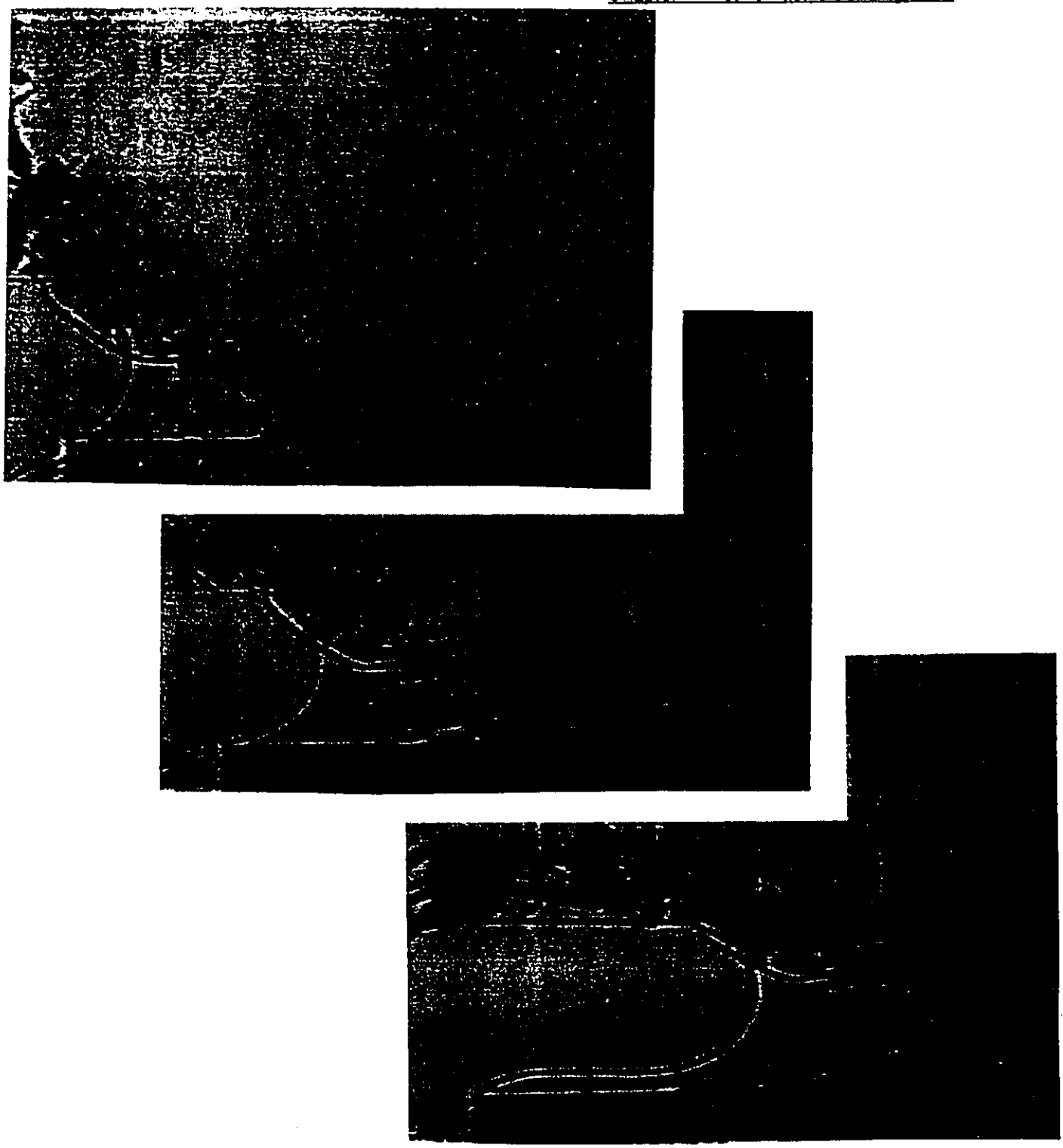


Figure 4.10 : Plane Tracking With Lateral Motion Experiment

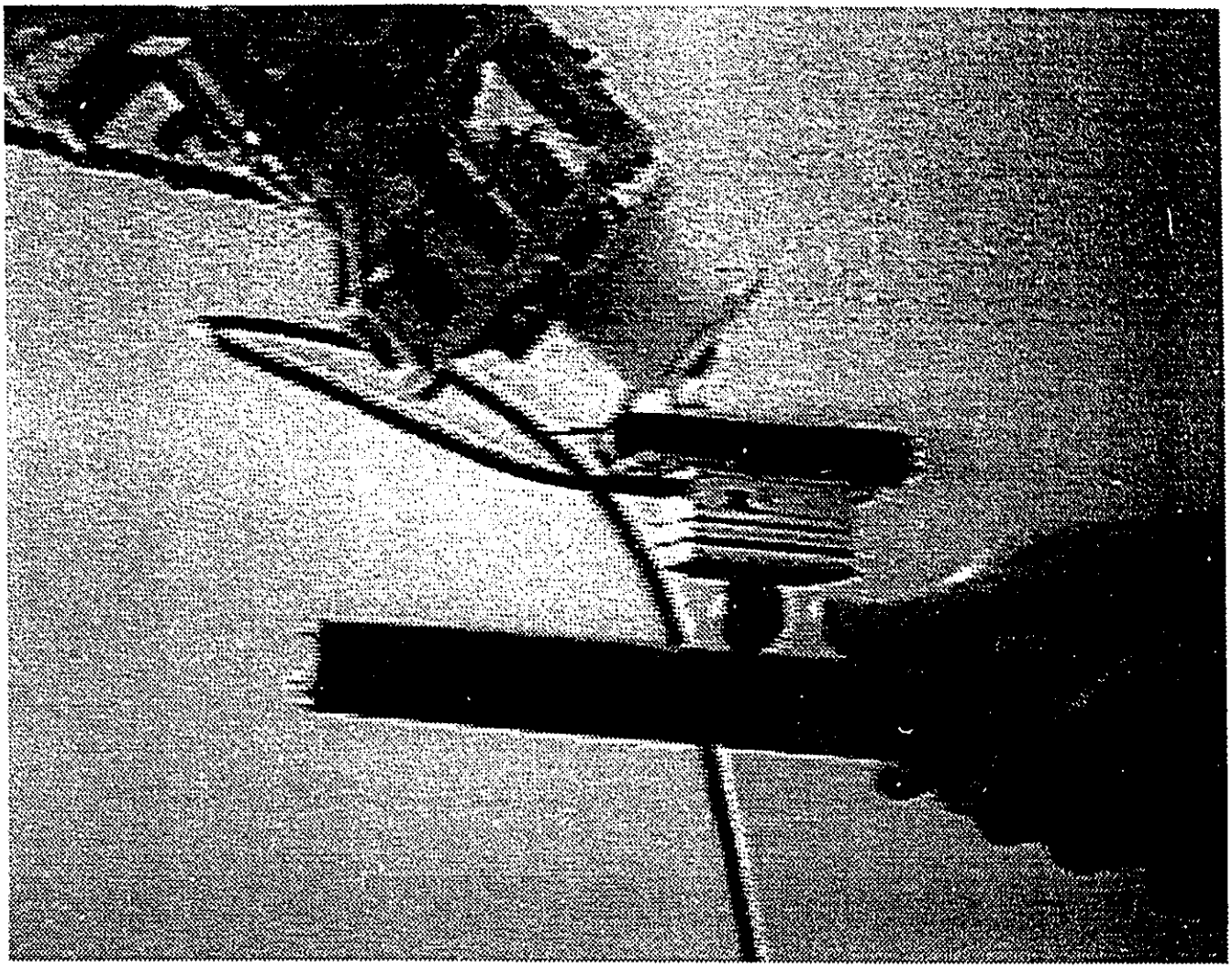


Figure 4.11 : Sphere Tracking Experiment

4.7 Conclusions

The main conclusion which was drawn from this experimental system was that cutaneous tactile image processing can be used effectively for robotic contact skills. One of the tracking skills, two d.o.f. plane tracking, has been implemented previously using Force Torque sensing. This development demonstrates that it can also be implemented using tactile sensing, with an estimate of the rotational information provided by a processing of the cutaneous tactile image.

The second skill, three d.o.f. feature tracking, is an example of a robot tracking skill which cannot be performed without the use of tactile sensing.

Having demonstrated the utility of tactile sensing for robotic tracking skills, there are a number of secondary conclusions pertaining to the performance evaluation of the system. From an analysis of the communication and computational overheads of the various processing steps, it was determined that the three main factors which limit the system response rate are; the communication overhead in serially transmitting the tactile data to the computer, the calculation of the IKS, and the slowness of the robot motion. Compared to either of these stages, the processing of the tactile data is computationally inexpensive.

Chapter 5 : Conclusions and Further Work

In this chapter the contributions of this thesis to the field of robotic tactile sensing are reviewed and conclusions are drawn. A number of possibilities to further the research in this area are also suggested.

The contribution of this work was the development and testing of two active tactile sensing skills. These skills facilitated the tracking of two contacted objects ; a plane in two d.o.f., and a sphere in three d.o.f. To the best of the author's knowledge, the implementation of these skills using a tactile array sensor has not been reported in the literature. As the development of sensor based skills is a current and important research area in robotics, it is believed that this work constitues a valid contribution to the field.

The development of the active skills involved the design and implementation of C program code for the cutaneous processing routines and the point-to-point kinematic control of the manipulator. The kinesthetic response to these functions was calibrated experimentally.

Directions for the extension of this work are derived from the performance analysis of the system. The response rates of both tracking skills are limited by the serial communication bandwidth of the tactile data, the calculation of the IKS, and slowness of robot motion. In order to further advance the developed tactile skills, it is necessary to

reimplement the system on a more industrial platform, such as PUMA 560, which has kinematic point-to-point response rates on the order of 28 msec.

It is also necessary to design a tactile sensor with a larger data communication bandwidth, implemented through a parallel communication interface. Each tactile image should be acquired and processed in under 10 msec. In order to achieve this fast rate, it may be necesssary to do some signal processing functions directly in the sensor's hardware circuitry.

Other possibilities include the investigation of the effects of the sampling rate on the control of the manipulator. It is accepted that good force control results from a time response on the order of 100 Hz to 1000 Hz.

This thesis proposes an alternative direction for future research in tactile sensing. Previous research has focussed on the use of tactile sensing for object recognition. Here, the emphasis has been on the use of active tactile sensing for the tracking of contacted objects. It is the author's belief that the practical advances in the field will arise from the development of tactile sensing skills for grasp and manipulation control. These capabilites are not only desirable for existing robotic manipulation problems, but object recognition capabilities will necessarily result as a spinoff. The identity of an object will be discernable from the series of sensory skills which successfully contact and explore the object.

Bibliography

General

[1] Petriu, E., Greenspan, M.A., McMath, W.S., Yeung, S.S.K., Active Tactile Perception of 3D Geometric Profiles. Proceedings of the IMTC/91 Instrumentation and Measurement Technology Conference, Atlanta, Georgia,1991, pp 464-467.

[2] Petriu, E., Greenspan, M.A., Gelinas, F., McMath, W.S., Yeung, S.S.K., An Active Tactile Perception System, Proceedings of the 6th CASI Conference on Astronautics, Ottawa, Onatrio ,1990, pp 59-67

[3] Petriu, E., Greenspan, M.A., Gelinas, F., McMath, W.S., Yeung, S.S.K., Tactile Sensing for 3-Dimensional Shape Estimation, Proceedings of the Canadian Conference on Electrical and Computer Engineering, Ottawa, Onatrio, 1990, pp 10.1.1.-10.1.4.

[4] Petriu, E., Greenspan, M.A., McMath, W.S., Yeung, S.S.K.,A Robotic Tactile Sensing System Concept, Proceedings of the IMTC/90 IEEE Instrumentation and Measurement Technology Conference, San Jose, CA, 1990, pp 132-135.

[5] Harmon, L.D., "Touch-Sensing Technology : A Review", Rovots V Conference, October 1980, pp 112-127.

[6] Harmon, L.D., "Tactile Sensing For Robotics", NATO ASI Series, vol. F11, Robotics and Artificial Intelligence, M. Brady ed., Springer-Verlag, Berlin, 1989, pp 107-157.

[7] Harmon, L.D.,"Automated Touch Sensing : A Brief Perspective and Several New Approaches", Rovots VII Conference, 1982, pp 326 - 331.

[8] Brady, M., "Artificial Intelligence and Robotics", Artificial Intelligence, 1985, pp 79-121.

[9] Davey, P.G., "Application and Requirements for Industrial Robotics", Robotics and Artificial Intelligence, NATO ASI Series, vol. F11, Springer-Verlag, 1984, pp 447-467.

[10] Nicholls, H.R., Lee, M.H., "A Survey of Robotic Tactile Sensing Technology", The International Journal of Robotics Research, vol. 8 no. 3, June 1989, pp 3-30.

[11] Seltzer, D. S., "Tactile Sensory Feedback for Difficult Robotic Tasks", Robot VI Proceedings, 1984, pp 467-478.

[12] Davey, P.G., "An Overview of the Basic Research Needed to Advance the State of Knowledge in Robotics", IEEE Transactions on Sytems, Man, and Cybernetics, vol. SMC-11 no. 8, Aug. 81, pp. 150-160.

[13] Jayawant, B.V., "Integration of Robotic Snesory Systems:, NATO ASI Series, vol. F43, Sensors and Sensory Systems for Advanced Robots, P. Dario ed., Springer-Verlag, 1988, pp 20-31.

[14] Dario, P., DeRossi, D.,"Tactile Sensors and the Gripping Challenge",IEEE Spectrum, Aug. 1985, pp 46-52.

[15] Pennywitt, K.E., "Robotic Tactile Sensing", BYTE, Jan. 1986, pp 177-200.

Tactile Transducers

[16] Boie, R.A., "Capacitive Impedance Readout Tactile Image Sensor", Proceedings of the International Conference on Robotics, Washington D.C., IEEE Computer Society Press, 1984, pp 370-378.

[17] Siegel, D.M., Druchen, S.M., Gorbieta, I., "Performance Analysis of a Tactile Sensor", Proceedings of the IEEE International Conference on Robotics and Automation, IEEE Computer Society Press, 1987, pp 1493-1499.

[18] Jayawant, B.V., Onori, M.A., McKWatson, J.D., "Robot Tactile Sensing : A New Array Sensor", Robot Sensors, vol 2., Tactile and Non-Vision, ed. A. Pugh, 1986, pp 130-145.

[19] Fan, L.S., White, R.M., Muller, R.S., 1984, "A Mutual Capacitive Normal and Shear-Sensitive Tactile Sensor", IEEE Electron Devices Magazine, 1984, pp 220 - 222.

[20] Pewa, "Tactile Sensors For Robotics and Medicine", ed. J.G. Webster, pp 201-232.

[21] Sato, N., Pugh, A., :A Method For Three-Dimensional Part Identification By Tactile Transducer", Robot Sensors, vol.2 TJ 211 R54 v.2 pp 133-143.

[22] Vranish, J.M., "Magnetoresistive Skin for Robots", Robot Sensors, vol 2., Tactile and Non-Vision, ed. A. Pugh, 1986, pp 99- 123.

[23] Hillis, W.D., "A High-Resolution Imaging Touch Sensor", The International Journal of Robotics Research, vol. 1 no. 2, Summer 1982, pp 33-44.

[24] Prubrick, J.A., "A Force Transducer Employing Conductive Silicon Rubber", Proceedings of the 1st International Conference on Robot Vision and Sensory Controls", Kempton, U.K., 1981, pp 73-80.

[25] Raibert, M.H., "An All Digital VLSI Tactile Array Sensor", Proceedings of the Internationai Conference on Robotics, Washington D.C., IEEE Computer Society Press, 1984, pp 317-319.

[26] Robertson, B.E., Walkden, A.J., "Tactile Sensor Systems For Robotics", Robot Sensors, vol 2., Tactile and Non-Vision, ed. A. Pugh, 1986, pp 89-97.

[27] Larcombe, M.H.E, "Why Carbon Fibres Can Give Robots A Strong Sense of Grip", Sensor Review, 1 (1), 1981, p. 58-59.

[28] Ristic, R., Benhabib, B., Goldenberg, A., "Analysis and Deisgn of a Modular Electrooptical Tactile Sensor", IEEE Transactions on Robotics and Automation, vol. 5 no. 3, June 1989, pp 362-368.

[29] Begej, S., "Planar and Finger-Shaped Optical Tactile Sensors for Robotic Applications", IEEE Journal of Robotics and Automation, vol. 4 no. 5, Oct. 1988.

[30] ---, "Piezo Film Yields Novel Transducers", Electronics Week, Aug. 6, 1984.

[31] Patterson, R.W., Nevillw, G.E., "Performance of an Induced Vibration Touch Sensor", Robots V, pp 11-108 to 11-118.

[32] Dario, P., Fiorillo, A., Buttazzo, G., Bergamasco, M., "A Piezoelectric Polymer Tactile Sensor System", Robot Sensors 2 : Tactile and Non-Vision, pp. 1-23.

[33] Luo, R.C., Wang, F., Liu, Y., "An Imaging Tactile Sensor with Magnetoresistive Transduction", Robot Sensors, vol 2., Tactile and Non-Vision, ed. A. Pugh, 1986, pp 113 - 131.

[34] Checinski, S., Agrawal, A.K., "Magnetoelastic Tactile Sensor", Robot Sensors, vol 2., Tactile and Non-Vision, ed. A. Pugh, 1986, pp 229 -241 .

[35] Webster, J.G., "Tactile Sensors For Robotics and Medicine", John Wiley and Sons, U.S.A, 1988.

[36] Cameron, A., Daniel, R., Durrant-Whyte, H., "Tactile Geometry for Images and Normals", Sensor Devices and Systems for Robotics, NATO ASI Series, vol. F52, A. Casals ed., Springer-Verlag, 1989, pp 67-77.

[37] Bicchi, A., Dario, P., "Intrinsic Tactile Sensing for Artificial Hands", Proceedings of the IV International Symposium on Robotics Research", 1988, pp 83-90.

[38] Speeter, T.H., "Tactile Sensing System for Robotic Manipulation", The International Journal of Robotics Research, 1988, pp. 144-162.

[39] Vaidyanathan, C.S., Wood, H.C., "A Practical Model For An FSR Based Tactile Sensor", Proceeding of the Canadian Conference on Electrical and Computer Engineering, Sept. 1989, pp 500-503.

[40] Pruski, A., Mutel, B., "Direct Contact Sensors Based on Carbon Fibre", Proceedings of the International Conference of Robotics and Factories of the Future, 1984, pp 409-415.

[41] Raibert, M.H., Tanner, J.E., "Design and Implementation of a VLSI Tactile Sensing Computer", International Journal of Robotics Research, 1 (3), 1982, pp 3-18.

Object Recognition Systems

[42] Muthakrishnan,C. Smith, D., Myers, D., Robman, J., Koiro, A., "Edge Detection In Tactile Images", Proceedings of the International Conference on Robotics and Automation, 1987, pp 1500-1505.

[43] Jurczyk, J. and Loparo, K.A.,"Mathematical Tranforms and Correlation Techniques for Object Recognition Using Tactile Data",IEEE Transactions on Robotics and Automation, vol. 5 no. 3, June 1989, pp 359-361.

[44] Allen, P.K., "Object Recongnition Using Vision and Touch", PhD Thesis Dissertation, University of Pennsylvania, 1985.

[45] Allen, P.K., "Surface Descriptions From Vision And Touch",IEEE Transaction on Robotics and Automation, 1986, pp 394 - 397.

[46] Allen, P.K., "Integrating Vision and Touch for Object Recognition Tasks", The International Journal of Robotics Research, vol. 7 no. 6, Dec. 1988, pp 15-33.

[47 DeMeter, E.C., Deisensroth, M.R., "The Integration of Visual and Tactile Sensing for the Definition of Regions Withina Robot Workcell", IEEE Transaction on Robotics and Automation, 1988, pp 33-57.

[48] Stansfield,S.A., "A Robotic Perceptual System Utilizong Passive Vision and Active Touch",The International Journal of Robotics Research, vol. 7 no. 6, Dec. 1988, pp 138-161.

[49] Gaston, P.C., Lozano-Perez, T., "Tactile Recognition and Localization Using Object Models: The Case Of Polyhedra on a Plane", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-6, no. 3, May 1984, pp 257-266.

[50] Grimson, W.E.L., Lozano-Perez, T.,"Model Based Recognition and Localization from Sparse Range or Tactile Data", The International Journal of Robotics Research, vol.3 no.3, Fall 1984, p. 3-35.

[51] Browse, R., "Feature Based Tactile Object Recognition", IEEE Transaction on Robotics and Automation, 1987, pp 19-32.

[52] Dario, P., Buttazzo, G., "An Anthropomorphic Robot Finger for Invesitigating Artificial Tactile Perception", The International Journal of Robotics Research, vol. 6 no. 3, Fall 1987, pp 25-48.

[53] Ogorek, M.,"Tactile Sensors", Manufacturing Engineering, Feb. 1985, pp 71-77.

[54] Bajczy, R., "What Can We Learn From One Finger Experiments?", International Journal of Robotics Research, 1983, p. 509-513.

[55] Whitney, D.E., "What is the Remote Compliance Center and What Can It Do ?", Robot Sensors, vol. 2, A. Pugh ed., pp. 3-15.

[56] Kinoshita, G., "A Pattern Classification By Dynamic Tactile Sense Imformation Processing", Pattern Recognition, Paragon Press, vol.7, 1975, pp 243-245.

[57] Ozaki, H., Waku, S. Mohri, A., Takata, M., "Pattern Recognition of a Grasped Object By Unit Vector Distribution", IEEE Transactions on Systems Man and Cybernetics, vol. SMC-12, no. 3, May/June 1982, pp 315-324.

[58] Allen, P.K., Roberts, K.S.,"Haptic Object Rcognition Using a MultiFingered Dexterous Hand",IEEE Transaction on Robotics and Automation, 1989, pp 53-69.

[59] Wang, Y.F., Aggarwal, J.K., :An Overview of Geometric Modelling Using Active Sensing",IEEE Control Systems Magazine, June 1988, pp 5-12.

[60] Dario, P., Bicchi, A., Fiorillo, A., Butazzo, G., Francesconi, R., "A Sensorized Scenario For Basic Investigation on Active Touch", Robot Sensors, vol. 2, Tactile and Non-Vision, A. Pugh ed., Springer-Verlag, 1986, pp 237-245.

[61] Pribadi, K., Bay, J.S., Hemamai, H., "Exploration and Dynamic Shape Estimation by a Robotic Probe", IEEE Transaction on Systems Man and Cybernetics, vol. 19 no. 4, July/Aug 1989, pp 840-845.

Kinematics

[62] Denavit, J., Hartenberg, R.S., "A Kinematic Notation For Lower-Pair Mechanisms Based On Matrices", Journal of Applied Mechanics, pp 215-221.

[63] Lee C.S.G., "Robot Arm Kinematics", Tutorial on Robotics, IEEE Computer Society Press, 1984, pp 47-65.

[64] Lee C.S.G., "Robot Arm Dynamics", Tutorial on Robotics, IEEE Computer Society Press, 1984, pp 93-102.

[65] Elgazzar, S., "Efficient Solution for the Kinematic Positions for the PUMA 560 Robot", National Research Council of Canada Internal Report No. 23952, Dec. 1984.

[66] Goldenberg, A.A., Benhabib, B., Fenton, R.G., "A Complete Generalized Solution to the Inverse Kinematics of Robots", IEEE Journal of Robotics and Automation, vol. RA-1 no. 1, March 1985, pp 14-20.

[67] Schilling, R.J., "Fundamentals of Robotics : Analysis and Control", Prentice Hall Inc., U.S.A, 1990.

[68] White, W.N. Niemann, D.D., Lynch, P.M., "The Presentation of Lagranges' Equations in Introductory Robotics Courses", IEEE Transactions on Education, vol. 32 no. 1, Feb. 1989, pp 39-46.

[69] RHINO XR Owner's Manual, Rhino Robots Inc. Publication, 1986.

[70] RHINO XR Software Manual, Rhino Robots Inc. Publication, 1986.

[71] RHINO XR RHINO-VAL Manual, Rhino Robots Inc. Publication, 1986.

Appendix A

Tactile Sensing Concepts : Anthropomorphic Considerations

(Internal report : 1989-Sept-20)

Michael A. Greenspan,
Emil M. Petriu,
Dept. of Electrical Engineering,
University of Ottawa,
Ottawa, Ontario, Canada,
K1N 6N5

Abstract

When developing an artificial tactile sensor system, it is useful to have an understanding of the nature of biological tactile sensing. Presented are some of the features of a biological sensing system: particularly, those aspect pertaining to the accumulation and processing of tactile data are emphasized.

1. Introduction

A study of the biological basis is invaluable when designing a tactile sensing system for a number of reasons. There may be biological tactile receptors which can be copied in a technological implementation of an artificial tactile system. More importantly, a knowledge of biological tactile perception will help us better understand the nature of the sense which we are attempting to emulate. Indeed, although everyone has a direct experiential understanding of tactile perception, to explicitly describe this understanding is a difficult task. It is more difficult, however, to design such a system which has not been explicitly described.

As an example, consider the design of a mechanical gripper to perform conventional dexterous manufacturing tasks. Having determined the two point threshold for human fingerprint skin (first investigated by Weber) to be around 1 mm at its most sensitive, it would surely be a wasted effort to strive for a far greater resolution in our artificial skin. If a greater degree of resolution is attainable in our design, however, we may conceive of an artificial perception which surpasses our own tactile abilities, and allows a greater flexibility in the manufacturing process.

There is also the danger when engineering an anthropomorphic system of becoming too enthralled with the biology of the system. When this occurs, the resultant design may tend to mimic the

structure of the biological system under consideration, without providing the useful functionality of that system : clearly, this is not our goal. Our admitted predisposition thus allows us to describe a selective overview of certain aspects of the biological somatosensory system which help us understand our task and, in some instances, guide us in our implementation.

2. Biological Mechanisms of Tactile Sensing

Historical roots of tactile sensing research are discussed in [1,2]. It is notable that in antiquity, Aristotle defined perception by touch ("haphe") as the most basic property of all living organisms, without which they could not exist. He wrote that tactile stimuli arose only from immediate contact with the object, touch appearing as the only type of sensation responding to an object placed directly on the perceiving organ. Another ancient Greek, Galen, was a pioneer of the modern "sensor fusion" approach. He described the hand as an organ equipped for the perception of all tactile qualities :

> " There was no use for one organ of apprehension, and another for touch, for holding, for lifting and transferring things, and another for discriminating warm, cold, hard, soft, and other distinguishing qualities of an object. However, by grasping, the hand can judge the nature of all these [impressions] together." [1]

2.1 Functional Innervation of Human Fingerprint Skin

Fingerprint skin, the organ area which is usually involved in active touch, contains four types of nerve configurations, which have been functionally related to four types of

mechanoreceptive, human skin also responds in a thermoreceptive and nociceptive fashion. Nociceptive response results from particularly stressful and potentially damaging mechanical or thermal agitation. Here, only the mechanoreceptive function is of interest, as thermoreception does not present a problem from a design point of view, and nociception can be viewed as a class of mechanical or thermal reception.

There have been four functional types of receptive fields identified in human digital skin, two of which respond only once to a prolonged stimulation, and two of which respond continuously to a prolonged stimulation [3,4,5,6]. These groupings are accordingly labeled rapidly adapting (RA) and slowly adapting (SA) fields. The receptive fields are further grouped according to the size and distribution of the stimulation threshold, as is described following.

The first type of rapidly adapting field, misguidingly labeled the RA field, is characterized by many points of maximal sensitivity, and a rapid rolloff in the sensitivity threshold. Thus, the RA field locates a stimulus within a small region of a few square mm. The receptors for the RA fields are the Meissner Corpuscles. These small encapsulations of nerve are located between the papillary ridges of the dermal epidermal interface. At maximum sensitivity, the Meissner Corpuscles will fire from a displacement of 6 microns.

The second rapidly adapting receptive field is labeled the PC field after its receptive neuron, the Pacinian Corpuscle. Located deeper down into the dermus, the Pacinian Corpuscles is 0.5 by 1.5 mm in size, the largest single receptor of the body. The PC field is much larger than the RA field, extending over a few square cm of skin. Also, this field has only one point of maximal sensitivity. The area surrounding this point very gradually diminishes in sensitivity. It is interesting to note that, while the Meissner Corpuscles share a number of nerve fibers, the Pacinian Corpuscles will often have a single neural connection.

The slowly adapting receptive fields, SAI and SAII, are in many ways analogous to the RA and PC fields. SAI, like RA, has multiple points of maximum sensitivity and a quick rolloff, while SAII has a single maximally sensitive point and a slower attenuation, similar to the PC field. The neural structure associated with the SAI field is the Merkel Disk, located at the tip of the epidermal ridge. The Ruffini Endings, said to be the receptors for the SAII field, are distributed deeper into the dermus. While the SAII field is very sensitive to skin stretch, SAI requires a more direct stimulation, and responds very little to stretching.

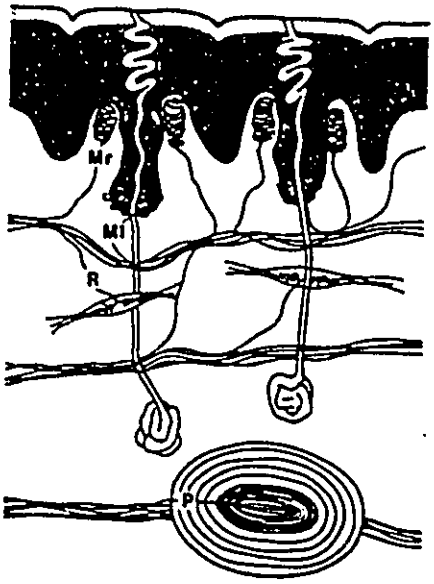


Figure 1 : Schematic Vertical Section of Human Hand Skin

(from Valbro and Johansson, [3]). The neural structures are identified as : Mr - Meissner Corpuscle, Ml - Merkel Complex, R - Ruffini ending, and P - Pacinian Corpuscle. Also shown are the dermal-epidermal interface, the intermediate and limiting ridges, and two sweat glands.

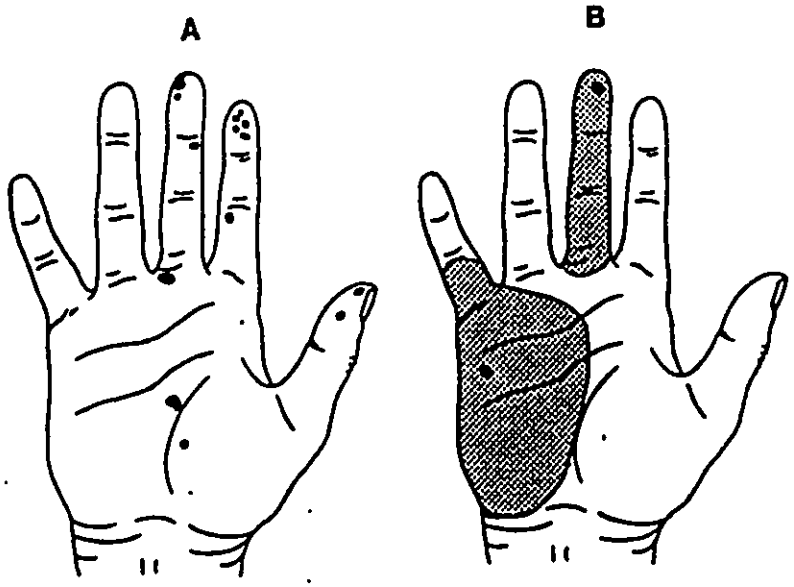


Figure 2 : A) RA Field , B) PC Field

(from Valbro and Johansson, [3]). In A), the receptive fields for 15 RA fields are shown. In B), the much larger receptive filed for two PC fields.

2.2 Texture Perception

Surprisingly little has been ascertained about the mechanisms underlying the perception of textured surfaces. It is considered to depend largely upon the mechanical properties of skin, particularly the glabrous skin in the fingertips [7]. Certain physical attributes of this skin, such as an oily coating, and the outer layer of dead epidermal cells, accentuate vibrations as it is passed lightly over surfaces. It is the nature of these vibrations which allow for the determination of varying degrees of roughness, hardness, stickiness, slipperiness, etc. [8].

Indeed, the perception of textured surfaces is entirely a dynamic sense, which can be carried out only when the skin is passed over a given surface. Only in the case of very coarse surfaces with protruberances can textural information be derived from a static placement of the fingers. This, of course, discludes the cases were auxiliary information, such as thermal and visual clues, can be used in conjunction with the mechanosenitive reception.

The Meissner and Merkel corpuscles have been identified as the dynamic receptors of these vibratory senses. The Meissner corpuscles are responsible for the sensation of "flutter" which is experienced when the surface oscillations are on the order of 5 - 40 Hz. Higher frequencies, from 40 - 200 Hz, are received by the more deeply embedded Pacinian corpuscles [9].

In one particular experiment [9,10], Lederman observed that by covering the fingertips with a thin piece of material (a glove or a piece of paper), a previously considered smooth surface would reveal numerous irregularities and protruberances. It seems that a masking effect filters out the high frequency components of the vibrations experienced. The structural explanation offered for this common effect resolves the forces of active touch into lateral and normal components, and considers the effect which these forces could have on the shallow intermediate and deeper limiting ridges which comprise the papillary ridges. By deforming these ridges, lateral or shearing forces tend to become amplified and the Merkel corpuscles at the bottom of the limiting ridge, as well as the Meissner corpuscle between the intermediate and limiting ridges, are ideal receptor sites for the wealth of combinations possible. As these units are excited by the high frequency oscillations, they become less sensitive to low frequency pertebations : hence , the masking effect.

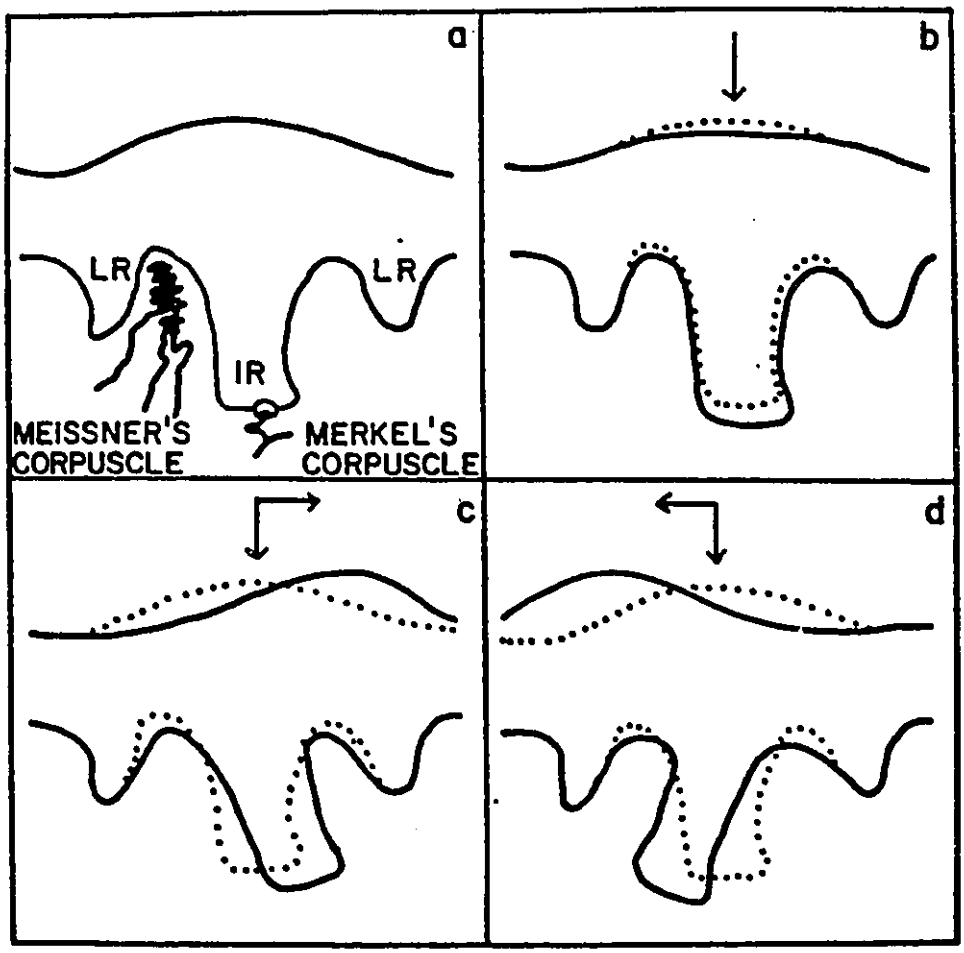


Figure 3 : Proposed Mechanism for the Detection of Shearing Forces in Human Digital Skin

(from Lederman, [9]). In a) the Meissner and Merkel corpuscles are shown as the respective receptors for motions from the Limiting and Intermediate Papillary Ridges. In b), a proposed mechanism for the reception of a normal force. In c) and d), proposed mechanisms for the reception of lateral forces.

3. Cortical Processing of Tactile Information

It was Galen who had recognized the brain as the seat of perception :

> " He understood that sensations of touch and temperature were subject to immediate recognition, whereas the perception of qualities such as dry and moist, or of size and motion seemed to require a conscious judgment. This presented and anticipation of the concept that immediate perception by specialized nerve-endings has to be distinguished from the evaluation of complex impressions." [1]

Having identified neural structures involved in the reception of tactile information, the question remains as to how this information is organized further up in the perceptual hierarchy. Studies by Sakata and Iwamura [11] of the first somatosensory area (SI) and the parietal association area (area 5) in cats and monkeys have offered insight about the brain's internal classification of cutaneous stimuli.

There are selective temporal patterns of cutaneous stimulation which elicit the excitation of individual cortical neurons. Some of these patterns are excitatory only when applied in a given direction. For example, with monkeys, a glass rod stroked across the palm of the hand causes an excitation in an area 2 neuron when moving in the proximal to distal direction (i.e., towards the fingertips). Movement in any other direction has no effect. With man, the application of a metal edge to the fourth finger caused the firing of an area 2 neuron only when it was oriented parallel to the axis of the finger and oscillated sideways.

Along with pure feature extraction, individual neurons have been isolated which allude to higher organizational principles. In the cat, receptive fields along the leg and paw correspond to the functional surfaces stimulated during various poses. This finding suggests that there are adaptive data reduction processes active.

In the monkey, neurons have been isolated which respond to active touch only. An area which causes vigorous neural excitation during active exploratory motions remains silent when stimulated passively. Further, individual neurons have been isolated which relate specifically to shape discrimination. In one case, neural excitation is recorded when the monkey handles cube shaped blocks, while spherical shaped blocks produce no response. Conversely, other neurons respond solely to the manipulation of spherical shapes, and are silent when cubes are handled.

The above described experimental results are just a small sample of the of work being carried out in this field. They are of interest to us for two reasons. First, the experiments refer to a particular interpretation of brain function. Specifically, the brain is considered as an adaptive hierarchical processor for the classification of tactile stimuli. Second, the manner in which the information is organized provides us some insight into the perception of tactile stimulation. Further studies have isolated the convergence of tactile neurons with proprioceptive and visual fibers [11].

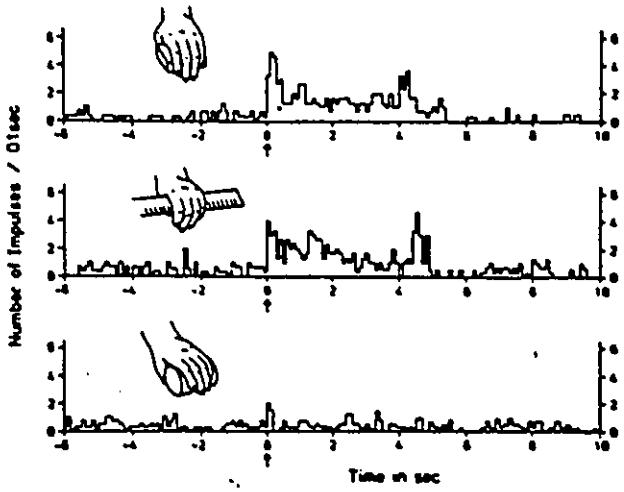


Figure 4. A) : Discharge vs. Time in a Single Cortical Neuron When Grasping Two Rectangular and One Cylindrical Block (from Sakata and Iwamura, [11]).

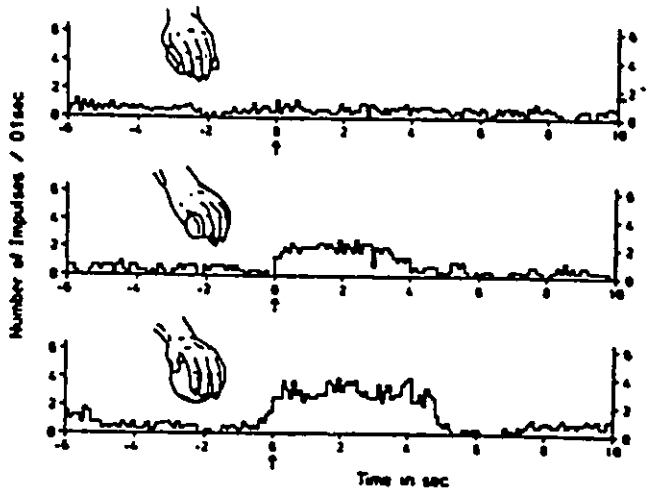


Figure 4. B) : Discharge vs. Time in a Single Cortical Neuron When Grasping Two Cylindrical and One Rectangular Block (from Sakata and Iwamura, [11]).

4. References

[1] R.E. Siegel, Galen on Sense Perception, 1970, ch V.

[2] E.H. Weber, The Sense of Touch, 1978.

[3] A.B. Vallbo and R.S. Johansson, "The Tactile Sensory Innervation of the Glabrous Skin of the Human Hand", Active Touch, 1978, pp 29-54.

[4] T.A. Quilliam, "The Structure of Finger Print Skin", Active Touch, 1978, pp 1-19.

[5] T.A. Quilliam, "Neuro-Cutaneous Relationships in Fingerprint Skin", The Somatosensory System, 1975, pp 193-199.

[6] C.E. Sherrick and J.C. Craig, "The Psychophysics of Touch",Tactual Perception : A Sourcebook, 1982.

[7] J. Petit and Y. Galifret, "Sensory Coupling Function and the Mechanical Properties of the Skin", Active Touch, 1978, pp 19-28.

[8] R.H. LaMotte and V.B. Mountcastle,"Neural Processing of Temporally-ordered Somesthetic Input: Remaining Capacity in Monkeys Folowing Lesions of the Parietal Lobe",Active Touch, 1978, pp 73-78.

[9] S. Lederman, "The Perception of Texture by Touch", Tactual Perception : A Sourcebook,1982.

[10] S. Lederman, "Heightening Tactile Impressions of Surface Texture", Active Touch, 1978, pp 205-214.

[11] H. Sakata and Y. Iwamura, "Cortical processing of Tactile Information in the First Somatosensory and Parietal Association Areas in the Monkey", Active Touch, 1978, pp 55-72.

[12] S. Millar, "Aspects of Memory for Information from Touch And Movement", Active Touch, 1978, pp 215-228.

[13] E.C. Carterette and M.P. Friedman, ed., Handbook of Perception, Vol VIB, 1978, ch 1-2.

Appendix B : Code Listing

This appendix contains the source code listings of the software system implemented to calibrate and execute the described tactile sensing skills. The code dependancies are illustrated in Figure B.1 ;

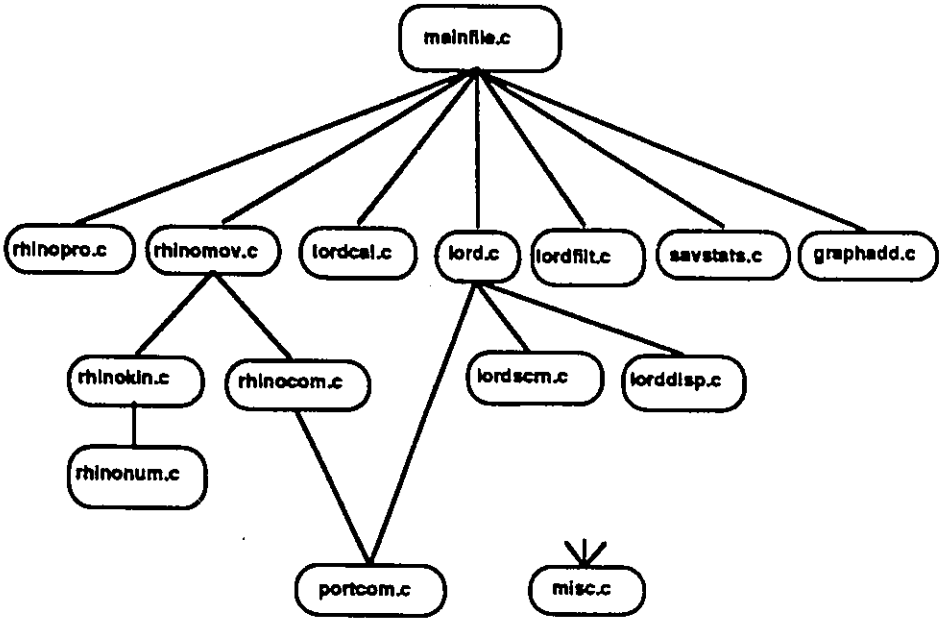


Figure B.1 : Modular Structure of Software System

```
/*****************************************************************/
/*                                                               */
/*          title : MAINFILE.C                                   */
/*          author : M.A. Greenspan                              */
/*          date : 14-MAY-90                                     */
/*                 31-MAY-90                                     */
/*                                                               */
/*****************************************************************/

#include <conio.h>
#include <graphics.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "c:\tc201\programs\robosens\include\vehtypo.h"
#include "c:\tc201\programs\robosens\include\vmisc.h"
#include "c:\tc201\programs\robosens\include\portcom.h"
#include "c:\tc201\programs\robosens\include\rs232com.h"
#include "c:\tc201\programs\robosens\include\graphadd.h"
#include "c:\tc201\programs\robosens\include\lord.h"
#include "c:\tc201\programs\robosens\include\lorddisp.h"
#include "c:\tc201\programs\robosens\include\lordscrn.h"
#include "c:\tc201\programs\robosens\include\lordcal.h"
#include "c:\tc201\programs\robosens\include\lordfilt.h"
#include "c:\tc201\programs\robosens\include\reflex.h"
#include "c:\tc201\programs\robosens\include\rhino.h"
#include "c:\tc201\programs\robosens\include\rhinocom.h"
#include "c:\tc201\programs\robosens\include\rhinomov.h"
#include "c:\tc201\programs\robosens\include\rhinokin.h"
#include "c:\tc201\programs\robosens\include\rhinonum.h"
#include "c:\tc201\programs\robosens\include\rhinopro.h"
#include "c:\tc201\programs\robosens\include\savstats.h"

#define START                           0
#define     PORT_INITIALIZE     10
#define LORD_INITIALIZE       20
#define RHINO_INITIALIZE      30
#define SCREEN_INITIALIZE     40
#define PAD_INITIALIZE        50
#define     CALIBRATE                   65
#define     PLANE_TRACKING              67
#define     FEATURE_TRACKING68
#define CHOOSE              70
#define RECONFIGURE         80
#define INCREMENT           90
#define     VECTOR              100
#define     ROTATE                      102
#define     FUNCTION_STATISTICS         105
#define TEST_TIME                 106
#define     DISPLAY_LOCATION    107
#define     DRIVE_THE_CAR               108
#define EXIT              110
#define BYE               120

#define EPSILON_PER_ROW_CENTER_OF_MASS 0.007 /* was 0.014 */
#define CM_PER_DELTA_TAXIL_SUM 0.002
#define     TRUE 1
#define FALSE 0

padtype lord_sensor ;
lord_stats_type min_threshold_stats, max_threshold_stats, lord_threshold_stats ;
reflex_type reflex ;

main() {
        int step ;
        int index ;
        int status ;
        int max_threshold, max, min_threshold, min ;
        int row_center_of_mass, col_center_of_mass, taxil_sum, last_taxil_sum, delta_taxil_sum ;
        int number_of_iterations, iteration ;
        float dummy_dcspace[3], dcspace[3], cspace[3] ;
        float row_epsilon, col_epsilon, delta_normal_direction ;
        float lateral_distance ;
        clock_t start_time, end_time ;
        double cycle_time ;
        char initcode ;
```

```
char resp ;
char filename[255] ;
int normal_reflex, rotational_reflex ;
arm_type currentarmstate, nextarmstate, deltaarm ;
vector_type orientation, dorientation, arm_length_axis, arm_width_axis ;
FILE *write_stream ;

step = START ;
printf("\nRunning program LORDLINE ...\n") ;
do {
        switch(step) {
                case START : {
                        step = PORT_INITIALIZE ;
                        break ;
                }
                case PORT_INITIALIZE : {
                        printf("\nInitializing ports COM0 and COM1 ...") ;
                        initcode = BAUD9600 | PARITYNO | STOPBIT2 | SIZE8 ;
                        port_init(LORDPORT,initcode) ;
                        initcode = BAUD9600 | PARITYEVEN | STOPBIT2 | SIZE7 ;
                        printf("\ninitializing port ...\n");
                        port_init(RHINOPORT,initcode);
                        step = LORD_INITIALIZE ;
                        break ;
                }
                case LORD_INITIALIZE : {
                        printf("\nReseting LORD sensor ... ") ;
                        lord_reset() ;
                        step = RHINO_INITIALIZE ;
                        break ;
                }
                case RHINO_INITIALIZE : {
                        for (index=0; index<5; index++) {
                                dcspace[index] = 0 ;
                        }
                        printf("\n\nreseting RHINO ... \n");
                        rhino_reset(RHINOPORT) ;
                        printf("\n\n(H)ARD Home or (S)OFT Home ? > ") ;
                        resp = getche() ;
                        if ((resp=='H')||(resp=='h')) {
                                currentarmstate.angle.a = ANGLE_A_HARD_HOME * M_PI_2/90;
                                currentarmstate.angle.b = ANGLE_B_HARD_HOME * M_PI_2/90;
                                currentarmstate.angle.c = ANGLE_C_HARD_HOME * M_PI_2/90 ;
                                currentarmstate.angle.d = ANGLE_D_HARD_HOME * M_PI_2/90;
                                currentarmstate.angle.e = ANGLE_E_HARD_HOME * M_PI_2/90 ;
                                currentarmstate.angle.f = ANGLE_F_HARD_HOME * M_PI_2/90 ;
                        }
                        else {
                                currentarmstate.angle.a = ANGLE_A_SOFT_HOME * M_PI_2/90;
                                currentarmstate.angle.b = ANGLE_B_SOFT_HOME * M_PI_2/90;
                                currentarmstate.angle.c = ANGLE_C_SOFT_HOME * M_PI_2/90 ;
                                currentarmstate.angle.d = ANGLE_D_SOFT_HOME * M_PI_2/90;
                                currentarmstate.angle.e = ANGLE_E_SOFT_HOME * M_PI_2/90 ;
                                currentarmstate.angle.f = ANGLE_F_SOFT_HOME * M_PI_2/90 ;
                        }
                        currentarmstate.step.a = 0 ;
                        currentarmstate.step.b = (int) (currentarmstate.angle.b * BSTEPSPERRAD) ;
                        currentarmstate.step.c = (int) (currentarmstate.angle.c * CSTEPSPERRAD) ;
                        currentarmstate.step.d = (int) (currentarmstate.angle.d * DSTEPSPERRAD) ;
                        currentarmstate.step.e = (int) (currentarmstate.angle.e * ESTEPSPERRAD) ;
                        currentarmstate.step.f = (int) (currentarmstate.angle.f * FSTEPSPERRAD) ;
                        orientation.i = -1 ;
                        orientation.j = 0 ;
                        orientation.k = -1 ;
                        step = DISPLAY_LOCATION ;
                        break ;
                }
                case SCREEN_INITIALIZE : {
                        graphics_initialize(VGA,VGAMED) ;
                        settextstyle(TRIPLEX_FONT,HORIZ_DIR,2) ;
                        clearviewport() ;
                        step = PAD_INITIALIZE ;
                        break ;
                }
                case PAD_INITIALIZE : {
                        general_pad_initialize(&lord_sensor, 50, 340, 15, 6, 6, HORIZONTAL) ;
                        pad_display_2d(&lord_sensor, NOISE_THRESHOLD, XOFFSET, YOFFSET) ;
                        step = PLANE_TRACKING ;
```

```
                break ;
        }
        case CALIBRATE : {
                printf("\n\nApply pressure for maximum threshold and press any key > ") ;
                resp = getche() ;
                printf("\n\nHold steady while calibrating stats for maximum threshold ...") ;
                lord_calibrate(&max_threshold_stats) ;
                printf("\n\nApply pressure for minimum threshold and press any key > ") ;
                resp = getche() ;
                printf("\n\nHold steady while calibrating stats for minimum threshold ...") ;
                lord_calibrate(&min_threshold_stats) ;
                printf("\n\n\tSUM\n\tCENTROID\n\tCOMASS") ;
                printf("\n\n\tROW\tCOL\n\tROW\tCOL") ;
                printf("\n\nmin\t%3.2f\n\t%3.2f\t%3.2f\n\t%3.2f\t%3.2f",
                        min_threshold_stats.sum_of_averages,
                        min_threshold_stats.centroid_row_average,
                        min_threshold_stats.centroid_col_average,
                        min_threshold_stats.center_of_mass_row_average,
                        min_threshold_stats.center_of_mass_col_average) ;
                printf("\n\nmax\t%3.2f\n\t%3.2f\t%3.2f\n\t%3.2f\t%3.2f",
                        max_threshold_stats.sum_of_averages,
                        max_threshold_stats.centroid_row_average,
                        max_threshold_stats.centroid_col_average,
                        max_threshold_stats.center_of_mass_row_average,
                        max_threshold_stats.center_of_mass_col_average) ;
                step = CHOOSE ;
                break ;
        }
        case PLANE_TRACKING : {
                dummy_dcspace[0] = 0 ;
                dummy_dcspace[1] = 0 ;
                dummy_dcspace[2] = 0 ;
                printf("\nEnter taxil sum value [0,4800] > ") ;
                scanf("%d",&last_taxil_sum) ;
                do {
                        lord_scan(&lord_sensor) ;
                        center_of_mass(lord_sensor,&row_center_of_mass,&col_center_of_mass,&taxil_sum) ;
                        delta_taxil_sum = taxil_sum - last_taxil_sum ;
                        if ((delta_taxil_sum > 1)||(delta_taxil_sum < 1)) {
                                delta_normal_direction = - (delta_taxil_sum * CM_PER_DELTA_TAXIL_SUM) ;
                                dcspace[0] = orientation.i * delta_normal_direction ;
                                dcspace[1] = orientation.j * delta_normal_direction ;
                                dcspace[2] = orientation.k * delta_normal_direction ;
                                normal_reflex = TRUE ;
                        }
                        else {
                                dcspace[0] = 0 ;
                                dcspace[1] = 0 ;
                                dcspace[2] = 0 ;
                                normal_reflex = FALSE ;
                        }
                        row_epsilon = (8 - row_center_of_mass) * EPSILON_PER_ROW_CENTER_OF_MASS ;
                        if ((row_epsilon > 0.01)||(row_epsilon < 0.01)) {

rotate_around_arm_width_axis(row_epsilon,&currentarmstate,orientation,cspace,cspace,&dorientation,dummy_dcspace) ;
                                rotational_reflex = TRUE ;
                        }
                        else {
                                donentation.i = 0 ;
                                donentation.j = 0 ;
                                donentation.k = 0 ;
                                rotational_reflex = FALSE ;
                        }
                        if (normal_reflex || rotational_reflex) {
                                if (!rhino_move_delta(dcspace, dorientation, cspace, &orientation,
&currentarmstate)) {
                                        printf("\ncartesian_to_joint failure : unreachable state ...\n");
                                }
                        }
                } while (!kbhit()) ;
                step = DISPLAY_LOCATION ;
                break ;
        }
        case FEATURE_TRACKING : {
                printf("\nEnter taxil sum value [0,4800] > ") ;
                scanf("%d",&last_taxil_sum) ;
                donentation.i = 0 ;
                donentation.j = 0 ;
```

```
                                        dorientation.k = 0 ;
                                        do {
                                                lord_scan(&lord_sensor) ;
                                                center_of_mass(lord_sensor,&row_center_of_mass,&col_center_of_mass,&taxil_sum) ;
                                                delta_taxil_sum = taxil_sum - last_taxil_sum ;
                                                if ((delta_taxil_sum > 50)||(delta_taxil_sum < -50)) {
                                                        delta_normal_direction = - (delta_taxil_sum * CM_PER_DELTA_TAXIL_SUM) ;
                                                }
                                                row_epsilon = (8 - row_center_of_mass) ;
                                                col_epsilon = (4 - col_center_of_mass) ;
                                                calculate_arm_length_axis(&currentarmstate,&arm_length_axis) ;
                                                calculate_arm_width_axis(&currentarmstate,&arm_width_axis) ;
                                                dcspace[0] = orientation.i*delta_normal_direction +
row_epsilon*arm_length_axis.i*DISTANCE_BETWEEN_ROWS + col_epsilon*arm_width_axis.i*DISTANCE_BETWEEN_COLUMNS ;
                                                dcspace[1] = orientation.j*delta_normal_direction +
row_epsilon*arm_length_axis.j*DISTANCE_BETWEEN_ROWS + col_epsilon*arm_width_axis.j*DISTANCE_BETWEEN_COLUMNS ;
                                                dcspace[2] = orientation.k*delta_normal_direction +
row_epsilon*arm_length_axis.k*DISTANCE_BETWEEN_ROWS + col_epsilon*arm_width_axis.k*DISTANCE_BETWEEN_COLUMNS ;
                                                if (!rhino_move_delta(dcspace, dorientation, cspace, &orientation, &currentarmstate)) {
                                                        printf("\ncartesian_to_joint failure : unreachable state ...\n");
                                                }
                                        } while (!kbhit()) ;
                                        step = DISPLAY_LOCATION ;
                                        break ;
                                }
                                case DRIVE_THE_CAR : {
                                        dummy_dcspace[0] = 0 ;
                                        dummy_dcspace[1] = 0 ;
                                        dummy_dcspace[2] = 0 ;
                                        printf("\nEnter taxil sum value [0,4800] > ") ;
                                        scanf("%d",&last_taxil_sum) ;
                                        printf("\nEnter lateral distance per step > ") ;
                                        scanf("%f",&lateral_distance) ;
                                        do {
                                                lord_scan(&lord_sensor) ;
                                                center_of_mass(lord_sensor,&row_center_of_mass,&col_center_of_mass,&taxil_sum) ;
                                                delta_taxil_sum = taxil_sum - last_taxil_sum ;
                                                delta_normal_direction = - (delta_taxil_sum * CM_PER_DELTA_TAXIL_SUM) ;
                                                calculate_arm_length_axis(&currentarmstate,&arm_length_axis) ;
                                                dcspace[0] = (orientation.i * delta_normal_direction) + (arm_length_axis.i * lateral_distance) ;
                                                dcspace[1] = (orientation.j * delta_normal_direction) + (arm_length_axis.j * lateral_distance) ;
                                                dcspace[2] = (orientation.k * delta_normal_direction) + (arm_length_axis.k *
lateral_distance) ;
                                                row_epsilon = (8 - row_center_of_mass) * EPSILON_PER_ROW_CENTER_OF_MASS ;

rotate_around_arm_width_axis(row_epsilon,&currentarmstate,orientation,cspace,cspace,&dorientation,dummy_dcspace) ;
                                                if (!rhino_move_delta(dcspace, dorientation, cspace, &orientation, &currentarmstate)) {
                                                        printf("\ncartesian_to_joint failure : unreachable state ...\n");
                                                }
                                        } while (!kbhit()) ;
                                        step = DISPLAY_LOCATION ;
                                        break ;
                                }
                                case CHOOSE : {
                                        printf("\n\nChoose one of...") ;
                                        printf("\n\t(R)econfigure") ;
                                        printf("\n\t(I)ncrement") ;
                                        printf("\n\t(V)ector") ;
                                        printf("\n\t(C)alibrate") ;
                                        printf("\n\t(P)lane Tracking") ;
                                        printf("\n\t(F)eature Tracking") ;
                                        printf("\n\t(D)rive the car") ;
                                        printf("\n\tFunction (S)tats") ;
                                        printf("\n\tR(O)tate") ;
                                        printf("\n\t(T)est Time") ;
                                        printf("\n\te(X)it") ;
                                        printf("\n\t\t\t\t> ") ;
                                        resp = getche() ;
                                        if ((resp=='R')||(resp=='r')) step = RECONFIGURE ;
                                        else if ((resp=='X')||(resp=='x')) step = EXIT ;
                                        else if ((resp=='I')||(resp=='I')) step = INCREMENT ;
                                        else if ((resp=='V')||(resp=='V')) step = VECTOR ;
                                        else if ((resp=='c')||(resp=='C')) step = CALIBRATE ;
                                        else if ((resp=='p')||(resp=='P')) step = PLANE_TRACKING ;
                                        else if ((resp=='f')||(resp=='F')) step = FEATURE_TRACKING ;
                                        else if ((resp=='d')||(resp=='D')) step = DRIVE_THE_CAR ;
                                        else if ((resp=='o')||(resp=='O')) step = ROTATE ;
                                        else if ((resp=='s')||(resp=='S')) step = FUNCTION_STATISTICS ;
```

```
                else if ((resp=='t')||(resp=='T')) step = TEST_TIME ;
                else step = CHOOSE ;
                break ;
        }
        case RECONFIGURE : {
                printf("\n\n\nEnter desired coordinates ...");
                printf("\nX\tY\tZ\t\tJ\tK\n") ;
                scanf("%f %f %f %f %f %f",
                        cspace, cspace+1, cspace+2,
                        (&orientation.i),(&orientation.j),(&orientation.k)) ;
                normalize_vectortype(orientation, &orientation) ;
                if (rhino_move_pt_to_pt(cspace, &orientation, &currentarmstate)) {
                        printf ("\n\rhino_move_pt_to_pt successful") ;
                }
                else
                        printf("\ncartesian_to_joint failure : unreachable state ...\n");
                step = DISPLAY_LOCATION ;
                break ;
        }
        case INCREMENT : {
                printf("\n\n\nEnter desired incremental move ...");
                printf("\nX\tY\tZ\t\tJ\tK\n") ;
                scanf("%f %f %f %f %f %f",
                        dcspace, dcspace+1, dcspace+2, &orientation.i, &orientation.j, &orientation.k );
                if (rhino_move_delta(dcspace, orientation, cspace, &orientation, &currentarmstate))
                        printf ("\n\rhino_move_increment successful") ;
                else
                        printf("\ncartesian_to_joint failure : unreachable state ...\n");
                step = DISPLAY_LOCATION ;
                break ;
        }
        case VECTOR : {
                printf("\n\n\nEnter incremental vector move ...");
                printf("\nX\tY\tZ\t\tJ\tK\n") ;
                scanf("%f %f %f %f %f %f",
                        dcspace, dcspace+1, dcspace+2, &orientation.i, &orientation.j, &orientation.k );
                printf("\n\nEnter number of steps > ") ;
                scanf("%d",&step) ;
                for (index = 0; index < step; index++) {
                        if (!rhino_move_delta(dcspace, orientation, cspace, &orientation, &currentarmstate)) {
                                printf("\ncartesian_to_joint failure : unreachable state ...\n");
                                printf("\nstep = %d",index) ;
                                break ;
                        }
                }
                step = DISPLAY_LOCATION ;
                break ;
        }
        case ROTATE : {
                printf("\nEnter angle for rotation around width axis > ") ;
                scanf("%f",&row_epsilon) ;

rotate_around_arm_width_axis(row_epsilon,&currentarmstate,orientation,cspace,cspace,&orientation,dcspace) ;
                if (!rhino_move_delta(dcspace, orientation, cspace, &orientation, &currentarmstate)) {
                        printf("\ncartesian_to_joint failure : unreachable state ...\n");
                }
                step = DISPLAY_LOCATION ;
                break ;
        }
        case FUNCTION_STATISTICS : {
                printf("\nEnter filename for output > ") ;
                scanf("%s",filename) ;
                write_stream = fopen(filename,"w+t") ;
                if (write_stream == NULL) {
                        printf("\nUnable to open file") ;
                }
                else {
                        printf("\n\n(I)ncrement or (R)otation ? > ") ;
                        resp = getche() ;
                        if ((resp=='i')||(resp=='I')) {
                                printf("\n\n\nEnter desired incremental move ...");
                                printf("\nX\tY\tZ\t\tJ\tK\n") ;
                                scanf("%f %f %f %f %f %f",
                                        dcspace, dcspace+1, dcspace+2, &orientation.i, &orientation.j,
&orientation.k );
                                printf("\n\nEnter number of samples > ") ;
                                scanf("%d",&number_of_iterations) ;
                                for (iteration = 0; iteration < number_of_iterations; iteration++) {
```

```
&currentarmstate))                                              if (rhino_move_delta(dcspace, dorientation, cspace, &orientation,

                                                                        printf ("\nrhino_move_increment successful") ;
                                                                else
                                                                        printf("\ncartesian_to_joint failure : unreachable state

...\n");

                                                                printf("\ncartesian coordinates : X=%3.2f, Y=%3.2f,

Z=%3.2f\n",cspace[0],cspace[1],cspace[2]);

                                                                printf("\norientation :\tI = %3.2f\t\tJ = %3.2f\t\tK = %3.2f",orientation.i,

orientation.j, orientation.k) ;

                                                                printf("\n\nHold steady while calibrating stats for maximum threshold

...");

                                                                lord_calibrate(&lord_threshold_stats) ;

save_lord_stats(cspace,orientation,lord_threshold_stats,write_stream) ;

                                                                if (kbhit()) break ;
                                                        }
                                                }
                                                else {
                                                        printf("\nEnter angle for rotation around width axis > ") ;
                                                        scanf("%f",&row_epsilon) ;
                                                        printf("\n\nEnter number of samples > ") ;
                                                        scanf("%d",&number_of_iterations) ;
                                                        for (iteration = 0; iteration < number_of_iterations; iteration++) {

rotate_around_arm_width_axis(row_epsilon,&currentarmstate,orientation,cspace,cspace,&orientation,dcspace) ;
                                                                if (rhino_move_delta(dcspace, dorientation, cspace, &orientation,
&currentarmstate))
                                                                        printf ("\nrhino_move_increment successful") ;
                                                                else
                                                                        printf("\ncartesian_to_joint failure : unreachable state

...\n");

                                                                printf("\ncartesian coordinates : X=%3.2f, Y=%3.2f,

Z=%3.2f\n",cspace[0],cspace[1],cspace[2]);

                                                                printf("\norientation :\tI = %3.2f\t\tJ = %3.2f\t\tK = %3.2f",orientation.i,

orientation.j, orientation.k) ;

                                                                printf("\n\nHold steady while calibrating stats for maximum threshold

...");

                                                                lord_calibrate(&lord_threshold_stats) ;

save_lord_stats(cspace,orientation,lord_threshold_stats,write_stream) ;

                                                                if (kbhit()) break ;
                                                        }
                                                }
                                                fclose(write_stream) ;
                                        }
                                        step = DISPLAY_LOCATION ;
                                        break ;
                                }
                                case TEST_TIME : {
                                        do {
                                                start_time = clock() ;
                                                lord_scan(&lord_sensor) ;
                                                end_time = clock() ;
                                                cycle_time = difftime(start_time, end_time) ;
                                                printf("\n\ncycle time = %e, start_time = %l, end_time = %l",cycle_time, start_time, end_time) ;
                                        } while (!kbhit()) ;
                                        step = CHOOSE ;
                                        break ;
                                }
                                case DISPLAY_LOCATION : {
                                        joint_to_cartesian(currentarmstate,cspace,&orientation) ;
                                        printf("\n\nlocation :\tX\tY\tZ\t\tI\t\tJ\t\tK") ;
                                        printf("\n\t\t%3.2f\t%3.2f\t%3.2f\t%3.2f\t%3.2f\t%3.2f",
                                                cspace[0],cspace[1],cspace[2],orientation.i, orientation.j, orientation.k) ;
                                        calculate_arm_length_axis(&currentarmstate,&arm_length_axis) ;
                                        calculate_arm_width_axis(&currentarmstate,&arm_width_axis) ;
                                        printf("\n\nlength axis :\t%3.2f\t%3.2f\t%3.2f",arm_length_axis.i,arm_length_axis.j,arm_length_axis.k) ;
                                        printf("\n\nwidth axis :\t%3.2f\t%3.2f\t%3.2f",arm_width_axis.i,arm_width_axis.j,arm_width_axis.k) ;
                                        step = CHOOSE ;
                                        break ;
                                }
                                case EXIT : {
                                        printf("\nHoming RHINO ...") ;
                                        nextarmstate.angle.a = 0 ;
                                        nextarmstate.angle.b = 0;
                                        nextarmstate.angle.c = -90 * M_PI_2/90 ;
                                        nextarmstate.angle.d = 0 ;
```

```
                                nextarmstate.angle.e = -90 * M_PI_2/90 ;
                                nextarmstate.angle.f = 0 ;
                                nextarmstate.step.a = 0 ;
                                nextarmstate.step.b = (int) (nextarmstate.angle.b * BSTEPSPERRAD) ;
                                nextarmstate.step.c = (int) (nextarmstate.angle.c * CSTEPSPERRAD) ;
                                nextarmstate.step.d = (int) (nextarmstate.angle.d * DSTEPSPERRAD) ;
                                nextarmstate.step.e = (int) (nextarmstate.angle.e * ESTEPSPERRAD) ;
                                nextarmstate.step.f = (int) (nextarmstate.angle.f * FSTEPSPERRAD) ;
                                orientation.i = -1 ;
                                orientation.j = 0 ;
                                orientation.k = -1 ;
                                normalize_vectortype(orientation, &orientation) ;
                                joint_to_cartesian(nextarmstate,cspace,&orientation) ;
                                if (rhino_move_pt_to_pt(cspace, &orientation, &currentarmstate))
                                        printf ("rhino_move_pt_to_pt successful") ;
                                else
                                        printf("\ncartesian_to_joint failure : unreachable state ...\n");
                                step = BYE ;
                                break ;
                        }
                        default : {
                                step =CHOOSE ;
                                break ;
                        }
                }
        } while (step != BYE) ;
        printf("\nExiting program ROBOSENS") ;
        exit(0) ;
}
```

```
/*****************************************************************
/*                                                               */
/*          title :     GRAPHADD.C                               */
/*          author : M.A. Greenspan                              */
/*          date :      30 August 89                             */
/*                                                               */
/*****************************************************************/

/*****************************************************************
/*
/*          Module Description : Contains a routine for the registration
/*                               of the graphics drivers and the initialization
/*                               the graphics facilities.
/*
/*          Routines :  graphics_initialize
/*
/*          Header :    graphadd.h
/*
/*****************************************************************/

#include <graphics.h>
#include "c:\tc201\programs\lordtest\include\graphadd.h"

/*****************************************************************
/*
/*          title : void graphics_initialize(int driver, int mode)
/*
/*          input :                 int driver :  graphics driver (e.g. CGA, EGAVGA)
/*                                  int mode :    graphics mode (e.g. EGA_LO)
/*
/*          output :    none
/*
/*          return value :          none
/*
/*          subroutines :           registerbgifont, registerbgidriver, initgraph,
/*                                  graphresult, grapherrormsg
/*
/*          description :           registers the graphics driver and initializes
/*                                  the graphics mode. If the graphics driver or
/*                                  mode are unattainable, an error msg is reported
/*                                  and the program is aborted.
/*
/*****************************************************************/

void graphics_initialize(int driver, int mode) {

  int g_driver, g_mode, g_error ;
  char g_path ;
  char fieldcolor[15] ;

/******************** REGISTER FONT DRIVERS **********************/

  if (registerbgifont(triplex_font) < 0) {
    printf("\nError in registerbgifont(triplex_font)");
    exit(0) ;
    }
  if (registerbgifont(small_font) < 0) {
    printf("\nError in registerbgifont(small_font)");
    exit(0) ;
    }
  if (registerbgifont(sansserif_font) < 0) {
    printf("\nError in registerbgifont(sansserif_font)");
    exit(0) ;
    }
  if (registerbgifont(gothic_font) < 0) {
    printf("\nError in registerbgifont(gothic_font)");
    exit(0) ;
    }

/********* REGISTER GRAPHICS DRIVERS ***************************/

  if (registerbgidriver(EGAVGA_driver) < 0) {
            printf("\nError in registerbgidriver(EGAVGA_driver)");
            exit(0) ;
            }

/********** INITIALIZE GRAPHICS MODE ***************************/
```

```
        initgraph(&driver,&mode,&g_path) ;
        g_error = graphresult() ;
        if (g_error < 0) {
                printf("initgraph error : %s.\n",grapherrormsg(g_error)) ;
                exit(0) ;
        }
}
```

```
/*****************************************************************/
/*                                                               */
/*      program : LORD.C                                         */
/*      author : M.A. GREENSPAN                                  */
/*      date :   30 August 89                                    */
/*                                                               */
/*****************************************************************/

/*****************************************************************
/*
/*      Module Description :  contains a set of routines for the
/*                            interface to the LORD LTS210 sensor.
/*
/*      routines :  lord_reset, lord_scan, lord_scan_site,
/*                            lord_scan_ft
/*
/*      header :    lord.h
/*
/*****************************************************************/

#include <stdlib.h>
#include <conio.h>
#include "c:\tc201\programs\robosens\include\rs232com.h"
#include "c:\tc201\programs\robosens\include\portcom.h"
#include "c:\tc201\programs\robosens\include\lord.h"

/*****************************************************************
/*
/*      title :                 void lord_reset(void)
/*
/*      input :                 none
/*
/*      output :    none
/*
/*      return_value :          none
/*
/*      subroutines :           port_send, port_read
/*
/*      description :  sends the proper sequence of characters to the
/*                            LORD sensor and monitors response. Correct reset
/*        is assured by the receiving of a 'L' and linefeed
/*                            from the sensor. If the reset is unsuccessful, a
/*                            message will be sent from the port_send or port_read
/*                            routines.
/*
/*****************************************************************/

void lord_reset(void) {

        char c ;
        char msg ;

        printf("\n") ;
        c=ESC;
        port_send(LORDPORT,c);
        c='c';
        port_send(LORDPORT,c);
        do {
                msg = port_read(LORDPORT);
/**             printf("%c",msg) ; **/
        } while (msg != '\n') ;
}
```

```
/*****************************************************************
/*
/*        title :          void lord_scan(padtype far *pad)
/*
/*        input :                   none
/*
/*        output :    padtype far *pad : scanned tactile image
/*
/*        return_value :             none
/*
/*        subroutines :              port_send, port_read
/*
/*        description :              scans the current value of the sensor array.
/*
/*****************************************************************/

void lord_scan(padtype far *pad) {

        char c ;
        int row,column,column_index ;
        float sigma ;

        for (row=0;row<ROWMAX;row++) {
                for (column=0;column<COLUMNMAX;column++) {
                        (*pad).taxil[row][column].last_value =
                                (*pad).taxil[row][column].value ;
                }
        }
    c=ESC;
        port_send(LORDPORT,c);
        c='[';
        port_send(LORDPORT,c);
        c='?';
        port_send(LORDPORT,c);
    c='1';
        port_send(LORDPORT,c);
        c='L';
        port_send(LORDPORT,c);
        for (row=0;row<ROWMAX;row++) {
                for (column=0;column<COLUMNMAX;column++) {
                        /**
                        ***             for some reason, the columns are
                        ***             not scanned in order
                        **/
                        if (column==0) column_index = 9 ;
                        else column_index = column - 1 ;
                        (*pad).taxil[row][column_index].value = port_read(LORDPORT) ;
                }
        }
}

/*****************************************************************
/*
/*        title :                   int lord_scan_site(int x, int y)
/*
/*        input :                   int x :       row of taxel to be scanned
/*
/*                                  int y : column of taxel to be scanned
/*
/*        output :    none
/*
/*        return_value :            value of scanned taxel (from 0 to 30)
/*
/*        subroutines :             port_send, port_read
/*
/*        description :             scans element (x,y) of tactile array.
/*
/*****************************************************************/

int lord_scan_site(int x, int y) {

        char c ;
        int result ;

    c=ESC;
        port_send(LORDPORT,c);
        c='[';
        port_send(LORDPORT,c);
```

```
        c='?';
        port_send(LORDPORT,c);
    c='4';
        port_send(LORDPORT,c);
        c=';';
        port_send(LORDPORT,c);
        c= (char) x;
        port_send(LORDPORT,c);
        c=';';
        port_send(LORDPORT,c);
        c= (char) y;
        port_send(LORDPORT,c);
        c='L';
        port_send(LORDPORT,c);
        result = port_read(LORDPORT) ;
        return(result) ;
}

/*****************************************************************
/*
/*      title :                 void lord_scan_ft(int far *forcetorque)
/*
/*      input :                 none
/*
/*      output :    int far *forcetorque : array of integers
/*
/*      return_value :          none
/*
/*      subroutines :   port_send, port_read
/*
/*      description :           scans force/torque values of the sensor and
/*                              stores them in array.
/*
/*****************************************************************/

void lord_scan_load(int forcetorque[12]) {

        unsigned char c ;
        int index;

    c=ESC;
        port_send(LORDPORT,c);
        c='[';
        port_send(LORDPORT,c);
        c='?';
        port_send(LORDPORT,c);
    c='2';
        port_send(LORDPORT,c);
        c='L';
        port_send(LORDPORT,c);
        for (index = 0; index < 12; index++) {
                forcetorque[index] = port_read(LORDPORT);
        }

}

void lord_zero(void) {

        unsigned char c ;

    c=ESC;
        port_send(LORDPORT,c);
        c='[';
        port_send(LORDPORT,c);
        c='?';
        port_send(LORDPORT,c);
        c='0';
        port_send(LORDPORT,c);
        c='L';
        port_send(LORDPORT,c);
}

int lord_contact(padtype pad) {

        int row, col ;
        int status ;

        status = 0 ;
```

```
        for (row=0; row<ROWMAX; row++) {
                for (col=0; col<COLUMNMAX; col++) {
                        if (pad.taxil[row][col].value>NOISE_THRESHOLD) {
                                status = 1 ;
                                break ;
                        }
                }
                if (status) break ;
        }
        return(status) ;
}

void lord_stats(padtype *pad, pad_stats_type *stats) {

        int row, col ;
        int p ;
        float average ;
        int max ;
        int oncount, maxcount ;

/*      printf("\nEntered lord_stats ...") ; */
        average = 0 ;
        max = 0 ;
        oncount = 0 ;
        maxcount = 0 ;
        for (row=0; row<ROWMAX; row++) {
                for (col=0; col<COLUMNMAX; col++) {
                        p = (*pad).taxil[row][col].value ;
                        average = average + p ;
                        if (p > max) max = p ;
                        if (p > NOISE_THRESHOLD) oncount++ ;
                        if (p > MAX_THRESHOLD) maxcount++ ;
                        }
        }
        average = average/160 ;
        (*stats).average = average ;
        (*stats).max = max ;
        (*stats).oncount = oncount ;
        (*stats).maxcount = maxcount ;
/*      printf("\nExiting pad_stats ...") ; */
}

void lord_or(padtype *pad1, padtype *pad2, padtype *andpad) {

        int row, col ;

        for (row=0; row<ROWMAX; row++) {
                for (col=0; col<COLUMNMAX; col++) {
                        if ((*pad1).taxil[row][col].value > (*pad2).taxil[row][col].value)
                                (*andpad).taxil[row][col].value = (*pad1).taxil[row][col].value ;
                        else (*andpad).taxil[row][col].value = (*pad2).taxil[row][col].value ;
                }
        }
}

void lord_zero_pad(padtype *pad) {

        int row, col ;

        for (row=0; row<ROWMAX; row++) {
                for (col=0; col<COLUMNMAX; col++) {
                        (*pad).taxil[row][col].value = 0 ;
                        (*pad).taxil[row][col].last_value = 0 ;
                }
        }
}

void lord_scan_bias(padtype *biaspad, padtype *pad) {

        int row, col ;

        lord_scan(pad) ;
        for (row = 0; row<ROWMAX; row++) {
                for (col=0; col<COLUMNMAX; col++) {
                        (*pad).taxil[row][col].value = (*pad).taxil[row][col].value - (*biaspad).taxil[row][col].value ;
                        if ((*pad).taxil[row][col].value < 0) (*pad).taxil[row][col].value = 0 ;
                }
        }
```

```
}

void lord_reset_bias(padtype *biaspad) {

        int row, col ;
        int status ;

        do {
                status = 1 ;
                lord_scan(biaspad) ;
                for (row = 0; row<ROWMAX; row++) {
                        for (col=0; col<COLUMNMAX; col++) {
                                if ((*biaspad).taxil[row][col].value > NOISE_THRESHOLD) {
                                        status = 0 ;
                                        break ;
                                }
                        }
                        if (!status) break ;
                }
        } while(!status) ;
}
```

```
/*****************************************************************/
/*                                                               */
/*      program : LORDCAL.C                                      */
/*      author : M.A. GREENSPAN                                  */
/*      date    30 August 89                                     */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "c:\tc201\programs\robosens\include\rs232com.h"
#include "c:\tc201\programs\robosens\include\portcom.h"
#include "c:\tc201\programs\robosens\include\lord.h"
#include "c:\tc201\programs\robosens\include\lordfilt.h"
#include "c:\tc201\programs\robosens\include\lordcal.h"

void lord_calibrate(lord_stats_type *lord_stats) {

        int index, row, col ;
        int centroid_row, centroid_col ;
        float centroid_row_sum, centroid_col_sum ;
        float center_of_mass_row_sum, center_of_mass_col_sum ;
        int center_of_mass_row, center_of_mass_col, center_of_mass_taxil_sum ;
        padtype pad[SIGNIFICANT_NUMBER] ;

        centroid_row = 0 ;
        centroid_col = 0 ;
        centroid_row_sum = 0 ;
        centroid_col_sum = 0 ;
        center_of_mass_row = 0 ;
        center_of_mass_col = 0 ;
        center_of_mass_row_sum = 0 ;
        center_of_mass_col_sum = 0 ;
        for (row = 0; row < ROWMAX; row++) {
                for (col = 0; col < COLUMNMAX; col++) {
                        (*lord_stats).taxil[row][col].min = 30 ;
                        (*lord_stats).taxil[row][col].average = 0 ;
                        (*lord_stats).taxil[row][col].max = 0 ;
                        (*lord_stats).taxil[row][col].sum = 0 ;
                        (*lord_stats).taxil[row][col].sum_of_squares = 0 ;
                }
        }
        for (index = 0; index < SIGNIFICANT_NUMBER; index++) {
                printf("\nindex = %d", index) ;
                lord_scan(&(pad[index])) ;
        }
        for (index = 0; index < SIGNIFICANT_NUMBER; index++) {
                centroid(pad[index],&centroid_row,&centroid_col) ;
                centroid_row_sum = centroid_row_sum + centroid_row ;
                centroid_col_sum = centroid_col_sum + centroid_col ;
                center_of_mass(pad[index],&center_of_mass_row,&center_of_mass_col,&center_of_mass_taxil_sum) ;
                center_of_mass_row_sum = center_of_mass_row_sum + center_of_mass_row ;
                center_of_mass_col_sum = center_of_mass_col_sum + center_of_mass_col ;
                for (row = 0; row < ROWMAX; row++) {
                        for (col = 0; col < COLUMNMAX; col++) {
                                (*lord_stats).taxil[row][col].sum = (*lord_stats).taxil[row][col].sum + pad[index].taxil[row][col].value ;
                                (*lord_stats).taxil[row][col].sum_of_squares = (*lord_stats).taxil[row][col].sum_of_squares +
(pad[index].taxil[row][col].value)*(pad[index].taxil[row][col].value) ;
                                if (pad[index].taxil[row][col].value < (*lord_stats).taxil[row][col].min) (*lord_stats).taxil[row][col].min =
pad[index].taxil[row][col].value ;
                                if (pad[index].taxil[row][col].value > (*lord_stats).taxil[row][col].max) (*lord_stats).taxil[row][col].max =
pad[index].taxil[row][col].value ;
                        }
                }
        }
        (*lord_stats).centroid_row_average = centroid_row_sum/SIGNIFICANT_NUMBER ;
        (*lord_stats).centroid_col_average = centroid_col_sum/SIGNIFICANT_NUMBER ;
        (*lord_stats).center_of_mass_row_average = center_of_mass_row_sum/SIGNIFICANT_NUMBER ;
        (*lord_stats).center_of_mass_col_average = center_of_mass_col_sum/SIGNIFICANT_NUMBER ;
        (*lord_stats).sum_of_averages = 0 ;
        for (row = 0; row < ROWMAX; row++) {
                for (col = 0; col < COLUMNMAX; col++) {
                        (*lord_stats).taxil[row][col].average = (float) (*lord_stats).taxil[row][col].sum/SIGNIFICANT_NUMBER ;
                        (*lord_stats).sum_of_averages = (*lord_stats).sum_of_averages + (*lord_stats).taxil[row][col].average ;
                        (*lord_stats).taxil[row][col].stdev = (float)
sqrt((1/SIGNIFICANT_NUMBER)*((*lord_stats).taxil[row][col].sum_of_squares - (SIGNIFICANT_NUMBER*(*lord_stats).taxil[row][col].average))) ;
                }
```

```
/*****************************************************************/
/*                                                               */
/*      Title : LORDDISP.C                                       */
/*      Author : M.A. Greenspan                                  */
/*      Date : 5-FEB-90                                          */
/*      Updated : 14-April-90 (added comments)                   */
/*                                                               */
/*      This source file contains graphic routines which display a 2-D*/
/*      representation of the sensor pad on the screen and show the*/
/*      local edge segment that has been identified. The display  */
/*      outlines the two endpoints and draws the fitted line between*/
/*      them. The header file associated with this source file is */
/*      LORDDISP.H.                                              */
/*                                                               */
/*****************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <graphics.h>
#include "c:\tc201\programs\lordtest\include\lord.h"
#include "c:\tc201\programs\lordtest\include\graphadd.h"
#include "c:\tc201\programs\lordtest\include\lorddisp.h"

void box(int left, int top, int right, int bottom, int bordercolor, int fillcolor) {

        int currentcolor ;

        currentcolor = getcolor() ;
        setcolor(bordercolor) ;
        rectangle(left, top, right, bottom) ;
        setcolor(currentcolor) ;
        setfillstyle(FILLPATTERN, fillcolor) ;
        bar(left+1, top+1, right-1, bottom-1) ;
}

void pad_display_2d(padtype *pad, int threshold, int xoffset, int yoffset) {

        int row, column ;
        int bordercolor, fillcolor ;
        int left, top, right, bottom ;
        int x, y ;

        bordercolor = TAXEL_BORDER_COLOR ;
        for (row=0; row<ROWMAX; row++) {
                for (column=0; column<COLUMNMAX; column++) {
                        if ((*pad).taxil[row][column].value>=threshold) fillcolor = TAXEL_ON_COLOR ;
                        else fillcolor = TAXEL_OFF_COLOR ;
                        calculate_taxel_border(row, column, xoffset, yoffset, &left, &top, &right, &bottom) ;
                        box(left, top, right, bottom, bordercolor, fillcolor) ;
                        calculate_taxel_center(row, column, xoffset, yoffset, &x, &y) ;
                        putpixel(x, y, EGA_YELLOW) ;
                }
        }
}

void calculate_taxel_border(int row, int column, int xoffset, int yoffset, int *left, int *top, int *right, int *bottom) {

        *left = row*XDELTA + xoffset ;
        *top = yoffset - YDELTA*(column+1) ;
        *right =(row+1)*XDELTA + xoffset ;
        *bottom = yoffset - YDELTA*column ;

}

void calculate_taxel_center(int row, int column, int xoffset, int yoffset, int *x, int *y) {

        int left, top, right, bottom ;

        calculate_taxel_border(row, column, xoffset, yoffset, &left, &top, &right, &bottom) ;
        *x = (int) ((right+left)/2 + 0.5) ;
        *y = (int) ((top+bottom)/2 + 0.5) ;

}

void draw_edge(int row1, int column1, int row2, int column2, int xoffset, int yoffset) {

        int left, top, right, bottom ;
```

```
        int x1, y1, x2, y2 ;
        int savecolor ;

        calculate_taxel_center(row1, column1, xoffset, yoffset, &x1, &y1) ;
        calculate_taxel_center(row2, column2, xoffset, yoffset, &x2, &y2) ;
        savecolor = getcolor() ;
        setcolor(LINE_COLOR) ;
        line(x1, y1, x2, y2) ;
        setcolor(savecolor) ;
}


/*
main() {
        int row, column ;

        padtype pad ;

        printf("\nEntered program LORDDISP") ;
        printf("\nCalling graphics_initialize ...") ;
        graphics_initialize(VGA,VGAMED) ;
        printf("\nResetting pad ...") ;
        for (row=0; row<ROWMAX; row++) {
                for (column=0; column<COLUMNMAX; column++) {
                        pad.taxil[row][column].value = 0 ;
                }
        }
        pad_display_2d(&pad, TAXEL_ON_THRESHOLD, 100, 100) ;
        pad.taxil[0][0].value = TAXEL_ON_THRESHOLD ;
        pad.taxil[1][1].value = TAXEL_ON_THRESHOLD ;
        pad.taxil[2][2].value = TAXEL_ON_THRESHOLD ;
        pad.taxil[3][3].value = TAXEL_ON_THRESHOLD ;
        pad.taxil[4][4].value = TAXEL_ON_THRESHOLD ;
        pad.taxil[5][5].value = TAXEL_ON_THRESHOLD ;
        pad.taxil[6][6].value = TAXEL_ON_THRESHOLD ;
        pad.taxil[7][7].value = TAXEL_ON_THRESHOLD ;
        getch() ;
        pad_display_2d(&pad, TAXEL_ON_THRESHOLD, 100, 100) ;
        getch() ;
        printf("\nCalling closegraph ...") ;
        closegraph() ;
        printf("\nExiting programs LORDDISP") ;
        exit(0) ;

}
*/
```

```
/*****************************************************************
/*                                                               */
/*        program : LORDFILTC                                    */
/*        author : M.A. GREENSPAN                                */
/*        date :     30 August 89                                */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "c:\tc201\programs\robosens\include\rs232com.h"
#include "c:\tc201\programs\robosens\include\portcom.h"
#include "c:\tc201\programs\robosens\include\veltype.h"
#include "c:\tc201\programs\robosens\include\lord.h"
#include "c:\tc201\programs\robosens\include\lordfilt.h"

void centroid(padtype pad, int *row, int *col) {

        int row_index, col_index ;
        int row_sum, col_sum ;
        int on_taxils ;

        row_sum = 0 ;
        col_sum = 0 ;
        on_taxils = 0 ;
        for (row_index = 0; row_index < ROWMAX; row_index++) {
                for (col_index = 0; col_index < COLUMNMAX; col_index++) {
                        if (pad.taxil[row_index][col_index].value > NOISE_THRESHOLD) {
                                row_sum = row_sum + row_index ;
                                col_sum = col_sum + col_index ;
                                on_taxils++ ;
                        }
                }
        }
        if (row_sum == 0) *row = 8 ;
        else *row = (int) (row_sum/on_taxils) ;
        if (col_sum == 0) *col = 5 ;
        else *col = (int) (col_sum/on_taxils) ;
}

void center_of_mass(padtype pad, int *row, int *col, int *sum_of_taxils) {

        int row_index, col_index ;
        int row_sum, col_sum ;
        int taxil_sum ;

        row_sum = 0 ;
        col_sum = 0 ;
        taxil_sum = 0 ;
        for (row_index = 0; row_index < ROWMAX; row_index++) {
                for (col_index = 0; col_index < COLUMNMAX; col_index++) {
                        if (pad.taxil[row_index][col_index].value > NOISE_THRESHOLD) {
                                row_sum = row_sum + row_index*pad.taxil[row_index][col_index].value ;
                                col_sum = col_sum + col_index*pad.taxil[row_index][col_index].value ;
                                taxil_sum = taxil_sum + pad.taxil[row_index][col_index].value ;
                        }
                }
        }
        if (row_sum == 0) *row = 8 ;
        else *row = (int) (row_sum/taxil_sum) ;
        if (col_sum == 0) *col = 5 ;
        else *col = (int) (col_sum/taxil_sum) ;
        *sum_of_taxils = taxil_sum ;
}
```

```
/*****************************************************************/
/*                                                               */
/*      title :     LORDSCRN.C                                   */
/*      author : M.A. Greenspan                                  */
/*      date :      30-Aug-89                                    */
/*                                                               */
/*****************************************************************/

/*****************************************************************
/*
/*      Module Description :   Contains a set of routines for the
/*                             graphical display of the tactile data.
/*
/*      routines :     pad_initialize, general_pad_initialize,
/*                             pad_refresh, pad_refresh_vertical
/*
/*      header :    lordscrn.h
/*
*****************************************************************/

#include <graphics.h>
#include "c:\tc201\programs\vobosens\include\menu.h"
#include "c:\tc201\programs\vobosens\include\lord.h"
#include "c:\tc201\programs\vobosens\include\graphadd.h"
#include "c:\tc201\programs\vobosens\include\lordscrn.h"

/*****************************************************************
/*
/*      title :         void pad_initialize(padtype far *pad, int x_offset)
/*
/*      input :                 int x_offset : horizontal offset of graphical
/*                                             display of *pad
/*
/*      output :    padtype far *pad : initialized tactile array
/*                                             data type
/*
/*      return value :          none
/*
/*      subroutines :           putpixel
/*
/*      description :           initializes mapping information of tactile array
/*                              to screen. The displayed array is vertial with the
/*                              taxel values extending to the right in the horizontal
/*                              direction. Each taxel value on the data type padtype
/*                              has a field x and a field y which locates the 0-level
/*                              output point of that taxel on the screen.
/*
*****************************************************************/

void pad_initialize(padtype far *pad, int x_offset) {

        int row, column ;
        int pixelcolor ;

        pixelcolor = EGA_LIGHTGREEN ;
        for (row = 0; row < ROWMAX; row++) {
                for (column = 0; column < COLUMNMAX; column++) {
                        (*pad).taxil[row][column].x = F2_LEFT + 20 + 10 * column ;
                        (*pad).taxil[row][column].y = F2_TOP + 10 + 5 * row + 2*(10-column) ;
                        (*pad).taxil[row][column].last_value = 0 ;
                        (*pad).taxil[row][column].value = 0 ;
                        putpixel((*pad).taxil[row][column].x + x_offset,
                                (*pad).taxil[row][column].y,
                                pixelcolor) ;
                }
        }
}

/*****************************************************************
/*
/*      title :         void general_pad_initialize(padtype far *pad,
/*                                      int xoffset, int yoffset,
/*                                      int deltax, int deltay,
/*                                      int slope, int direction)
/*
/*      input :                 int xoffset :  horizontal offset of graphical display
/*
/*                              int yoffset :  vertical offset of graphical display
```

```
/*
/*                              int deltax :   horizontal spacing between taxels in
/*                                             graphical display
/*
/*                              int deltay :   vertical spacing between taxels in
/*                                             graphical display
/*
/*                              int slope :    offset between successive rows/columns
/*                                             in graphical display
/*
/*                              int direction :        orientation of graphical display, i.e.
/*                                                     VERTICAL or HORIZONTAL
/*
/*    output :          padtype far *pad :     initialized tactile array
/*
/*          return value :          none
/*
/*          subroutines :           putpixel
/*
/*          description :           initializes tactile array data type for graphical
/*                                  display in either the horizontal or vertical
/*                                  directions.
/*
/*****************************************************************/

void general_pad_initialize(padtype far *pad, int xoffset, int yoffset, int deltax, int deltay, int slope, int direction) {

        int row, column ;
        int pixelcolor ;

        pixelcolor = EGA_LIGHTGREEN ;
        for (row = 0; row < ROWMAX; row++) {
                for (column = 0; column < COLUMNMAX; column++) {
                        if (direction == VERTICAL) {
                                (*pad).taxil[row][column].x = deltax * column + xoffset ;
                                (*pad).taxil[row][column].y = deltay * row + slope*(deltay*column) + yoffset ;
                        }
                        else {
                                (*pad).taxil[row][column].x = deltax * row + slope*(deltax*column) + xoffset ;
                                (*pad).taxil[row][column].y = yoffset - deltay * column ;
                        }
                        (*pad).taxil[row][column].last_value = 0 ;
                        (*pad).taxil[row][column].value = 0 ;
                        putpixel((*pad).taxil[row][column].x,
                                (*pad).taxil[row][column].y,
                                pixelcolor) ;
                }
        }
}

/*****************************************************************
/*
/*          title :                 void pad_refresh(padtype far *pad, int x_offset)
/*
/*          input :                 padtype far *pad : tactile array to be displayed
/*
/*                                  int xoffset : horizontal offset of displayed array
/*
/*          output :    none
/*
/*          return value :          none
/*
/*          subroutines :  line, putpixel
/*
/*          description :  refreshes graphical representation of tactile
/*                                  array in the horizontal direction.
/*
/*****************************************************************/

void pad_refresh(padtype far *pad, int x_offset) {

        int row, column ;
        int pixel ;
        int endpixel ;
        int linecolor, endcolor, zerocolor, blankcolor, initcolor;

        linecolor = EGA_LIGHTRED ;
        endcolor = EGA_WHITE ;
```

```
        zerocolor = EGA_LIGHTGREEN ;
        blankcolor = EGA_BLACK ;
        initcolor = getcolor() ;

        for (row = 0; row < ROWMAX; row++) {
                for (column = 0, column < COLUMNMAX; column++) {
                if ((*pad).taxil[row][column].last_value > (*pad).taxil[row][column].value)
                        endpixel = (*pad).taxil[row][column].last_value ;
                else endpixel = (*pad).taxil[row][column].value ;
                setcolor(blankcolor) ;
                line((*pad).taxil[row][column].x + x_offset + 1,
                        (*pad).taxil[row][column].y,
                        (*pad).taxil[row][column].x + x_offset + endpixel,
                        (*pad).taxil[row][column].y) ;
                if ((*pad).taxil[row][column].y == 0) {
                        putpixel((*pad).taxil[row][column].x + x_offset,
                                (*pad).taxil[row][column].y,
                                zerocolor) ;
                        }
                else {
                        setcolor(linecolor) ;
                        line((*pad).taxil[row][column].x + x_offset,
                                (*pad).taxil[row][column].y,
                                (*pad).taxil[row][column].x + x_offset + (*pad).taxil[row][column].value - 1,
                                (*pad).taxil[row][column].y) ;
                        putpixel((*pad).taxil[row][column].x + x_offset + (*pad).taxil[row][column].value,
                                (*pad).taxil[row][column].y,
                                endcolor) ;
                }
                (*pad).taxil[row][column].last_value = (*pad).taxil[row][column].value ;
                }
        }

        setcolor(initcolor) ;

}
/*************************************************************************
/*
/*      title :         void pad_refresh_vertical(padtype far *pad,
/*                                      int xoffset, int yoffset, float scale)
/*
/*      input :                 padtype far *pad : tactile image to be displayed
/*
/*                              int xoffset : offset for display in x direction
/*
/*                              int yoffset : offset for display in y direction
/*
/*                              float scale : multiplicative factor for display
/*                                              of image
/*
/*      output :        none
/*
/*      return value :          none
/*
/*      subroutines :   setcolor, line, putpixel
/*
/*      description :   Outputs graphically the current value of the
/*                              tactile array. The array will be displayed
/*                              horizontally, with the array values extending
/*                              vaerically.
/*
/*************************************************************************/

void pad_refresh_vertical(padtype far *pad, int xoffset, int yoffset, float scale) {

        int row, column ;
        int pixel ;
        int endpixel ;
        int linecolor, endcolor, zerocolor, blankcolor, initcolor;

        linecolor = EGA_WHITE ;         /* EGA_BLACK ; */
        endcolor = EGA_LIGHTRED ;
        zerocolor = EGA_LIGHTGREEN ;            /* EGA_BLACK ; */
        blankcolor = EGA_BLACK ;
        initcolor = getcolor() ;
```

```
        for (row = 0; row < ROWMAX; row++) {
                for (column = 0; column < COLUMNMAX; column++) {
                if ((*pad).taxii[row][column].last_value > (*pad).taxii[row][column].value)
                        endpixel = (*pad).taxii[row][column].last_value * scale ;
                else endpixel = (*pad).taxii[row][column].value * scale ;
                setcolor(blankcolor) ;
                line((*pad).taxii[row][column].x + xoffset,
                        (*pad).taxii[row][column].y - 1 + yoffset,
                        (*pad).taxii[row][column].x + xoffset,
                        (*pad).taxii[row][column].y - endpixel + yoffset) ;
                if ((*pad).taxii[row][column].y == 0) {
                        putpixel((*pad).taxii[row][column].x + xoffset,
                                (*pad).taxii[row][column].y + yoffset,
                                zerocolor) ;
                        }
                else {
                        setcolor(linecolor) ;
                        line((*pad).taxii[row][column].x + xoffset,
                                (*pad).taxii[row][column].y + yoffset,
                                (*pad).taxii[row][column].x + xoffset,
                                (*pad).taxii[row][column].y - (*pad).taxii[row][column].value*scale + 1 + yoffset) ;
                        putpixel((*pad).taxii[row][column].x + xoffset,
                                (*pad).taxii[row][column].y - (*pad).taxii[row][column].value*scale + yoffset,
                                endcolor) ;
                }
                (*pad).taxii[row][column].last_value = (*pad).taxii[row][column].value ;
                }
        }

        setcolor(initcolor) ;

}
```

```
/*****************************************************************/
/*                                                               */
/*          title : MISC.C                                       */
/*          author : M.A. GREENSPAN                              */
/*          date : 29-MAY-90                                     */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <math.h>
#include <alloc.h>
#include "c:\tc201\programs\vobosens\include\misc.h"

float round(float number, int prec) {

        char sign ;
        double *whole ;
        double frac ;
        double result ;
        double mult ;

        sign = POSITIVE ;
        if (number < 0) {
                sign = NEGATIVE ;
                number = number * -1.0 ;
                }
        whole = (double *) malloc(sizeof(double)) ;
        *whole = 0 ;
        mult = pow10(prec) ;
        number = number * mult ;
        frac = modf(number,whole) ;
        if (frac > 0.5)  *whole = *whole + 1 ;
        if ((frac == 0.5)&&(fmod(*whole,2)!=0.0)) *whole = *whole + 1 ;
        result = *whole / mult ;
        result = (float) result ;
        if (sign == NEGATIVE) result = result * -1.0 ;
        return(result) ;

}

void bubble_sort_key(float *keylist, float *memberlist, int members) {

        int index1, index2 ;
        float temp ;

        for (index1 = 0; index1 < members-1; index1++) {
                for (index2=0; index2 < members-1-index1; index2++) {
                        if (*(keylist+index2) > *(keylist+index2+1)) {
                                temp = *(keylist+index2) ;
                                *(keylist+index2) = *(keylist+index2+1) ;
                                *(keylist+index2+1) = temp ;
    temp = *(memberlist+index2) ;
                                *(memberlist+index2) = *(memberlist+index2+1) ;
                                *(memberlist+index2+1) = temp ;
                                }
                        }
                }

}
```

```
/*****************************************************************
/*                                                               *
/*      title : PORTCOM.C                                        *
/*      author : M.A. Greenspan                                  *
/*      date :   30-August-89                                    *
/*                                                               *
/*****************************************************************/

/*****************************************************************
/*
/*      Module Description :    Contains low level communication routines
/*                              for RS232 serial communication
/*
/*      routines :      port_init, kbhit, port_stat, port_send,
/*                      int_to_hex, port_send_int, port_read
/*
/*      header :    portcom.h
/*
/*****************************************************************/

#include <dos.h>
#include <stdlib.h>
#include <ctype.h>
#include "c:\tc201\programs\rhintest\include\rs232com.h"
#include "c:\tc201\programs\rhintest\include\portcom.h"


/*****************************************************************
/*
/*      title :             void port_init(int port,unsigned char code)
/*
/*      input :             int port : serial com port (i.e. COM0, COM1,
/*                              COM2, or COM3)
/*
/*                          unsigned char code :  initialization code
/*
/*      output :    none
/*
/*      return value :      none
/*
/*      subroutines :       int86
/*
/*      description :       Initializes com port for serial communications
/*                          according to the attribute in code
/*
/*****************************************************************/

void port_init(int port,unsigned char code)
{
        union REGS r;

        r.x.dx = port;
        r.h.ah = 0;
        r.h.al = code;
        int86(0x14, &r, &r);
}

/*****************************************************************
/*
/*      title :     int kbhit(void)
/*
/*      input :     none
/*
/*      output :    none
/*
/*      return value :      0 = false, 1 = true
/*
/*      subroutines :       int86
/*
/*      description :       if keyboard has been hit, returns TRUE. Otherwise,
/*                          return FALSE.
/*
/*****************************************************************/

int kbhit(void) {

        union REGS r ;
```

```
        r.h.ah = SERVICE1 ;
    int86(0x16, &r, &r) ;
        if (!(r.x.flags & 64)) {
                r.h.ah = SERVICE0 ;
                int86(0x16, &r, &r);
      return(1) ;
                }
        else return(0) ;
}

/*************************************************************************
/*
/*      title :                 int port_stat(int port)
/*
/*      input :                 int port :      serial com port
/*
/*      output :        none
/*
/*      return value :  status word
/*
/*      subroutines :           int86
/*
/*      description :           Returns status word of serial com port.
/*
/*************************************************************************/

int port_stat(int port)
{
        union REGS r;

        r.x.dx = port ;
    r.h.ah = SERVICE3 ;
        int86(0x14, &r, &r) ;
        return r.x.ax ;
}

/*************************************************************************
/*
/*      title :                 void port_send(int port, char c)
/*
/*      input :                 int port :      serial com port
/*
/*                              char c :        character to send
/*
/*      output :        none
/*
/*      return value :          none
/*
/*      subroutines :           int86
/*
/*      description :           sends a character to the com port.
/*
/*************************************************************************/

void port_send(int port, char c) {

        int status ;
        union REGS r;

        r.x.dx = port;
        r.h.al = c;
        r.h.ah = SERVICE1;
        int86(0x14, &r, &r);
        if (r.h.ah & 128) {
                printf("\nport_send error\n");
                exit(1);
        }
}

/*************************************************************************
/*
/*      title :                 char int_to_hex(int value)
/*
/*      input :                 int value :     input integer value
/*
/*      output :        none
/*
/*      return value :          ascii code of input integer value
```

```
/*
/*          subroutines :          none
/*
/*          description :          Returns hexadecimal ascii code of input
/*                                 integer value between 0 and 15.
/*
/*************************************************************************/

char int_to_hex(int value)
{
        switch (value) {
                case 1 : return('1') ;
      case 2 : return('2') ;
                case 3 : return('3') ;
                case 4 : return('4') ;
                case 5 : return('5') ;
                case 6 : return('6') ;
                case 7 : return('7') ;
                case 8 : return('8') ;
                case 9 : return('9') ;
                case 10 : return('A') ;
                case 11 : return('B') ;
                case 12 : return('C') ;
                case 13 : return('D') ;
                case 14 : return('E') ;
                case 15 : return('F') ;
                default : return('0') ;
                }
}

/*************************************************************************
/*
/*          title :                void port_send_int(int port, int value)
/*
/*          input :                int port : serial com port
/*
/*                                 int value : integer value
/*
/*          output :      none
/*
/*          return value :         none
/*
/*          subroutines :          strcpy, itoa, toascii, port_send
/*
/*          description :          outputs the ascii string of the input integer to the
/*                                 serial com port specified
/*
/*************************************************************************/

void port_send_int(int port, int value)
{
        int radix ;
        int index ;
        char result[10] ;

        index = 0;
        radix = 10;
        strcpy(result,"");
        itoa(value,result,radix);
        while (toascii(*(result+index))!=NULL) {
      port_send(port,*(result+index));
                index++;
                }
}

/*************************************************************************
/*
/*          title :                char  port_read(int port)
/*
/*          input :                int port :    serial com port
/*
/*          output :      none
/*
/*          return value :         character read from serial com port
/*
/*          sburoutines :          int86
/*
/*          description :          returns character read from serial com port
```

```
/*
/*.....*/

char  port_read(int port)
{
        int status ;
        union REGS r ;
        int count ;

        count = 0 ;
        do {
                status = port_stat(port) ;
                count++;
        } while (!(status & 256) && (count < 30000)) ;
        r.x.dx = port;
        r.h.ah = SERVICE2;
        int86(0x14, &r, &r);
        if (r.h.ah & 128) printf("\nport  read error %d, %d \n",r.h.ah,r.h.al);
        return r.h.al;
}
```

```
/*************************************************************/
/*                                                           */
/*        title : PROBESAVC                                  */
/*        author : M.A. GREENSPAN                            */
/*        date : 26-Sept-90                                  */
/*                                                           */
/*************************************************************/

#include <stdio.h>
#include "c:\proglang\tc201\programs\michael\work\include\lord.h"
#include "c:\proglang\tc201\programs\michael\work\include\rhino.h"
#include "c:\proglang\tc201\programs\michael\work\include\probesav.h"

extern int patch_no, probe_no ;
extern FILE *probestream ;

void patch_save(padtype *pad, vector_type padlength, vector_type padwidth, vector_type padnormal, float offset[3]) {

        int row, col ;
        int status ;

        patch_no++ ;
        fprintf(probestream,"patch_name(\"probe_%d\",\"patch_%d\").\n",probe_no,patch_no) ;
        fprintf(probestream,"patch_length(\"probe_%d\",\"patch_%d\",normal(%3.2f,%3.2f,%3.2f)).\n",probe_no, patch_no, padlength.i, padlength.j,
padlength.k) ;
        fprintf(probestream,"patch_width(\"probe_%d\",\"patch_%d\",normal(%3.2f,%3.2f,%3.2f)).\n",probe_no, patch_no, padwidth.i, padwidth.j,
padwidth.k) ;
        fprintf(probestream,"patch_normal(\"probe_%d\",\"patch_%d\",normal(%3.2f,%3.2f,%3.2f)).\n",probe_no, patch_no, padnormal.i,
padnormal.j, padnormal.k) ;
        fprintf(probestream,"patch_offset(\"probe_%d\",\"patch_%d\",offset(%3.2f,%3.2f,%3.2f)).\n",probe_no, patch_no, offset[0], offset[1],
offset[2]) ;
        fprintf(probestream,"contact_array(\"probe_%d\",\"patch_%d\",[",probe_no, patch_no) ;
        for (row = 0; row < ROWMAX; row++) {
                for (col = 0; col <= COLUMNMAX; col++) {
                        fprintf(probestream,"[%d,%d,%d]",row,col,(*pad).taxil[row][col].value) ;
                        if (!((row == ROWMAX)&&(col < COLUMNMAX))) {
                                fprintf(probestream,",") ;
                        }
                }
        }
        fprintf(probestream,"]).") ;
}
/*
main() {

        int patchno ;
        int probeno ;
        char filename[20] ;
        padtype pad ;

        printf("\nEnter filename > ") ;
        scanf("%s",filename) ;
        probeno = 1 ;
        patchno = 10 ;
        patch_save(probeno,patchno,&pad,filename) ;
}
*/
```

```
/*****************************************************************/
/*                                                               */
/*      program : REFLEX.C                                       */
/*      author : M.A. GREENSPAN                                  */
/*      date :   30 August 89                                    */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <conio.h>
#include <math.h>
#include "c:\tc201\programs\robosens\include\rs232com.h"
#include "c:\tc201\programs\robosens\include\portcom.h"
#include "c:\tc201\programs\robosens\include\veltype.h"
#include "c:\tc201\programs\robosens\include\lord.h"
#include "c:\tc201\programs\robosens\include\reflex.h"

int control_function_1(padtype pad, lord_stats_type min_lord_stats, lord_stats_type max_lord_stats) {

        int row, col ;
        int positive_count, negative_count ;

        positive_count = 0 ;
        negative_count = 0 ;
        for (row = 0; row < ROWMAX; row++) {
                for (col = 0; col < COLUMNMAX; col++) {
                        if (pad.taxil[row][col].value <= min_lord_stats.taxil[row][col].average) negative_count++ ;
                        if (pad.taxil[row][col].value >= max_lord_stats.taxil[row][col].average) positive_count++ ;
                }
        }
        if (positive_count > negative_count) return(POSITIVE_REFLEX) ;
        else if (positive_count < negative_count) return(NEGATIVE_REFLEX) ;
        else return(NO_REFLEX) ;
}

int control_function_2(padtype pad, lord_stats_type min_lord_stats, lord_stats_type max_lord_stats) {

        int row, col ;
        float sum_of_taxil_values ;

        sum_of_taxil_values = 0 ;
        for (row = 0; row < ROWMAX; row++) {
                for (col = 0; col < COLUMNMAX; col++) {
                        sum_of_taxil_values = sum_of_taxil_values + pad.taxil[row][col].value ;
                }
        }
        if (sum_of_taxil_values > max_lord_stats.sum_of_averages) return(POSITIVE_REFLEX) ;
        else if (sum_of_taxil_values < min_lord_stats.sum_of_averages) return(NEGATIVE_REFLEX) ;
        else return(NO_REFLEX) ;
}


reflex_type control_function_3(padtype pad, lord_stats_type min_lord_stats, lord_stats_type max_lord_stats) {

        int row, col ;
        int row_sum, col_sum ;
        int weighted_row_sum, weighted_col_sum ;
        int on_taxils ;
        int row_centroid, col_centroid ;
        int row_center_of_mass, col_center_of_mass ;
        float sum_of_taxil_values ;
        int z_reflex, row_reflex, col_reflex, row_rotation_reflex, col_rotation_reflex ;
        reflex_type reflex ;

        row_sum = 0 ;
        col_sum = 0 ;
        weighted_row_sum = 0 ;
        weighted_col_sum = 0 ;
        on_taxils = 0 ;
        sum_of_taxil_values = 0 ;
        for (row = 0; row < ROWMAX; row++) {
                for (col = 0; col < COLUMNMAX; col++) {
                        sum_of_taxil_values = sum_of_taxil_values + pad.taxil[row][col].value ;
                        if (pad.taxil[row][col].value > NOISE_THRESHOLD) {
                                row_sum = row_sum + row ;
                                col_sum = col_sum + col ;
                                weighted_row_sum = weighted_row_sum + row*pad.taxil[row][col].value ;
                                weighted_col_sum = weighted_col_sum + col*pad.taxil[row][col].value ;
```

```
                                        on_taxils++ ;
                                }
                        }
                }
                /**
                ***                     z reflex
                **/
                reflex.z = 0 ;
                if (sum_of_taxil_values > max_lord_stats.sum_of_averages) reflex.z = sum_of_taxil_values - max_lord_stats.sum_of_averages ;
                else if (sum_of_taxil_values < min_lord_stats.sum_of_averages) reflex.z = sum_of_taxil_values - min_lord_stats.sum_of_averages ;
                /**
                ***                     row reflex
                **/
                if ((row_sum == 0)||(on_taxils == 0)) row_centroid = 8 ;
                else row_centroid = (int) (row_sum/on_taxils) ;
                reflex.row = ((min_lord_stats.centroid_row_average + max_lord_stats.centroid_row_average)/2) - row_centroid ;
                /**
                ***                     col reflex
                **/
                if ((col_sum == 0)||(on_taxils == 0)) col_centroid = 8 ;
                else col_centroid = (int) (col_sum/on_taxils) ;
                reflex.col = ((min_lord_stats.centroid_col_average + max_lord_stats.centroid_col_average)/2) - col_centroid ;
                /**
                ***                     row_rotation reflex
                **/
                if ((row_sum == 0)||(sum_of_taxil_values == 0)) row_center_of_mass = 8 ;
                else row_center_of_mass = (weighted_row_sum/sum_of_taxil_values) ;
                reflex.row_rotation = ((min_lord_stats.center_of_mass_row_average + max_lord_stats.center_of_mass_row_average)/2)
row_center_of_mass ;
                /**
                ***                     col_rotation reflex
                **/
                if ((col_sum == 0)||(sum_of_taxil_values == 0)) col_center_of_mass = 8 ;
                else col_center_of_mass = (weighted_col_sum/sum_of_taxil_values) ;
                reflex.col_rotation = ((min_lord_stats.center_of_mass_col_average+ max_lord_stats.center_of_mass_col_average)/2) -
col_center_of_mass ;
                return(reflex) ;
}
```

```
/*****************************************************************/
/*                                                               */
/*          title : RHINOCOM.C                                   */
/*          author : M.A. GREENSPAN                              */
/*          date : 29-MAY90                                      */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <math.h>
#include <alloc.h>
#include "c:\tc201\programs\robosens\include\vs232com.h"
#include "c:\tc201\programs\robosens\include\portcom.h"
#include "c:\tc201\programs\robosens\include\rhino.h"
#include "c:\tc201\programs\robosens\include\rhinocom.h"

int rhino_query(char motor, int port) {

        int result ;

        port_send(port, motor) ;
        port_send(port, QUERY) ;
        port_send(port, CR) ;
        result = port_read(port);
        if (result<32) result = result + 96 ;
        else result = result - 32 ;
        return(result) ;

}

int rhino_motor_idle(char motor, int port) {

        int remainder;
        int zerocount ;
    int index ;
        int result ;

        zerocount = 0 ;
        result = 0 ;
        for (index = 0; index < 3; index++) {
                remainder = rhino_query(motor,port) ;
                if ((remainder == 0)||(remainder == 1)) zerocount++ ;
                }
        if (zerocount > 2) result = 1;
        return(result) ;

}

void rhino_stop(char motor, int port) {

        port_send(port, motor) ;
        port_send(port,X) ;
        port_send(port CR) ;

}


void rhino_reset(int port) {

        port_send(port, Q) ;
        port_send(port, CR) ;

}
```

```
/*****************************************************************/
/*                                                               */
/*          title : RHINOKIN.C                                   */
/*          author : M.A. GREENSPAN                              */
/*          date : 29-MAY-90                                     */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <math.h>
#include "c:\tc201\programs\robosens\include\rhino.h"
#include "c:\tc201\programs\robosens\include\misc.h"
#include "c:\tc201\programs\robosens\include\rhinocom.h"
#include "c:\tc201\programs\robosens\include\rhinomov.h"
#include "c:\tc201\programs\robosens\include\rhinonum.h"
#include "c:\tc201\programs\robosens\include\rhinokin.h"

int similar_space(arm_type arm, float *cspace) {

        float armcspace[3] ;
        float epsx, epsy, epsz ;
        vector_type normal ;

        armcspace[0] = 0 ;
        armcspace[1] = 0 ;
        armcspace[2] = 0 ;

        joint_to_cartesian(arm,armcspace,&normal) ;

        epsx = fabs(*cspace-*armcspace) ;
        epsy = fabs(*(cspace+1)-*(armcspace+1)) ;
        epsz = fabs(*(cspace+2)-*(armcspace+2)) ;

        if ((epsx < FINEGRAIN)&&(epsy < FINEGRAIN)&&(epsz < FINEGRAIN)) return(1) ;
        else return(0) ;

}

float square_difference(float *cspace1, float *cspace2) {

        float x1, y1, z1 ;
        float x2, y2, z2 ;
        float xdiff, ydiff, zdiff ;
        float sqdist ;

    x1 = *cspace1 ;
        y1 = *(cspace1 + 1) ;
        z1 = *(cspace1 + 2) ;
        x2 = *cspace2 ;
        y2 = *(cspace2 + 1) ;
        z2 = *(cspace2 + 2) ;

        xdiff = x1 - x2 ;
        ydiff = y1 - y2 ;
        zdiff = z1 - z2 ;

        sqdist = pow(xdiff,2) + pow(ydiff,2) + pow(zdiff,2) ;
        return(sqdist) ;

}

void calculate_d_e(float cspace[3], float c, float *d, float *e) {

        float x, y, z ;
        float xhat, zhat ;
        float l ;
        float temp1, temp2 ;
        float beta, rho ;

        x = cspace[0] ;
        y = cspace[1] ;
        z = cspace[2] ;
/*      xhat = (float) (sqrt(pow(x,2) + pow(y,2)) - CA*cos(c) - ap*sqrt(pow(normal.i,2) + pow(normal.j,2))) ; */
        xhat = (float) (sqrt(pow(x,2) + pow(y,2)) - CA*cos(c)) ;
        xhat = round(xhat,4) ;
/*      zhat = z - UE - CA*sin(c) - normal.k*ap ; */
        zhat = z - UE - CA*sin(c) ;
        zhat = round(zhat,4) ;
```

```
        l = (float) sqrt(round(pow(xhat,2),4) + round(pow(zhat,2),4)) ;
        l = round(l,4) ;

        temp1 = l/(2.0 * DC) ;
        if (temp1>1) temp1 = 1 ;
        if (temp1 < -1) temp1 = -1 ;

        beta = M_PI_2 - asin(temp1) ;

        temp2 = xhat/l ;
        if (temp2 > 1) temp2 = 1 ;
        if (temp2 < -1) temp2 = -1 ;
        rho = asin(temp2) ;

        *d = M_PI_2 - rho - beta ;
        *e = M_PI_2 - rho + beta ;
}

int new_inverse_transform(float cartesian[3], arm_type *arm, vector_type normalor) {

        int status ;
        float x,y ;
        float a,b,c,d,e,f ;
        float xhat, zhat ;
        float l, temp1, beta, temp2, rho ;
        arm_type temparm ;

        status = 0 ;
        x = cartesian[0] ;
        y = cartesian[1] ;
        a = 0 ;
        f = atan2(y,x) ;
        if (calculate_b_c(normalor.i, normalor.j, normalor.k, f, &b, &c)) {
                calculate_d_e(cartesian, c, &d, &e) ;
                temparm.angle.a = a ;
                temparm.angle.b = b ;
                temparm.angle.c = c ;
                temparm.angle.d = d ;
                temparm.angle.e = e ;
                temparm.angle.f = f ;
                temparm.step.a = a ;
                temparm.step.b = b*BSTEPSPERRAD ;
                temparm.step.c = c*CSTEPSPERRAD ;
                temparm.step.d = d*DSTEPSPERRAD ;
                temparm.step.e = e*ESTEPSPERRAD ;
                temparm.step.f = f*FSTEPSPERRAD ;
                if (similar_space(temparm, cartesian)) {
                        status = 1 ;
                        *arm = temparm ;
                }
        }
        return(status) ;
}

int incremental_transform(float dcspace[3], vector_type dor, float cspace[3], vector_type *normalor, arm_type *arm) {

        int status ;
        float x, y, z ;
        float b, c, d, e, f ;
        float alpha, beta, gamma ;
        float cartesian[3] ;
        vector_type newor ;
        arm_type temparm ;

        status = 0 ;
        x = cspace[0] + dcspace[0] ;
        y = cspace[1] + dcspace[1] ;
        z = cspace[2] + dcspace[2] ;
        cartesian[0] = x ;
        cartesian[1] = y ;
        cartesian[2] = z ;
        (*normalor).i = (*normalor).i + dor.i ;
        (*normalor).j = (*normalor).j + dor.j ;
        (*normalor).k = (*normalor).k + dor.k ;
        b = (*arm).angle.b ;
        c = (*arm).angle.c ;
        normalize_vectortype(*normalor, normalor) ;
```

```
        alpha = (*normalor).i ;
        beta = (*normalor).j ;
        gamma = (*normalor).k ;
        f = atan2(y,x) ;
        if (new_call_bisect_b_c(alpha, beta, gamma, f, &b, COARSE_B_SWING, &c, COARSE_C_SWING)) {
                calculate_d_e(cartesian, c, &d, &e) ;
                temparm.angle.a = 0 ;
                temparm.angle.b = b ;
                temparm.angle.c = c ;
                temparm.angle.d = d ;
                temparm.angle.e = e ;
                temparm.angle.f = f ;
                temparm.step.a = 0 ;
                temparm.step.b = (int) (b*BSTEPSPERRAD) ;
                temparm.step.c = (int) (c*CSTEPSPERRAD) ;
                temparm.step.d = (int) (d*DSTEPSPERRAD) ;
                temparm.step.e = (int) (e*ESTEPSPERRAD) ;
                temparm.step.f = (int) (f*FSTEPSPERRAD) ;
                if (similar_space(temparm, cartesian)) {
                        status = 1 ;
                        *arm = temparm ;
                }
        }
        return(status) ;
}

void normalize_vectortype(vector_type vector, vector_type *normalvector) {

        float length ;

        length = sqrt(pow(vector.i,2) + pow(vector.j,2) + pow(vector.k,2)) ;
        (*normalvector).i = vector.i/length ;
        (*normalvector).j = vector.j/length ;
        (*normalvector).k = vector.k/length ;
}

void joint_to_cartesian(arm_type arm, float *cartesian, vector_type *normal) {

        float x,y,z ;
        float i,j,k ;

        i = (float) M_SQRT_2*(cos(arm.angle.f)*sin(arm.angle.c)*cos(arm.angle.b) - sin(arm.angle.f)*sin(arm.angle.b) +
cos(arm.angle.f)*cos(arm.angle.c)) ;
        j = (float) M_SQRT_2*(sin(arm.angle.f)*sin(arm.angle.c)*cos(arm.angle.b) + cos(arm.angle.f)*sin(arm.angle.b) +
sin(arm.angle.f)*cos(arm.angle.c)) ;
        k = (float) M_SQRT_2*(sin(arm.angle.c) - cos(arm.angle.c)*cos(arm.angle.b)) ;
/*      x = (float) (ED*cos(arm.angle.e) + DC*cos(arm.angle.d) + CA*cos(arm.angle.c))*cos(arm.angle.f) + AP*i ;
        y = (float) (ED*cos(arm.angle.e) + DC*cos(arm.angle.d) + CA*cos(arm.angle.c))*sin(arm.angle.f) + AP*j ;
        z = (float) (UE + ED*sin(arm.angle.e) + DC*sin(arm.angle.d) + CA*sin(arm.angle.c)) + AP*k ;
*/
        x = (float) (ED*cos(arm.angle.e) + DC*cos(arm.angle.d) + CA*cos(arm.angle.c))*cos(arm.angle.f) ;
        y = (float) (ED*cos(arm.angle.e) + DC*cos(arm.angle.d) + CA*cos(arm.angle.c))*sin(arm.angle.f) ;
        z = (float) (UE + ED*sin(arm.angle.e) + DC*sin(arm.angle.d) + CA*sin(arm.angle.c)) ;

        *(cartesian) = x ;
        *(cartesian+1) = y ;
        *(cartesian+2) = z ;
        (*normal).i = i ;
        (*normal).j = j ;
        (*normal).k = k ;

}
```

```
/*****************************************************************/
/*                                                               */
/*        title : RHINOMOVC                                      */
/*        author : M.A. GREENSPAN                                */
/*        date : 29-MAY-90                                       */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "c:\tc201\programs\robosens\include\portcom.h"
#include "c:\tc201\programs\robosens\include\rs232com.h"
#include "c:\tc201\programs\robosens\include\rhino.h"
#include "c:\tc201\programs\robosens\include\rhinocom.h"
#include "c:\tc201\programs\robosens\include\rhinonum.h"
#include "c:\tc201\programs\robosens\include\rhinokin.h"
#include "c:\tc201\programs\robosens\include\rhinomov.h"


/*
int rhino_move_increment(float dcspace[6], vector_type *orientation, arm_type *currentarmstate) {

        int status ;
        float grad[6][5], newgrad[5][5], djspace[5], newdcspace[5] ;
        arm_type deltaarm ;

        printf("\nEntered rhino_move_increment") ;
        status = 0 ;
        rhino_grad(*currentarmstate,grad) ;
        convert_grad_to_5_by_5(grad,*orientation, newgrad) ;
        convert_cspace(dcspace, newdcspace) ;
        if (cramer_5_by_5(newgrad,djspace,newdcspace)) {
                deltaarm.angle.a = 0 ;
                deltaarm.angle.b = djspace[0] ;
                deltaarm.angle.c = djspace[1] ;
                deltaarm.angle.d = djspace[2] ;
                deltaarm.angle.e = djspace[3] ;
                deltaarm.angle.f = djspace[4] ;
                deltaarm.step.a = 0 ;
                deltaarm.step.b = djspace[0]*BSTEPSPERRAD ;
                deltaarm.step.c = djspace[1]*CSTEPSPERRAD ;
                deltaarm.step.d = djspace[2]*DSTEPSPERRAD ;
                deltaarm.step.e = djspace[3]*ESTEPSPERRAD ;
                deltaarm.step.f = djspace[4]*FSTEPSPERRAD ;
                if (rhino_reconfigure(deltaarm,RHINOPORT)) {
                        (*currentarmstate).angle.a = (*currentarmstate).angle.a + deltaarm.angle.a ;
                        (*currentarmstate).angle.b = (*currentarmstate).angle.b + deltaarm.angle.b ;
                        (*currentarmstate).angle.c = (*currentarmstate).angle.c + deltaarm.angle.c ;
                        (*currentarmstate).angle.d = (*currentarmstate).angle.d + deltaarm.angle.d ;
                        (*currentarmstate).angle.e = (*currentarmstate).angle.e + deltaarm.angle.e ;
                        (*currentarmstate).angle.f = (*currentarmstate).angle.f + deltaarm.angle.f ;
                        (*currentarmstate).step.a = (*currentarmstate).step.a + deltaarm.step.a ;
                        (*currentarmstate).step.b = (*currentarmstate).step.b + deltaarm.step.b ;
                        (*currentarmstate).step.c = (*currentarmstate).step.c + deltaarm.step.c ;
                        (*currentarmstate).step.d = (*currentarmstate).step.d + deltaarm.step.d ;
                        (*currentarmstate).step.e = (*currentarmstate).step.e + deltaarm.step.e ;
                        (*currentarmstate).step.f = (*currentarmstate).step.f + deltaarm.step.f ;
                        (*orientation).i = (*orientation).i + dcspace[3] ;
                        (*orientation).j = (*orientation).j + dcspace[4] ;
                        (*orientation).k = (*orientation).k + dcspace[5] ;
                        status = 1 ;
                }
        }
        return(status) ;
}

*/

int rhino_move_pt_to_pt(float cspace[3], vector_type *orientation, arm_type *armstate) {

        int status ;
        arm_type initialarmstate, nextarmstate, deltaarm ;

        status = 0 ;
        initialarmstate = *armstate ;
        if (new_inverse_transform(cspace,&nextarmstate,*orientation)) {
                deltaarm.angle.a = nextarmstate.angle.a - initialarmstate.angle.a ;
```

```
                deltaarm.angle.b = nextarmstate.angle.b - initialarmstate.angle.b ;
                deltaarm.angle.c = nextarmstate.angle.c - initialarmstate.angle.c ;
                deltaarm.angle.d = nextarmstate.angle.d - initialarmstate.angle.d ;
                deltaarm.angle.e = nextarmstate.angle.e - initialarmstate.angle.e ;
                deltaarm.angle.f = nextarmstate.angle.f - initialarmstate.angle.f ;
                deltaarm.step.a = nextarmstate.step.a - initialarmstate.step.a ;
                deltaarm.step.b = nextarmstate.step.b - initialarmstate.step.b ;
                deltaarm.step.c = nextarmstate.step.c - initialarmstate.step.c ;
                deltaarm.step.d = nextarmstate.step.d - initialarmstate.step.d ;
                deltaarm.step.e = nextarmstate.step.e - initialarmstate.step.e ;
                deltaarm.step.f = nextarmstate.step.f - initialarmstate.step.f ;

                if (!(rhino_reconfigure(deltaarm,RHINOPORT))) {
                        *armstate = initialarmstate ;
                        joint_to_cartesian(*armstate,cspace,orientation) ;
                }
                else {
                        *armstate = nextarmstate ;
                        joint_to_cartesian(*armstate,cspace,orientation) ;
                        status = 1 ;
                }
        }
        return(status) ;
}

int rhino_move_delta(float dcspace[3], vector_type dor, float cspace[3], vector_type *normalor, arm_type *armstate) {

        int status ;
        arm_type initialarmstate, nextarmstate, deltaarm ;

        status = 0 ;
        initialarmstate = *armstate ;
        nextarmstate = initialarmstate ;
        if (incremental_transform(dcspace, dor, cspace, normalor, &nextarmstate)) {
                deltaarm.angle.a = nextarmstate.angle.a - initialarmstate.angle.a ;
                deltaarm.angle.b = nextarmstate.angle.b - initialarmstate.angle.b ;
                deltaarm.angle.c = nextarmstate.angle.c - initialarmstate.angle.c ;
                deltaarm.angle.d = nextarmstate.angle.d - initialarmstate.angle.d ;
                deltaarm.angle.e = nextarmstate.angle.e - initialarmstate.angle.e ;
                deltaarm.angle.f = nextarmstate.angle.f - initialarmstate.angle.f ;
                deltaarm.step.a = nextarmstate.step.a - initialarmstate.step.a ;
                deltaarm.step.b = nextarmstate.step.b - initialarmstate.step.b ;
                deltaarm.step.c = nextarmstate.step.c - initialarmstate.step.c ;
                deltaarm.step.d = nextarmstate.step.d - initialarmstate.step.d ;
                deltaarm.step.e = nextarmstate.step.e - initialarmstate.step.e ;
                deltaarm.step.f = nextarmstate.step.f - initialarmstate.step.f ;

                if (!(rhino_reconfigure(deltaarm,RHINOPORT))) {
                        *armstate = initialarmstate ;
                        joint_to_cartesian(*armstate,cspace,normalor) ;
                }
                else {
                        *armstate = nextarmstate ;
                        joint_to_cartesian(*armstate,cspace,normalor) ;
                        status = 1 ;
                }
        }
        return(status) ;
}

float steps_per_rad (char motor) {

        float steps ;

        switch (motor) {
                case B : steps = BSTEPSPERRAD ; break ;
                case C : steps = CSTEPSPERRAD ; break ;
                case D : steps = DSTEPSPERRAD ; break ;
                case E : steps = ESTEPSPERRAD ; break ;
                case F : steps = FSTEPSPERRAD ; break ;
                default : steps = 0 ;
                }
        return(steps) ;
}

void rhino_user_reconfigure(arm_type *arm) {
```

```
        arm_type deltaarm ;
        float cspace[3] ;
        vector_type normal ;

        cspace[0] = 0 ;
        cspace[1] = 0 ;
        cspace[2] = 0 ;
        printf("\nEnter degree values for movement ... \n") ;

        printf("\n\n\nA\t\tB\t\tC\t\tD\t\tE\t\tF\n");
        scanf("%f %f %f %f %f
%f",&deltaarm.angle.a,&deltaarm.angle.b,&deltaarm.angle.c,&deltaarm.angle.d,&deltaarm.angle.e,&deltaarm.angle.f);

    deltaarm.angle.a = deltaarm.angle.a * (2*M_PI/360) ;
        deltaarm.angle.b = deltaarm.angle.b * (2*M_PI/360) ;
        deltaarm.angle.c = deltaarm.angle.c * (2*M_PI/360) ;
        deltaarm.angle.d = deltaarm.angle.d * (2*M_PI/360) ;
        deltaarm.angle.e = deltaarm.angle.e * (2*M_PI/360) ;
        deltaarm.angle.f = deltaarm.angle.f * (2*M_PI/360) ;

        (*arm).angle.a = deltaarm.angle.a + (*arm).angle.a ;
        (*arm).angle.b = deltaarm.angle.b + (*arm).angle.b ;
        (*arm).angle.c = deltaarm.angle.c + (*arm).angle.c ;
        (*arm).angle.d = deltaarm.angle.d + (*arm).angle.d ;
        (*arm).angle.e = deltaarm.angle.e + (*arm).angle.e ;
        (*arm).angle.f = deltaarm.angle.f + (*arm).angle.f ;
        (*arm).step.a = deltaarm.step.a + (*arm).step.a ;
        (*arm).step.b = deltaarm.step.b + (*arm).step.b ;
        (*arm).step.c = deltaarm.step.c + (*arm).step.c ;
        (*arm).step.d = deltaarm.step.d + (*arm).step.d ;
        (*arm).step.e = deltaarm.step.e + (*arm).step.e ;
        (*arm).step.f = deltaarm.step.f + (*arm).step.f ;

        rhino_reconfigure(deltaarm,RHINOPORT) ;

        joint_to_cartesian(*arm,cspace,&normal) ;
        printf("\ncartesian coordinates : X=%3.2f, Y=%3.2f, Z=%3.2f\n",
                cspace[0],cspace[1],cspace[2]);

}

void rhino_move(char motor, int distance, int port) {

        char sign ;
        if (distance >= 0) sign=POSITIVE_MOTION ;
        else sign=NEGATIVE_MOTION ;

/*      printf("\nrhino_move %c %d",motor,distance); */
        distance = abs(distance) ;
        port_send(port, motor) ;
        port_send(port, sign) ;
        port_send_int(port,distance) ;
        port_send(port, CR) ;

}


void rhino_move_wait(char motor, int distance, int port) {

        char sign ;
        int remaining ;
        int index ;
        int zerocount ;

        if (distance >= 0) sign=POSITIVE_MOTION ;
        else sign=NEGATIVE_MOTION ;
        distance = abs(distance) ;
        while (distance > MAXSTEPMOVE) {
/*      printf("\nrhino_move_wait : %c %c MAXSTEPMOVE",motor,sign); */
                port_send(port, motor) ;
                port_send(port, sign) ;
                port_send_int(port,MAXSTEPMOVE) ;
                port_send(port, CR) ;
                distance = distance - MAXSTEPMOVE ;
      do {} while (!rhino_motor_idle(motor,port)) ;
        }
        if (distance > 0) {
/*              printf("\nrhino_move_wait : %c %c %d",motor,sign,distance); */
```

```
                port_send(port, motor) ;
                port_send(port, sign) ;
                port_send_int(port,distance) ;
                port_send(port, CR) ;
        }
}

int rhino_reconfigure(arm_type arm, int port) {

        char motor ;
        int ar,br,cr,dr,er,fr ;
        int adone,bdone,cdone,ddone,edone,fdone ;
        arm_type temparm ;

        adone = 0 ;
        bdone = 0 ;
        cdone = 0 ;
        ddone = 0 ;
        edone = 0 ;
        fdone = 0 ;
        temparm = arm ;

        do {
                if ((!adone)&&(rhino_motor_idle(A,port))) {
                        if (temparm.step.a > MAXSTEPMOVE) ar = MAXSTEPMOVE ;
                        else {
                                if (temparm.step.a < -MAXSTEPMOVE) ar = -MAXSTEPMOVE ;
                                else {
                                        ar = temparm.step.a ;
                                        adone = 1 ;
                                        }
                                }
                        temparm.step.a = temparm.step.a - ar ;
            if (ar != 0) rhino_move(A,ar,port) ;
                }
                if ((!bdone)&&(rhino_motor_idle(B,port))) {
                        if (temparm.step.b > MAXSTEPMOVE) br = MAXSTEPMOVE ;
                        else {
                                if (temparm.step.b < -MAXSTEPMOVE) br = -MAXSTEPMOVE ;
                                else {
                                        br = temparm.step.b ;
                                        bdone = 1 ;
                                        }
                                }
                        temparm.step.b = temparm.step.b - br ;
            if (br != 0) rhino_move(B,br,port) ;
                }
                if ((!cdone)&&(rhino_motor_idle(C,port))) {
                        if (temparm.step.c > MAXSTEPMOVE) cr = MAXSTEPMOVE ;
                        else {
                                if (temparm.step.c < -MAXSTEPMOVE) cr = -MAXSTEPMOVE ;
                                else {
                                        cr = temparm.step.c ;
                                        cdone = 1 ;
                                        }
                                }
                        temparm.step.c = temparm.step.c - cr ;
            if (cr != 0) rhino_move(C,cr,port) ;
                }
                if ((!ddone)&&(rhino_motor_idle(D,port))) {
                        if (temparm.step.d > MAXSTEPMOVE) dr = MAXSTEPMOVE ;
                        else {
                                if (temparm.step.d < -MAXSTEPMOVE) dr = -MAXSTEPMOVE ;
                                else {
                                        dr = temparm.step.d ;
                                        ddone = 1 ;
                                        }
                                }
                        temparm.step.d = temparm.step.d - dr ;
             if (dr != 0) rhino_move(D,dr,port) ;
                }
                if ((!edone)&&(rhino_motor_idle(E,port))) {
                        if (temparm.step.e > MAXSTEPMOVE) er = MAXSTEPMOVE ;
                        else {
                                if (temparm.step.e < -MAXSTEPMOVE) er = -MAXSTEPMOVE ;
                                else {
                                        er = temparm.step.e ;
```

```
                                                edone = 1 ;
                                                }
                                        }
                                temparm.step.e = temparm.step.e - er ;
        if (er != 0) rhino_move(E,er,port) ;
                        }
                        if ((!fdone)&&(rhino_motor_idle(F,port))) {
                                if (temparm.step.f > MAXSTEPMOVE) fr = MAXSTEPMOVE ;
                                else {
                                        if (temparm.step.f < -MAXSTEPMOVE) fr = -MAXSTEPMOVE ;
                                        else {
                                                fr = temparm.step.f ;
                                                fdone = 1 ;
                                                }
                                        }
                                temparm.step.f = temparm.step.f - fr ;
        if (fr != 0) rhino_move(F,fr,port) ;
                        }
        } while (!((adone)&&(bdone)&&(cdone)&&(ddone)&&(edone)&&(fdone))) ;
        return(1) ;
}

void rhino_move_wait_rads(char motor, float rads, int port) {

    int steps ;

    steps = (int) (rads * steps_per_rad(motor)) ;
        rhino_move_wait(motor,steps,port) ;

}

int rhino_reconfigure_rads(arm_type *deltaarm, int port) {

        arm_type arm ;

/*
        printf("\nEntered rhino_reconfigure_rads") ;
        printf("\n\n\tA\tB\tC\tD\tE\tF") ;
        printf("\n\t%3.2f\t%3.2f\t%3.2f\t%3.2f\t%3.2f\t%3.2f",
                (*deltaarmrads).a,(*deltaarmrads).b,(*deltaarmrads).c,(*deltaarmrads).d,(*deltaarmrads).e,(*deltaarmrads).f) ;
*/
        arm.step.a = (int) ((*deltaarm).angle.a * steps_per_rad(A)) ;
        arm.step.b = (int) ((*deltaarm).angle.b * steps_per_rad(B)) ;
        arm.step.c = (int) ((*deltaarm).angle.c * steps_per_rad(C)) ;
        arm.step.d = (int) ((*deltaarm).angle.d * steps_per_rad(D)) ;
        arm.step.e = (int) ((*deltaarm).angle.e * steps_per_rad(E)) ;
        arm.step.f = (int) ((*deltaarm).angle.f * steps_per_rad(F)) ;

        rhino_reconfigure(arm, port) ;
        return(1) ;
}

int rhino_restore_arm(float savecspace[3], vector_type saveor, arm_type savearm, float restorecspace[3], vector_type *restoreor, arm_type
*restorearm) {

        int status ;
        arm_type deltaarm ;

        deltaarm.angle.a = savearm.angle.a - (*restorearm).angle.a ;
        deltaarm.angle.b = savearm.angle.b - (*restorearm).angle.b ;
        deltaarm.angle.c = savearm.angle.c - (*restorearm).angle.c ;
        deltaarm.angle.d = savearm.angle.d - (*restorearm).angle.d ;
        deltaarm.angle.e = savearm.angle.e - (*restorearm).angle.e ;
        deltaarm.angle.f = savearm.angle.f - (*restorearm).angle.f ;
        deltaarm.step.a = savearm.step.a - (*restorearm).step.a ;
        deltaarm.step.b = savearm.step.b - (*restorearm).step.b ;
        deltaarm.step.c = savearm.step.c - (*restorearm).step.c ;
        deltaarm.step.d = savearm.step.d - (*restorearm).step.d ;
        deltaarm.step.e = savearm.step.e - (*restorearm).step.e ;
        deltaarm.step.f = savearm.step.f - (*restorearm).step.f ;
        status = rhino_reconfigure(deltaarm, RHINOPORT) ;
        if (status) {
                *restorearm = savearm ;
                *restoreor = saveor ;
                restorecspace[0] = savecspace[0] ;
                restorecspace[1] = savecspace[1] ;
                restorecspace[2] = savecspace[2] ;
        }
```

```
        return(status) ;
}

void rhino_save_arm(float cspace[3], vector_type or, arm_type arm, float savecspace[3], vector_type *saveor, arm_type *savearm) {

        *savearm = arm ;
        *saveor = or ;
        savecspace[0] = cspace[0] ;
        savecspace[1] = cspace[1] ;
        savecspace[2] = cspace[2] ;
}
```

```
/*************************************************************/
/*                                                           */
/*        title : RHINONUM.C                                 */
/*        author : M.A. GREENSPAN                            */
/*        date : 29-MAY-90                                   */
/*                                                           */
/*************************************************************/

#include <math.h>
#include <stdio.h>
#include <conio.h>
#include "c:\tc201\programs\robosens\include\rhino.h"
#include "c:\tc201\programs\robosens\include\rhinocom.h"
#include "c:\tc201\programs\robosens\include\rhinokin.h"
#include "c:\tc201\programs\robosens\include\rhinonum.h"

float calculate_gamma(float b, float c) {

        float gamma ;

        gamma = -(1/sqrt(2))*cos(c)*cos(b) + (1/sqrt(2))*sin(c) ;
        return(gamma) ;
}

float calculate_alpha(float b, float c, float f) {

        float al, ca ;

        alpha = (float) (1/sqrt(2))*(cos(f)*sin(c)*cos(b) - sin(f)*sin(b) + cos(f)*cos(c)) ;
        return(alpha) ;
}

float calculate_beta(float b, float c, float f) {

        float beta ;

        beta = (1/sqrt(2))*(sin(f)*sin(c)*cos(b) + cos(f)*sin(b) + sin(f)*cos(c)) ;
        return(beta) ;
}

int call_bisect_b_c(float alpha, float beta, float gamma, float f, float *b, float bswing, float *c, float cswing) {

        int status ;

        printf("\nEntered call_bisect_b_c") ;
        status = bisect_b_c(alpha, beta, gamma, f, b, bswing, c, cswing) ;
        return(status) ;
}

int new_call_bisect_b_c(float alpha, float beta, float gamma, float f, float *b, float bswing, float *c, float cswing) {

        int status ;
        float error ;

/*      printf("\nEntered call_bisect_b_c") ; */

        status = 0 ;
        do {
                error = new_bisect_b_c(alpha, beta, gamma, f, b, bswing, c, cswing) ;
                bswing = bswing/2 ;
                cswing = cswing/2 ;
        } while ((error > FINE_ABG_ERROR)&&(bswing > MIN_B_SWING)&&(cswing > MIN_C_SWING)) ;
        if (error <= FINE_ABG_ERROR) status = 1 ;
        return(status) ;
}

float new_bisect_b_c(float alpha, float beta, float gamma, float f, float *b, float bswing, float *c, float cswing) {

        int bindex, cindex ;
        float b0, c0 ;
        float tempb ;
        float tempc ;
        float tempalpha ;
        float tempbeta ;
        float tempgamma ;
        float error, minerror ;

        b0 = *b ;
```

```
        c0 = *c ;
        minerror = LARGE_NUMBER ;
        for (cindex=-1; cindex<=1; cindex++) {
                tempc = cindex * cswing + c0 ;
                for (bindex=-1; bindex<=1; bindex++) {
                        tempb = bindex * bswing + b0 ;
                        tempalpha = calculate_alpha(tempb,tempc,f) ;
                        tempbeta = calculate_beta(tempb,tempc,f) ;
                        tempgamma = calculate_gamma(tempb,tempc) ;
                        error = fabs(tempalpha - alpha) + fabs(tempbeta - beta) + fabs(tempgamma - gamma) ;
                        if (error < minerror) {
                                minerror = error ;
                                *b = tempb ;
                                *c = tempc ;
                        }
                }
        }
/*      printf("\nbisect_b_c : (b,c,error) = ( %f, %f, %f)", *b, *c, minerror) ; */
        return(minerror) ;
}

static int near pascal bisect_b_c(float alpha, float beta, float gamma, float f, float far *b, float bswing, float far *c, float cswing) {

        int status ;
        int bindex, cindex ;
        float b0, c0 ;
        float tempb ;
        float tempc ;
        float tempalpha ;
        float tempbeta ;
        float tempgamma ;
        float error, minerror ;

/*      printf("\nEntered bisect_b_c : (b,c) = (%f,%f)", *b, *c) ; */
        status = 0 ;
        b0 = *b ;
        c0 = *c ;
        minerror = LARGE_NUMBER ;
        for (cindex=-1; cindex<=1; cindex++) {
                tempc = cindex * cswing + c0 ;
                for (bindex=-1; bindex<=1; bindex++) {
                        tempb = bindex * bswing + b0 ;
                        tempalpha = calculate_alpha(tempb,tempc,f) ;
                        tempbeta = calculate_beta(tempb,tempc,f) ;
                        tempgamma = calculate_gamma(tempb,tempc) ;
                        error = fabs(tempalpha - alpha) + fabs(tempbeta - beta) + fabs(tempgamma - gamma) ;
                        if (error < minerror) {
                                minerror = error ;
                                *b = tempb ;
                                *c = tempc ;
                        }
                }
        }
        printf("\nbisect_b_c : (b,c,error) = ( %f, %f, %f)", *b, *c, minerror) ;
        if (error < FINE_ABG_ERROR) {
                status = 1 ;
        }
        else {
                bswing = bswing/2 ;
                cswing = cswing/2 ;
                if ((cswing>=MIN_C_SWING)||(bswing>=MIN_B_SWING)) {
                        status = bisect_b_c(alpha, beta, gamma, f, b, bswing, c, cswing) ;
                }
        }
        return(status) ;
}


float linear_approximate_b_c(float alpha, float beta, float gamma, float f, float lob, float hib, float bstep, float loc, float hic, float cstep, float *b, float *c) {

        float tempb, tempc ;
        float error, minerror ;

        minerror = LARGE_NUMBER ;
        error = minerror ;
        tempb = lob ;
        do {
```

```
                tempc = loc ;
                do {
                        error = fabs(calculate_alpha(tempb,tempc,f)-alpha) + fabs(calculate_beta(tempb,tempc,f)-beta) +
fabs(calculate_gamma(tempb,tempc)-gamma) ;
                        if (error < minerror) {
                                *b = tempb ;
                                *c = tempc ;
                                minerror = error ;
                        }
/*                      printf("\n\t(tempb , tempc , error) = ( %f, %f, %f)", tempb, tempc, error) ;
                        tempc = tempc + cstep ;
                } while (tempc < hic) ;
                tempb = tempb + bstep ;
        } while (tempb < hib) ;
        return(minerror) ;
}


int calculate_b_c(float alpha, float beta, float gamma, float f, float *b, float *c) {

        int status ;
        float error ;

/*      printf("\nEntered calculate_b_c") ; */
        status = 0 ;
/*      printf("\ncoarse approximate ...") ; */
        error = linear_approximate_b_c(alpha, beta, gamma, f, -M_PI, M_PI, COARSE_B_STEP, -M_PI_2, M_PI_2, COARSE_C_STEP, b, c) ;
        if (error < COARSE_ABG_ERROR) {
/*              printf("\nfine approximate ...") ; */
                status = new_call_bisect_b_c(alpha, beta, gamma, f, b, COARSE_B_SWING, c, COARSE_C_SWING) ;
        }
        return(status) ;
}

/*
main() {

        float alpha, beta, gamma, b, c, f ;
        float error ;
        char resp ;

        do {
                printf("\n\nEnter value of alpha beta gamma > ") ;
                scanf("%f %f %f", &alpha, &beta, &gamma) ;
                printf("\nEnter value of f > ") ;
                scanf("%f", &f) ;
                printf("\ncalculating b c ...") ;
                f = f*M_PI/180 ;
                if (calculate_b_c(alpha, beta, gamma, f, &b, &c)) {
                        printf("\nsuccess") ;
                        printf("\t(b,c) = ( %f, %f)", b*180/M_PI,c*180/M_PI) ;
                }
                else printf("\nfailure") ;
                printf("\nRestart or Exit (R/X) > ") ;
                resp = getch() ;
        } while ((resp=='r')||(resp=='R')) ;
}
*/
```

```
/*****************************************************************/
/*                                                               */
/*        title : RHINOPRO.C                                     */
/*        author : M.A. GREENSPAN                                */
/*        date : 4-JUNE-90                                       */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include <conio.h>
#include <math.h>

#include "c:\tc201\programs\robosens\include\rhino.h"
#include "c:\tc201\programs\robosens\include\rhinokin.h"
#include "c:\tc201\programs\robosens\include\rhinomov.h"
#include "c:\tc201\programs\robosens\include\rhinopro.h"

void calculate_arm_length_axis(arm_type *arm, vector_type *padlength) {

        float root2 ;
        float b,c,f ;

        b = (*arm).angle.b ;
        c = (*arm).angle.c ;
        f = (*arm).angle.f ;
        root2 = (float) sqrt(2) ;

        (*padlength).i = (float) ((1/root2)*(-cos(f)*sin(c)*cos(b) + sin(f)*sin(b) + cos(f)*cos(c))) ;
        (*padlength).j = (float) ((1/root2)*(-sin(f)*sin(c)*cos(b) + cos(f)*sin(b) + sin(f)*cos(c))) ;
        (*padlength).k = (float) ((1/root2)*(cos(c)*cos(b) + sin(c))) ;
}

void calculate_arm_width_axis(arm_type *arm, vector_type *padlength) {

        float b,c,f ;

        b = (*arm).angle.b ;
        c = (*arm).angle.c ;
        f = (*arm).angle.f ;

        (*padlength).i = (float) ((-cos(f)*sin(c)*sin(b) + sin(f)*cos(b))) ;
        (*padlength).j = (float) -((-sin(f)*sin(c)*sin(b) + cos(f)*cos(b))) ;
        (*padlength).k = (float) ((cos(c)*sin(b))) ;
}


void rotate_around_arm_width_axis(float epsilon, arm_type *arm, vector_type or, float cspace[3], float padcspace[3], vector_type *dor, float
dcspace[3]) {

        float root2 ;
        float b,c,f ;
        float alpha, beta, gamma ;
        float dist ;
        vector_type rotatedor ;

        dist = (float) (pow(cspace[0]-padcspace[0],2) + pow(cspace[1]-padcspace[1],2) + pow(cspace[2]-padcspace[2],2)) ;
        dist = (float) sqrt(dist) ;
        b = (*arm).angle.b ;
        c = (*arm).angle.c ;
        f = (*arm).angle.f ;
        alpha = (float) (cos(M_PI_4 + epsilon)*cos(b)) ;
        beta = (float) (cos(M_PI_4 + epsilon)*sin(b)) ;
        gamma = (float) sin(M_PI_4 + epsilon) ;
        rotatedor.i = (float) (cos(f)*sin(c)*alpha - sin(f)*beta + cos(f)*cos(c)*gamma) ;
        rotatedor.j = (float) (sin(f)*sin(c)*alpha + cos(f)*beta + sin(f)*cos(c)*gamma) ;
        rotatedor.k = (float) (-cos(c)*alpha + sin(c)*gamma) ;
        normalize_vectortype(rotatedor, &rotatedor) ;
        (*dor).i = rotatedor.i - or.i ;
        (*dor).j = rotatedor.j - or.j ;
        (*dor).k = rotatedor.k - or.k ;
        dcspace[0] = padcspace[0] - rotatedor.i*dist - cspace[0] ;
        dcspace[1] = padcspace[1] - rotatedor.j*dist - cspace[1] ;
        dcspace[2] = padcspace[2] - rotatedor.k*dist - cspace[2] ;
}
```

```
/*****************************************************************/
/*                                                               */
/*          program : SAVSTATS.C                                 */
/*          author : M.A. GREENSPAN                              */
/*          date :      30 August 89                             */
/*                                                               */
/*****************************************************************/

#include <stdlib.h>
#include <stdio.h>
#include "c:\tc201\programs\robosens\include\lord.h"
#include "c:\tc201\programs\robosens\include\lordhlt.h"
#include "c:\tc201\programs\robosens\include\rhino.h"
#include "c:\tc201\programs\robosens\include\savstats.h"

void save_lord_stats(float cspace[3], vector_type orientation, lord_stats_type lord_stats, FILE *write_stream) {

        fprintf(write_stream,"\n----------------------------------------------") ;
        fprintf(write_stream,"\nLocation\t:\tX\t\tY\t\tZ") ;
        fprintf(write_stream,"\n\t\t\t%3.2f\t\t%3.2f\t\t%3.2f",cspace[0],cspace[1],cspace[2]) ;
        fprintf(write_stream,"\nOrientation\t:\ti\t\tJ\t\tK") ;
        fprintf(write_stream,"\n\t\t\t%3.2f\t\t%3.2f\t\t%3.2f",orientation.i,orientation.j,orientation.k) ;
        fprintf(write_stream,"\nCentroid Row Average = %3.2f",lord_stats.centroid_row_average) ;
        fprintf(write_stream,"\nCentroid Col Average = %3.2f",lord_stats.centroid_col_average) ;
        fprintf(write_stream,"\nCenter of Mass Row Average = %3.2f",lord_stats.center_of_mass_row_average) ;
        fprintf(write_stream,"\nCenter of Mass Col Average = %3.2f",lord_stats.center_of_mass_col_average) ;
        fprintf(write_stream,"\nSum of Taxil Averages = %3.2f",lord_stats.sum_of_averages) ;
/*
        for (row = 0; row < ROWMAX; row++) {
                for (col = 0; col < COLUMNMAX; col++) {
                        (*lord_stats).taxil[row][col].average = (float) (*lord_stats).taxil[row][col].sum/SIGNIFICANT_NUMBER ;
                        (*lord_stats).sum_of_averages = (*lord_stats).sum_of_averages + (*lord_stats).taxil[row][col].average ;
                        (*lord_stats).taxil[row][col].stdev = (float)
sqrt((1/SIGNIFICANT_NUMBER)*((*lord_stats).taxil[row][col].sum_of_squares - (SIGNIFICANT_NUMBER*(*lord_stats).taxil[row][col].average))) ;
                }
        }
*/
}
```