UNIVERSITÉ D'OTTAWA
UNIVERSITY OF OTTAWA

MODEL-BASED TACTILE OBJECT RECOGNITION USING PSEUDO-RANDOM ENCODING

by

Stephen Siu Kau Yeung

A thesis submitted to the School of Graduate Studies and Research
in partial fulfilment of the requirements for the degree of
Doctor of Philosophy
in Electrical Engineering

1996

OTTAWA-CARLETON INSTITUTE FOR ELECTRICAL ENGINEERING

Department of Electrical Engineering
Faculty of Engineering

University of Ottawa, Ontario, CANADA

Abstract

An original model-based tactile 3-D object recognition system is presented. The development of this system was motivated by the need for a "blind" back-up solution to maintain the object recognition capability of the Space Shuttle robot manipulator when the sun light blinds the video-cameras normally used for object recognition.

Tactile object recognition requires the identification of the explored object as well as the recovery of its 3-D position and orientation. Such a complex task calls for a robotic tactile sensing system whose parameters (spatial resolution, probing compliance, tactile image processing) are beyond those usually offered by the emergent tactile sensing technology. The development of the tactile sensing system described in this thesis has resulted in a number of contributions: (i) novel tab-shaped elastic overlay which reduces the cross-talk errors in the tactile sensor, (ii) instrumented passive compliant wrist for more efficient object exploration, and (iii) 2-D correlation method for the integration of the local tactile images in a global image of the explored object surface.

The original model-based tactile object recognition paradigm in this thesis can be summarized as follows: "Given a set of 3-D objects having their surfaces Braille-like embossed with terms of a Pseudo-Random Array (PRA) for which is a *priori*

known how the unfolded faces of geometrical models of these objects are mapped into the PRA plane, and given the tactile image of the explored object surfaces, determine the identity, position, and orientation of that object." Tactile object recognition is thus reduced to the recognition of a small set of symbols embossed on object surfaces. The use of multi-valued (instead of the usual binary) pseudo-random encoding allows to reduce the number of embossed symbols which actually are needed to recognized in the tactile image. An original contribution is made by the development of a Pseudo-Random Multi-Valued Product Array (PRMVPA) encoding and related pseudo-random/natural code conversion algorithm.

While it is inherently limited to a given set of embossed objects, the described "blind" method allows for a simpler and faster 3-D object recognition using solely tactile sensing. Such an object recognition method has potential applications in controlled environments such as the Space Shuttle, nuclear stations, underwater maintenance of the oil platforms and pipelines.

Acknowledgements

I would like specially to thank Professor Emil M. Petriu for his kind guidance enabling this thesis to be realized.

I also thank Dr. Sunil Das as my second thesis supervisor.

Many thanks to the Space Mechanics Machine Shop for fabricating the necessary hardware pieces and Nick Trif for his assistance.

I would like to thank Howard Reynaud for reviewing the final draft.

This thesis is dedicated to my wife, IRENE for her full support in making it happen

Table of Contents

		Page
CHAPTER 1	**INTRODUCTION**	
1.0	Background	1
1.1	Comments on The Human Sense of Touch From An Engineering Perspective	1
1.2	Tactile Sensing Technology Perspective	4
1.3	Objectives	9
1.4	Summary of The Main Contributions	10
1.5	Thesis Outline	11
CHAPTER 2	**MODEL-BASED OBJECT RECOGNITION**	
2.0	Introduction	
2.1	Model-Based Object Recgnition	13
2.2	Model-Based Tactile Recognition Using Pseudo-Random Encoding of Object Surfaces	14 18
CHAPTER 3	**TACTILE OBJECT RECOGNITION**	
3.0	Introduction	
3.1	Tactile Sensing	21
3.2	Exploratory Object Sensing	22
3.2.1	Problem Definition	28 28
CHAPTER 4	**EXPERIMENTAL TACTILE SENSING SYSTEM**	
4.0	Introduction	31
4.1	Robot Tactile Sensing System	32
4.2	Tactile Probe	38
4.2.1	Elastic Overlay Behaviour	40
4.2.2	Tab-shaped Elastic Overlay	43

		Page
4.3	Correlation Technique for Tactile Data Integration	49
CHAPTER 5	**PSEUDO-RANDOM ENCODING**	
5.1	Pseudo-Random Binary Sequences	58
5.2	Pseudo-Random Binary Arrays	69
5.3	Pseudo-Random Product Array Encoding	71
5.4	Introduction to Pseudo-Random Multi-Valued Sequences	75
5.4.1	Fields and Rings of Pseudo-Random Polynomial	75
5.4.2	Pseudo-Random Multi-Valued Sequences	76
5.4.3	Code Conversion for Pseudo-Random Multi-Valued Sequences	79
5.5	Pseudo-Random Multi-Valued Array	83
CHAPTER 6	**MODEL-BASED TACTILE OBJECT RECOGNITION USING PSEUDO-RANDOM ENCODING: EXPERIMENTAL RESULTS**	
6.1	Tactile Recognition of Pseudo-Random Binary Symbols	93
6.2	Tactile Recognition of Multi-Valued Pseudo-random Symbols	99
6.3	Model-Based Recognition of The Encoded Objects	107
6.4	Summary	115
CHAPTER 7	**CONCLUSIONS, CONTRIBUTIONS, AND FUTURE RESEARCH DIRECTIONS**	
7.0	Conclusions	116
7.1	Contributions	117
7.2	Future Research Directions	118

APPENDICES **Page**

APPENDIX I	Biological Aspects of Human Tactile Sensing	A1.1
APPENDIX 2	Current Tactile Sensor Technology	A2.1
APPENDIX 3	Extraction of 3d Linear Object Features Using Hough Transform	A3.1
APPENDIX 4	Computer Program Source Code for Tactile Recognition	A4.1
APPENDIX 5	Overview of Chinese Remainder Theorem	A5.1
REFERENCES		REF.1

LIST OF FIGURES

		Page
Figure 2.1:	Model-based Object Recognition System (adapted from [Lee and Fu [2.3])	15
Figure 2.2	Pseudo-Random Binary Array Symbols embossed on object surfaces	20
Figure 3.1	Contact-type measuring instrument: tactile sensor	23
Figure 3.2	Object and sensor direction	29
Figure 4.1	Robotic tactile system	33
Figure 4.2	Link coordinate diagram of the five-axis manipulator	34
Figure 4.3	Instrumented passive-compliant wrist	35
Figure 4.4	Functional diagram of the wrist	36
Figure 4.5	Tactile sensor interface to the host computer	39
Figure 4.6	Experimental diagram showing the strain/stress single tab response of the elastic overlay	41
Figure 4.7	Tactile sensor with the tab shaped elastic overlay	44
Figure 4.8:	Protruding round tabs of the elastic overlay provide spatial sampling	45
Figure 4.9	The 16-by-16 (median filtered) tactile image of a circular washer produced by a tactile sensor with a tab-shaped elastic overlay	47
Figure 4.10	The 16-by-16 (median filtered) tactile image of the same washer produced by a tactile sensor with an one-piece elastic overlay	48

Figure 4.11	Overlapped tactile scanning strategy	52
Figure 4.12	A Composite 32-by-16 tactile Image. (a) After Median Filtering. (b) After The Application Of Sobel Gradient Operators	54
Figure 4.13	Position Of The Edge Recovered (Estimation) From Figure 4.16(b) Compared With The Actual Position Of The Investigated Object Edge	56
Figure 5.1	PRBS generation using a modulo-two feedback shift register	60
Table 5.1:	Direct and reverse feedback equations for some binary shift registers	61
Figure 5.2	Serial-parallel pseudorandom/natural code conversion	65
Figure 5.3	Relative time performance of different pseudorandom/natural code conversion methods.	66 62
Figure 5.4	Serial-parallel code conversion costs as a function of the distance between milestone	67
Table 5.2	The Optimal number of milestones in serial-parallel code conversion as a function of the relative equipment/temporal cost and the measurement resolution (adapted from [5.20]}	68
Figure 5.5	Folding of the PRBS to produce a PRBA	70
Figure 5.6	Generation of a $(2^{k1} - 1)$-by-$(2^{k2} - 1)$ Pseudo-Random Product Array (PRPA)	72
Figure 5.7	The $\wp$ "product law"	73
Figure 5.8	A 15-by-63 quaternary PRPA {A(i,j)= SX(i) $\wp$ SY(j)|i=0,1,...,14;j=0,1,...,62}	74
Table 5.3	Primitive polynomials over GF(q)= {0,1,A,..., A^{q-2}}	77

Figure 5.9	Serial-parallel Code Conversion for Pseudo-Random Multi-Value Sequence	81
Figure 5.10	An example of a PRMVS (with $q=4$, $n=5$) of length 1024 with milestones marked at every 32^{th}. The milestones are bold faced and underlined.	82
Figure 5.11	PRMVPA formed by the cross product of A and B.	85
Figure 6.1	Polyhedral model of the PRA encoded object and its 3-D reference frame.	89
Figure 6.2	The geometric models of two objects (indexed k and r) are unfolded on the encoding PRA.	90
Figure 6.3	The recovered PRA window allows identification of the specific model face to which that window belongs	92
Figure 6.4	The two symbols used to mark the binary values "0" and "1" within the PRBA	94
Figure 6.5	Experimental tactile image of binary symbol "0"	95
Figure 6.6	Experimental tactile image of binary symbol "1"	95
Figure 6.7	Features describing a binary symbol	96
Figure 6.8	Acceptance test for neighbouring symbols in the case of PRBA encoding	98
Figure 6.9	A 15-by-15 PRMVPA obtained by multiplying two PRMVS defined over GF(4)	100
Figure 6.10	Braille-like encoded object surface used for experiments on tactile object recognition	101
Figure 6.11	The experimental tactile image of the Braille-like encoded surface	102
Figure 6.12	The four symbols used to mark the GF(4) elements within the PRMVPA	103
Figure 6.13	Experimental tactile images of the four symbols (0, 1, A, A^2)	106

Figure 6.14 Winged-edge representation for parallelepiped and cube. **Note:** In the case of a cube all faces are square (S). For a parallelepiped the faces could be either rectangular with unequal sides (R) or square (S) with the condition that pairs of faces marked with the same number of asterisks (*) are similar and at least one pair is of type R 108

Figure 6.15 Winged-edge representation for the triangular prism 109

Figure 6.16 File relationship in the data base 110

Figure 6.17 Experimental results illustrating tactile model-based recognition of a cube 112

Figure 6.18 Experimental results illustrating tactile model-based recognition of a prism 113

Figure 6.19 Experimental results illustrating tactile model-based recognition of a triangular prism 114

Figure 7.1 Artificial skin 120

Chapter 1

Introduction

1.0 Background

Tactile sensing technology currently is being developed to enhance the sensing capability of robots by improving their ability to identify and measure the position and orientation of objects, during robotic manipulation operations.

Robotic tactile sensing essentially emulates the mechanisms of human tactile perception. This is a complex process with two distinct modes: passive touch, produced by the "cutaneous" field of "mechanoreceptive" transducers covered by an elastic material which provides contact force, contact geometric profile, and temperature information, and active tactile sensing, which integrates cutaneous sensory data and "kinesthetic" information (i.e. limb/joint positions and the states of motor muscles).

1.1 Comments on the Human Sense of Touch From an Engineering Perspective

A large body of literature in biology is available on the subject of human tactile perception. Many experiments have been carried out in order to evaluate the performance of the human sense of touch.

Human tactile sensing is the result of a deliberate exploratory act of perception, rather than mere passive sensing, [1.1], [1.2], [1.3]. This integral approach of tactile sensing is clearly formulated in [1.3] as follows:

> *"An artificial tactile sensing system based on this anthropomorphic model should be provided with sensory and motor organs, along with low-level and high-level processing and control capabilities. ... Perception requires skilled use of manipulative functions and implies purposive behaviour. In particular, tactile exploration is based on hand and finger movements directed to elicit tactile stimuli that, when properly organised, allow the brain to associate the explored object with an element of a database of existing models, or to construct a new model."*

The complex human tactile (`haptic') perception constitutes two components: which are **cutaneous** (providing force, touch, slip, and temperature information) and **kinesthetic** response characteristics (including sensing of the limb/joint position and command of muscle motor action). [1.2]

Our efforts to develop an artificial tactile perception system will certainly benefit from the study of the biological

structure, tactile sensing mechanisms, and cortical processing of the human skin. This will help us to understand the intrinsic character of tactile perception in order to emulate artificial touch sensing.

It should be noted that while anthropomorphic considerations may serve as inspiration to the engineering of an artificial integrated tactile perception system; too much enthusiasm for the biology of the skin could jeopardize the functionality of an engineered tactile sensor. A selective overview of the biological and physiological functions of the skin will help and lead us to the successful development of the sensing system.

The human sensory system has a highly structured organization, the details of which are outside the scope of this thesis. However, some predominant features are discussed further: Sensory functions at the lowest level are carried out by receptors responding to various stimuli; such as mechanical, optical, and thermal. These receptors may be seen as specialized nervous cells or neurons. Neurons specialized in the transmission of information are collected in fibers, which in turn form pathways which eventually reach the central nervous system. It should be noted that despite specialization, the function of every single neuron is multiple : metabolic, computation, and transmission.

Further details are presented in Appendix 1.

Another interesting aspect was noted in the work of Moore [1.4] whose purpose was to find a relationship between the visco-elastic characteristics of the skin and tissues and the vibration applied. These characteristics are related to the impedance of the surface of the body. The resistance per unit area remains relatively constant over a range of frequencies. The reactance is frequency-dependent, it is determined by the elasticity and the mass of the tissues. At low frequencies, the elasticity prevails but for high frequencies the mass component has more influence.

There are evidences that both tactile and kinesthetic senses are combined to provide for haptic perception. This perception is obtained through a variety of cues coming from phenomenal numerous mechanoreceptors distributed throughout the structure of the system (joints, muscles, tendons, derma, skin). Using these cues, one can experience the simultaneous sensation of position of, and forces applied to, an object touched by human fingers or hands. As for any human perceptual channels, it is generally accepted that a stable representation in the central nervous system of the motion of a held object results from a combination of several channels, in particular position and force.

Experiments have shown that tactile perception of the environment can reach a very high level of accuracy, whereas the perception of position or movement is rather approximate.

There are various cutaneous mechanoreceptors located in the outer layers of the skin. The free nerve endings are the most numerous and may play an active role in the perception of pain, cold and warmth. Most of the receptors are encapsulated. They differ in the hairy and glabrous skin as described in Appendix A1.3.

Each class of mechanoreceptors respond preferentially over a frequency range : the minimum threshold (highest sensitivity) for the Pacinian Corpuscles (PC) units is around 250-300 Hz but they respond from 30 Hz to very high frequencies. The Rapidly Adapting, (RA) units effective frequency range is between 10 and 200 Hz, with more sensitivity below 100 Hz. The Slowly Adapting (SA) units respond at low frequencies. Further details of PC, RA and SA are explained in Appendix 1. The sensitivity threshold for touch is about 6-20 μm for the depth of indentation on the hand. A relationship between the threshold value and the frequency reveals that RA's units are likely to be responsible for the perception under 40-50 Hz and the PCs units are responsible of the detection at higher frequencies. The receptive fields characteristics (which are described in Appendix 1) are larger for the PC receptors (their density is

lower). The previous remark leads us to conclude that when the frequency increases, our ability to detect the localization and the direction of a vibration decreases. The experiments characterizing the detection of the direction of a movement on the skin showed that the SA units are responsible for the detection of stretch and they present a force vector (for a 20 g indentation force, the stretch is easily detected). The PCs and RAs units are more likely to detect the relative motion of slip. The stretch enhances the ability to detect movement on the skin surface, at 1 cm s^{-1} , the threshold is 0.6 mm.

The frequencies of interest for kinesthetics are much lower than for the tactile perception. It is however difficult to quote a clear value for the frequency above which the number of degree of freedom can be reduced. For the tactile sensor, there will be multiple simultaneous contact areas and it can be expected that the performance will depend upon the configuration of the sensor.

1.2 Tactile Sensing Technology Perspective

In 1980, L.D. Harmon [1.1] defined artificial tactile sensing (taction) as the continuous sensing of variable contact forces over an area within which there is a spatial resolution.

A set of requirements for a mature tactile sensor was specified as follows: spatial resolution of 2 mm, bandwidth of 0 to 100 Hz, moderate linearity (nonlinearity to be corrected by the further signal processing), low hysteresis, compliance and robustness.

Two main topical areas are usually considered as distinct when discussing tactile sensing: a) the operating principle and the fabrication technology for tactile sensors (transducers); b) the sensory data processing and the pattern recognition for tactile perception.

The operating principle for tactile sensors have been categorized into four types: conductive elastomer, optoelectronic, strain gauge and piezoelectric.

Conductive elastomer sensor operation is based on changes in electrical resistance caused by pressure. Noise, nonlinearity, hysteresis, fatigue, long time constants, low sensitivity, and drift are serious drawbacks of these sensors [1.5].

Optoelectronic sensors rely on the modulation of the light intensity by the mechanical deformation of an elastic material. These sensors have a very high sensitivity and they are less sensitive than electromagnetic tactile sensors which are

commercially available today [1.6].

Strain gauge sensors also are based on a change in electrical resistance caused by pressure [1.2], [1.7]. This is a proven technology. These sensors have high sensitivity, linearity, reliability, versatility, without the hysteresis, long-term creep and fatigue problems of elastomer sensors. They are available in a multitude of shapes and sizes. Recent development has made available miniature solid state silicon strain gauges that have a very promising high resolution, but which are still fragile and stiff. Both commercial strain gauge tactile sensors and solid state silicon strain gauges are already available.

Piezoelectric sensors utilize polymeric materials that generate an electric charge in response to an applied force. Piezoelectric polymers represent a very promising technology [1.6]: they can be arranged in very high resolution arrays and are flexible, rugged, and inexpensive. They are also commercially available. However, further development of the sensors is required for practical applications.

Which type is best suited for robotic application is not yet certain. The operation of optimal sensor principle may depend upon the specific functional requirements.

The second topic addresses sensory data processing and pattern recognition for tactile perception. Usually, such a discussion is limited to general, low-level aspects: electronic circuitry for measurement, scanning, amplification, signal conditioning, analog/digital conversion, and simple pattern recognition techniques borrowed from image processing.

1.3 Objectives

The motivation of this thesis comes from space-based robotics applications. The cameras mounted on the Space Shuttle Manipulator are frequently blinded by the bright light in certain orientation, especially, when they are directly facing the sun. The same situation will be experienced by the cameras which will be installed on the manipulator system in the International Space Station. The threat of instantaneous loss of information on the orientation and position of an object to be docked (or undocked) is highly possible. In order to complement the vision capability, studies on tactile and proximity sensors have been initiated by the Strategic Technology on Automation and Robotics (STEAR) section of the Canadian Space Station Program Office. This research was done by the Canadian Space Agency focuses on the hardware design and the implementation of the sensors itself without looking into the sensor working environment and computation power available for the sensor from

the Station.

1.4 Summary of the Main Contributions

This thesis presents an original solution to the blind tactile object recognition based on a permanent Braille-like encoding of object surface. The development of the tactile sensing system and of the related object recognition pseudo-random symbols embossed on objects has resulted in a number of contributions as follows:

(a) Theoretical and experimental studies to investigate the effects of the elastic layer mounted on top of the force-sensitive tactile transducer array. A novel tab shaped design was developed to reduce the cross-talk in the elastic layer of a high resolution and sampling tactile sensor.

(b) Development of an instrumented passive compliant wrist which allows the experimental robotic tactile sensing system to explore more efficiently the surface of the 3-D objects.

(c) Correlation technique for the integration of a sequence of local tactile images into a global tactile image of the explored object surface.

(d) Model-based object recognition method using a Braille-like pseudo-random encoding of object surfaces. Tactile object recognition is reduced to the identification and position recovery of a very small set of symbol types embossed on object surfaces.

(e) Extension of the pseudo-random binary code conversion methods to pseudo-random multi-valued sequences and arrays allowing a higher encoding resolution than the Pseudo-Random Binary Arrays.

1.5 Thesis Outline

Chapter 2 provides a short review of model-based object recognition in general and introduces the model-based tactile recognition of 3-D objects using pseudo-encoding.

Chapter 3 discussed specific problems of the tactile sensing and its application to object recognition.

Chapter 4 presents the development of the experimental tactile sensing system which includes the instrumented passive-compliant wrist, the design of the tactile sensing probe and its elastic overlay, the correlation technique for the integration of the global tactile images acquired during object surface exploration, and tactile signal processing methods for object features.

Chapter 5 presents the basics of pseudo-random binary sequences and arrays. Consequently, properties of pseudo-random multi-value sequences and arrays are discussed. An original code conversion method based on the window properties of a pseudo-random multi-valued sequences is discussed. An application of this code version is finally presented for absolute index recovery in pseudo-random multi-valued product arrays.

Chapter 6 presents the original pseudo-random encoding paradigm for model based tactile object recognition. Experimental results illustrate the performance of the new blind, model-based tactile recognition method for objects having their surface embossed with Braille-like pseudo-random binary arrays and pseudo-random multi-valued array.

Chapter 7 presents conclusions of the research and suggests new directions for future research.

Chapter 2

Model-Based Object Recognition

2.0 Introduction

Object recognition means not only the ability to identify the object according to its shape but also the ability to find the orientation and location of this object relative to a given reference co-ordinate system.

This chapter first will present certain general rules of the object modelling process which can be used for any type of sensor object recognition. Model-based object recognition is based on information acquired from the real world by sensors. Sensor data are then processed and object features are extracted to allow for the recognition of the object from a data base containing feature-based description models of all given objects. It is also assumed that the physical object maintains its dimensions, and its shape will not change. Model-based object recognition can be seen as matching the pattern of the measured features with the model patterns stored in the data base.

As summarized in [2.1], any object recognition system of practical value should have the capacity of examining; (1)

complicated real objects, (2) objects with different orientation and location, (3) noisy sensed data, (4) analysis of the scene in a fast and correct manner, (5) the process to interpret the sensed data with respect to the model, and (6) errors such as miss, false alarm, location and orientation.

2.1 Model-Based Object Recognition

Many researchers define the model-based object recognition as a two stage process: learning and matching [2.2]. In the learning stage, a description model is constructed for each object to be recognized. This description should allow for some descriptor variation as it may exist in the real world. In the matching stage, the object features recovered from sensor data are compared against the model features stored in the object-model data base. If no match occurs, an update function will add the new feature-based object description to the model data base.

Lee and Fu [2.3] have proposed a general interactive model-based 3-D object recognition system. The system contains three basic steps: (i) object description or model generation, (ii) model retrieval and (iii) model verification. The input sensor data is first processed by the "Primitive and Relation Finder" module. If there is no match with any stored model, the input data will be sent to the "Primitive and Relation Extractor"

where the object features will be extracted. Then these features will be grouped by the "Primitive and Relation Organizer" module and stored as a new model in the "Knowledge Base for Object Model" module. If a match is found, the input data will be associated with an existing model and the "Class Interpreter" will indicate the specific model corresponding to the input data.

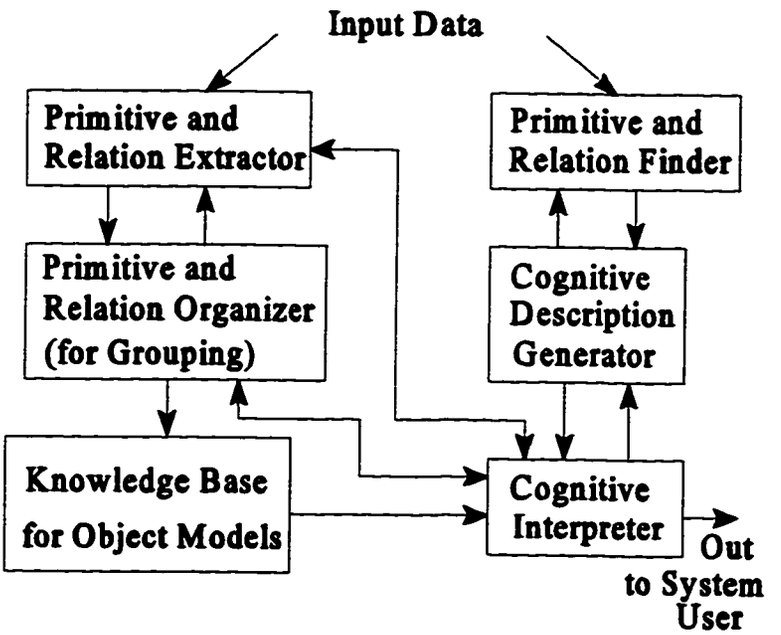


Figure 2.1: Model-based Object Recognition System (adapted from [Lee and Fu [2.3])

The functions of the different modules in the diagram are summarized as follows:

(a) Primitive and Relation Extractor:

A low level knowledge module which responds to a request from the Primitive and Relation Organizer and extracts primitive information from the input data and relation information for the Primitive and Relation Organizer module.

(b) Primitive and Relation Organizer:

In response to a request from the Cognitive Interpreter, this module uses the information from the Primitive and Relation Extractor to produce an organized set of primitives and relations.

(c) Knowledge Base for Object Models:

Using an associative memory mechanism, this module produces candidate object models compatible with the set of primitives and relations received from Primitive and Relation Organizer.

(d) Cognitive Interpreter:

A decision making module which takes primitive information

from Primitive and Relation Extractor, the organized sets of primitives and relations from Primitive and Relation Organizer, and from Knowledge Base for Object Models, and the object evaluations from the Cognitive Description Generator module. It produces object models to be further verified by the Cognitive Description Generator to distinguish objects with similar features. If further reorganization of the object primitives and relations is required, a request will be sent to Primitive and Relation Organizer. When the decision making process stops, this module generates the final output image description for the system user.

(e) Cognitive Description Generator:

The Generator takes the object models to be verified from Cognitive Interpreter and the results of verification search from the Primitive and Relation Finder. It generates an request to find a particular primitive or relation by the Primitive and Relation Finder. It also provides the evaluation of an object model for Cognitive Interpreter.

(f) Primitive and Relation Finder:

It is a special purpose module which takes the input sensed data and the request from Cognitive Description Generator to produce the results from verification search in the Cognitive Interpreter module.

2.2 Model-Based Tactile Recognition Using Pseudo-Random Encoding of Object Surfaces

It is generally believed that input data arrays with higher dimensions will provide better pattern recognition of the object features. However, the increased number of measurement may burden the data acquisition system and require expensive computer hardware. This will make real time operation impractical, and sometimes impossible to response and control.

Some researchers [2.4] have proposed an optimal encoding method to select features of an object to minimize pattern recognition computation time. The method reduces the complexity and overall equipment cost, however, it still requires considerable computation time which makes real time operation impractical. On the other hand, excessive reduction of model features may lead to wrong classification, since errors are always present in any object recognition method.

This thesis will present *an original model-based object recognition method by tactile sensing based on a permanent Pseudo Random Array (PRA) encoding of the object surfaces.* The PRA code elements are represented by special symbols which are Braille-like embossed on the object surface as shown in Figure

2.2. The shape of the embossed symbols is specially designed for easy tactile recognition as shown in Figure 2.2. These symbols are embossed on the surfaces of all objects which have their geometric models mapped on the encoding PRA frame. *This compact encoding requires only one symbol per quantization interval.* This is a novel application of the pseudo-random encoding which we have previously used for visual object recognition [2.5, 2.6]. This method will simplify considerably the tactile object recognition of 3-D objects. The sensor data processing is reduced in this case to the recognition of a reduced set of symbol types embossed on object surfaces, and therefore, it is significantly faster and cost effective. Shorter computation time will result enabling real time object recognition.

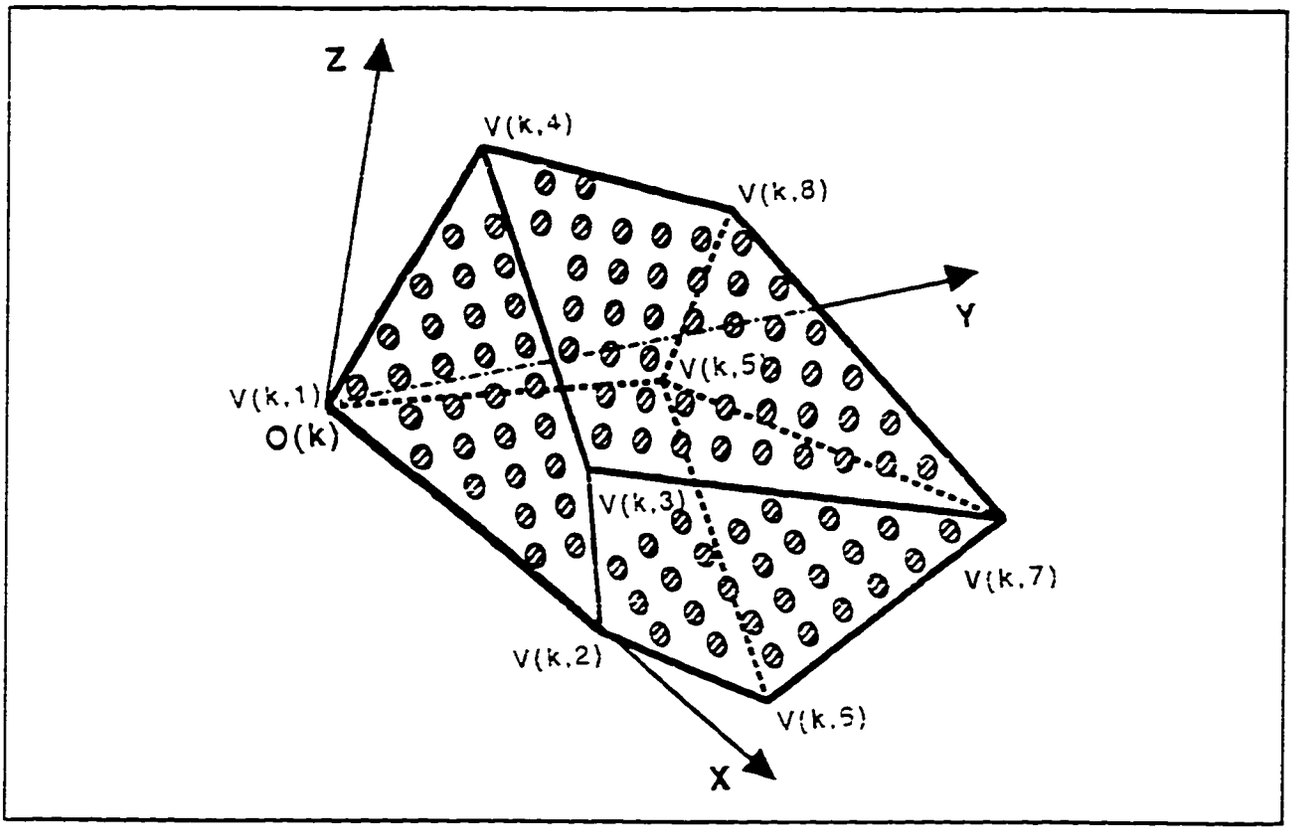


Figure 2.2 **Pseudo-random array symbols embossed on object surfaces**

Chapter 3

Tactile Object Recognition

3.0 Introduction

Tactile sensing provides a powerful tool for contact-type object recognition and object manipulation functions. The tactile functions can be divided into three general categories:

(1) Simple pressure determination and slip sensing: These capabilities are necessary for the most common industrial applications involving workpieces without causing damage.

(2) Determination of object orientation and position: This is required for more complex applications in unstructured-environments, such as picking an object from a bin, orienting it into a new position, and assembling it with other objects.

(3) Object identification or recognition: This function is necessary for advanced applications in which a robot may be working in a relatively unknown environment (such as undersea and space explorations)

and may be required to classify or identify an object based solely on tactile information.

Each of these applications involves a different design approach and different computational requirements. The first application is technically the simplest to implement, and industry has found several workable approaches. The latter two are more challenging, and it is the focus of most of the current research.

3.1 Tactile Sensing

Tactile sensing generally refers to skin-like properties where force-sensitive or displacement-sensitive probes are capable of reporting signals with different level of values according to the applied force, and the features of the object being touched.

Tactile sensing may be viewed as two-step process: transduction and data processing. Transduction occurs when the features of an object being examined are converted from signals of some physical form (pressure, displacement, temperature, etc,...) into electrical signals. Data processing then interprets these signals to obtain useful information about the features of interest.

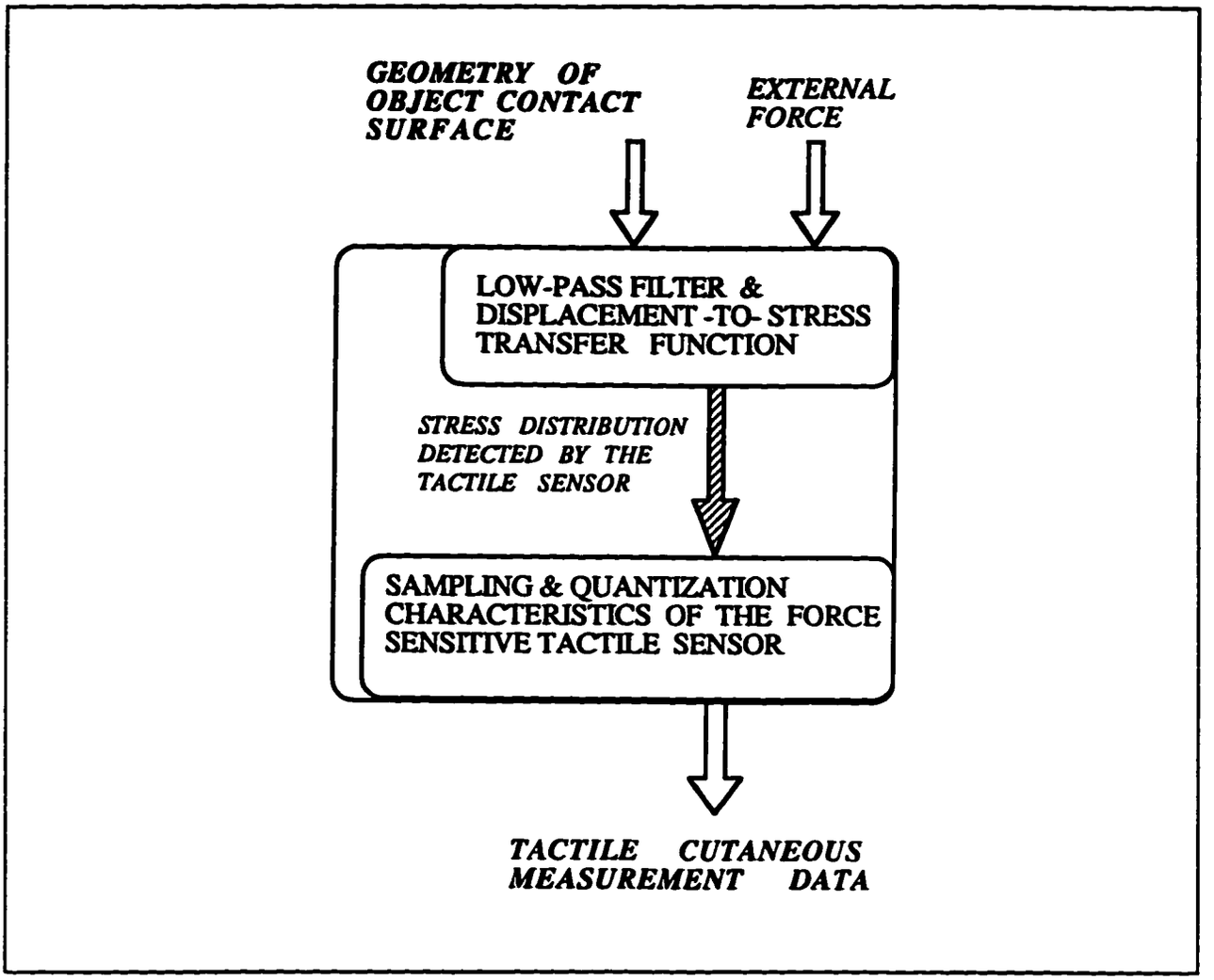


Figure 3.1 Contact-type measuring instrument: tactile sensor

As a contact-type measuring instrument, the tactile sensor requires an external force to impose the contact with the touched object (Figure 3.1). Under the pressure of this force, the local geometric profile of the object's surface indents an elastic overlay. Because of the relatively high stress induced

at the overlay's surface it is preferable to measure the stress beneath it rather than on its surface, [3.1].

Since spatial sampling of the tactile image introduces uncertainty in the edge location, and may result in loss of metric properties of object features during geometric transformations, [3.2] there is the requirement to increase, as much as possible, the 2-D sampling speed of tactile sensors which, for technical reasons, have a poorer sampling resolution than vision sensors. It is important to remark that the overall sampling resolution of a tactile sensor is the integral of both the pressure transducer's and elastic overlay's sampling characteristics.

Details on different types of tactile sensors are presented in Appendix 2. The most practical type is the matrix tactile sensor, composed of an array of sensitive elements. Compared with the simplest one-point tactile probe, the matrix type tactile sensor has the advantage of parallelism allowing for more surface probing points to be sensed at the same time.

A matrix tactile sensor produces an array of measurements that are a function of the pressure distribution over the sensor. The exact relationship of these measurements with reject to properties of the object is very complex and depends on the particular sensor design [3.3, 3.4, 3.5]. In practice, the

presence of electrical noise, vibrations, limited resolution, and complex modelling process of an elastic overlay make it difficult to determine the exact input/output transfer function of the tactile sensor. Because of these sensor technology problems, more elaborate tactile object recognition techniques should be developed.

Several tactile object recognition methods have been proposed to build a partial description from the sensed data and to match this description to other given object model. Each approach differs on the type of descriptors which are used.

One object recognition approach emulates the feature-based approach which has successfully been used in vision systems. The pattern of tactile measurements can be used to identify global object features, such as holes, edges, vertices, and pits [3.5, 3.4, 3.6]. These features may be difficult to be located and identified from objects which are significantly larger in size than the sensor.

Another approach attempts to *build surface models*, either from pressure distributions on matrix sensors [3.7], or from the displacements of an array of needle-like sensors [3.8, 3.9]. These methods face the rather complex problem of matching local surface descriptions obtained from the data with those of a model. A similar approach that simplifies matching is to build a representation of subsets of an object's cross sections and

match them to object models [3.10, 3.11]. The method described in [3.11] is particularly interesting in this respect as it represents both objects and data as a sequence of unit surface tangents indexed by angle. This representation is invariant with translations and simply shifts with rotation, thus simplifying the matching process. The data used for recognition and localization are estimates of the position and normal vector of a few points on the surface of the touched object. In many cases, the tactile sensors are used only to detect contact; it is the relative position of sensors to the objects that is the actual source of data. The method described in this chapter also uses relative positions to complement the tactile data provided by the sensor.

On the basis of sensor readings, some points on the sensor can be identified as being in contact with external objects. In real sensors, there is some uncertainty as to the actual contact point, but its position can be constrained within some small area. If the sensor's shape and location in space are known, one can determine the position of some point on the touched object, to within some uncertainty volume.

At the contact points, the known surface normal to the sensor must be the negative of the object's surface normal at that point. This is exactly true only for both a rigid sensor and object in the absence of measurement error. Under weaker but

still useful constraints, the surface normal can still be recovered.

Position and normal information can be obtained reliably only if the tactile sensors have high spatial resolution relative to features of object; as for instance the 16 × 16 array sensor (with 256 sensitive points) on an area of one inch square.

The first approach is that much of the existing work on tactile object recognition is based on *statistical pattern recognition* for classification. Some researchers have relied on the contact patterns on matrix sensors for object recognition. [3.12, 3.13]. The rational motivating this line of research has been that the individual (local) data elements are not repeatable and only their statistical parameters can be counted on. The measured statistics are then compared to reference statistics for the known object types. The resulting methods are limited to discriminations among a few simple types of objects.

A second approach to *statistical tactile recognition* uses patterns of the positions in which the fingers of articulated dexterous hands come to rest against the object. A number of researchers have used the joint angles of the fingers as their primary data [3.14, 3.15, 3.16, 3.17]. A related approach classifies the pattern of activation of on-off contacts placed

on the finger links [3.10].

3.2 Exploratory Object Sensing

Tactile sensors, by their very nature, provide information over a relatively small area of an object. This limitation is overcome either by mechanically exploring the object, which can be slow; or by using multiple sensors.

To explore an object with a tactile sensor, it is necessary to move the sensor along the surface and record the sensory data. After the whole surface is explored, the input sensory data is put together in order to complete the image of object surface. In the following section, we discuss how multiple sensor data are integrated to form the final image.

3.2.1 Problem Definition

The objective of the model-based tactile object recognition, is to identify an object from among a set of known objects and to measure its position and orientation relative to a gripper which is equipped with tactile sensors.

During the exploration process, sensor paths usually are

parallel to, but possibly at different normal distances from a prespecified support plane [Figure 3.2]. The gripper frame and the positions of the sensor relative to the gripper frame are known to high accuracy. Each sensor image is processed to obtain one contact point known to be on the object surface, within some error bound, and a range of feasible surface normals at the point of contact.

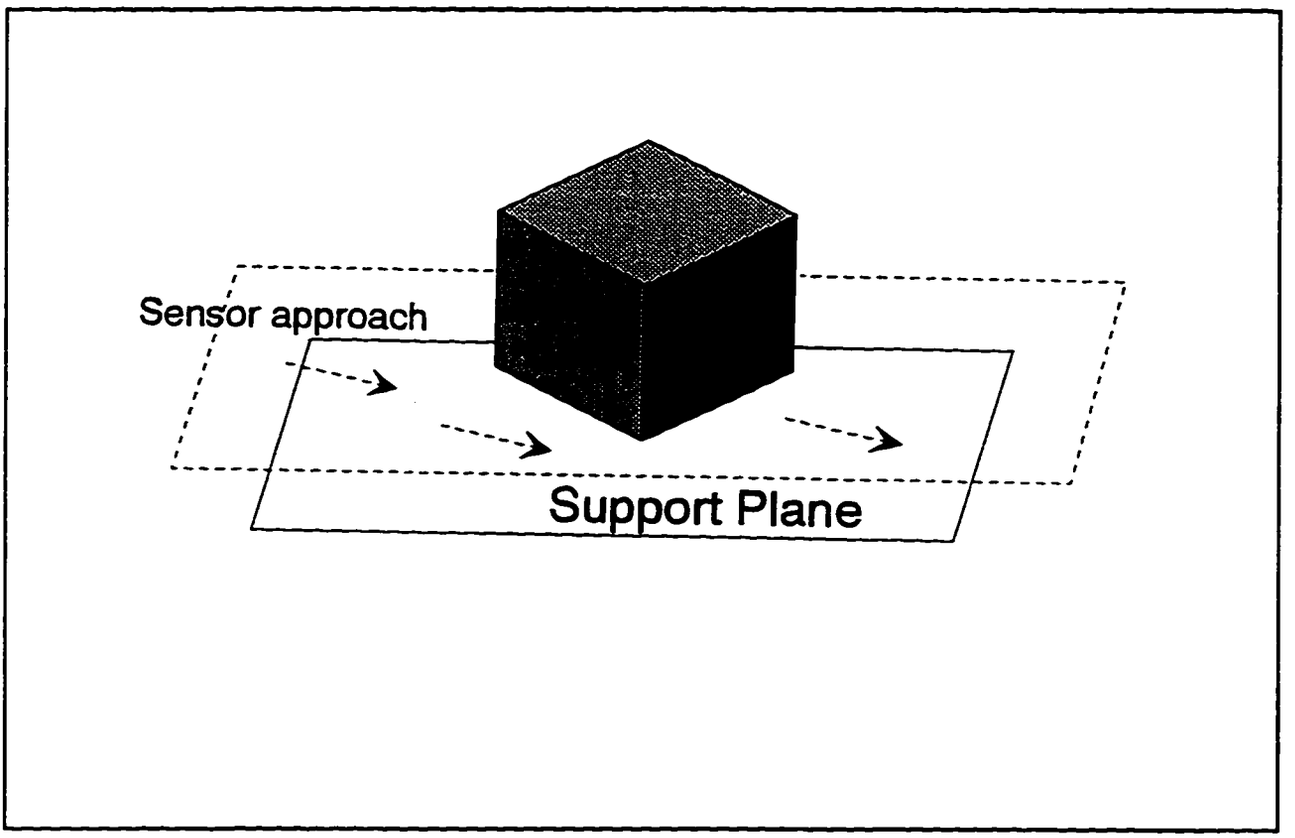


Figure 3.2 Object and sensor direction

To simplify the data-base used in this thesis, the explored

object is assumed to be polyhedral. The object is supposed to stand in a stable state on a support plane. Hence the object has three degrees of positional freedom, x, y, and θ (which is rotational angle of the object with respect to the normal vector to the plane where it sits), relative to the frame of the support plane. The object configuration is defined as the vector of parameters that uniquely specifies the position and orientation. The different stable states of the object are treated, conceptually, as if they were separate objects. This set of assumptions is similar to those used in many binary vision systems [3.18].

Limiting the number of degrees of positional freedom of the object relative to the gripper is the key limitation in this problem definition. If one can locate any planar surface on an object, e.g. by aligning a planar sensor with it, then the resulting localization problem is reduced to three degrees of freedom relative to the surface.

Chapter 4

Experimental Tactile Sensing System

4.0 Introduction

Tactile sensing is an emerging technology which is still in a laboratory stage. There are not yet available industrial tactile sensors with good resolution as object recognition requires. This chapter presents development aspects of an experimental tactile sensor system used for the blind tactile recognition of 3-D objects. All object surfaces are permanently encoded (using distinct geometric symbols embossed on object surfaces) with the terms of a pseudo-random array (PRA). Special properties of this array allows identification of the absolute coordinates of the touched surface area within the PRA. Knowing how the object models were originally mapped on the encoding array it is then possible to recover the identity of the touched object by a simple consultation to the data-base storing this mapping relationship.

4.1 Robot Tactile Sensing System

The tactile sensing system is shown in Figure 4.1 and consists of a commercial robot manipulator, specially designed instrumented compliant wrist and tactile sensor.

The five-axis manipulator has the kinematic diagram shown in figure 4.2 with the joint distances $(d_i | i = 1,2, \ldots 5)^T = (260.4, 0, 0, 0, 75)^T$ and link lengths $(a_i | i = 1, 2, \ldots, 5)^T = (0, 228.6, 228.6, 9.5, 0)^T$, in centimetres [4.1]. The arm matrix is

$$T_{base}^{gripper} = \begin{bmatrix} C_1C_{234}C_5+S_1S_5 & -C_1C_{234}S_5+S_1C_5 & -C_1S_{234} & C_1(a_2C_5a_3C_{23}a_4C_{234}-d_5S_{234}) \\ S_1C_{234}C_5-C_1C_5 & -S_1C_{234}S_5-C_1C_5 & -S_1S_{234} & S_1(a_2C_2+a_3C_{23}+a_4C_{234}-d_5S_{234}) \\ -S_{234}C_5 & S_{234}S_5 & -C_{234} & d_1-a_2S_2-a_3S_{23}-a_4S_{234}-d_5C_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where C_i is the notation for $cos\ (q_i)$, C_{ij} for $cos\ (q_i + q_j)$, C_{ijk} for $cos\ (q_i + q_j + q_k)$, S_i for $sin\ (q_i)$, S_{ij} for $sin\ (q_i + q_j)$, and S_{ijk} for $sin\ (q_i + q_j + q_k)$.

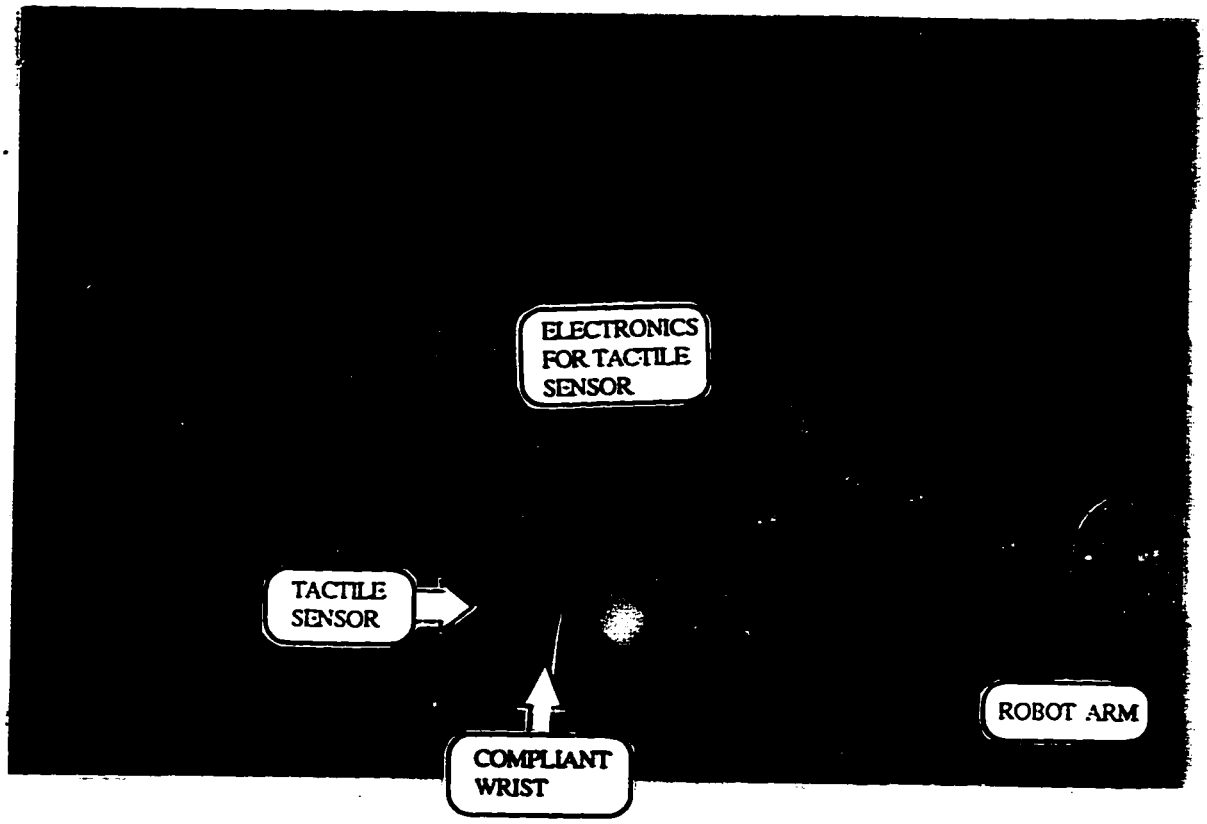


Figure 4.1 Robotic tactile system

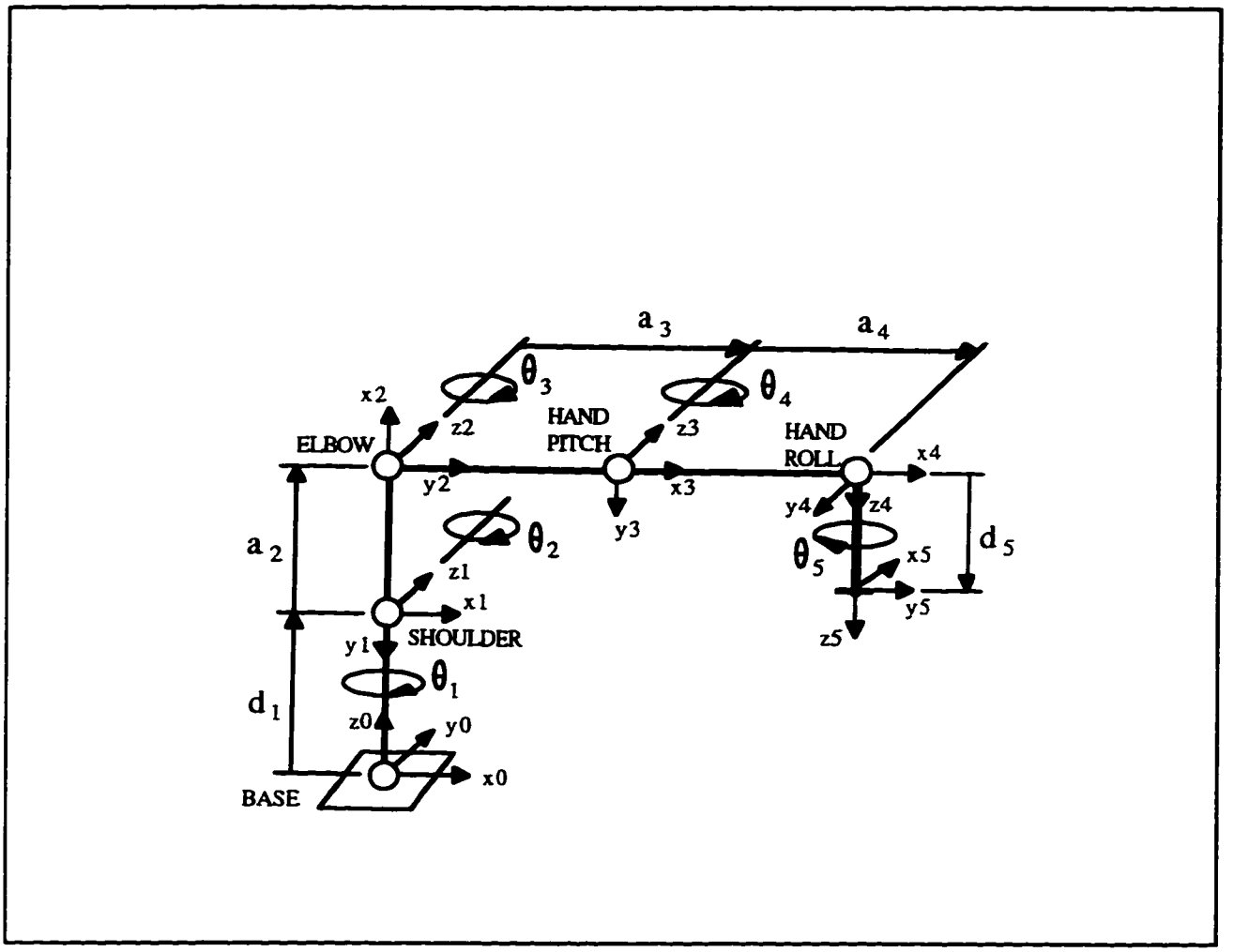


Figure 4.2 Link coordinate diagram of the five-axis manipulator

The Instrumented Passive Compliant Wrist

In tactile sensing, information is collected by placing the probe on the object surface. The amount of acquired information depends on how well the probe accommodates the constraints of

the local geometry of the object. An original instrumental passive compliant wrist, shown in figure 4.3, was developed in order to allow the tactile probe to comply to the object surface constraints [4.2].

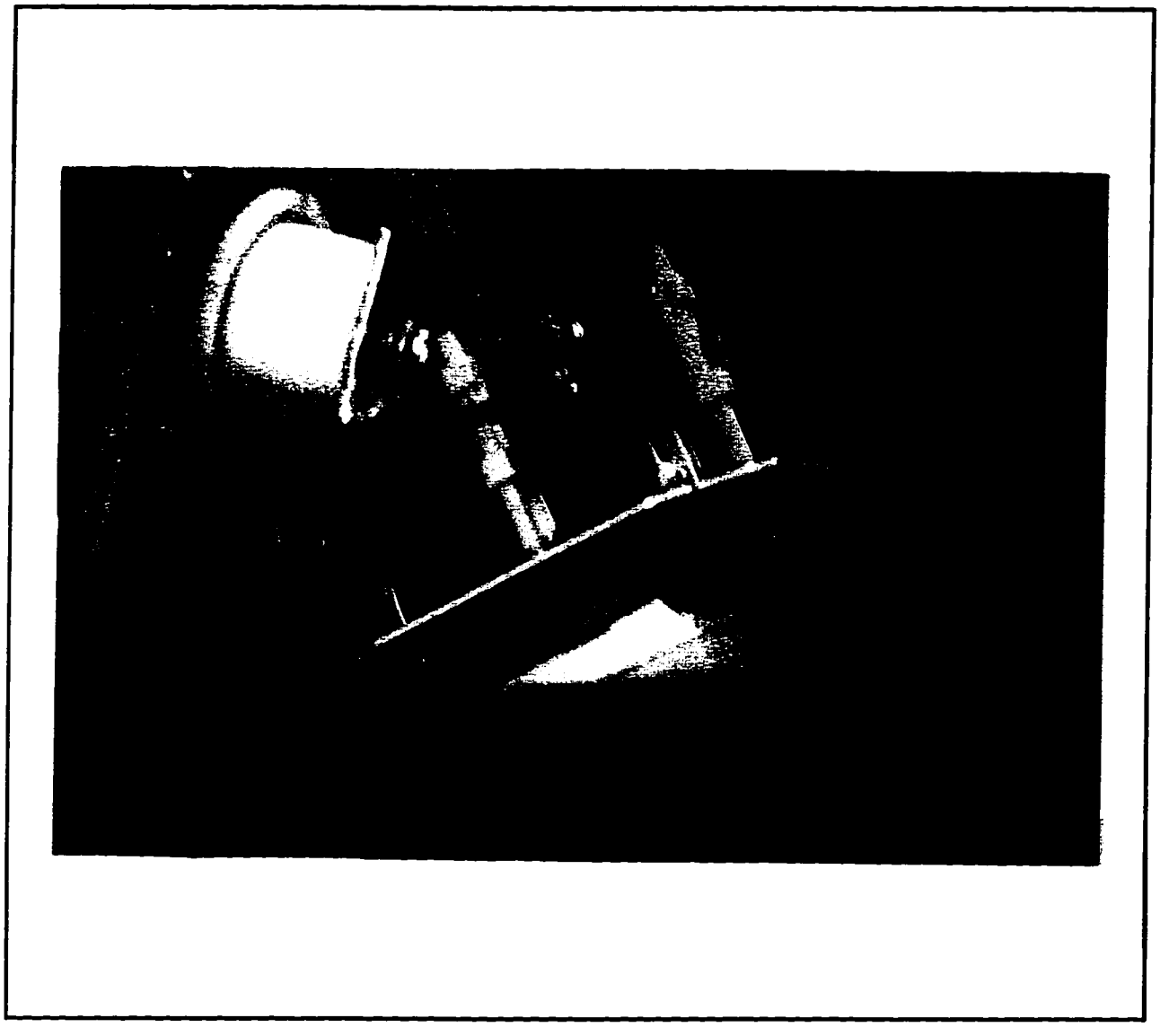


Figure 4.3 Instrumented passive-compliant wrist

As illustrated in Figure 4.4, the wrist has three degrees of freedom: two rotations (θx and θy) and one displacement (D_z). The ranges for these degrees of freedom are: $\theta x = \theta y = \pm 10^0$, and $OO_z' = 0$ to 10 mm.

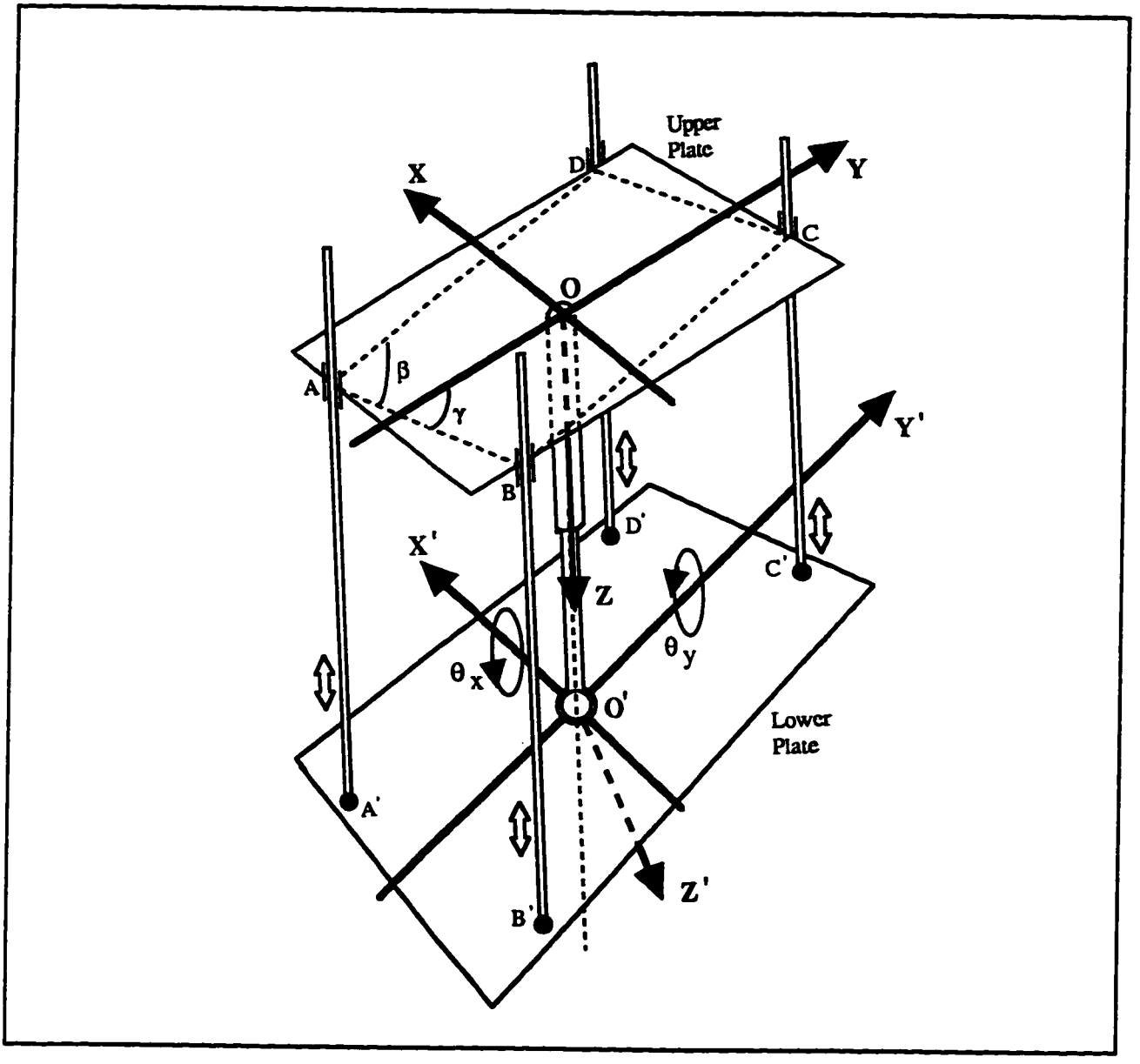


Figure 4.4 Functional diagram of the wrist

Four resistive displacement transducers measure the AA', BB', CC', and DD' distances between the two plates of the wrist, which are then used to update the wrist's homogeneous transformation matrix:

$$T_{base}^{probe} = \begin{bmatrix} C_2\cos\gamma & S_1S_2\cos\gamma+C_1\sin\gamma & C_1S_2\cos\gamma-S_1\sin\gamma & 0 \\ C_2\sin\gamma & -S_1S_2\sin\gamma+C_1\cos\gamma & -C_1S_2\sin\gamma-S_1\cos\gamma & 0 \\ -S_2 & S_1C_2 & C_1C_2 & (AA'+CC')/2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

In this matrix equation C_1 is the notation for cos (θx), C_2 for $cos\ (\theta y)$, S_1 for $sin\ (\theta x)$, and S_2 for $sin\ (\theta y)$. The current values of the two rotation angles θx and θy are calculated from the following system of equations:

$$tan\ (\theta x) = (AA' - BB' - CC' + DD')/(2 \times AB\ sin\ \gamma);$$

$$tan\ (\theta y) = (AA' + BB' - CC' - DD')/(2 \times AD \times sin\ \beta);$$

where γ and β are defined in Figure 4.4

Knowing the robot's kinematic transformations and measuring its joint angles and the relative displacements between the wrist's plates, the current position and orientation of the tactile probe relative to the robot base can be calculated.

4.2 Tactile Probe

The tactile probe is built around the Interlink force sensitive transducer consisting of a 16-by-16 array of force-sensitive resistor (FSR) elements arranged on a 6.5 cm^2 (one square inch) area.

The piezoresistive polymer film FSR elements exhibit exponentially decreasing electrical resistance with applied normal force. For a pressure range of 1 N/cm^2 to 100 N/cm^2, the element resistance will change by two orders of magnitude. These elements sense compressive forces and thus should be placed on a rigid backing.

The FSR transducer array is interfaced to a host computer through two analog multiplexers which select the row and the column of any desired element of the array, as shown in Figure 4.5. The resulting voltage is converted by an A/D converter on the host computer data acquisition board.

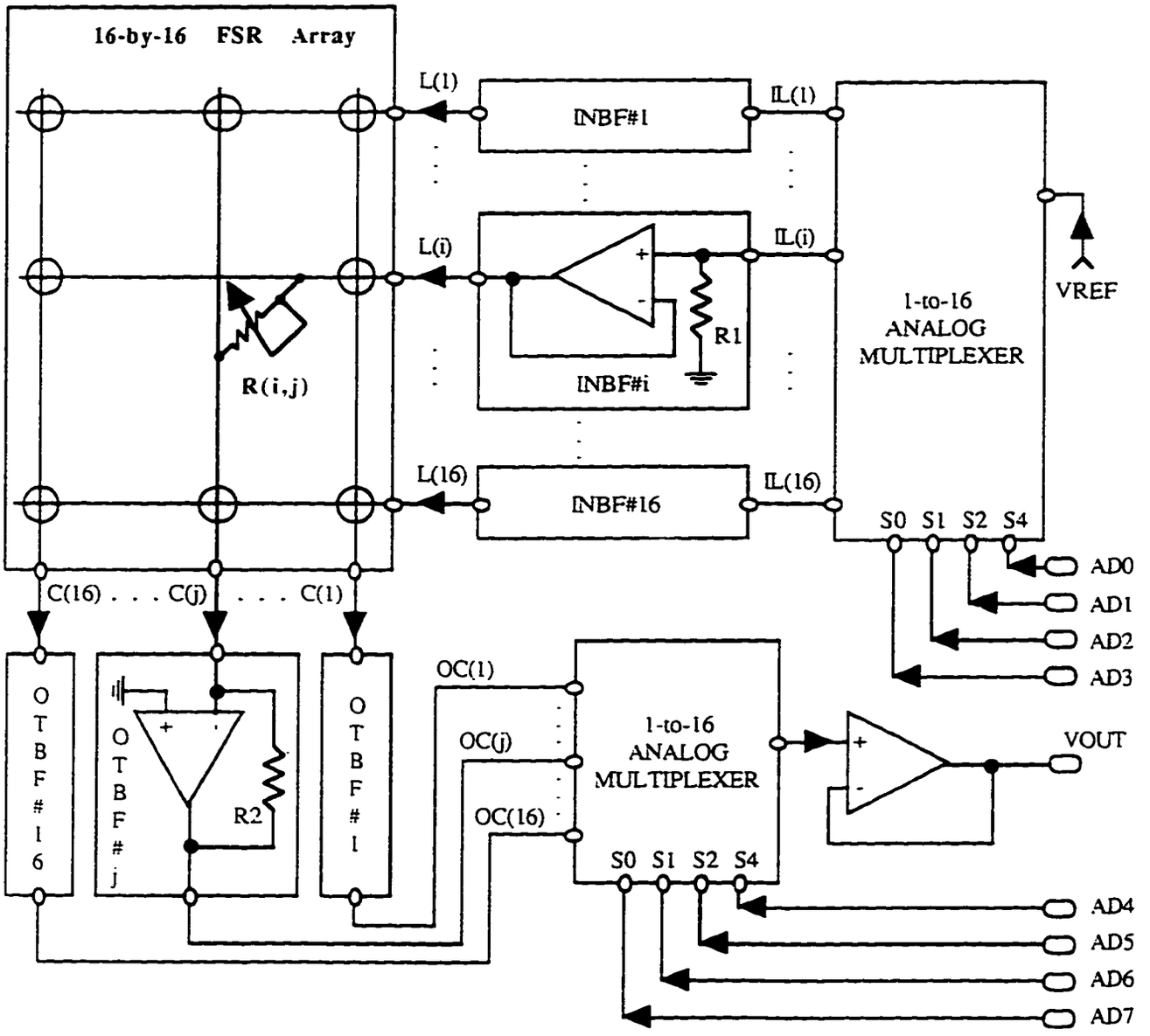


Figure 4.5 Tactile sensor interface to the host computer

4.2.1 Elastic Overlay Behaviour

The elastic overlay has a protective damping effect against impulsive contact forces, [4.3] and [4.4], and its elasticity returns to its shape when the sensor ceases to touch the object. There also are complex relationships between the geometric profile and the strain the overlay's elastic material may cause considerable distortion (blurring) in the sensing process if they are not properly compensated.

Figure 4.6 shows an experimental diagram of the strain/stress average response of a single tab of this elastic overlay.

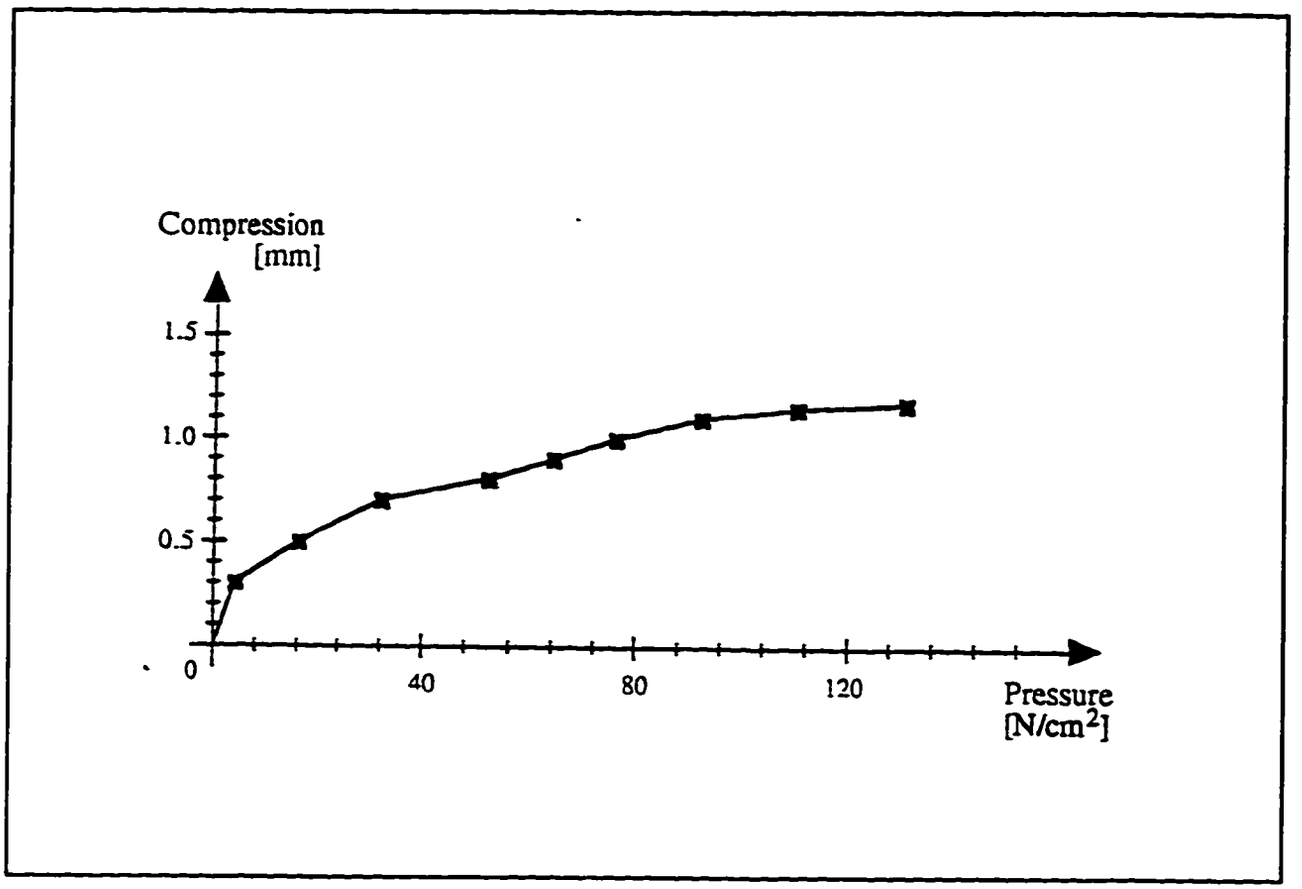


Figure 4.6 Experimental diagram showing the strain/stress single tab response of the elastic overlay

In principle, a geometrically fully constrained one-piece elastic pad cannot be compressed. Let us apply Hooke's law to a small unit cube of homogeneous elastic material, [4.5]:

$$\varepsilon_x = (\sigma_x - (\sigma_y + \sigma_z)/2)/E$$

$$\varepsilon_y = (\sigma_y - (\sigma_z + \sigma_x)/2)/E$$

$$\varepsilon_z = (\sigma_z - (\sigma_x + \sigma_y)/2)/E$$

where E is the modulus of elasticity, σ is the stress and ε is the strain.

Assume that the x-y plane is aligned with the pad surface, and that forces are applied along the z axis. If the pad is laterally constrained, then the x and y axes of the cube cannot undergo any positional change when a pressure is applied to its surface:

$$\varepsilon_x = (\sigma_x - (\sigma_y + \sigma_z)/2)/E = 0$$

$$\varepsilon_y = (\sigma_y - (\sigma_z + \sigma_x)/2)/E = 0$$

which gives:

$$\sigma_x = (\sigma_y + \sigma_z)/2$$

$$\sigma_y = (\sigma_z + \sigma_x)/2$$

and finally:

$$\sigma_x = \sigma_y = \sigma_z$$

It results then that:

$$\varepsilon_z = (\sigma_z - (\sigma_x + \sigma_y)/2)/E = 0$$

This means that the fully constrained one-piece elastic pad cannot be compressed in the z direction, and thus it cannot take on the shape of the touched object surface.

Generally, one-piece elastic pads can be compressed because they are only partially constrained. However, these pads are faced with the "inverse problem" of determining the profile of the stress applied at the pad surface from the strain vectors measured by transducers embedded in the elastic pad.

4.2.2 Tab-shaped Elastic Overlay

Attempts made to solve the crosstalk problem have resulted in complicated signal processing solutions, [4.6] and [4.7]. A simpler solution proposed here is to abandon the one-piece structure of the elastic pad. The new custom-designed elastic overlay, illustrated in Figure 4.7, consists of a relatively thin membrane with protruding round tabs. This allows the material to expand without any stress in the x and y directions, ($\sigma_x = \sigma_y = 0$), making possible its compression in the z direction proportionally with the normal stress component ($\varepsilon_z = \sigma_z/E$), as illustrated in Figure 4.8.

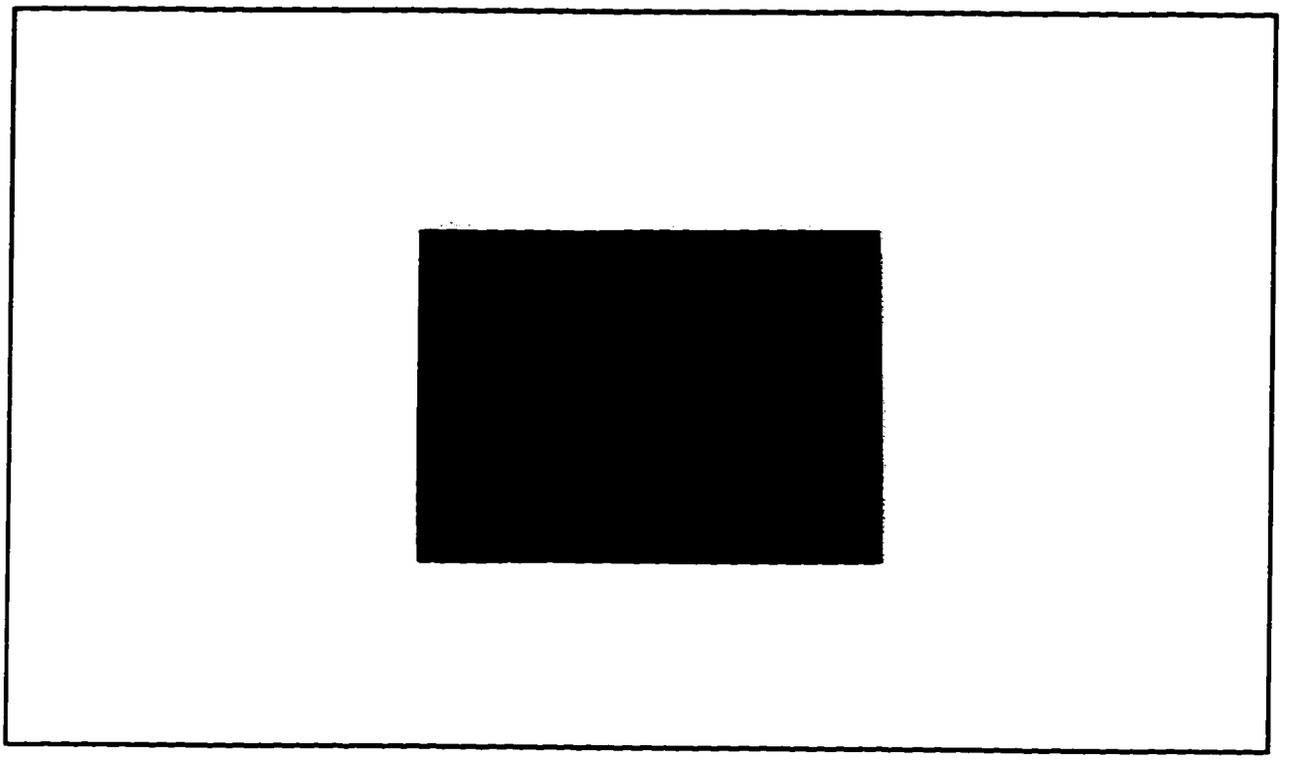


Figure 4.7 Tactile sensor with the tab shaped elastic overlay

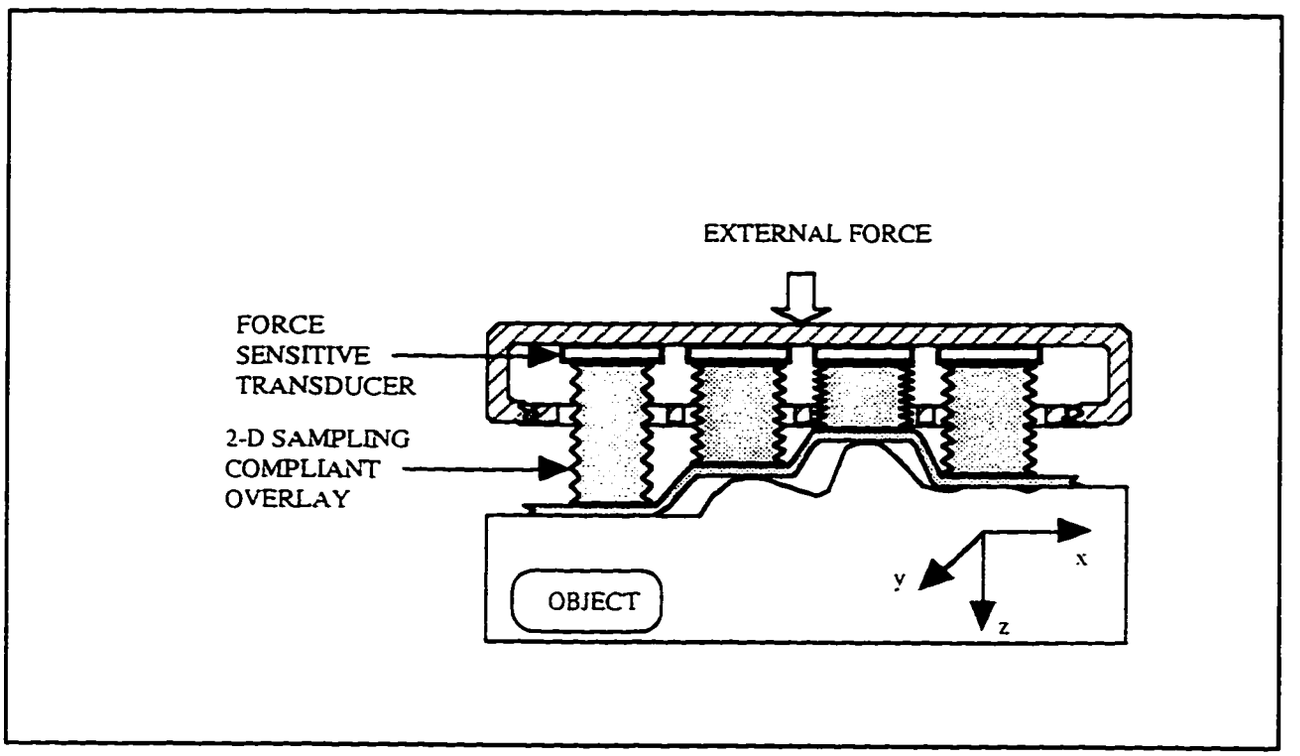


Figure 4.8 **Protruding round tabs of the elastic overlay provide spatial sampling**

The tabs are arranged in a 16-by-16 array in such a way that there is a tab superimposed on top of each sensing element of the FSR matrix. The spatial sampling provided by these pads matches that of the FSR matrix. The 2-D sampling error can be defined as the difference between the actual contact area of the object indenting the elastic pad and its digitized image obtained by chain linking the centers of the pixels along the object edges. The physical shape and size of the tabs has considerable effect on this sampling error. Based on recommendations made in [4.8] and [4.9], circular tabs have been selected which occupy 50% of each 2-D sampling area.

The developed tactile sensor provides a high spatial sampling resolution (x and y directions) and a limited resolution for the measurement of a touched object surface details (z direction). Figure 4.9 shows the 16-by-16 (median filtered) tactile image of a circular washer with an exterior diameter of 15 mm and an interior diameter of 7 mm which has been pressed with a force of 80 N on a tab-shaped elastic overlay. Figure 4.10 shows the 16-by-16 (median filtered) tactile image of the same washer pressed with the same force on an one-piece elastic overlay. A comparison of the two images illustrates the significant improvement due to positive effect of the new pad design.

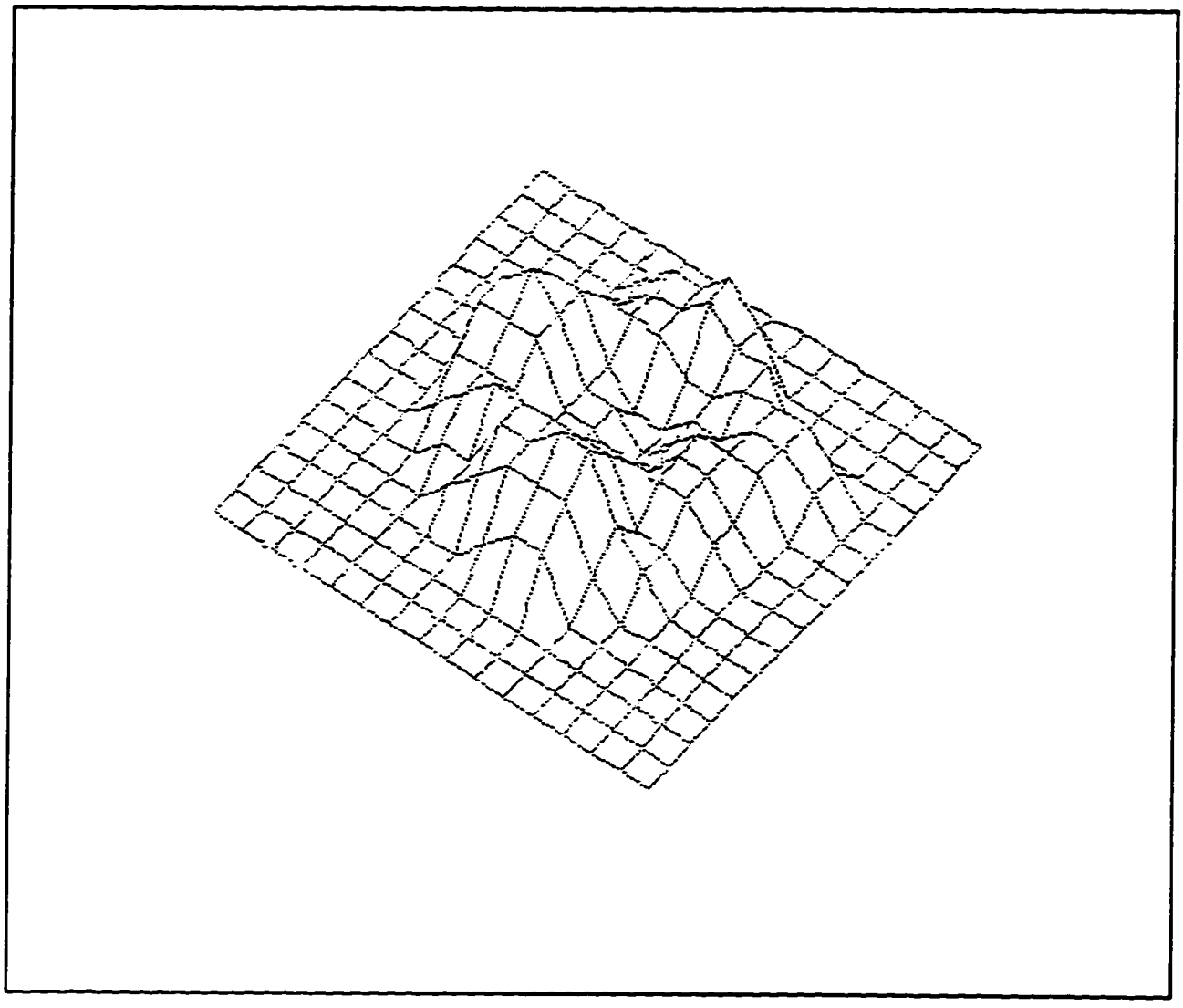


Figure 4.9 The 16-by-16 (median filtered) tactile image of a circular washer produced by a tactile sensor with a tab-shaped elastic overlay.

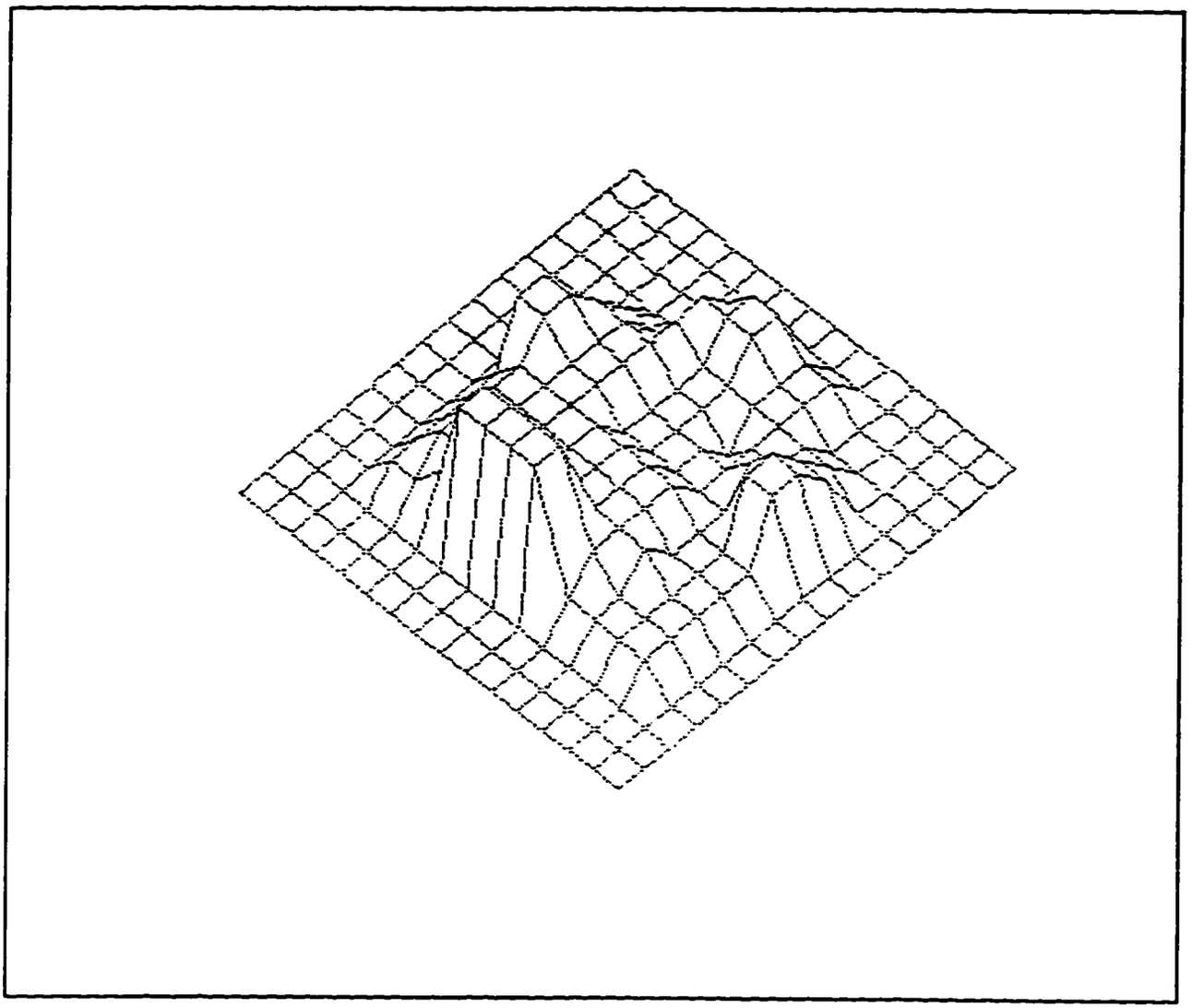


Figure 4.10 The 16-by-16 (median filtered) tactile image of the same washer produced by a tactile sensor with an one-piece elastic overlay.

4.3 Correlation Technique for Tactile Data Integration

Since, for non-trivial applications, the probe is smaller than the explored surface, a "global tactile image", (GTI), has to be assembled from a series of smaller "tactile probe images", (TPIs). Each of these TPIs is obtained through a guarded motion of the robot until the tactile probe makes a firm contact with the object. Then the sensor is lifted up , move a short distance away from the first point, and come in contact again on the surface. The action is repeated in a sequential manner as described in the later section. The following information is recorded for each tactile pose:

1) kinesthetic data from the robot, consisting of three position and two orientation parameters;
2) kinesthetic data from the compliant wrist, as provided by the displacement transducers;
3) cutaneous data, represented by the TPI.

The accuracy of the kinesthetic data is of utmost importance for the integration of the sequence of local tactile images into a global tactile image of the explored object shape [4.10], [4.11]. The incremental character of the GTI makes its production very sensitive to positional errors of its TPI components. A correlative method was developed for the recovery

of an object shape from the relative position errors between consecutive tactile poses.

This error correction method is based on the idea that TPI's produced by distinct but overlapping tactile poses (impressed with equal bias forces) must have identical contents in the common areas. If there is any measurement error of the planar relative-position of two overlapping poses, then this is reflected by a similar shift of the maximum value of the 2-D correlation function of the identical image areas.

The GTI of a planar polyhedron face is obtained by the sequentially merging overlapping TPI local images. An initial TPI is chosen as a reference image to which all subsequent TPIs will be added. The 3-D positional parameters of this reference image are used as global kinesthetic parameters of the entire explored surface. A "left-to-right and top-to-bottom" scanning strategy is implemented by overlapping (by 75%) the current pose on the previous pose as illustrated in Figure 4.11.

If the relative pose position errors have to be recovered with a subpixel resolution, then the 2-D correlation function has to be calculated with the same subpixel resolution. This can be obtained from appropriately magnified TPI images.

For the probe used in this thesis, the 16-by-16 TPI images

are magnified to a 64-by-64 format which has one quarter of the initial pixel resolution. This is obtained, as given in [4.12], by a second-order linear interpolation produced by padding each line and column of the initial image by two rows and two columns of zeros and then convolving it two times with the 3 x 3 array:

$$H=\begin{bmatrix} 1/4 & 1/2 & 1/4 \\ 1/2 & 1 & 1/2 \\ 1/4 & 1/2 & 1/4 \end{bmatrix}$$

The 2-D cross correlation function $R(m,n)$ between two s x t rectangular magnified-images zones {TP1(i+p, j+q)p = 0,..., s-1; q = 0,..., t-1} is given by the following relation:

$$R(m,n)=\sum_{p=0}^{s-1}\sum_{q=0}^{t-1} TPI1(i+p, j+q)\times TPI2(k+p+m, l+q+n).$$

If the two magnified-image zones overlap the same planar shape (which does not have to be perfectly smooth, then, as a theoretical (error-free) case, the 2-D correlation function R(m, n) will have a maximum for m = n = 0. Since in practice there are always some planar position measurement errors, e_x and e_y, the 2D correlation function will have a maximum $R(m_1, n_1)$ for some nonzero m_1 and n_1, A maximum $R(m_1, n_1)$ represents the quantized estimates of e_x and e_y, respectively, in the magnified-image grid.

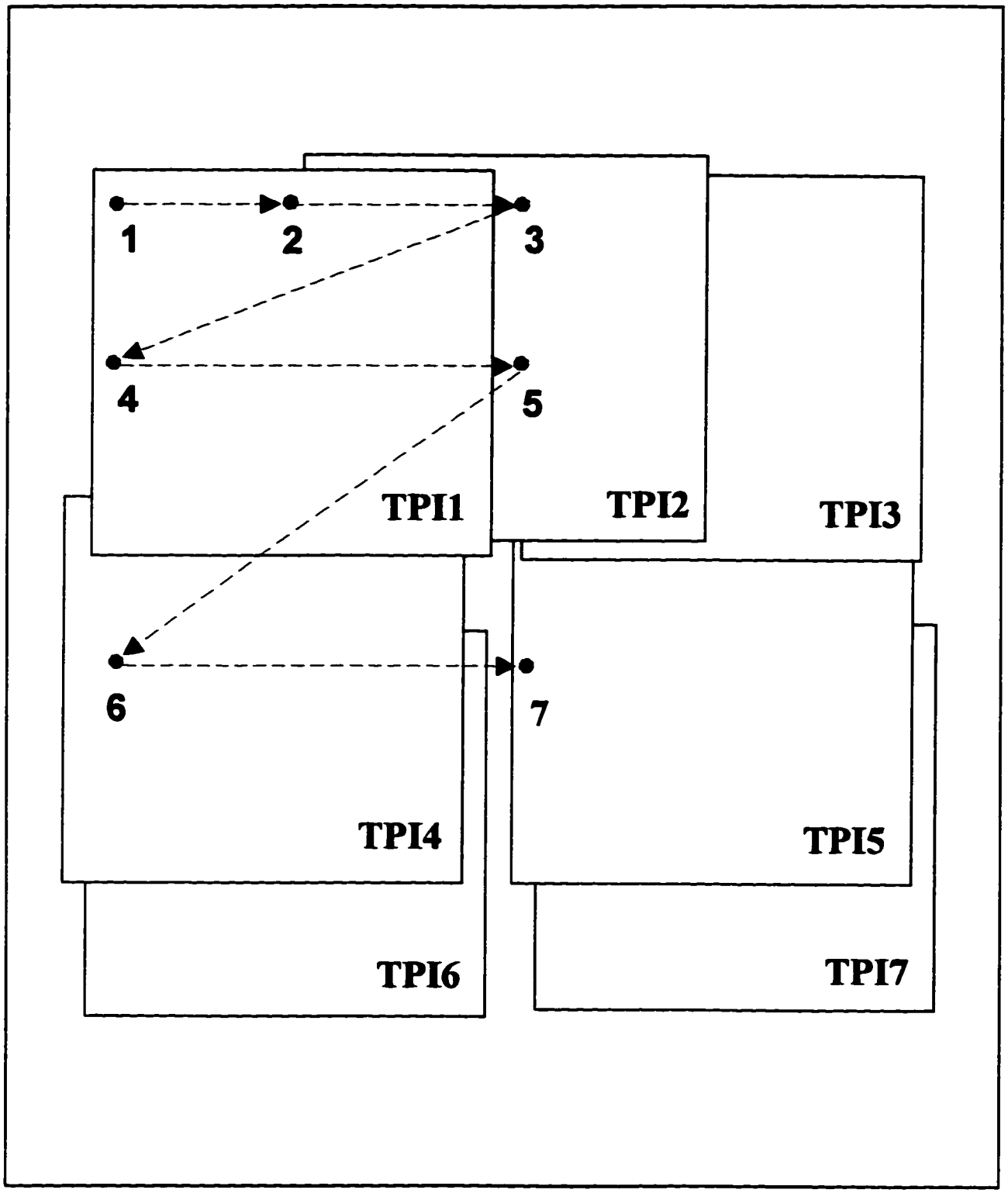


Figure 4.11 Overlapped tactile scanning strategy

The composite image is processed by a series of algorithms [4.13] and [4.14] in order to identify the edges and their positional parameters. The noise inherent to edge detection can be reduced if a median filtering operation is implemented before applying the edge operator. The median filter replaces the intensity of a tactel by the median of the intensities in a 3-by-3 neighborhood centered by that tactel. This filtering removes the noise without affecting the edges in the tactile image. As an example, Figure 4.12(a) shows a median-filtered 32-by-16 tactile image.

For edge enhancement we use local Sobel gradient operators. For a 3-by-3 neighborhood centered at (i, j) these Sobel gradient operators are [4.15]:

$$Gi = [f(i+1, j-1) + 2 \cdot f(i+1,j) + f(i+1, j+1)] - [f(i-1, j-1) + 2 \cdot f(i-1,j) + f(i-1, j+1)]$$

$$Gj = [f(i-1, j+1) + 2 \cdot f(i,j+1) + f(i+1, j+1)] - [f(i-1, j-1) + 2 \cdot f(i,j-1) + f(i+1, j-1)]$$

The resulting image, shown in Figure 4.12(b), is then thresholded and edge line segments are extracted using the modified adaptive Hough transform algorithm [4.16], specially conceived to handle multiple lines of arbitrary slope in images with very low signal-to-noise ratio. Multiple lines are extracted iteratively one by one. The algorithm always extracts

the longest segment first and then removes its data points before the next iteration. The algorithm stops when there are no more segments left. The algorithm provides noise filtering by imposing a minimum number (three in our case) of data points which define a line segment.

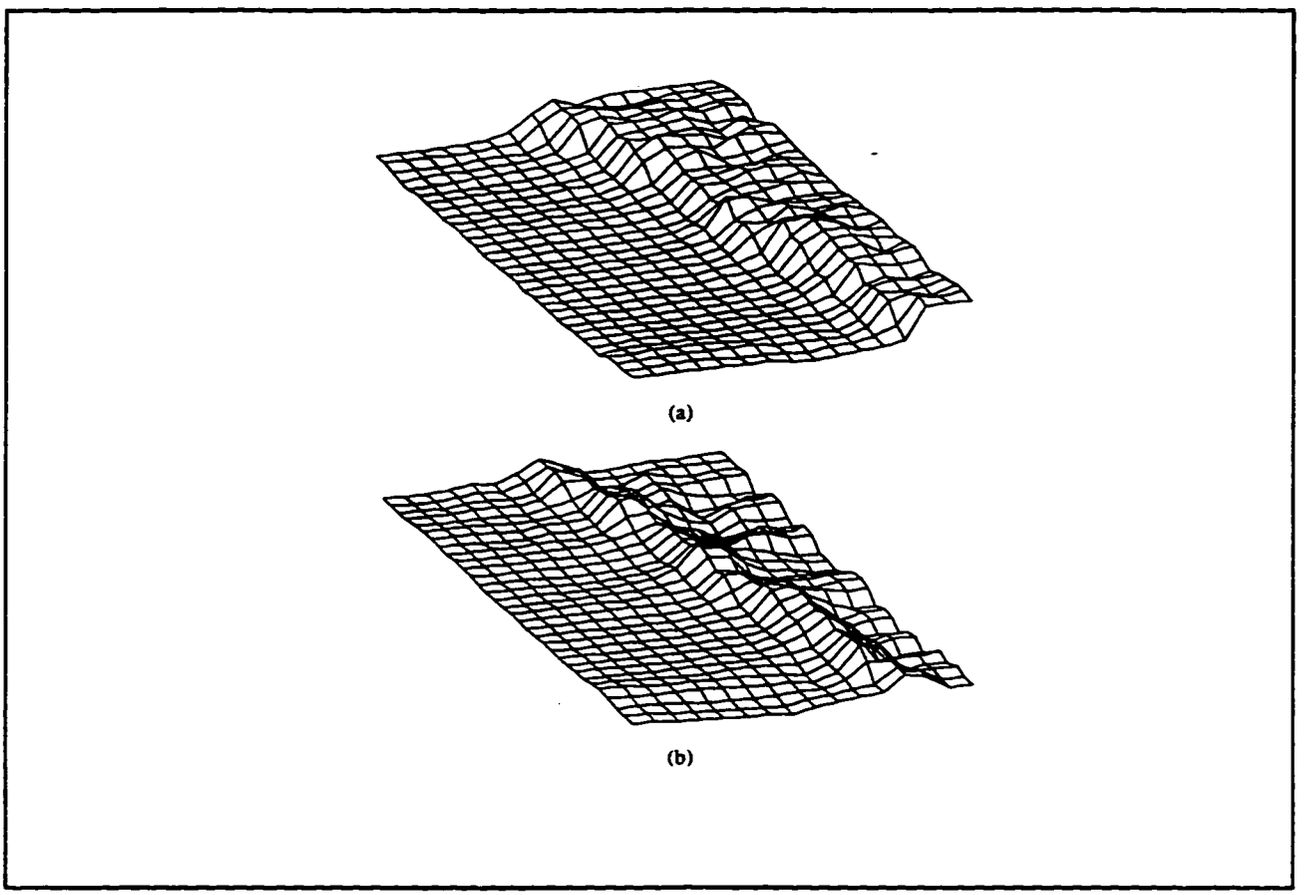


Figure 4.12(a)&(b) A Composite 32-by-16 tactile Image. (a) After Median Filtering. (b) After The Application Of Sobel Gradient Operators.

The syntactic description method [4.17] is then used to

fuse the extracted line segments according to the following production rules:

⟨ EDGE ⟩ := ⟨ EDGE1 ⟩ ∨ ⟨ EDGE2 ⟩ • (angle EDGE1, EDGE2 > α)

⟨ EDGE ⟩ := ⟨ EDGE1 ⟩ ∨ ⟨ BREAK ⟩ • (angle EDGE1, BREAK > α)

Here EDGE primitive indicates a relatively long linear edge segment, BREAK indicates a short edge segment with no regular shape, ∨ is the union operator, and α is an angle threshold value. Figure 4.13 illustrates the object edge recovered from figure 4.12(b) with a maximum estimation error of 1.52 mm (0.96 sampling step).

Object vertices are defined by the intersection of edges using the syntactic production rule ⟨ VERTEX ⟩:=⟨ EDGE1 ⟩ ∧ ⟨ EDGE2 ⟩ • (angle EDGE1, EDGE2 < α), where ∧ is the intersection operator. A more detailed presentation of this polygonal boundary estimation method is given in [4.18].

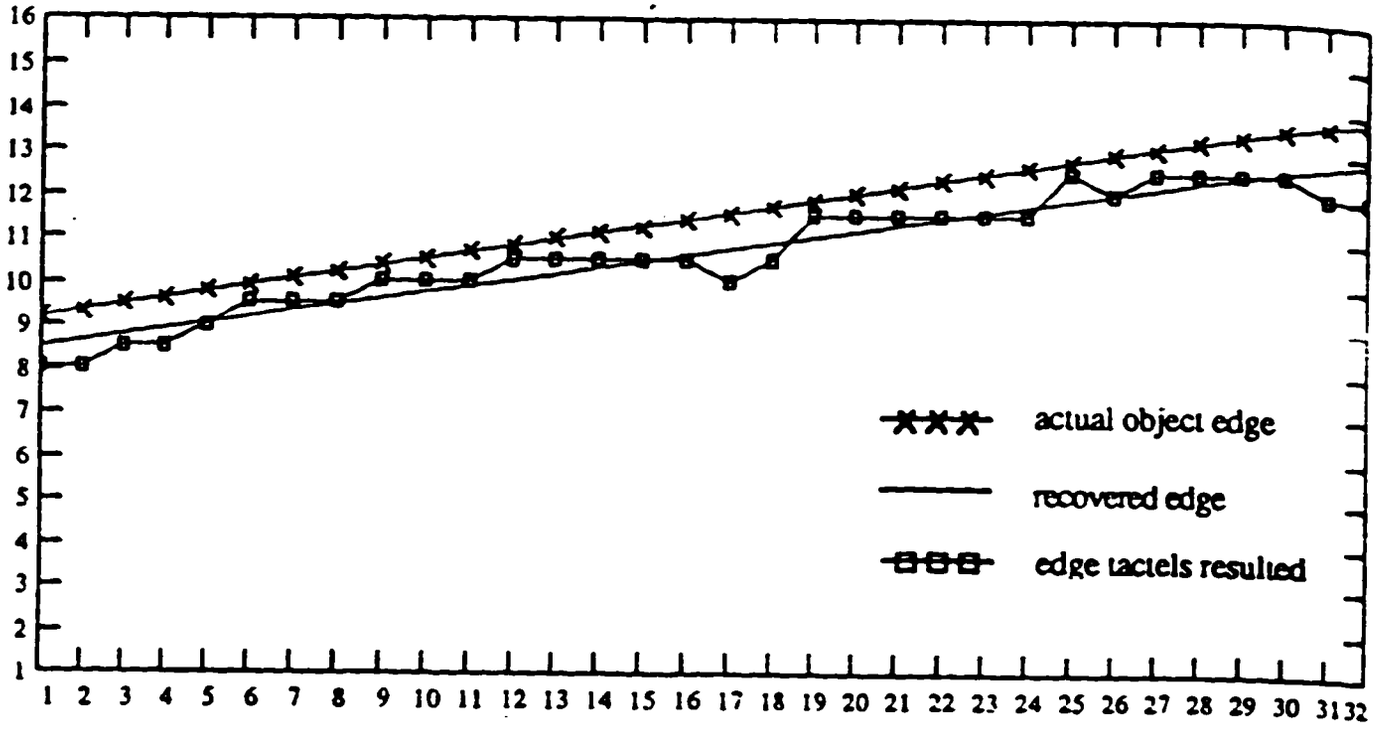


Figure 4.13 Position of the edge recovered (estimation) from figure 4.12(b) compared with the actual position of the investigated object edge.

Chapter 5

Pseudo-Random Encoding

Pseudo-Random Binary Sequences (PRBS) are also known as binary maximal-length linear feedback shift register sequences (M-sequences) or Pseudo-Noise sequences (PN-sequences), [5.1]-[5.4]. A considerable body of literature exists showing how these sequences have been employed over the years in communications, [5.5]-[5.11], system identification [5.12]-[5.14], for position measurement applications [5.15]-[5.23], and more recently for 3-D object recognition using vision [5.24]-[5.28]. As far as we know our work is the first attempt to apply pseudo-random sequences to the 3-D object recognition using tactile sensing [5.29].

The communication applications of the pseudo-random binary sequences are essentially in the area of direct-sequence spread spectrum (DS/SS) synchronization. The synchronization process consists of two steps: acquisition and tracking. Acquisition requires generation of a local replica of the PRBS code in the receiver and coarse alignment of this local code with that contained in the in coming signal. Tracking registers finely aligning the two codes and maintaining this alignment.

Different requirements have motivated the absolute position measurement and 3-D object recognition applications of the pseudo-random sequences. The reason for these applications resides in an obvious need for an economical position encoding method. It may also be worthwhile mentioning that the position measurement and 3-D object recognition applications grew up to use the higher density multi-valued sequences and arrays as while the communication applications are basically using binary pseudo-random sequences.

Absolute 1-D or 2-D position recovery with a *n*-bit resolution usually requires a *n*-bit code to mark each quantization step, [5.30] and [5.31]. The size of this code represents an obvious limitation of this natural encoding for most practical applications. This limitation of the absolute encoding can be overcome by using pseudo-random encoding which requires only one bit of code per quantization step, [5.31].

The simplest form of pseudo-random encoding is based on pseudo-random binary sequences (PRBS)for the 1-D case, [5.2]-[5.4] and pseudo-random binary arrays (PRBA) for 2-D case, [5.2].

5.1 Pseudo-Random Binary Sequences

A PRBS { S(p) | p = 0,1, ..., 2^n - 2)} is generated by a modulo-two n-bit shift register {R(n), ..., R(1)}, shown in Figure 5.1, having the direct feedback equations given in Table 5.1.

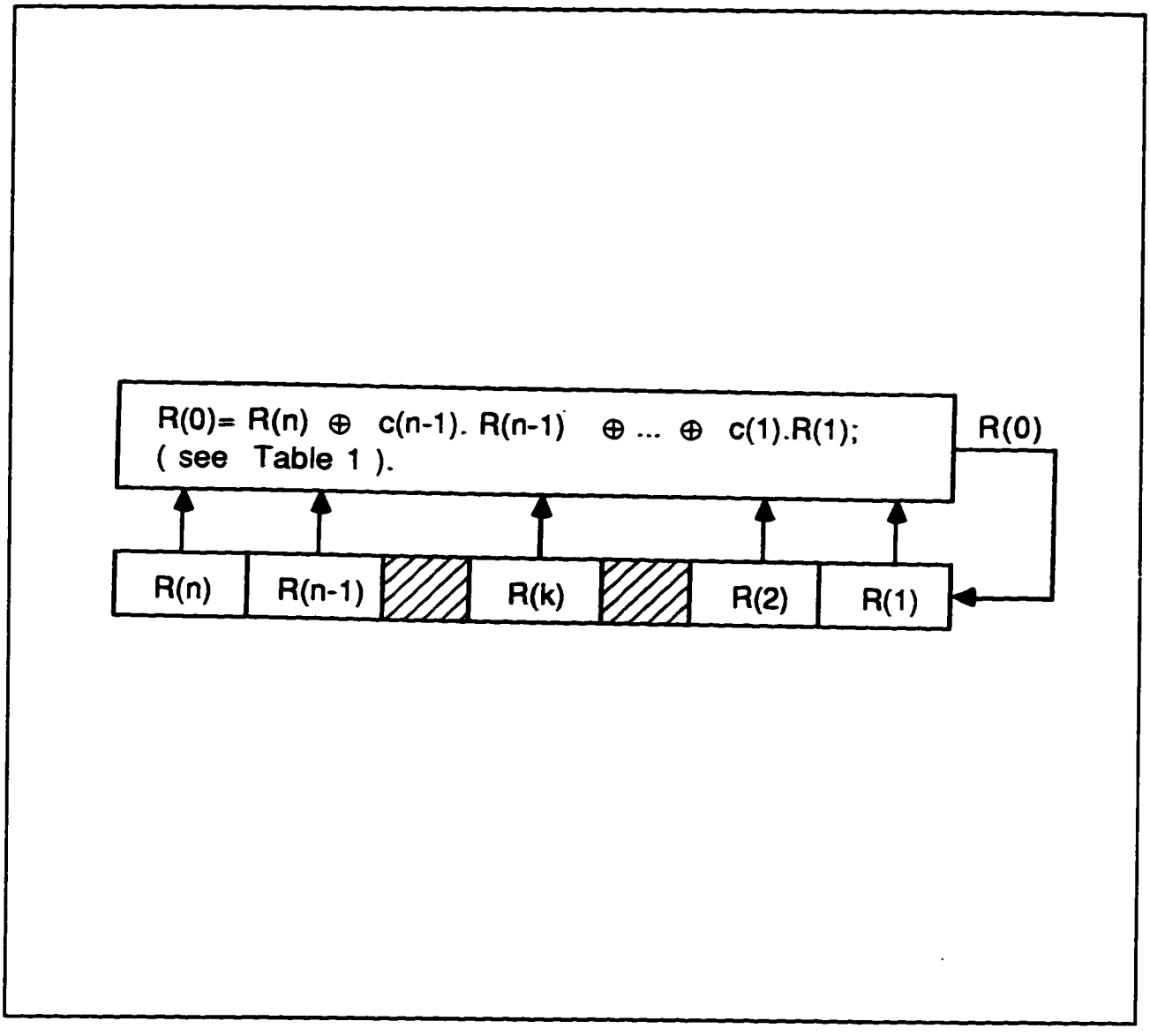


Figure 5.1 PRBS generation using a modulo-two feedback shift register

Shift register length n	Feedback for direct PRBS R(0)= R(n) ⊕ c(n-1)·R(n-1) ⊕...⊕ c(1)·R(1)	Feedback for reverse PRBS R(n+1)= R(1) ⊕ b(2)·R(2) ⊕...⊕ b(n)·R(n)
4	R(0) = R(4) ⊕ R(1)	R(5) = R(1) ⊕ R(2)
5	R(0) = R(5) ⊕ R(2)	R(6) = R(1) ⊕ R(3)
6	R(0) = R(6) ⊕ R(1)	R(7) = R(1) ⊕ R(2)
7	R(0) = R(7) ⊕ R(3)	R(8) = R(1) ⊕ R(4)
8	R(0) = R(8) ⊕ R(4) ⊕ R(3) ⊕ R(2)	R(9) = R(1) ⊕ R(3) ⊕ R(4) ⊕ R(5)
9	R(0) = R(9) ⊕ R(4)	R(10) = R(1) ⊕ R(5)
10	R(0) = R(10) ⊕ R(3)	R(11) = R(1) ⊕ R(4)

Table 5.1: Direct and reverse feedback equations for some binary shift registers

This PRBS has the so called "window property" which is very useful for position recovery applications. According to this, any n-tuple {S(p+n-k) | k = n, ..., 1} scanned by a window {x(k) | k = n, ..., 1) is unique and fully identifies the current index "p" of the window, [5.2]and [5.3].

A translation from the pseudo-random binary code into a more convenient natural binary representation is necessary for

practical applications. This binary conversion problem is similar with the acquisition problem encountered in the DS/SS synchronization case. The time required for the pseudo-random/natural binary code conversion is critical for real time applications.

The code conversion methods vary in the extent of their degree of parallelism and corresponding computation time/equipment cost. A strictly parallel solution will use a code conversion table stored in ROM. This method is equipment expensive for applications requiring high encoding resolutions. At the other extreme, a strictly serial code conversion exploits the reversibility of the PRBS generating algorithm [5.16] and [5.17]. This method is based on the idea that it is possible to find the natural value associated with any pseudo-random n-tuple by simply counting the number of reverse feedback shifts (Table 5.1) that it takes for the given n-tuple to arrive back into the "zero position" pseudo-random pattern. This idea is essentially similar with the serial search used for the sequential acquisition of PN sequences for DS/SS communications [5.8] and [5.10]. In this case the solution is less equipment expensive but more time expensive for high resolution measurements.

A compromise solution is a combination of the serial and parallel methods, [5.19] and [5.20], and is exemplified in Figure 5.2. Consider a pseudo-random encoded track where

certain positions (uniformly distributed with a period of t) are employed as "milestones," The code conversion for any position $p = m \cdot t + r$, where $m \cdot t$ is position of the nearest milestone $Q(m)$ and r represents the position relative to this milestone, will be discussed. The natural code for r is found by counting the steps required to shift the initial register contents to the assigned nearest milestone $Q(m)$. All intermediate states of the shift register contents are checked (using a field-programmable logic array, FPLA) against all possible milestone pseudo-random patterns. As a result, the code conversion of the relative position r distance is found serially while the milestone code conversion is done in parallel. While the idea of this hybrid binary code- conversion method is similar with that discussed in [5.7] and [5.11] the specific solution proposed for position measurement applications [5.19] and [5.20] is different from the implementation point of view (strictly hardware implementation, FPLA).

Figure 5.3 shows the time performance of this serial-parallel code conversion method as compared with the other two extreme conversion methods.

Equipment and temporal costs are used to decide on the optimal degree of parallelism to be used for a specific implementation. Given below are the definitions of the variables and constants required for the cost analysis.

n	Bit length of each pseudo-random binary code.
$2^n - 1$	Total length of the PRBS.
t	Uniformly distributed period between successive milestones.
$k1$	Equipment cost associated with each milestone.
$k2$	Equipment cost for the serial back shift operations.
$k3$	Temporal cost for a fully parallel solution.
$k4$	Temporal cost associated with each back shift operation.

Note: $k1, k2, k3, k4$ are constant for a given n and are functions of the employed technology.

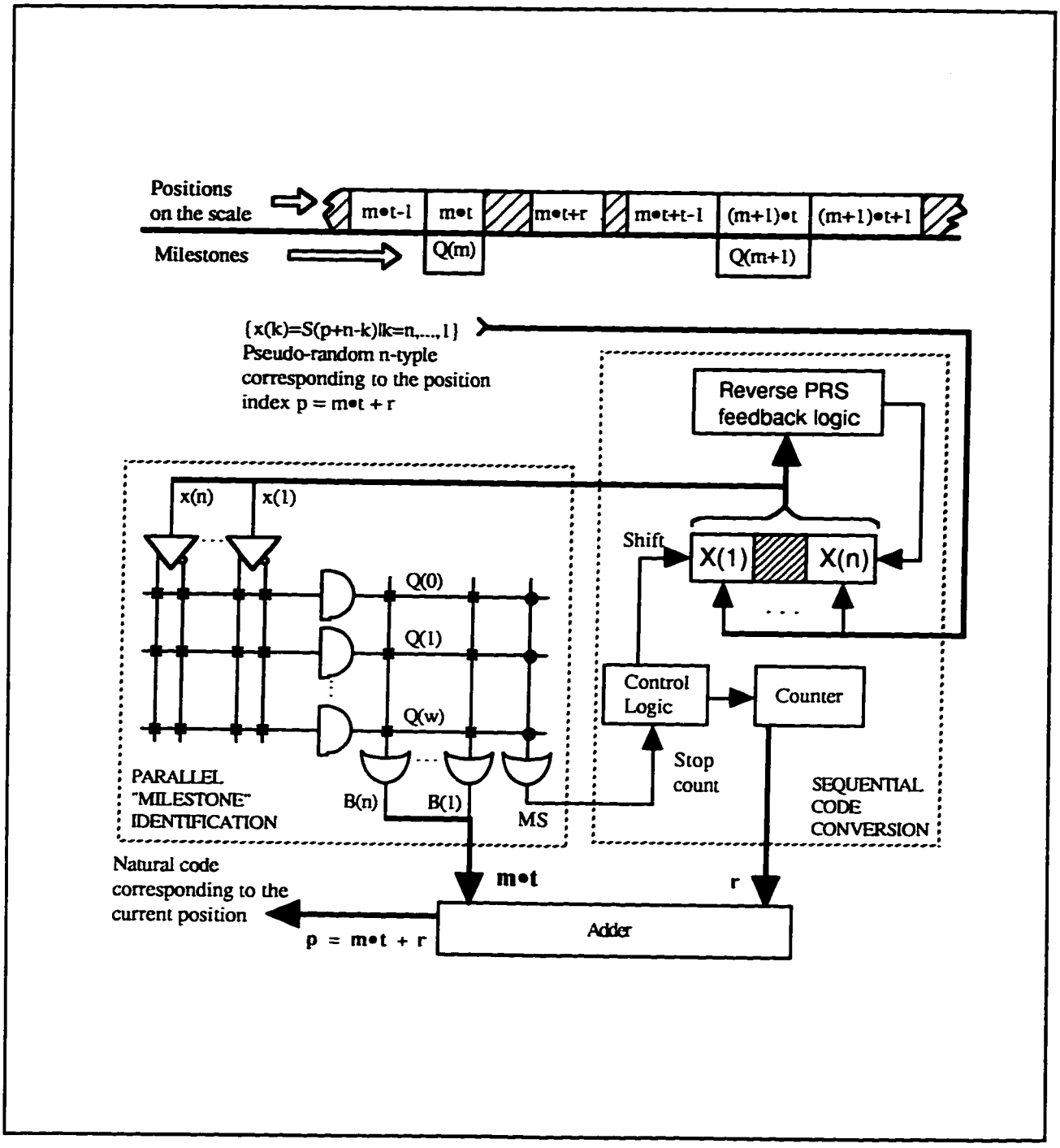


Figure 5.2 Serial-parallel pseudo-random/natural binary code conversion

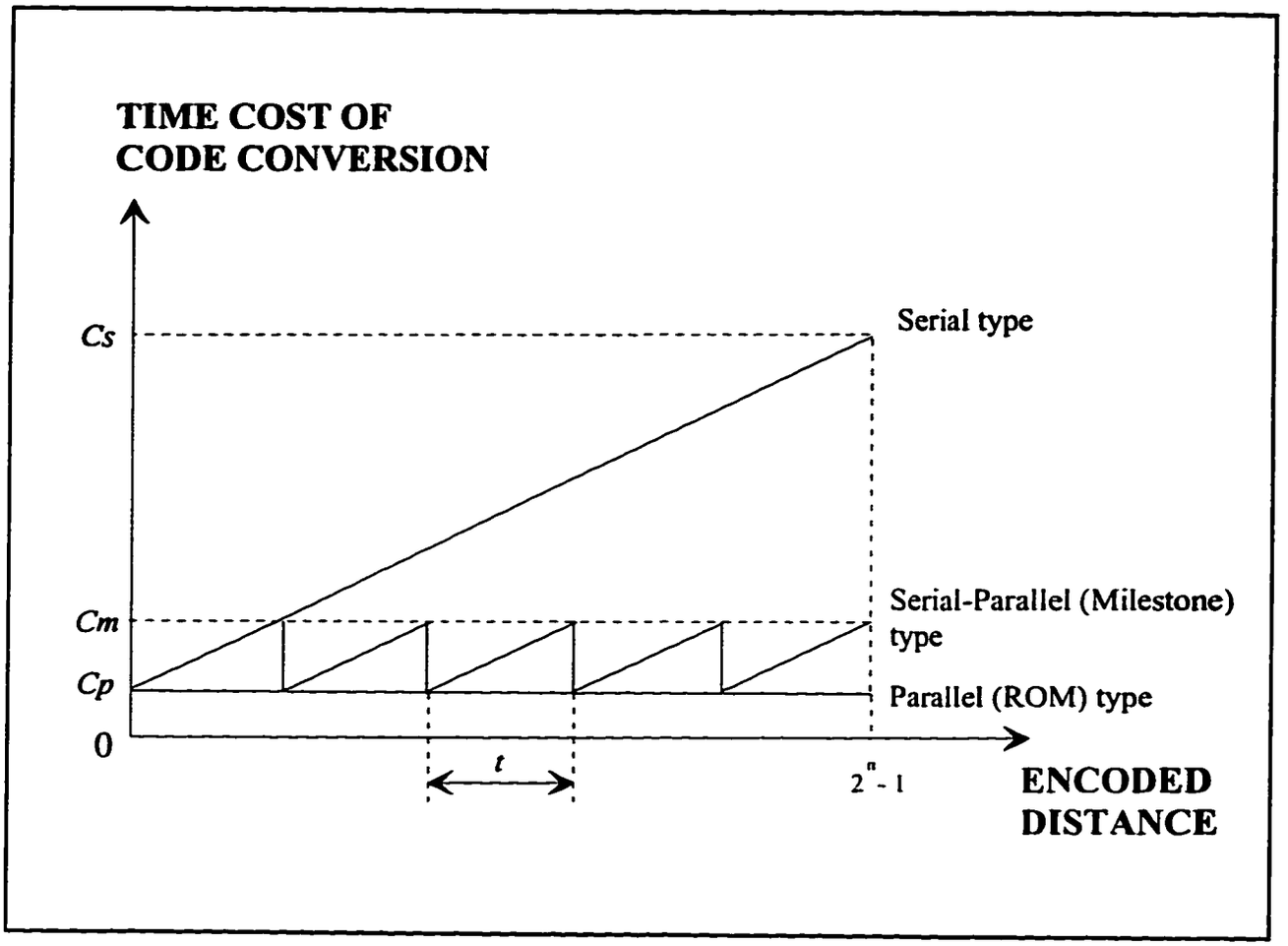


Figure 5.3 Relative time performance of different pseudo-random/natural binary code conversion methods.

The total cost of the serial-parallel binary code conversion will then be

$$total\ cost = equipment\ cost + temporal\ cost$$

$$total\ cost = k1 \cdot \int (2^n - 1/t) + k2 + k3 + k4 \cdot t.$$

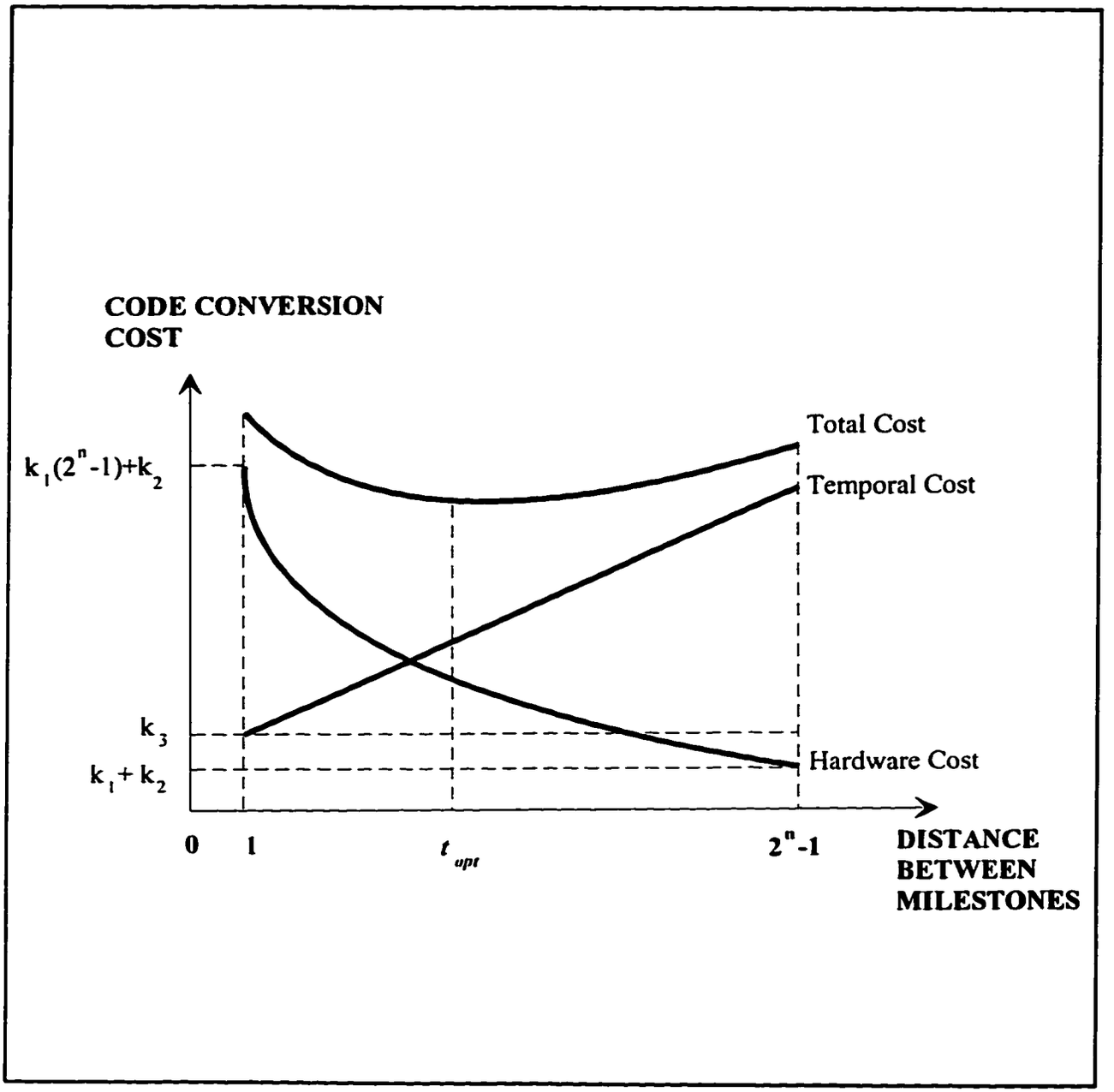


Figure 5.4 Serial-parallel binary code conversion costs as a function of the distance between milestone

Requested Number of Milestones $(2^n - 1)/t$		Measurement Resolution			
		8 bits	16 bits	24 bits	32 bits
Relative equipment/ temporal cost k_1/k_4	1/4	32	512	8192	131072
	4/1	8	128	2048	32768

Table 5.2 The Optimal number of milestones in serial-parallel binary code conversion as a function of the relative equipment/temporal cost and the measurement resolution (adapted from [5.20])

Figure 5.4 shows a graph of the equipment cost, temporal cost and total cost as a function of *t*. The temporal cost associated with serial recovery of *r* increases linearly with the independent variable *t* while the equipment cost is inversely proportional with this variable. The combined total cost graph does have a minimum between 1 and $2^n - 1$. The value of *t* which minimizes the total cost function is the following:

$$t_{opt} = \int [(k1/k4 \cdot 2^n - 1)^{1/2}]$$

Depending on the relative ratio of the constants *k1* and *k4* (for

a given *n*), the optimal distance between milestones will vary. This cost analysis provides a guide for choosing the number of milestones for a specific application.

A number of examples of this optimization procedure are summarized in Table 5.2. As one can expect, a higher degree of parallelism is requested when the temporal cost is relatively higher than the equipment cost. A higher measuring resolution will always lead to an increase of the number of milestones in order to maintain the same code conversion speed.

5.2 Pseudo-Random Binary Arrays

A Pseudo-Random Binary Array (PRBA) with *n1* rows and *n2* columns is usually constructed, by folding a $(2^n - 1)$-term PRBS $\{S(p) \mid p = 0,1, \ldots, 2^n - 2\}$ as shown in Figure 5.5, [5.2]. The following relations hold:

$$2^n - 1 = 2^{k1 \cdot k2} - 1$$

$$n1 = 2^{k1} - 1$$

$$n2 = (2^n - 1)/n1$$

where n1 and n2 must be relatively prime.

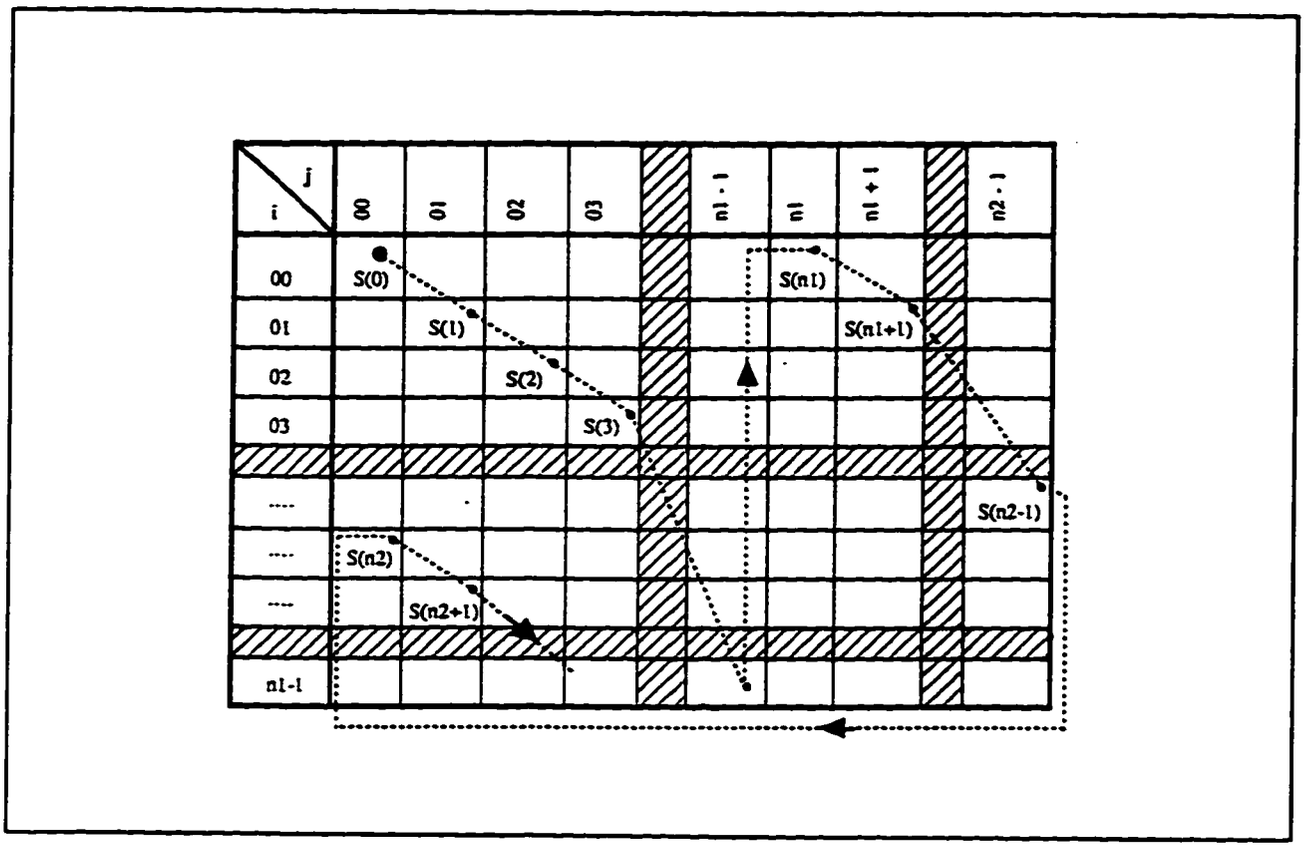


Figure 5.5 Folding of the PRBS to produce a PRBA

The row/column index recovery is based on the PRBA window property. Any k1-by-k2 nonzero binary pattern seen through a k1-by-k2 window sliding over the PRBA is unique and will fully identify the window absolute position index (i,j) within the PRBA.

Because only the look-up table code conversion is presently known, this encoding method may be too expensive for some practical applications which require high encoding resolutions.

5.3 Pseudo-Random Binary Product Array Encoding

A original 2-D pseudo-random encoding method was developed in our laboratory [5.23] as solution which allows for a more convenient pseudo-random/natural binary code conversion than the look-up table used for the previously described PRBA.

In this case, a $(2^{k1} -1)$-by-$(2^{k2} -1)$ array (Figure 5.6) with elements $A(i, j)$, is generated by a bit-by-bit multiplication of two PRBS, one along the X coordiante axis, another along the Y coordinate axis:

$$[A(i,j) = SX(i) \wp SY(j) \mid i = 0,1,...,2^{k1} -2;\ j = 0,1,...2^{k2} -2]$$

where $SX(i)$ are the elements of the $[SX(i)\ i=0,1,...,2^{k2} -2]$ PRBS generated by a $k1$-bit shift register, $SY(j)$ are the elements of the $[SY(j)\ j=0,1,...,2^{k2} -2]$ PRBS generated by a $k2$-bit shift register and $\wp$ is the product operator defined in Figure 5.7. Due to the non-commutative character of this law it is always possible to recover the binary values of the two factors $SX(i)$ and $SY(j)$ from the quaternary value of any $A(i,j)$ product.

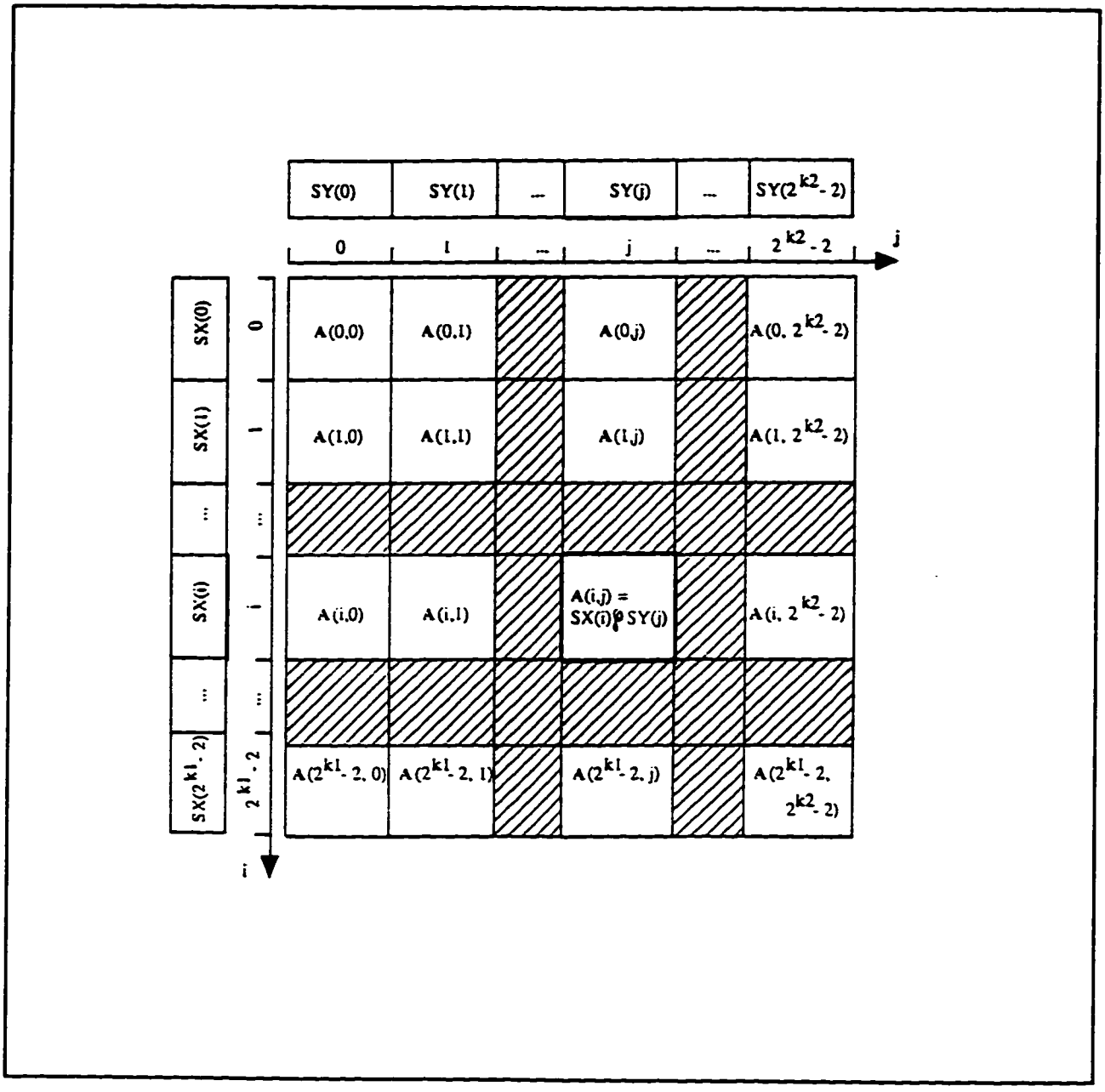


Figure 5.6 Generation of a $(2^{k1} - 1)$-by-$(2^{k2} - 1)$ Pseudo-Random (binary) Product Array (PRPA)

SY(j) \ SX(i)	0	1
0	0	1
1	2	3

Figure 5.7 The ρ "product law"

The quaternary pattern [A(k,m) = SX(k) ρ SY(m) | k = i,i+k1-1; m = j,j+1,...,j+k2-1] seen through a *k1*-by-*k2* window sliding over the array is the product of the k1-tuple [SY(m) | m=j,j+1,...,j+k2-1]. If at least one element of each row and column of the *k1*-by-*k2* window is recovered, then it is possible to find (using the "product law" from figure 5.7) the binary values of all the elements of the 1-D pseudo-random binary patterns [SX(k)|k=i,i+1,...,i+k1-1] and [SY(m)|m=j,j+1,...,j+k2-1] which fully identify the coordinates (i,j) of the 2-D window.

In Figure 5.8 is it shown, as an example, the 15-by-63 quaternary PRPA [A(i) = SX(i) ρ SY(j)|i=0,1,...,14; j=0,1,...,62]. Any 4-by-6 window (k1=4 and k2=6) sliding over this array is unique and may be used to identify the window's coordinates. The window highlighted in the figure identifies

the coordinates i=5 and j=16.

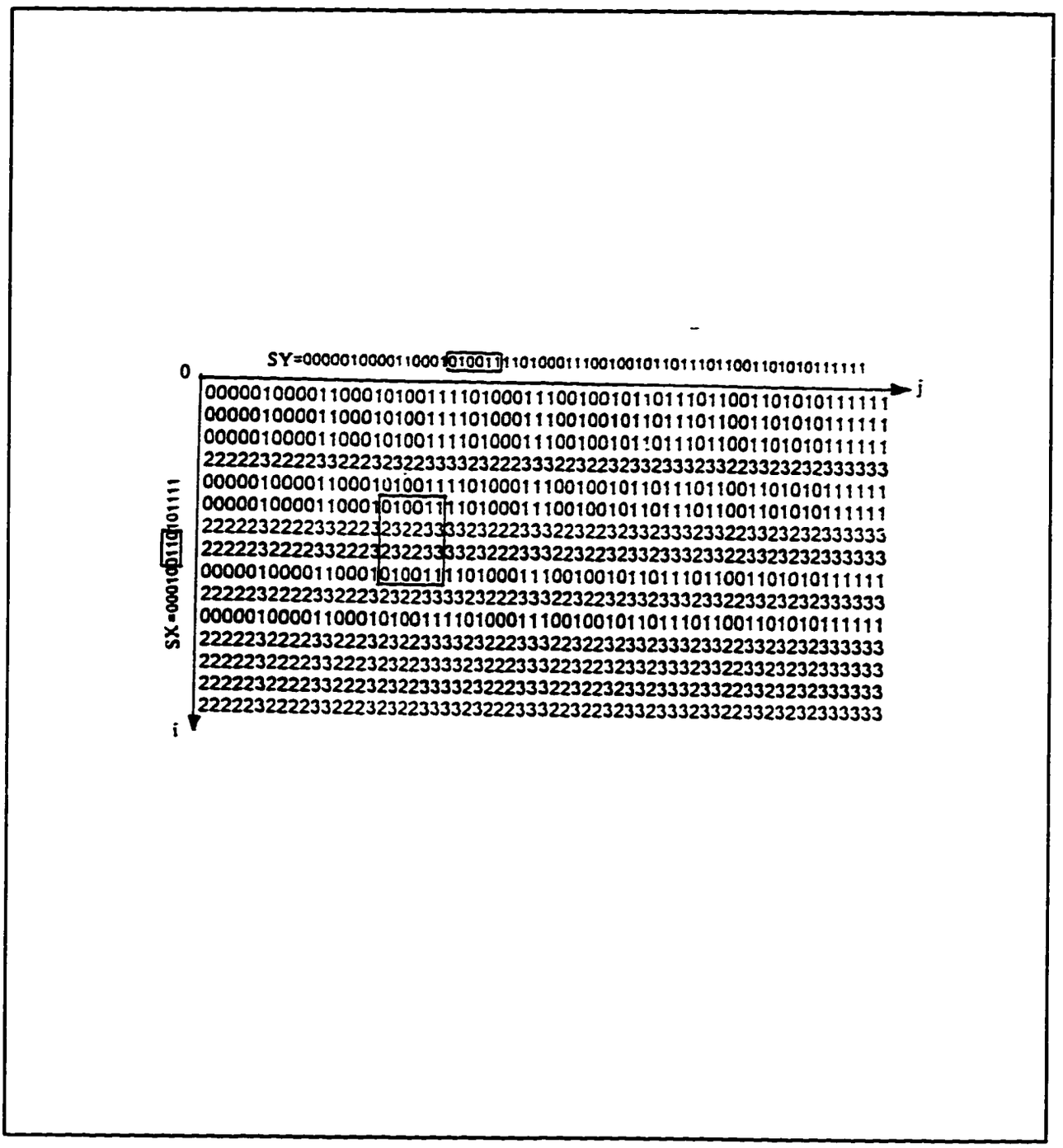


Figure 5.8 A 15-by-63 quaternary PRPA {A(i,j)= SX(i) ρ

SY(j)|i=0,1,...,DJ=0,1,...,62}

The major advantage of PRPA encoding is the fact that the code conversion is reduced to the 1-D code binary conversion of the pseudo-random binary patterns identifying the i and j coordinates of the window. This is a simpler problem for which more convenient solutions are known. These solutions vary in the extent of their degree of parallelism and corresponding time/equipment relative cost as discussed in section 5.1.

5.4 Introduction to Pseudo-Random Multi-Valued Sequences

This section discusses properties of the generalized "pseudo-random multi-valued sequence" (PRMVS) and "pseudo-random multi-valued array" (PRMVA), [5.2], [5.32] and [5.33]. The multi-valued sequences/arrays allow for even higher encoding density than the previously discussed PRBS/PRBA and PRPA encoding methods.

5.4.1 Fields and Rings of Pseudo-Random Polynomial

This section will discuss some of the mathematical

background for a better understanding of the PRMVS properties. A finite Galois field, GF(p) is a set of p elements with operations of "add" and "multiply" which satisfies the following axioms.

(1) GF(p) is commutative with $\oplus$ operation.

(2) If the additive identity (the zero element) is removed from GF(p), the rest of elements in the set are commutative with multiply operation.

(3) All elements in GF(p) obey the distributive law.

If GF(p) = {0, 1,2,..., (p-1)}, where p is a prime number, the GF(p) arithmetic operation is in modulo p or written as (mod (p)).

A commutative ring is the same as that of a field except for the multiplicative inverse for every non-zero elements.

5.4.2 Pseudo-Random Multi-Valued Sequences

The entries of a PRMVS are entries taken from a multi-valued alphabet of q symbols, where q is a prime or a power of a prime. Such a $(q^n - 1)$-term sequence is generated by an n-position shift register with a feedback path specified by a

primitive polynomial, $h(x) = x^n+h_{n-1}\,x^{n-1}+\ldots+h_1\,x+h_0$ of degree n with coefficients from the binary Galois field *GF*(q), [5.2], [5.32] and [5.33]. A number of primitive polynomials over *GF(q)* are given in Table 5.3 for GF(3), GF(4). GF(8), and GF(9).

n	q=3	q=4	q=8	q=9
2	x^2+x+2	x^2+x+A	x^2+Ax+A	x^2+x+A
3	x^3+2x+1	x^3+x^2+x+A	x^3+x+A	x^3+x+A
4	x^4+x+2	$x^4+x^2+Ax+A^2$	x^4+x+A^3	x^4+x+A^5
5	x^5+2x+1	x^5+x+A	$x^5+x^2+x+A^3$	x^5+x^2+A
6	x^6+x+2	x^6+x^2+x+A	x^6+x+A	x^6+x^2+Ax+A
7	$x^7+x^6+x^4+1$	$x^7+x^2+Ax+A^2$	$x^7+x^2+Ax+A^3$	x^7+x+A
8	x^8+x^5+2	x^8+x^3+x+A		
9	$x^9+x^7+x^5+1$	x^9+x^2+x+A		
10	$x^{10}+x^9+x^7+2$	$x^{10}+x^3+A(x^2+x+1)$		

The following relations apply,

for GF(4)= GF(2^2): $A^2+A+1=0$, $A^2=A+1$, and $A^3=1$;

for GF(8)= GF(2^3): $A^3+A+1=0$, $A^3=A+1$, $A^4=A^2+A$, $A^5=A^2+A+1$, $A^6=A^2+1$, and $A^7=1$;

for GF(9)= GF(3^2): $A^2+2A+2=0$, $A^2=A+1$, $A^3=2A+1$, $A^4=2$, $A^5=2A$, $A^6=2A+2$, $A^7=A+2$, and $A^8=1$;

Table 5.3 Primitive polynomials over GF(q) = {0,1,A,A^2, ...,A^{q-2}}

When *q* is prime, the integers modulo-q form the Galois field *GF(q)* = *{0,1,2,...,p-1}* in which the addition,

subtraction, multiplication and division are carried out modulo-q. When q is a power of a prime, $q=p^m$, the integers modulo-q do not form a field and the Galois field elements are expressed as the first $q-1$ powers of some primitive element, labelled here for convenience by the letter A: $GF(q) = \{0,1,a,a^2,\ldots,a^{q-2}\}$. The primitive polynomials used for different PRMVS generation depend on the nature of the addition/substraction and multiplication/division tables adopted for each particular Galois field.

As for the previously discussed for the PRBS, a PRMVS also has the window property which states that any q-valued contents observed through a n-position window sliding over the PRMVS is unique and fully identifies the current position of the window, [5.2].

Let us consider, for instance, a two stage shift register, n=2, having the feedback path defined by the primitive polynomial $h(x)=x^2+x+A$ over $GF(4) =\{0,1,A,A^2\}$, with $A^2+A+1=0$ and $A^3=1$, which generates the 15-term PRMVS:$\{0,1,1,A^2,1,0,A,A,1,A,0,A^2,A^2,A,A^2\}$. Examining this PRMVS it is easy to verify that any 2-tuple seen through a 2-position window sliding over the sequence is unique.

5.4.3 Code Conversion for Pseudo-Random Multi-Valued Sequences

Practical applications require the conversion of the pseudo-random q-valued code into a more convenient natural representation. This can obviously be done in **parallel** using a memory stored look-up table. A **serial** conversion method is based on the idea that it is possible to find the natural value associated with any q-valued pseudo-random n-tuple by simply counting the number of reverse feedback shifts that it takes for the given n-tuple to arrive back into the "zero position" pseudo-random pattern.

The **serial-parallel code conversion** algorithm used for PRBS can also be extended for PRMVS. Multi-valued n-positions with multi-milestones distributed at regular intervals within PRMVS. The multi-valued pseudo-random n-tuple to be converted is loaded in a n-position feedback multi-valued shift register (Figure 5.9). The register is then back-shifted until its contents match the multi-valued n-tuple assigned to a milestone. The position ($Position_{window}$) of the initial n-tuple window within the PRAMVS can be found by adding the counts ($Count_{reverse\ shift}$) of the back shifts and milestones ($Milestone_{position}$):

$$Position_{window} = Count_{reverse\ shift} + Milestone_{position}$$

The source code for finding the position of the multi-valued window is given in Appendix 4.

Figure 5.10 illustrates how the position of a desired window is found in a PRMVS of length 1024 (q=4, n=5). Milestones are placed at regular intervals (in this case, at every 32^{th} interval. If the location of a certain window is required to be identified, back shifts are counted till the current contents of the shift register match a milestone multi-valued n-tuple.

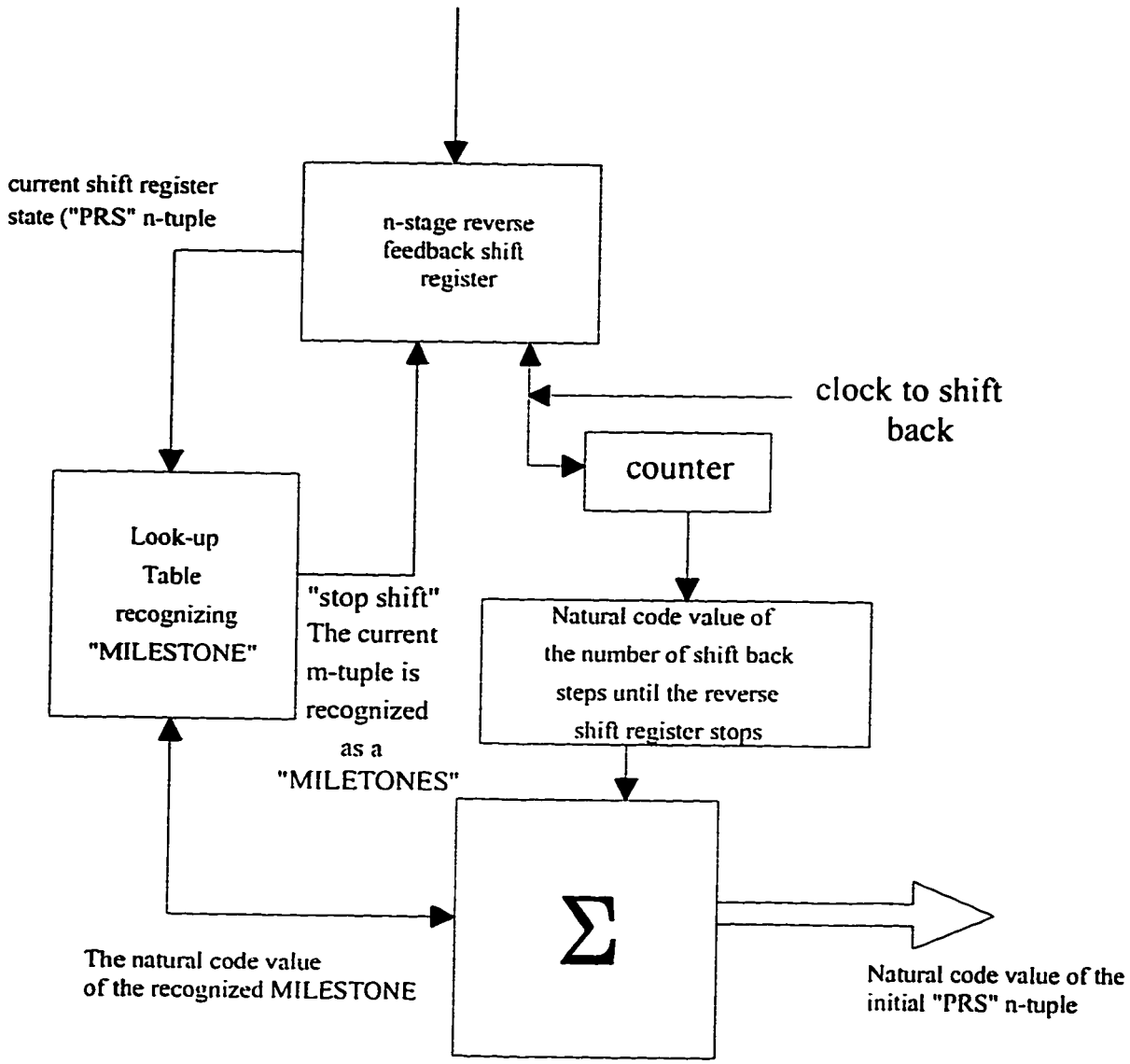


Figure 5.9 The new Serial-parallel Code Conversion for Pseudo-Random Multi-Value Sequence

00001000120010301231100032003330**<u>32213</u>**312120020302331 2

02003230330**<u>13211</u>**132331302011231300011001320**<u>11331</u>**31201

00312030033103202333202**<u>23321</u>**023222031133031213002110 2

2**<u>32210</u>**312230010101212102022232110**<u>32323</u>**303021312210012

20101312110023202033233230230120110313230130**<u>11111</u>**3333

12220011301311110333232**<u>23031</u>**013021112233010211222301 1

01**<u>13213</u>**132101322113123100002000230**<u>02010</u>**2312200013001 1

10133211232300**<u>3010</u>**311230300131011021-3-222131121**<u>0302</u>**231

21000220021302211212302020**<u>1230</u>**100112013031113033113203

133**<u>30122</u>**110123210032203313320123310**<u>02020</u>**232320303331 3

22013131101032**<u>12332</u>**002330202123220031303011311**<u>3103</u>**102

30220121310210222221111123**<u>33002</u>**210212222011131331012 02

1032**<u>22331</u>**102032233310220221321213202**<u>13322</u>**123120000300

03100302031233000**<u>210</u>**022202113223131001020122310**<u>0021</u>**2

02203213332122320103312320**<u>00330</u>**032103322323103002310 2

00223**<u>02101</u>**222101122130121110233220231**<u>32001</u>**330112113 02

31120030303131301**<u>0111</u>**213302121220201323113003110**<u>30323</u>**

13300121010221221201203103**<u>02321</u>**203203333322223111003

32032333302221211202303201333112203013311120330332132

32130321**<u>13323</u>**123

Figure 5.10 An example of a PRMVS (with $q=4$, $n=5$) of length 1024 with multi-valued milestones distributed with a 32 step-length. The milestones are bold faced and underlined.

5.5 Pseudo-Random Multi-Valued Product Array

Pseudo-Random Multi-Valued Arrays (PRMVA) can be obtained from PRMVS in a similar manner as the generation of PRBA from PRBS, A straight forward method to generate a PRMVA would be to fold a PRMVS in a similar way as shown in Figure 5.5 for the PRBA. However, a PRMVA obtained in this way, will require a look-up code conversion which is not practical for real life applications as the size of the ROM used to implement this conversion is increasing exponentially with the *n*-tuple size.

A Pseudo-Random Multi-Valued Product Array (PRMVPA) represents an alternative encoding method which results in a better code conversion performance than PRMVA. The multi-valued element A(i,j) of a PRMVPA is generated as illustrated in Figure 5.11 by a cross multiplication of the multi-valued terms of a PRMVS along the X coordinate axis: S(i) and another PRMVS along the Y coordinate axis S(j): [A(i,j) = S(i) ρ S(j)]

For the example shown in Figure 5.11, the specific case: q1 = 2, q2 = 4, n1 = 4, n2 = 2 which results in

$$\text{Length}_{\text{SEQUENCE 1}} = q_1^{n1} - 1 = 15;$$
$$\text{Length}_{\text{SEQUENCE 2}} = q_2^{n2} - 1 = 15.$$

The sequences, S(i) and S(j) are

S(i)= 0 0 0 1 0 0 1 1 0 1 0 1 1 1 1

S(j)= 0 1 1 A^2 1 0 A A 1 A 0 A^2 A^2 A A^2

The table needed to generate this PRMVPA is:

Product law

S(i) \ S(j)	0	1	A	A^2
0	s0	s1	s2	s3
1	s4	s5	s6	s7

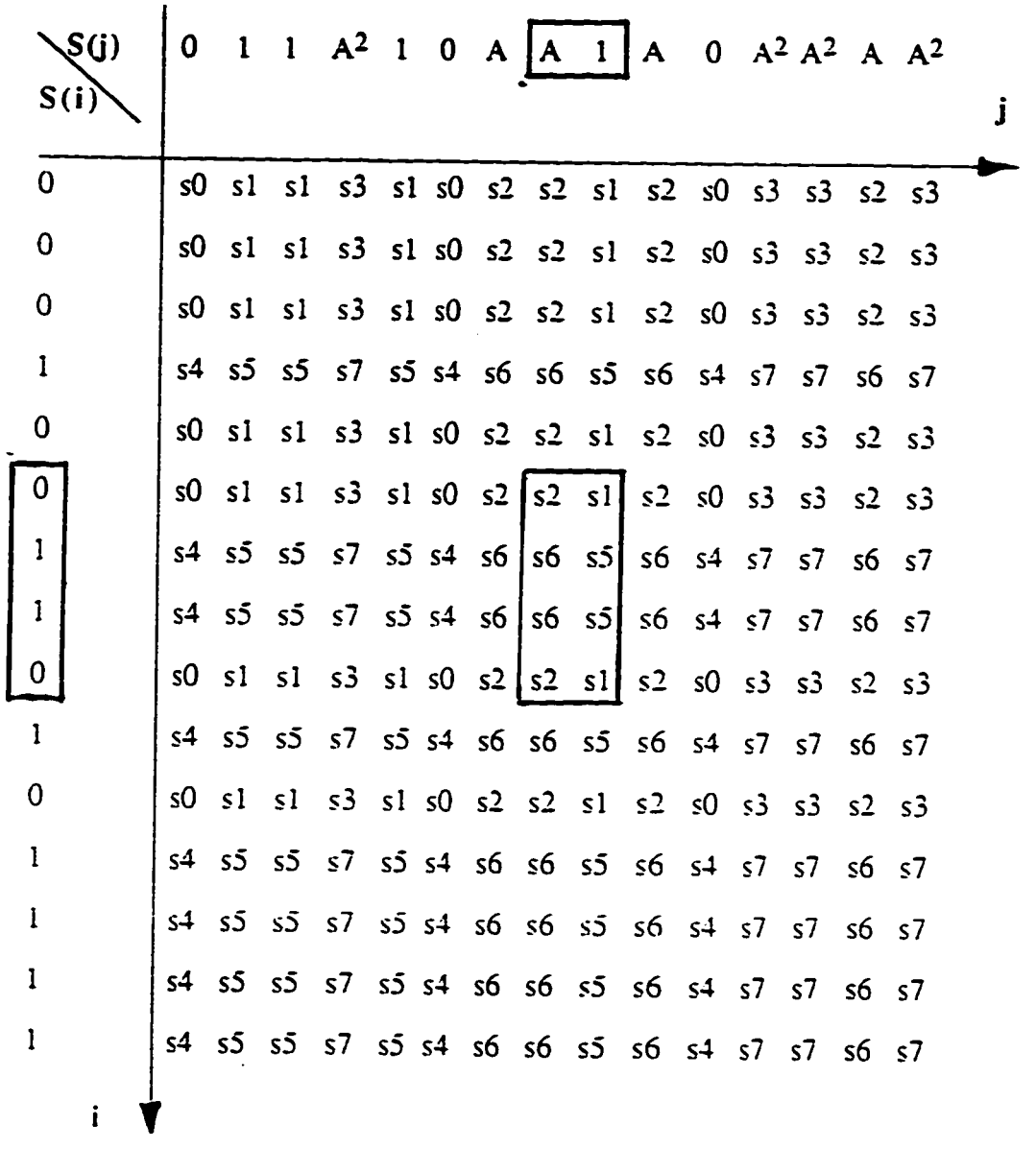


Figure 5.11 **Example of Pseudo-Random Multi-Valued Product Array (PRMVPA)**

To recover a position in this array using a 2-by-4 window, it is required to have at least a sufficient number of symbols to be recognized to permit independent x and y recovery. An example window is shown below:

s2 s1

s6 s5

s6 s5

s2 s1

This window contains p × q symbols, but the minimum will need only p + q - 1 symbols for recovery. The other symbols are redundant and can be used for error correction and increasing the probability of correct recovery.

In order to identify the position of a window within an array, we have to find the index values of ***i*** and ***j*** of the upper left cover element of the window. The 2-D code conversion is reduced to two 1-D code multi-valued conversions which are done in the serial/parallel code conversion as discussed in section 5.4.3.

As a closing remark, it may be interesting to compare the communications and respectively the position measurement approach to the pseudo-random/natural code conversion (or index

recovery) problem.

In the communications case the incoming pseudo-random n-tuple is usually corrupted by noise and there is not practically possible to discard it and ask to be sent again. Index recovery is done by looking for the local pseudo-random *m*-tuple local replica (of known index) which gives the optimal statistical match (e.g. correlation coefficient) with the incoming *m*-tuple.

The code conversion problem is simpler in the position measurement case when more than the minimum required *m* symbols can actually be read. This allows to discard any erroneous *m*-tuple and to look for a "good" m-tuple (i.e. one which has correct pseudo-random relationships with its neighbours). Index recovery is done in this case by looking for the local pseudo-random m-tuple local replica (of known index) which gives the perfect match with the "good" incoming *m*-tuple. The code converter can be implemented simple as a pseudo-random shift-register preloaded with the incoming m-tuple. The relatively fewer constrains attached to the position measurement applications of the pseudo-random encoding allow for the use of the multi-valued pseudo-random sequences and arrays. When supported by high resolution sensors such a multi-valued encoding results in a higher encoding resolution as required for practical 3-D object recognition applications.

Chapter 6

Model-based Tactile Object Recognition Using Pseudo-Random Encoding:

Experimental Results

The tactile object recognition paradigm can be formally stated as follows: *Given a set of 3-D objects having their surfaces Braille-like embossed with terms of a Pseudo-Random Array (PRA) for which it is a priori known how the unfolded faces of the geometrical models of these objects are mapped into the "physical" PRA plane, and given a partial tactile image of an object surface, determine the identity of the touched object.*

This paradigm is illustrated in Figures 6.1 and 6.2. As shown in Figure 6.1, the investigated object surfaces are permanently encoded in a manner similar to Braille coding (used by blind people) with elements of a large PRA in such a way that any region of the array is used only once.

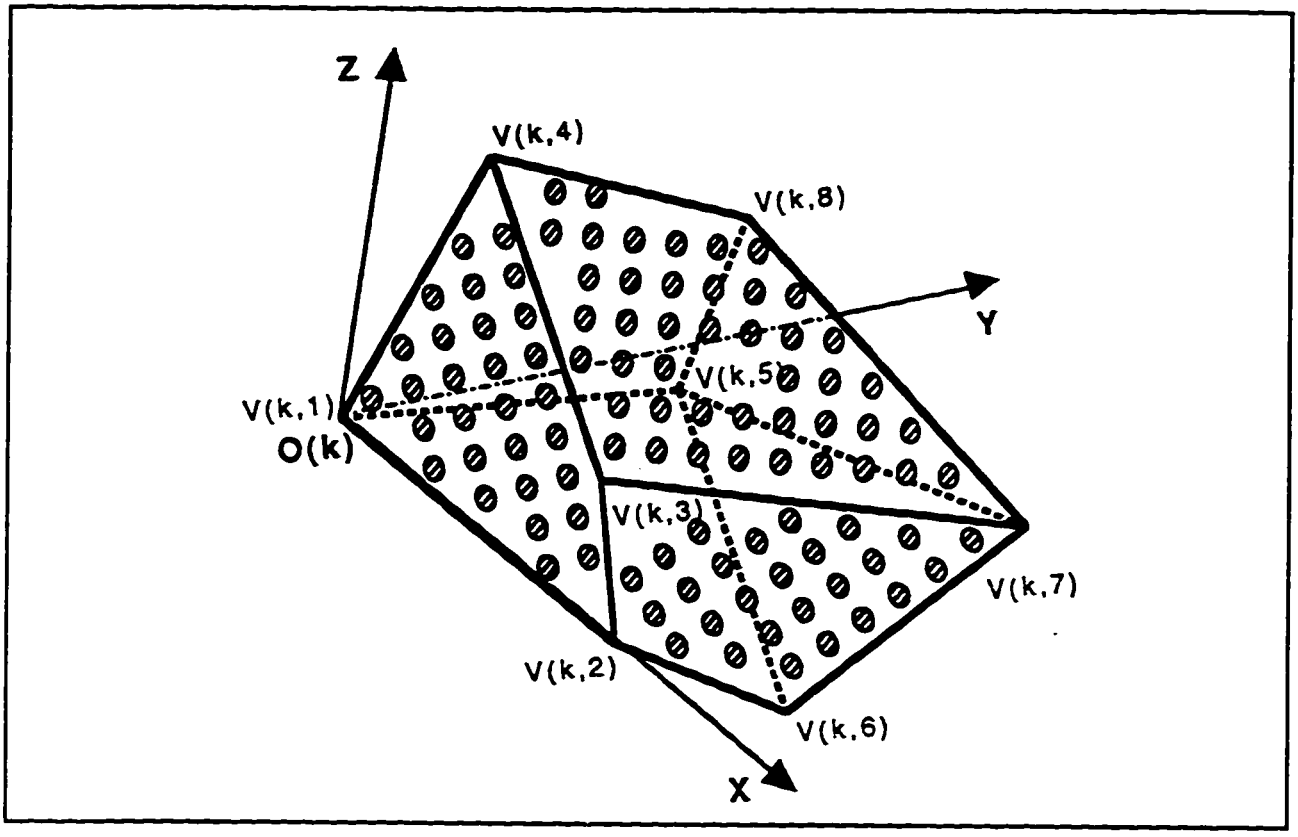


Figure 6.1 Polyhedral model of the PRA encoded object and its 3-D reference frame.

Using polyhedral geometric models of the 3-D objects the object surfaces are flattened and then unfolded, as illustrated in figure 6.2, to be mapped on the encoding PRA.

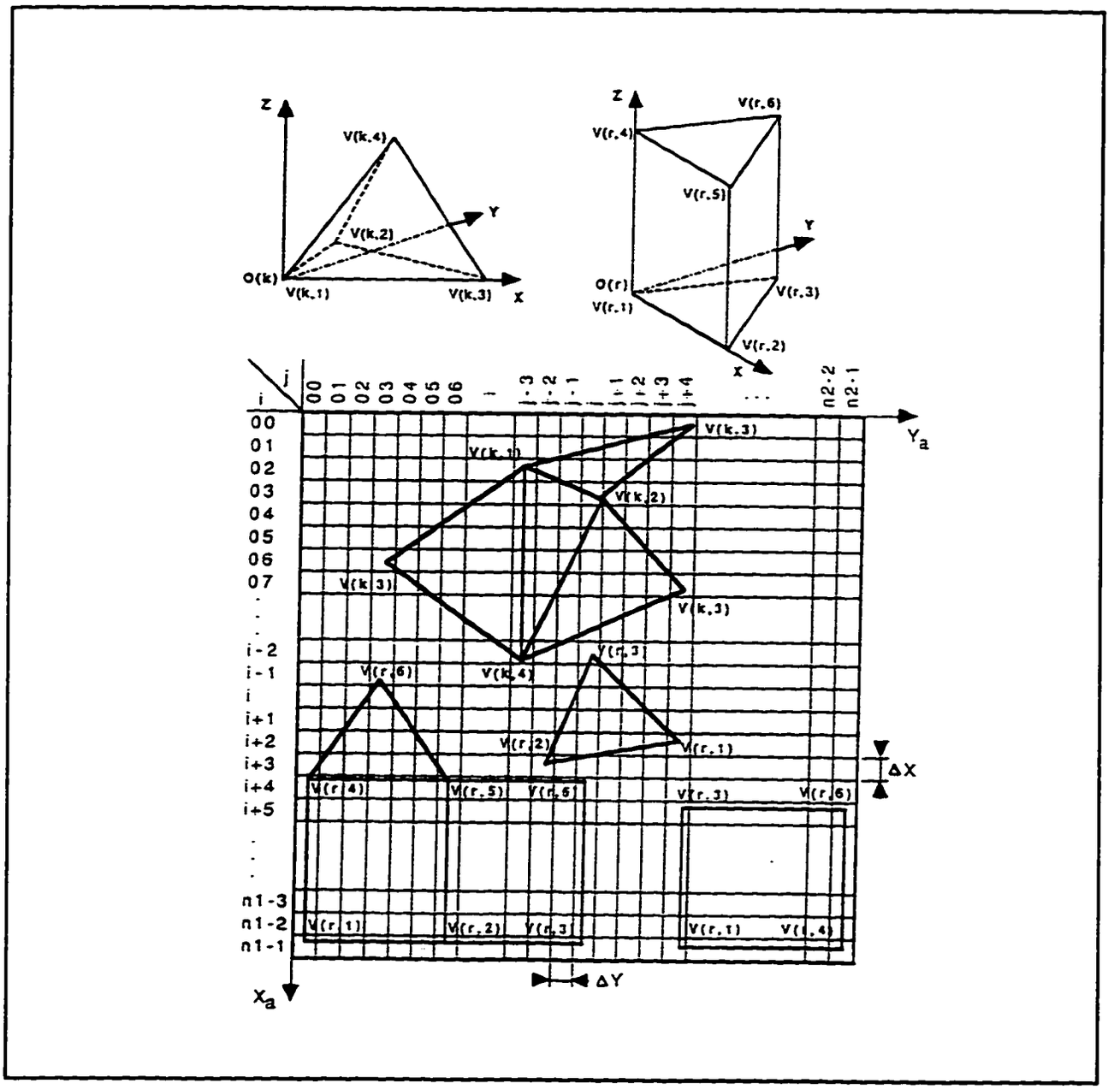


Figure 6.2 The geometric models of two objects (indexed k and r) are unfolded on the encoding PRA.

The mapping of the models on the PRA is implemented as a

relational database. There is only one polygonal face which may contain a given symbol A(i,j). The database system will report the identity of object type, and the specifics of the object where a given point is located. The tactile sensing is actually reduced to the recognition of only two types of symbols embossed at different locations on the encoded object surface.

Usually, only part of the encoding symbols marked on the explored object surface are actually recognized. Based on the recognized symbols, a broken portion of the PRBA can be reconstructed and then inspected to find a complete *k1-by-k2* window. The pseudo-random/natural code conversion of the recovered window contents yields the (i,j) coordinates of the origin P(i,j) of that window within the encoding PRA.

Consulting the database which stores the "object geometry/PRA" mapping, it is possible to recognize the specific polygonal object face (and thus the object) containing the recovered window as illustrated in Figure 6.3.

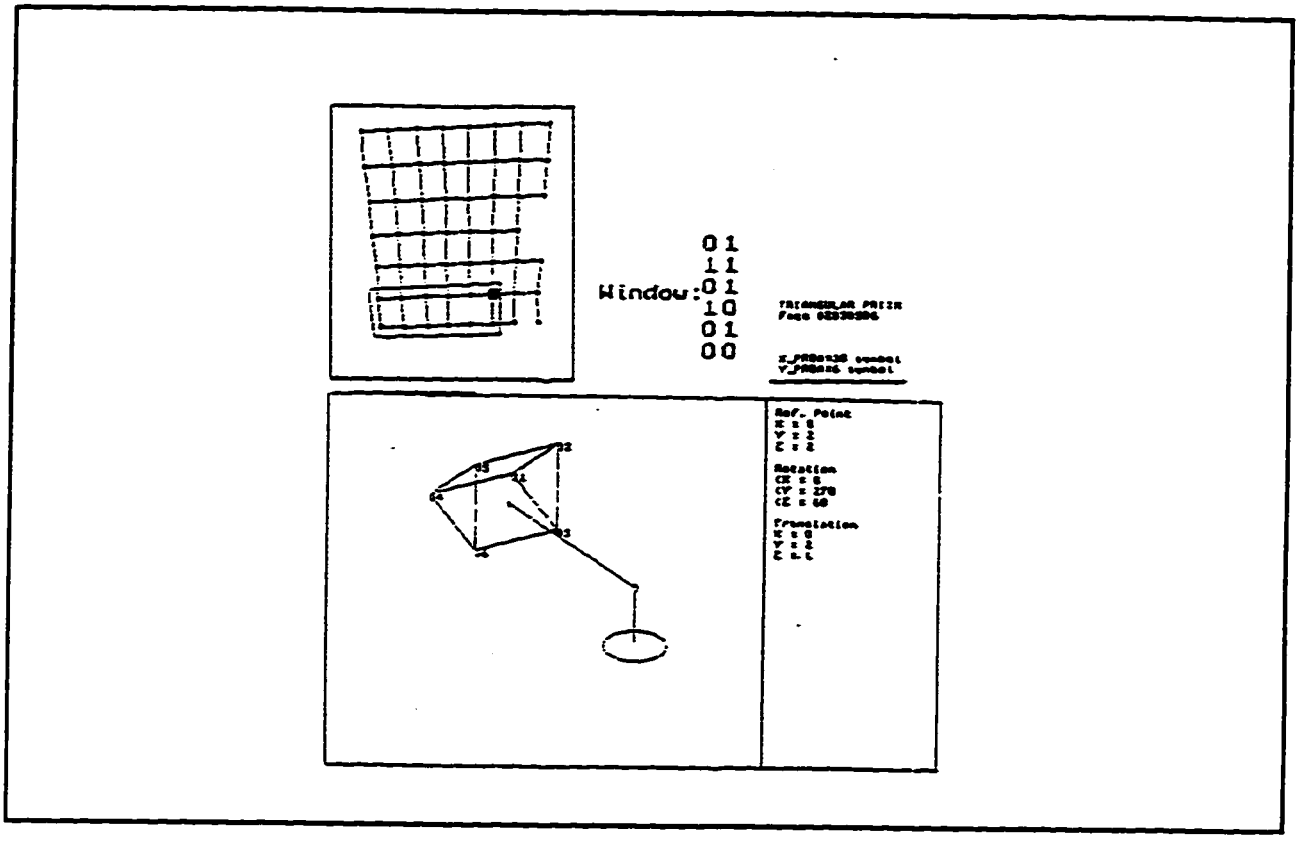


Figure 6.3 The recovered PRA window allows identification of the specific model face to which that window belongs

Two experiments were conducted : one using a PRBA based on only two types of symbols and another using a PRMVA based on four types of symbols.

6.1 Tactile Recognition of Pseudo-Random Binary Symbols

In the simplest case of PRBA encoding, two geometrically distinct symbols are used to mark the object surface with the binary values of the PRBA elements. These binary symbols are recognized from tactile images using specific symbol definition properties.

For an efficient pattern recognition, the particular shape of the binary symbols were selected in such a way to meet the following demands [6.1]:

(1) there is enough information at the symbol level to provide an immediate indication of the grid orientation;

(2) the symbol recognition procedure is invariant to position, and orientations;

(3) the symbols have a certain peculiarity so that other objects in the scene will not be mistaken for encoding symbols.

The binary symbols which we used to mark "0" and "1" are recognized on the basis of the number of end-points and

vertices. As shown in Figure 6.4, the symbol representing "0" has 2 end-points and 1 vertex-point and the symbol for "1" has 3 end-point and 2 vertex-points.

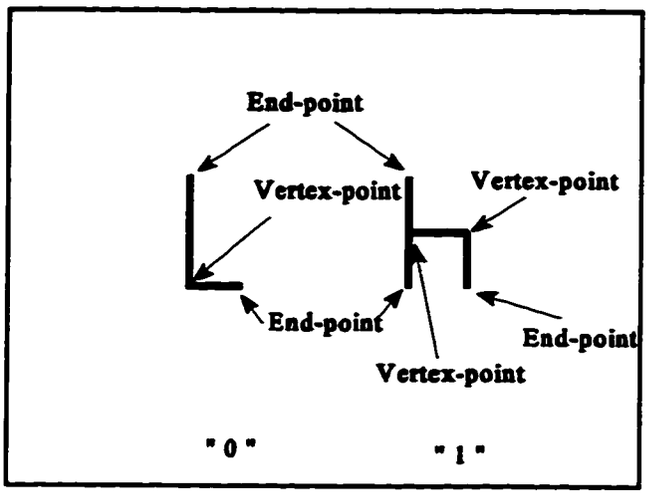


Figure 6.4 The two symbols used to mark the binary values "0" and "1" within the PRBA

Figures 6.5 and 6.6 show examples of the experimental tactile images of the binary symbols "0" and "1" respectively.

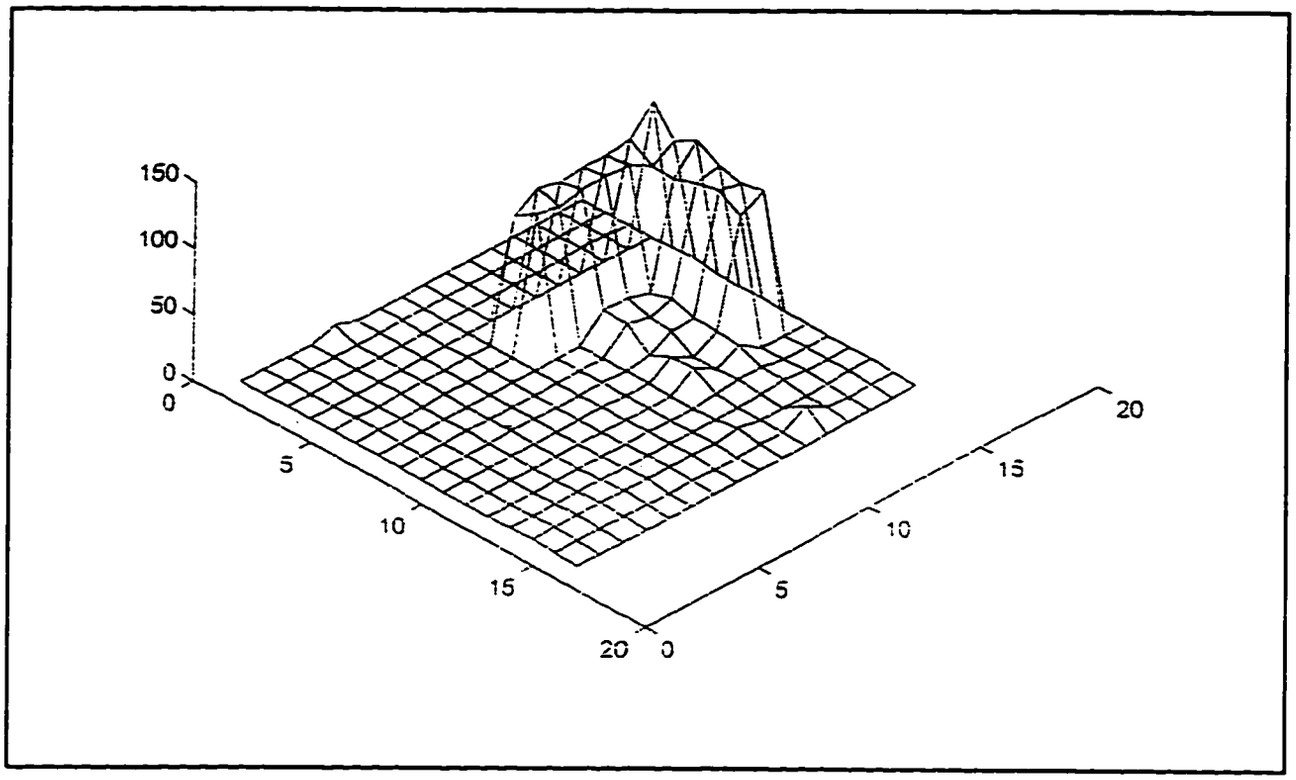


Figure 6.5 Experimental tactile image of binary symbol "0"

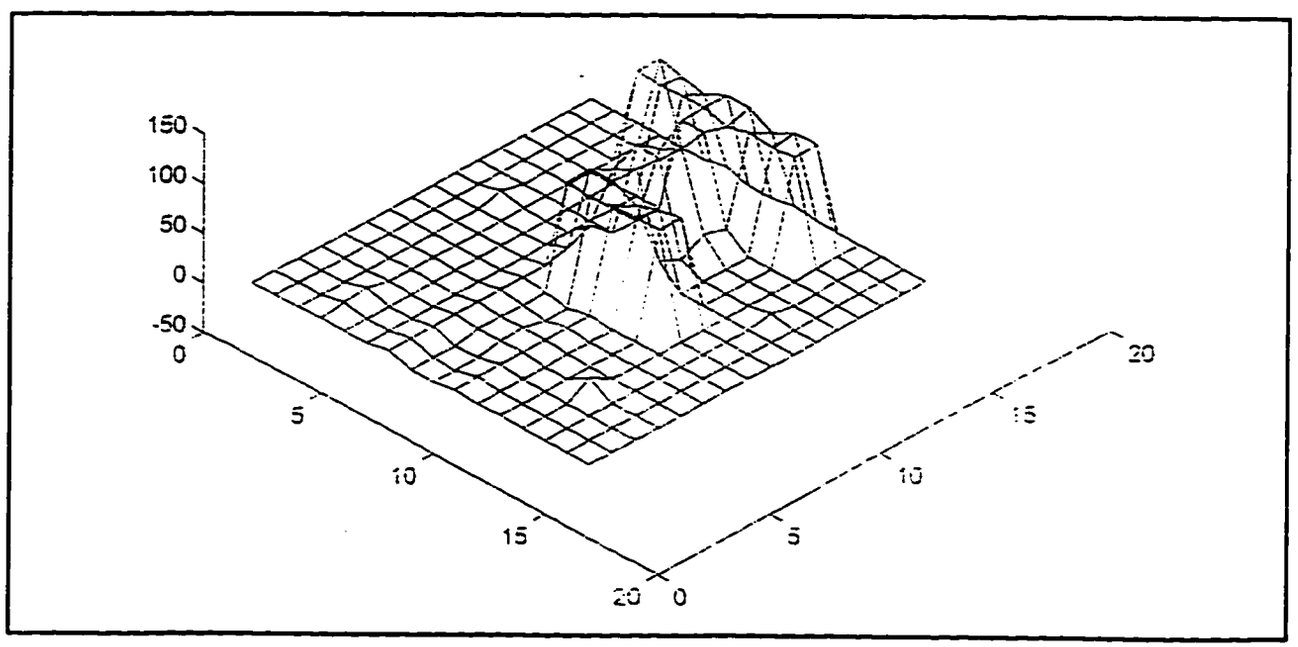


Figure 6.6 Experimental tactile image of binary symbol "1"

The features used to recover the position and orientation of the symbols sensed on explored object are shown in Figure 6.7:

- the x- and y-coordinates of the symbol position;
- the directions of the symbol x and y axes;
- the expected distances along the symbol axes to the neighboring symbols.

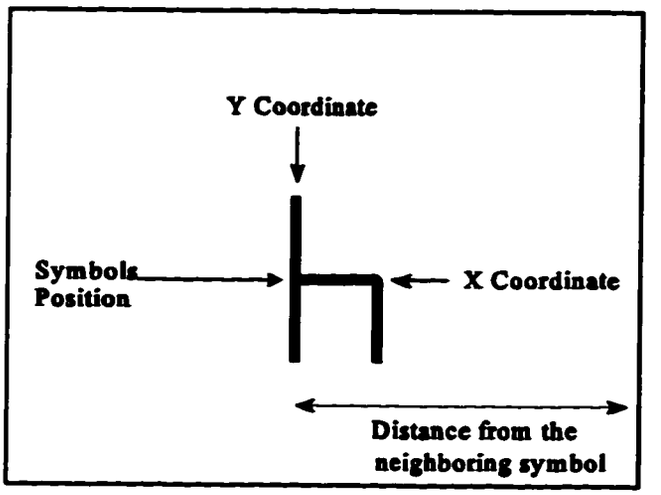


Figure 6.7 Features describing a binary symbol

The PRBA reconstruction is independent of the employed binary symbol shapes and the corresponding symbol recognition technique. Its input consists of one set for each extracted symbol features, and deals with their organization into a list

of symbol structures and the recognition of the two dimensional grid pattern. When a symbol is discovered, its features are added to the linked list of symbols already recognized. From this list, the expected coordinates of the current neighboring symbol are calculated (Figure 6.8) and then compared with actual measured coordinates of these neighbors. If the difference between the calculated coordinate expectations and the compared actual symbol coordinates does not exceed a certain threshold, a neighbor candidate is accepted for integration into the PRBA grid. In this way, erroneously recognized symbols are expelled before being integrated in the PRBA. If more candidates are found for a given grid node, then the 'best fit' is chosen as shown in Figure 6.8.

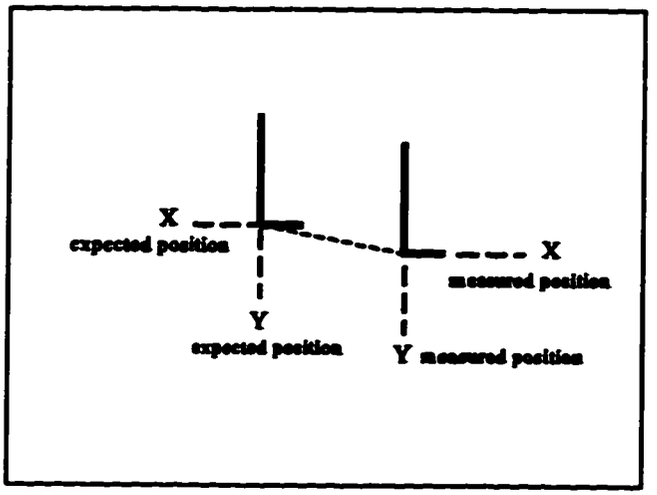


Figure 6.8 (a)

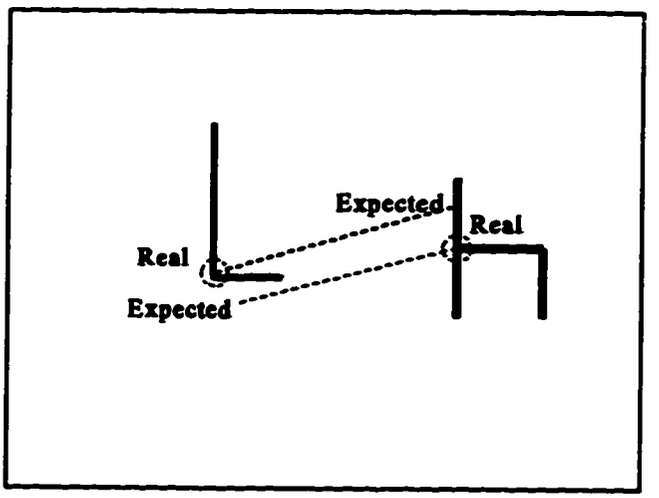


Figure 6.8(b)

Figure 6.8(a)& (b) Acceptance test for neighbouring symbols in the case of PRBA encoding

6.2 Tactile Recognition of Multi-valued Pseudo-random Symbols

Multi-valued encoding results in a higher encoding density than simple binary encoding. In this case, Pseudo-Random Multi-Valued Arrays (PRMVA) or a Pseudo-Random Multi-Valued Product Array (PRMVPA) are embossed on the object surfaces. Of course, the pattern recognition problem becomes more complex as more than two symbol types have to be recognized.

Let us consider for instance, an encoding PRMVPA with n1=15 lines (k1=4, and k2=4) and n2=15 columns, as shown in Figure 6.9. This PRMVPA was obtained by multiplication of two PRMVS, generated by a n=4 stage shift register over GF(4) = [0, 1, A, A^2]. The 2-D position index in this PRMVPA is recovered by the unique contents of a 4-by-4 window.

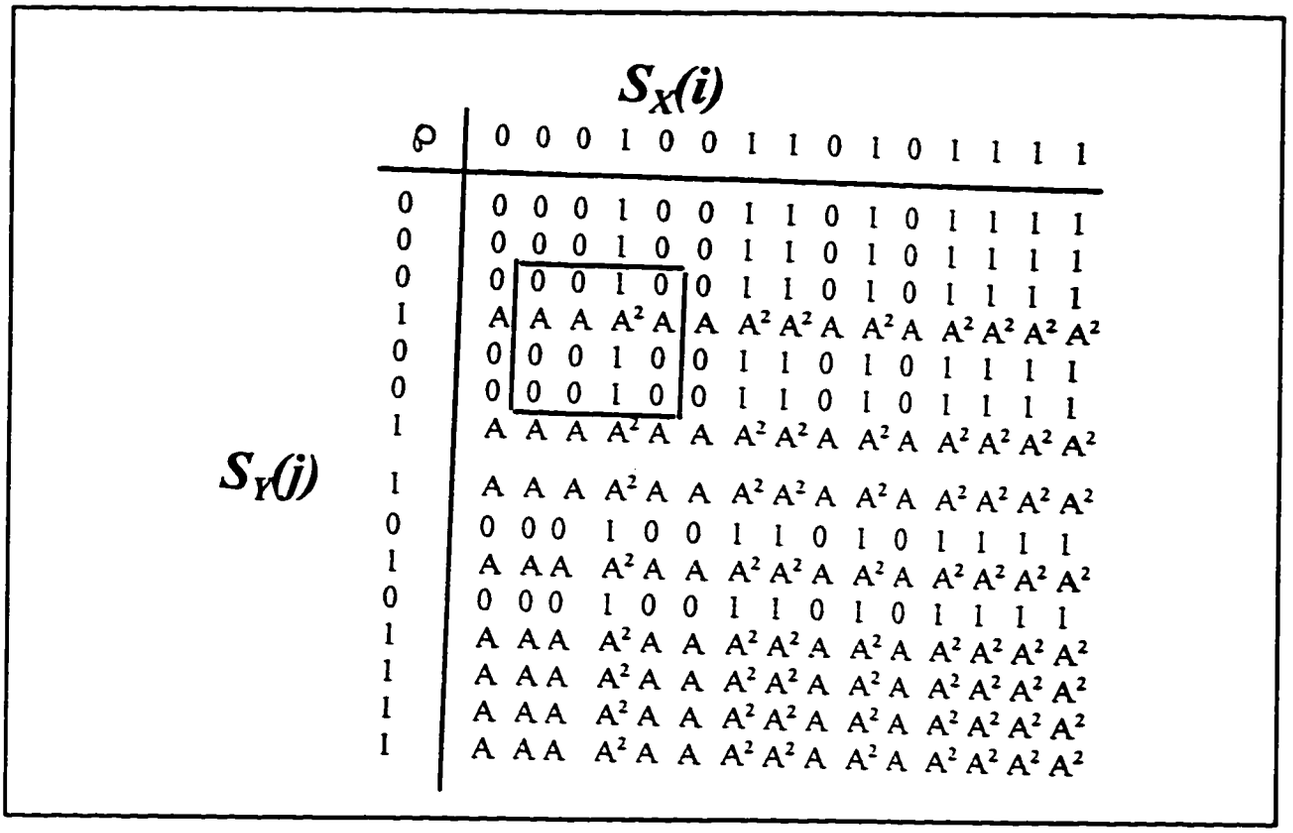


Figure 6.9 A 15-by-15 PRMVA obtained by multiplying two PRMVS defined over GF(4)

The picture in Figure 6.10 illustrates what an actual PRMVPA encoded object surface looks like. Figure 6.11 shows an experimental global tactile image (GTI) obtained by the integration of nine tactile probe images (TPI) of a PRMVPA encoded object.

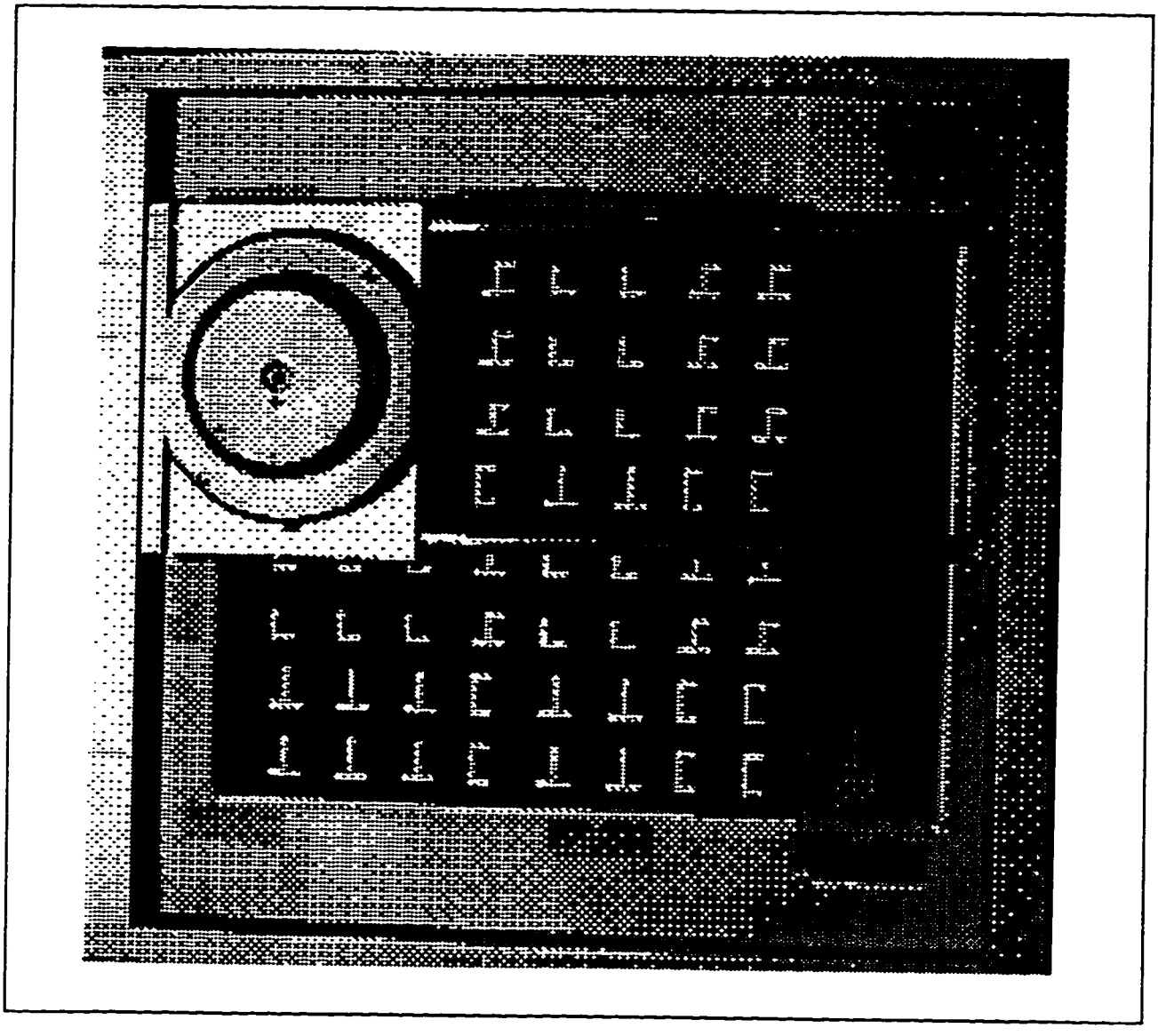


Figure 6.10 Braille-like encoded object surface used for experiments on tactile object recognition

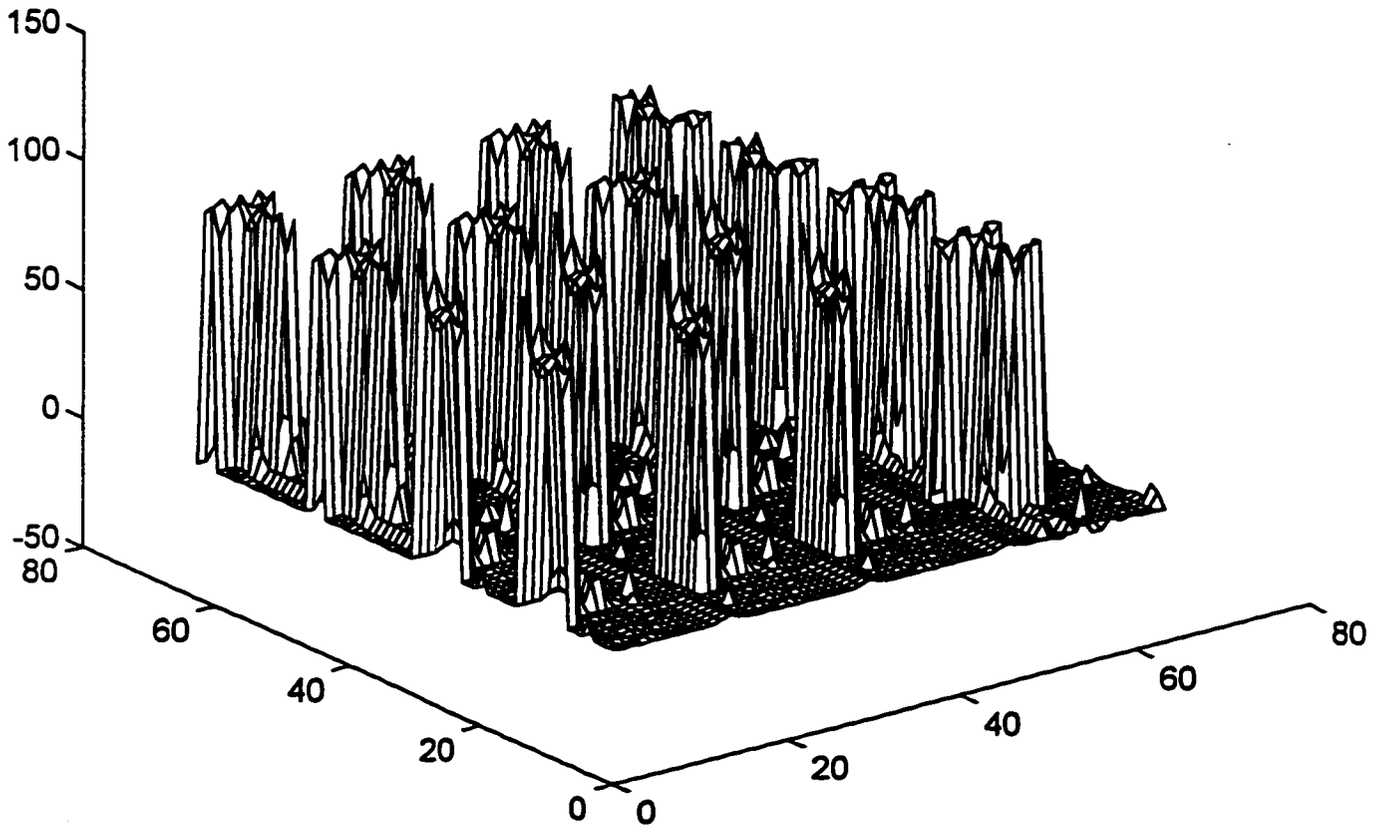


Figure 6.11 The experimental tactile image of the Braille-like encoded surface.

As shown in figure 6.12, four distinct graphical symbols are used to identify the four elements of the encoding PRMVPA. In order to simplify the pattern recognition process, the particular shapes of these graphical symbols were selected to meet the following demands(same as in the case of the binary symbols):

- there is enough information at the symbol level to provide an immediate indication of the grid orientation;
- the symbol recognition procedure is invariant to position and orientation;
- they have a certain peculiarity so that other objects in the scene won't be mistaken for encoding symbols.

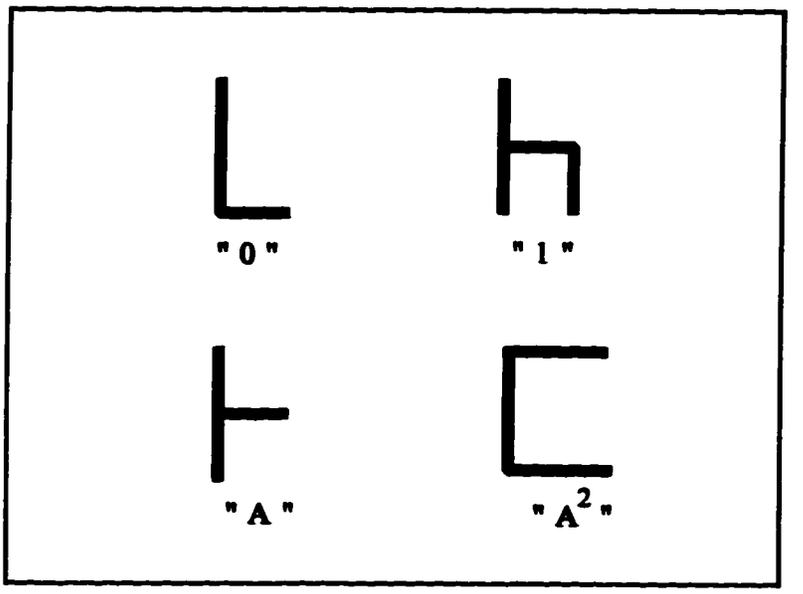


Figure 6.12 The four symbols used to mark the encoding elements within the PRMVA

The four symbols are recognized by the number of end-points and vertices. The symbol representing "0" has 2 end-points and 1 vertex-point, the symbol for "1" has 3 end-points and 3 vertex-points, the symbol for "A" has 3 end-points and 2 vertex-points, and the symbol for "A^2" has 2 end-points and 2 vertex-points.

Figure 6.13 shows, as an example, the experimental tactile images of the binary symbols "0", "1", "A", "A^2".

The features used to recover the position and orientation of the symbols sensed on the explored object surface are:

- the x- and the y-coordinates of the symbol's position;
- the directions of the symbol's x- and the y-axes;
- the expected distances along the symbol's axes to the neighboring symbols.

As in the case of PRBA, the PRMVPA reconstruction is independent of the employed symbol shapes and the corresponding symbol recognition technique. When a symbol is discovered, its features are added to a linked list of symbols already recognized. From this list, the expected coordinates of the current neighboring symbol are calculated and then compared with actual measured coordinates of these neighbors. If the

difference between the calculated coordinate values and the measured symbol coordinates doesn't exceed a certain threshold, the neighbor candidate is accepted for integration into the PRMVPA grid. In this way, erroneously recognized symbols are rejected before being integrated in the PRMVA. If more candidates are found for a given PRMVA grid node, then the best fit is chosen.

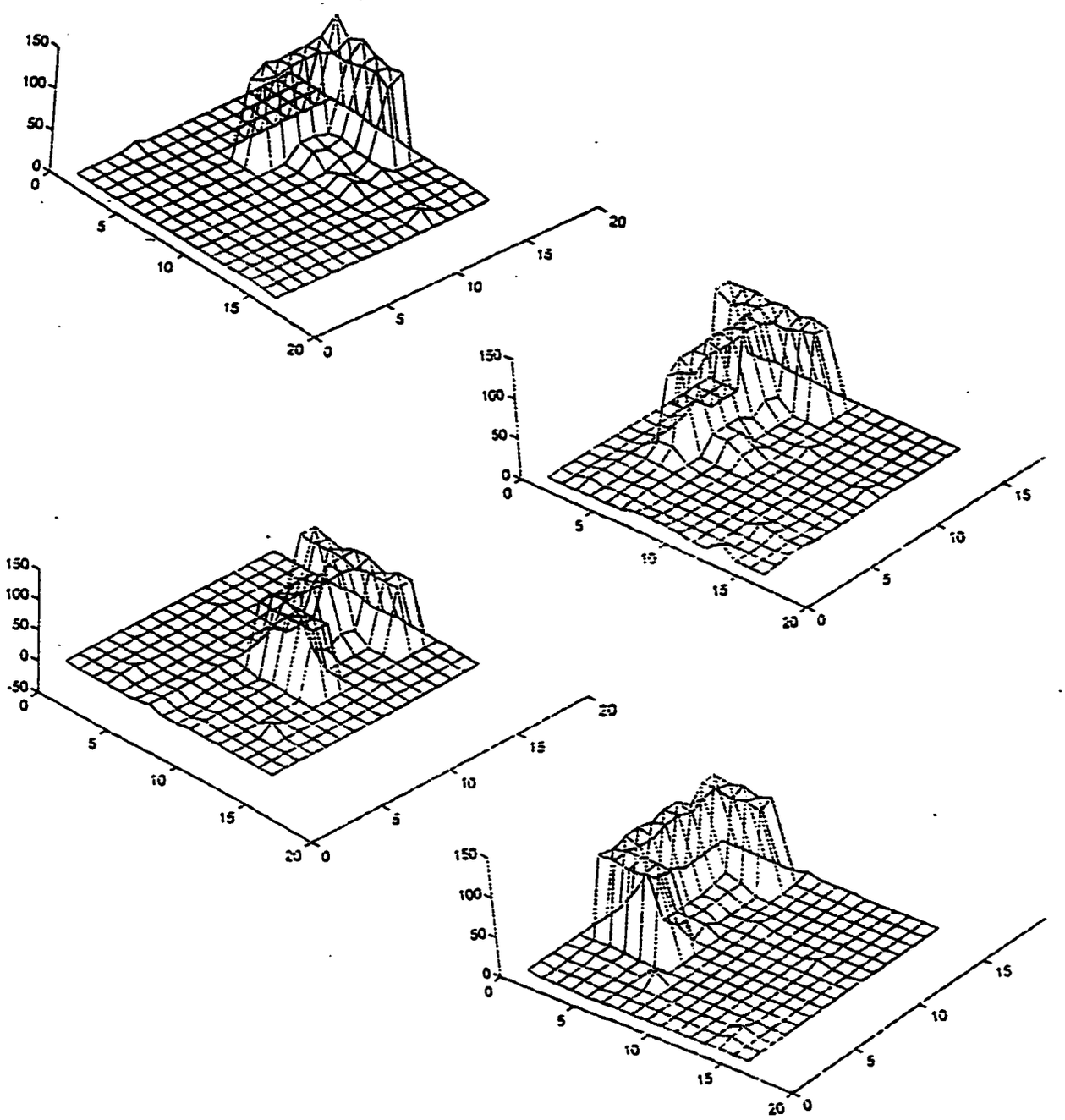


Figure 6.13 Experimental tactile images of the four encoding symbols

6.3 Model-Based Recognition of The Encoded Objects

Our experiments considered four representative types of objects: parallelepiped, cube, triangular prism and tetrahedron. The winged-edge polyhedral models of these objects [6.2] are shown in Figure 6.14 to 6.16. As it can be seen, in a winged-edge representation each edge has associated four links specifying the two object faces separated by that edge and the two vertices delimiting the edge. These models were implemented as a relational database using the scheme shown in Figure 6.16.

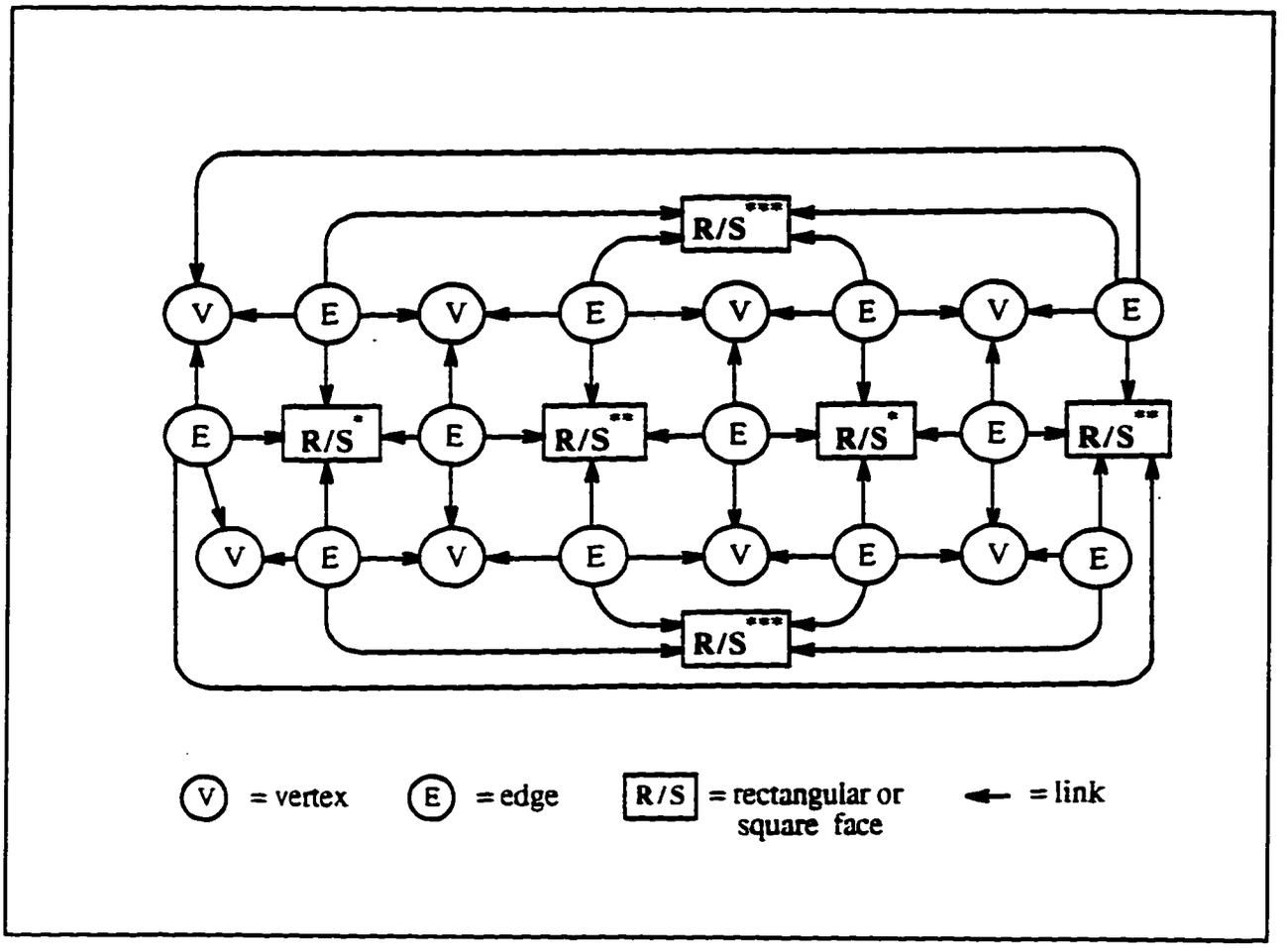


Figure 6.14 **Winged-edge representation for parallelepiped and cube. Note: In the case of a cube all faces are square (S). For a parallelepiped the faces could be either rectangular with unequal sides (R) or square (S) with the condition that pairs of faces marked with the same number of asterisks (*) are similar and at least one pair is of type R.**

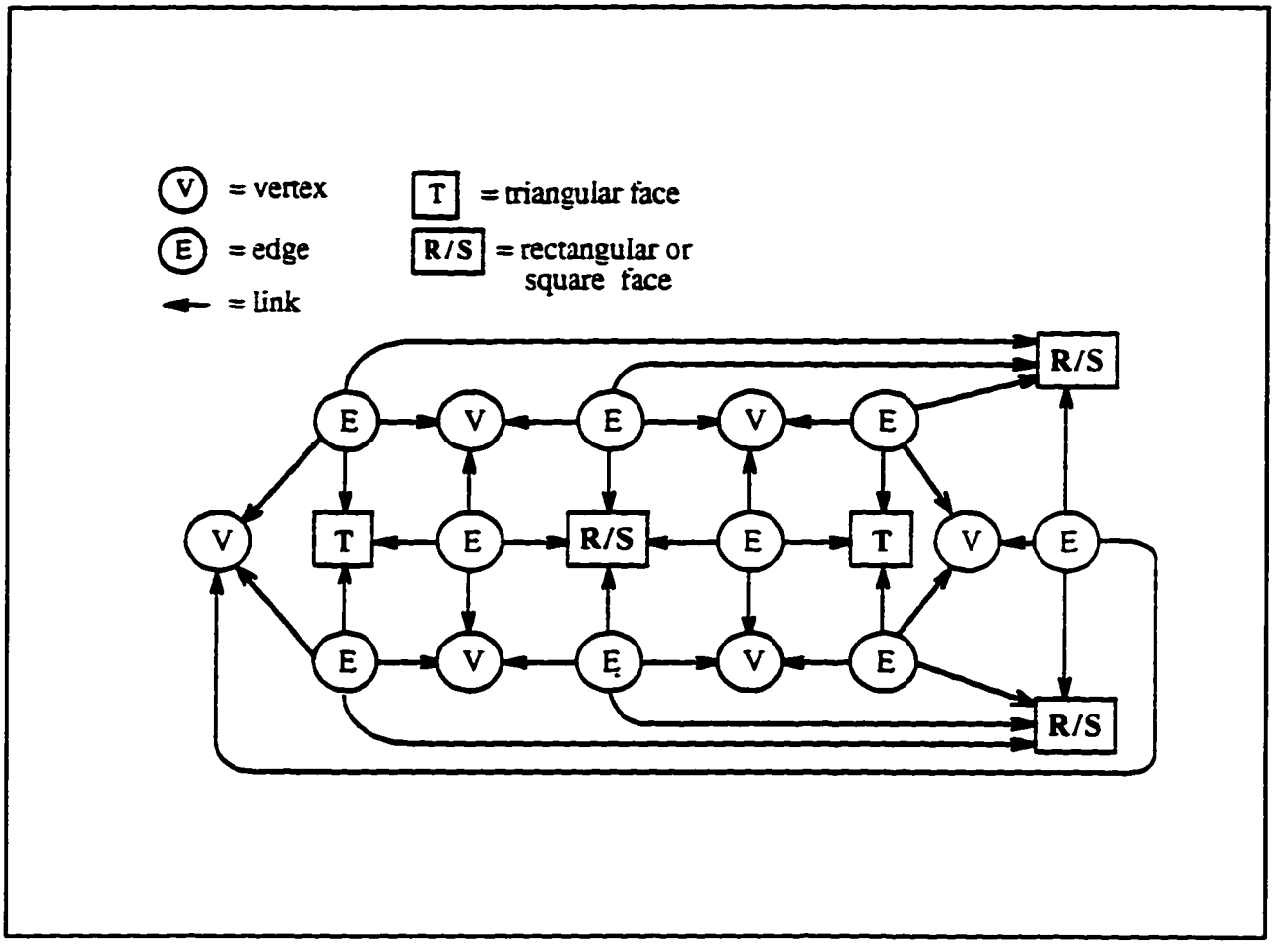


Figure 6.15 Winged-edge representation for the triangular prism

The database variables are classified as follows:

OBJECTS: object_id, object_type, number_of_vertices, number_of_faces;

VERTICES: vertex_id, object_id, x_vertex, y_vertex, z_vertex;

FACES: face_id, object_id, face_type, number_of_edges, face_area;

EDGES: edge_id, face_id, edge_length;

FACES_TO_VERTICES: vertex_id, face_id;

EDGES_TO_VERTICES: vertex_id, edge_id.

FACES_TO_VERTICES and EDGES_TO_VERTICES are connector files used to transform the many-to-many relationships between faces, edges and vertices to "one-to-many" relationships, as shown in Figure 6.16.

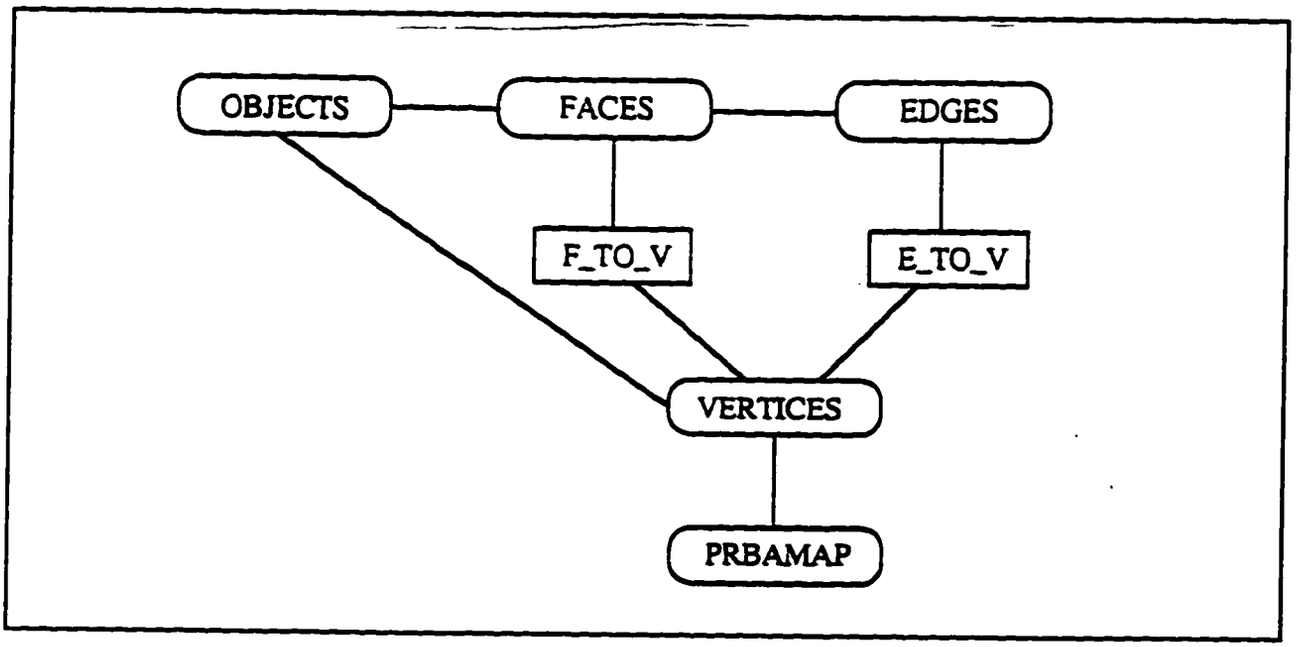


Figure 6.16 File relationship in the database

There is only one polygonal face which can contain a given point P(i,j). A simple algorithm allows one to decide if a recovered window of origin P(i,j) belongs or not to a given convex polygon: if the sum of the areas of all triangles formed by the P(i,j) and each edge of an object face is equal to the area of the same object face, then the P(i,j) belongs to this face; if the sum of these areas is greater than the area of the object face then P(i,j) does not belong to this face.

The database system reports the recognized object_id and vertex_id of all object vertices. As an example figures 6.17 to 6.19 show the wire frame model of the recognized object named "Cube", "Rectangular Prism", and "Triangular Prism" In each case. it also shows the current 3D position of the tactile probe relative to the robot frame.

Essentially, the proposed model-based tactile object recognition method replaces the classical feature-based 3-D object recognition by the simpler recognition of the symbols which are used to mark labels embossed on object surfaces. Once enough symbols are recognized to make sense as an object label search in a database will tell which is the specific object which has been marked with the recovered label. Obviously any error in labelling (mislabelling or ambiguous labelling)will result in error in object recognition.

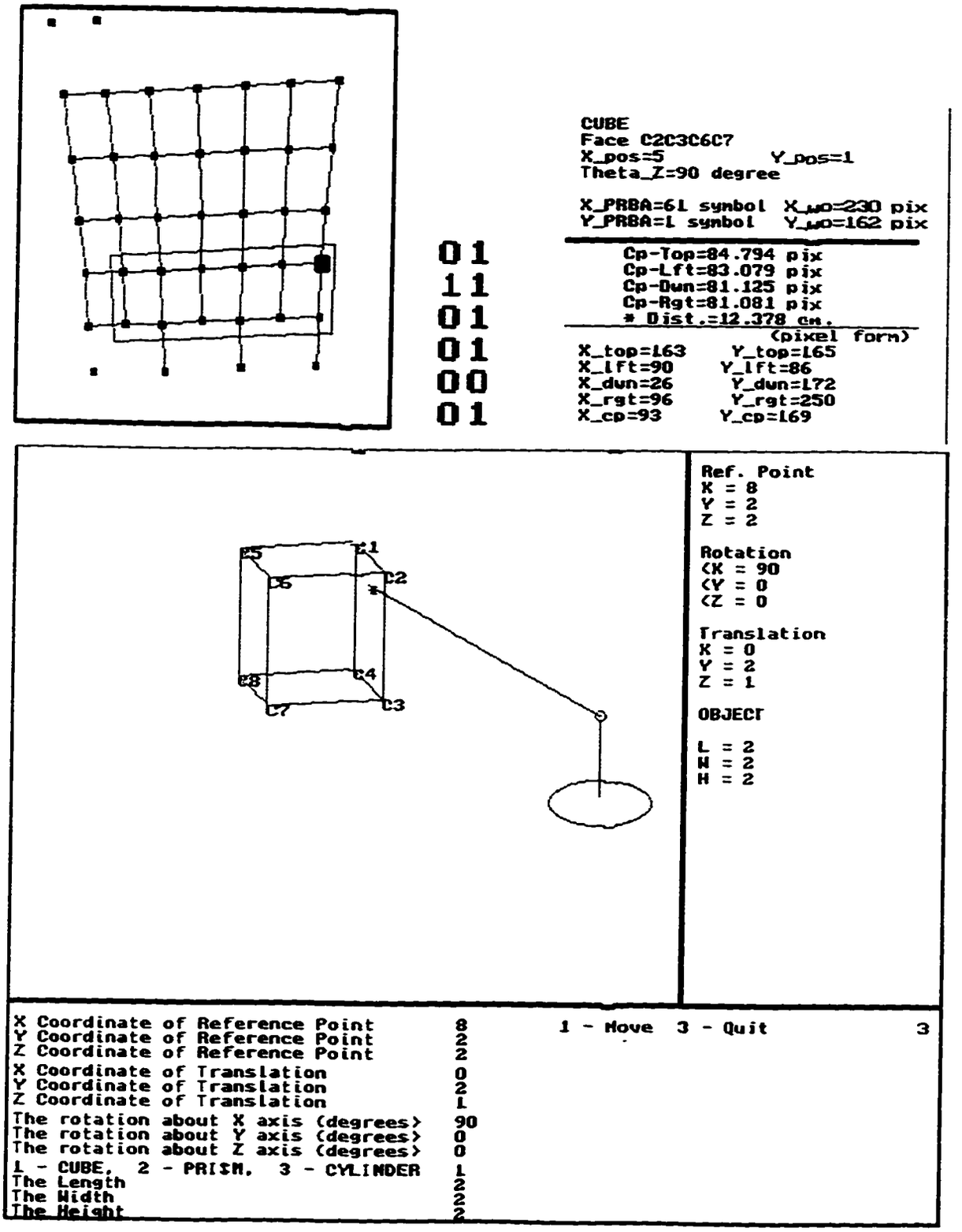


Figure 6.17 Experimental results illustrating tactile model-based recognition of a cube

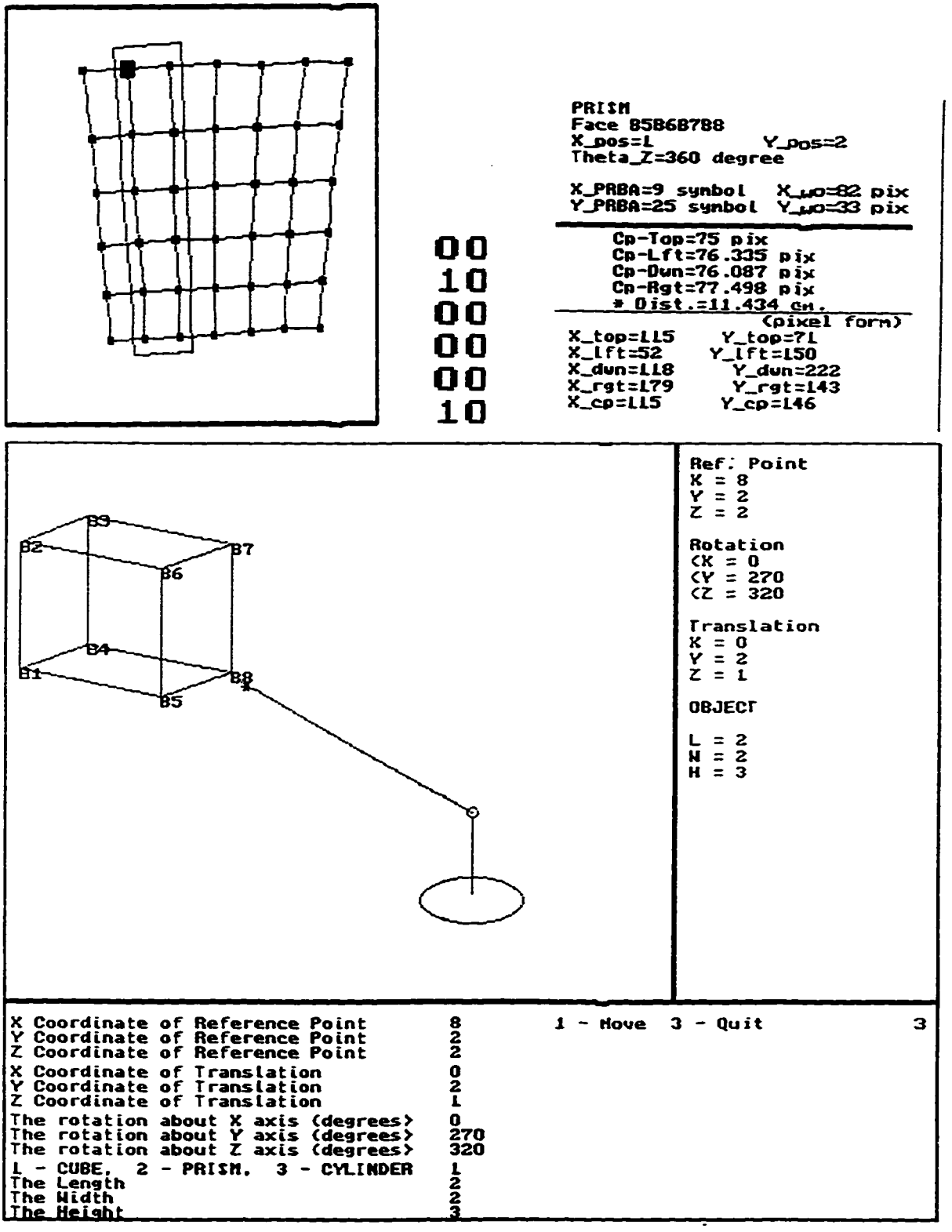
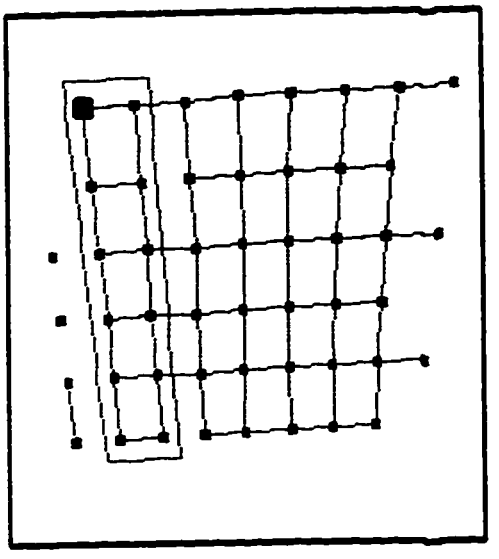


Figure 6.18 Experimental results illustrating tactile model-based recognition of a prism



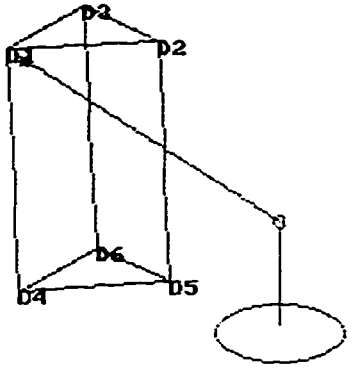
00
01
11
01
01
10

TRIANGULAR PRISM
Face D1D2D4D5
X_pos=1 Y_pos=5
Theta_Z=357 degree

X_PRBA=25 symbol X_μo=35 pix
Y_PRBA=5 symbol Y_μo=40 pix

Cp-Top=74.04 pix
Cp-Lft=73.918 pix
Cp-Dwn=76.087 pix
Cp-Rgt=73.837 pix
* Dist.=11.171 cm.
(pixel form)

X_top=156 Y_top=71
X_lft=97 Y_lft=149
X_dwn=161 Y_dwn=221
X_rgt=219 Y_rgt=143
X_cp=158 Y_cp=145



Ref. Point
X = 8
Y = 2
Z = 2

Rotation
<X = 356
<Y = 0
<Z = 345

Translation
X = 0
Y = 2
Z = 1

OBJECT

H = 3
L = 2

X Coordinate of Reference Point	8	1 - Move 3 - Quit	3
Y Coordinate of Reference Point	2		
Z Coordinate of Reference Point	2		
X Coordinate of Translation	0		
Y Coordinate of Translation	2		
Z Coordinate of Translation	1		
The rotation about X axis (degrees)	356		
The rotation about Y axis (degrees)	0		
The rotation about Z axis (degrees)	345		
1 - CUBE, 2 - PRISM, 3 - CYLINDER	2		
The Height	3		
The Side Length	2		

Figure 6.19 Experimental results illustrating tactile model-based recognition of a triangular prism

6.4 Summary

Experimental results verify the model-based tactile object recognition concept. Binary symbols (used for PRBA encoding) and quaternary symbols (used for PRMVPA encoding) were considered. Special features of the encoding symbol types were selected for the convenient symbol recognition in the relatively poor quality tactile images.

It was shown that the complex tactile object recognition problem can be reduced to a simple tactile recognition of few types of special symbols and using a knowledge data base which maps all encoded object surfaces into a given Pseudo-Random Array.

Chapter 7

Conclusions and Future Research Directions

7.0 Conclusions

This thesis has discussed an original method for the blind recognition of 3-D objects using tactile sensors. This object recognition method combines (1) model-based object recognition, (2) tactile sensing and (3) pseudo-random encoding of the object surfaces.

Essentially, the proposed model-based tactile object recognition method replaces the classical feature-based 3-D object recognition by the simpler recognition of the symbols which are used to mark labels embossed on object surfaces. Once enough symbols are recognized to make sense as an object label search in a database will tell which is the specific object which has been marked with the recovered label. Obviously any error in labelling (mislabelling or ambiguous labelling)will result in error in object recognition.

An experimental robotic tactile sensor system was developed to prove the concept and methodology of the model-based object recognition which uses pseudo-random encoding of the objects.

While it is inherently limited to a given set of embossed objects, the proposed method allows for a simpler and faster "blind" 3-D object recognition using only tactile sensing.

Despite all these limitions, the new tactile object recognition method has potential practical blind robotic applications in space, nuclear stations, underwater, etc. where the cost of permanent object encoding is of no concern and where tactile sensing is the only sensing capability which is available.

7.1 Contributions

The major contributions of this thesis are summarized as follows:

(1) The design and implementation of a physical tactile sensor system, readily used for experiments. Theoretical and experimental studies have been conducted to investigate the effects of the elastic layer mounted on the top of a force-sensitive tactile transducer. Finite Element Method (FEM) simulations were used to understand the characteristics of this elastic layer under stress. This understanding was used further for the design of the tab shaped elastic layer. This design isolates most of the cross-talk between the focus sensing element and the adjacent elements as well as providing a good signal-to-noise level.

(2) Development of an instrumented passive compliant wrist which allows the experimental robotic tactile sensing system to explore more efficiently (higher exploration speed, safer contact,...) the surface of the 3-D objects using touch.

(3) Development of a 2-D correlation technique for the integration of a sequence of local tactile images in a global tactile image of the explored object surface.

(4) Model-based object recognition applications using a Braille-like Pseudo-Random Encoding on the object surfaces. Tactile object recognition is reduced to the identification and position recovery of a very small set of symbol types embossed on object surfaces (sometimes only two types of symbols).

(5) Development of a generalized Pseudo-Random Multi-Valued Product Array (PRMVPA) encoding technique for absolute position recovery. This technique provides a higher resolution than the Pseudo-Random Binary Arrays.

7.2 Future Research Directions

There are a number of promising directions in which this research can be extended:

(1) 3-D Tactile sensor

The current tactile sensor is an array of force-sensitive resistors arranged in a plane. The measurement data are the result of a projection of the stress forces induced in a elastic medium/layer on a rigid 3-D force sensitive transducer array. A novel 3-D sensitive tactile sensor is proposed in Figure 7.1. The sensing elements are embedded inside the elastic layer in such a way to measure the induced stress (the force amplitude and, hopefully, the direction) at different points of the 3-D elastic medium. This tactile sensor structure will be more like a human skin. This "artificial skin" can be shaped as desired to suit applications. A detailed FEM modelling will allow one to understand the complex process of using 3-D distributed sensor data to recover the explored object shaped from the elastic medium deformation.

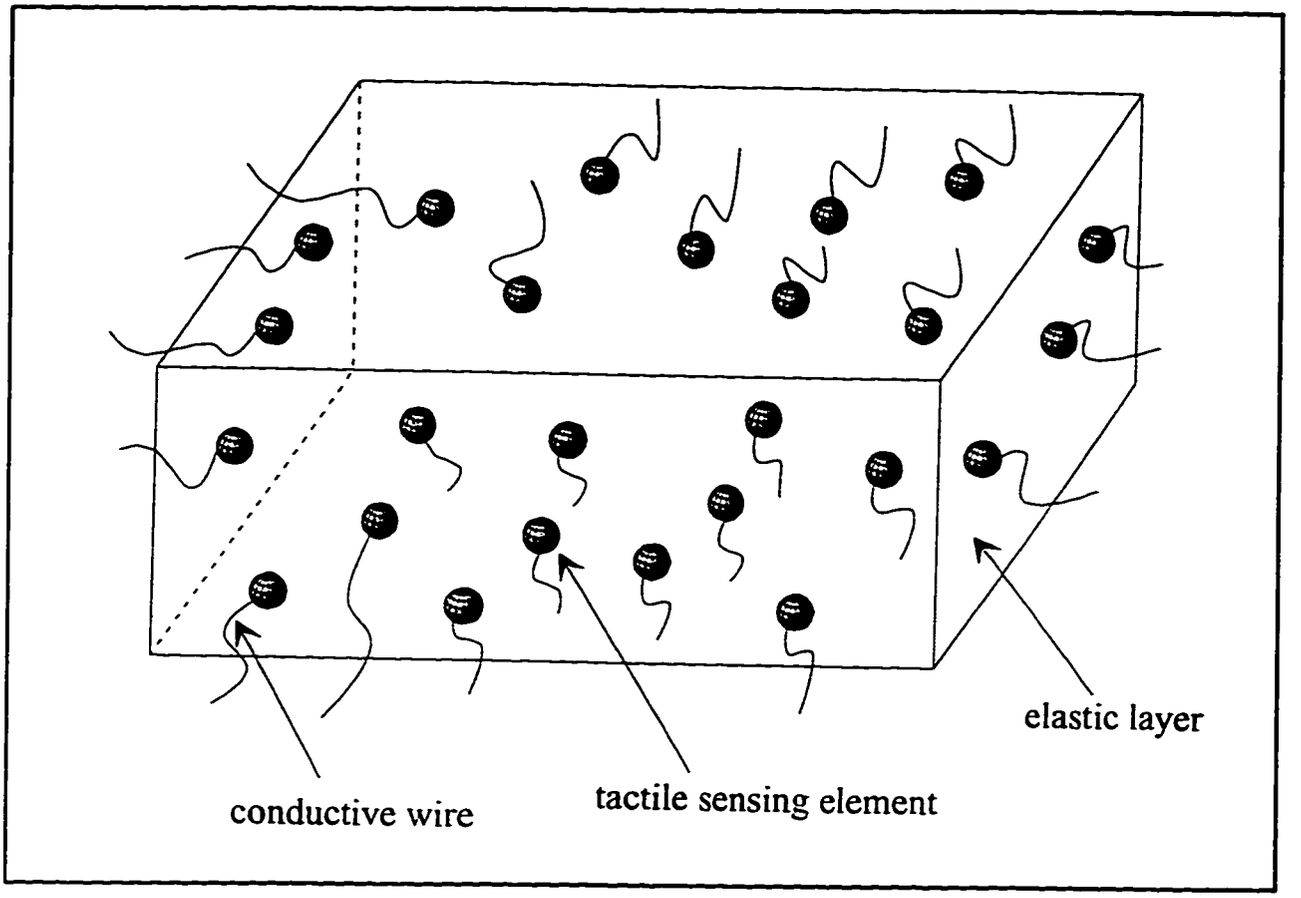


Figure 7.1 Artificial Skin

(2) Neural Networks Pattern Recognition

As suggested in [7.1], a neural network (NN) can be used to learn and recognize an object from the 3-D signature of the distributed tctile sensor. A hardware NN architecture will be needed for real time object recognition.

(3) Development of new a PRMVA natural code conversion algorithm

PRMVA is obtained by folding PRMVS as discussed on Chapter 5. The major drawback of PRMVA is the fact that only a look-up table method for code conversion is known today (which may be quite equipment expensive for high resolutions). A more efficient "PRMVA/ natural" code conversion algorithm can be generated by using an extension of the Chinese Remainder Theorem. The code conversion can be defined by straight modulus division. An illustration is described in Appendix 5.

Appendix I

Biological Aspect of Human Tactile Sensing

A.1.1 Spinal Cord Mechanisms

The sensory systems responsible for the kinesthetic and tactile sensations are organized to receive information from the periphery, process it and relay it to higher neural levels [A1.1].

The first neuron of the chain is a receptor neuron which encodes stimuli into neural signals. It only encodes the information coming from a small region which is called its receptive field. The receptor neurons converge to second-order neurons in the central nervous system. These may be located in the medulla or in the dorsal horn of the spinal cord (that is, the grey matter of the spinal cord). There are four groups of dorsal horn neurons. The Class 1 dorsal horn neurons are dedicated to the cutaneous mechanoreceptors only. Neurons of Class 2 are excited both by mechanoreceptors and nociceptors. The Class 3 corresponds to the neurons excited by the nociceptors only. In Class 4, the neurons are mainly excited by the thermoreceptors.

Information is then transmitted to third-order neurons in the thalamus via parallel pathways: the dorsal columns-medial lemniscus system, the spinothalamic tract, the spinocervical tract, or the

spinoreticular tract (see [A1.2], [A1.1] or [A1.3] for details). The dorsal column-medial lemniscus system differs from the cells of the dorsal horn in its discreteness, its lack of convergence between fibers of different specifications and its weakness to descending control. In addition, it transmits only the information from skin contained in a group of large afferent fibers. Conversely, cells of the spinal cord transmit information from all afferent fibers [A1.4]. The role of the dorsal columns is not clearly outlined, and contradictory interpretations are found in the literature. In [A1.4], it is described that the dorsal columns allow to control the analysis of the amount of information arriving from the other pathways.

The final destination for the sensory inputs is the cerebral cortex [A1.2]. There are two areas of the cortex devoted to somatosensory functions : area S-I and area S-II.

The pathways for conscious perception just described are distinct from the pathway for reflex acts for which the information is transmitted to a motor neuron either directly or through connector neurons, forming a much shorter path in all respects.

Inhibition may occur. It has been classified in three main categories. The surround inhibition is observed when the response of a "cortical unit" is reduced in response to a stimulus applied to an area adjacent to the receptive field. The complementary

inhibition occurs when a stimulus is applied to the contralateral area of the skin. The cross-modality inhibition results from the presentation of another kind of stimuli close to the first receptive field. These phenomenons are collectively referred to as masking.

A1.1.1 Kinesthesia and Cutaneous Senses

The tactual perception includes three senses: cutaneous, kinesthetic and haptic. Cutaneous senses refer to tactile sensations, thermal sensations as well as to pain sensations. It is related solely by variations in the cutaneous stimulation. Kinesthesia refers to the sense of movement but often includes a sense of static position of the limbs. It may also be called Proprioception. In the haptic perception, both tactile and kinesthetic senses convey information.

These are distributed senses, their receptors are far-flung missionaries from the central nervous system. They are located in muscular, tendinous, bony and cutaneous tissue. The receptors are generally excited by tissues deformation. There are several parallel channels to carry the impulses : 7 cranial and 31 spinal nerves.

For hearing and the vestibular system, which also sense mechanical signals, there is only one nerve. Like vision, they are highly organized and localized systems. Kinesthesia and cutaneous senses are therefore quite different in that they extend over large spatial regions.

In the next paragraphs we will see how information is processed through the receptors and we will attempt to summarize human performance.

A1.2 Human Sense of Touch

A1.2.1 Anatomy

The skin is a system of layers composed of:

- The epidermis.
- The dermis or corium which is bellow the dermis. This layer contains the nerve endings, encapsulated or free.
- The reticulated dermis. The combined thickness of the epidermis and the dermis is about 1 or 2 mm.
- Beneath begins the subcutaneous tissue which is often fatty.

Various types of mechanoreceptors are located in the skin and they are different in the hairy skin or in the glabrous skin. The structure of the skin is illustrated by the figures (figure A1.1) below.

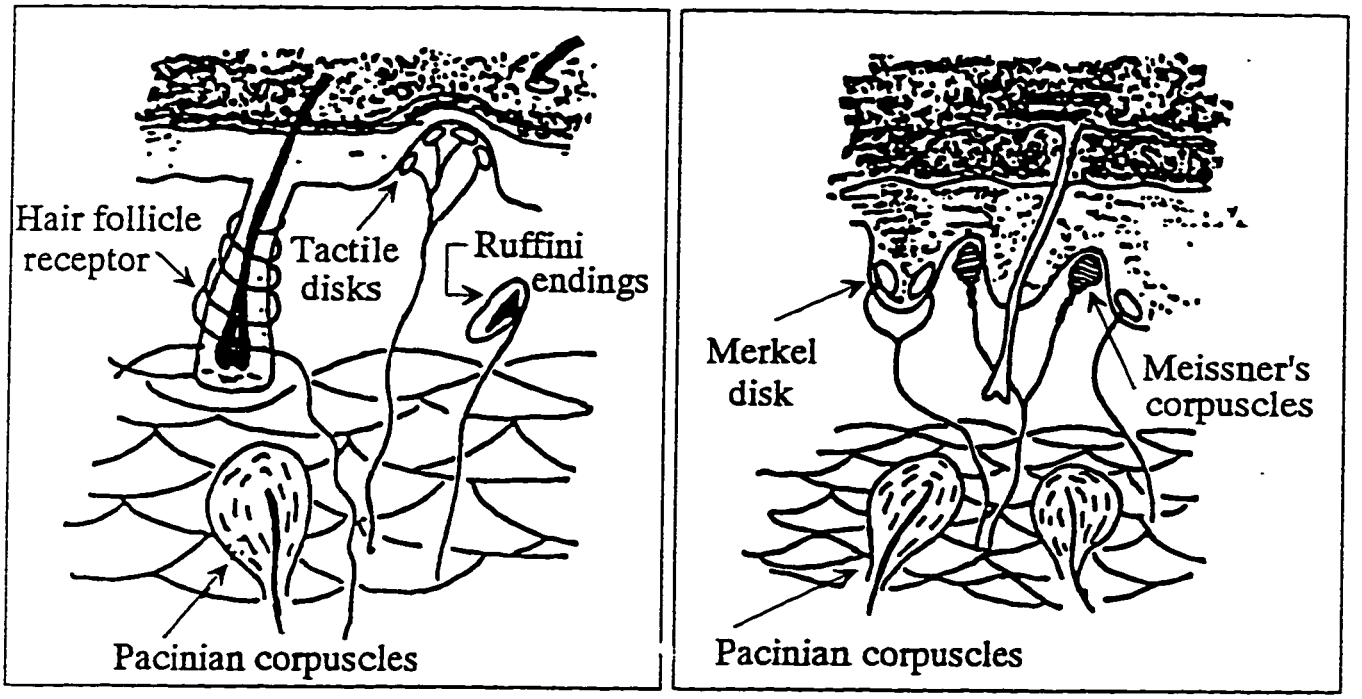


Figure A1.1: Structure of Hairy (left) and Glabrous (right) skin.

All these mecanoreceptors, except one, are innervated by myelinated, rapidly conducting afferent fibers of group II (diameter 5-12 μm, conduction velocity 30-70 m s^{-1}).

The receptors in hairy skin are :

- the hair follicle receptors are basket endings surrounding the follicles,
- the Merkel's disks are near the hairs, grouped in tactile discs,
- Ruffini endings are spindle shaped capsules,
- the Pacinian corpuscles,
- and the C-mechanoreceptors (free-nerve endings).

In the glabrous skin, encapsulated receptors are common :

- the Meissner's corpuscles (tiny corpuscles),
- the Merkel receptor complexes,
- the Ruffini endings,
- the Pacinian corpuscles,
- and the free-nerve endings (not encapsulated).

A1.2.1.1 The Merkel's cells

They are located at the base of the epidermis. The Merkel's cell is associated with an expanded termlnal of the afferent fiber. This subdivides freely when entering the receptor. There are 50-70 nerve terminal discs in a single spot-like receptor. At the base of each cell, there is a disc-shaped expansion of a branch of a myelinated sensory axon : the Merkell's disc.

A1.2.1.2 The free-nerve endings

The prevalent types are the A-fibers and the C-fibers. Type A has a relatively large diameter and is myelinated, fibers of the second type are more numerous. The C-mechanoreceptors are nerve fibers with free endings located in the outer layer of the skin, the epidermis. They are mostly found in hairy skin [A1.1]. The afferent fibers of the C-mechanoreceptors are non-myelinated fibers, with g small diameter and a conduction speed about 2 m s-1 [A1.5], [A1.6], [A1.3].

A1.2.1.3 The Meissner's corpuscles

They are encapsulated receptors with myelinated afferent fiber present in the dermis of glabrous ken. They are ovoid with the axis perpendicular to the surface of the skin (30 x 80 μm). They Bit 0.5 mm below the skin surface [A1.1].

A1.2.1.4 The Hair Follicle Receptors

They are innervated by myelinated efferent fibers which are arranged in a circumference with complex of endings around the hair root, below the sebaceous gland, sometime called the basket ending [A1.1]. Their nerve terminal are elongated rods running parallel to

the hair and the root sheath and encircling it.

A1.2.1.5 The Krause End Bulbs

They are found in the glabrous skin of non-primates and exist in two varieties. That's s simple lamellated capsule. The axon ends into the capsule of the receptor either as a single rod-like extension in the cylindrical form or as an intertwined spiral in the globular form [A1.1].

A1.2.1.6 The Ruffini endings

They are encapsulated receptors in the dermis of both hairy and glabrous skin. They lie more deeply than Meissner and Krause endings. They can also be found in protective sheath at articular joint. Each corpuscle is an elongated spindle of 0.1 mm on 0.5-2 mm.

A1.2.1.7 The Pacinian corpuscles

They are considered as equivocal cutaneous receptors because they are lying 2-3 mm below the surface of the skin (Figure A1.2 from([A1.1]). They can also be found in muscles, joints and mesentery. This is the largest structure to be found in the skin :1 x 2 mm. For this reason it has been extensively studied [A1.1],

[A1.5]. They are grey pearl-shaped structure. The onion like lamellar structure is formed of non nervous tissue. The core contains an elongated nerve terminal. The afferent fiber is myelinated and the last node of Ranvier is in the encapsulated receptor.

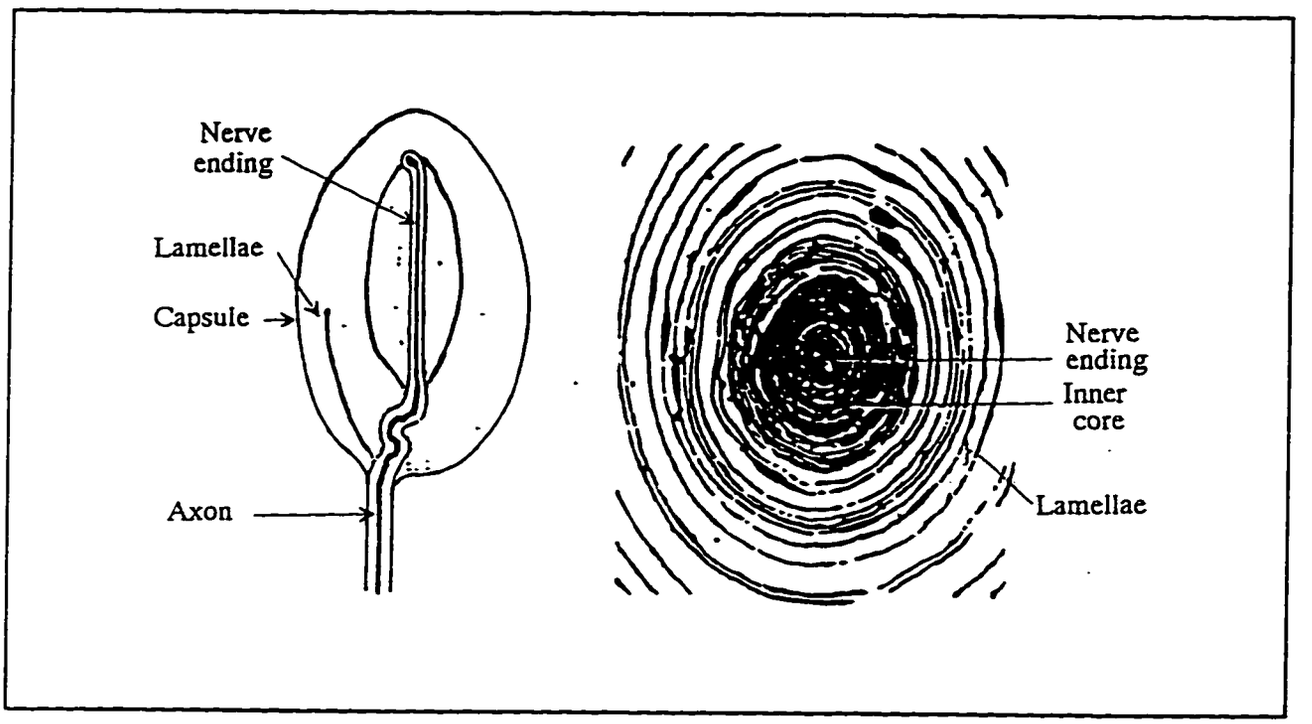


Figure A1.2 Pacinian Corpuscle

A1.2.2 Physiology of the Mechanoreceptor in the Skin

The current classification of the receptor systems takes into account their rate of discharge in response to a constant pressure stimulus. The two classes of receptors are the rapidly adapting receptors and the slowly adapting ones. They may also be described

depending on their frequency response, they may be described as displacement-, velocity-, acceleration-, or jerk-sensitive [A1.3]. The relationship between the rate of discharge and the intensity of the stimulus in terms of indentation can be written as follow (equation 1) [A1.1] :

$$R = a\,S^{b} \qquad (1)$$

Here R stands for the rate of discharge of impulses, a is a constant, S is depth of indentation as a measure of the stimulus, and b a real exponent.

Differentiation between nerve fibers depends on their conduction velocities and on the fact they are myelinated or not [A1.3].

Table A1.1 presents the innervation density of the skin expressed in units per cm^2 it is extracted from [A1.7].

Region of the hand	RA	SA I	PC	SA II	Relative density
Finger tip	140.5	70.2	21.4	9.2	4.2
Rest of the finger	37.1	29.7	9.5	13.8	1.6
Palm	24.5	8.0	9.3	15.7	1

Table A1.1: Innervation density of the skin in units per cm^2

Figure A1.3, extracted from [A1.2] p 297, represents the receptive fields for (a) : the Meissner's corpuscles, (b) : the Pacinian corpuscles, (c) : the Merkel's receptors and (c) : the Ruffini endings. The arrows refer to the direction in which the rates of response to stretch are higher, this characteristic will be developed in a further subsection.

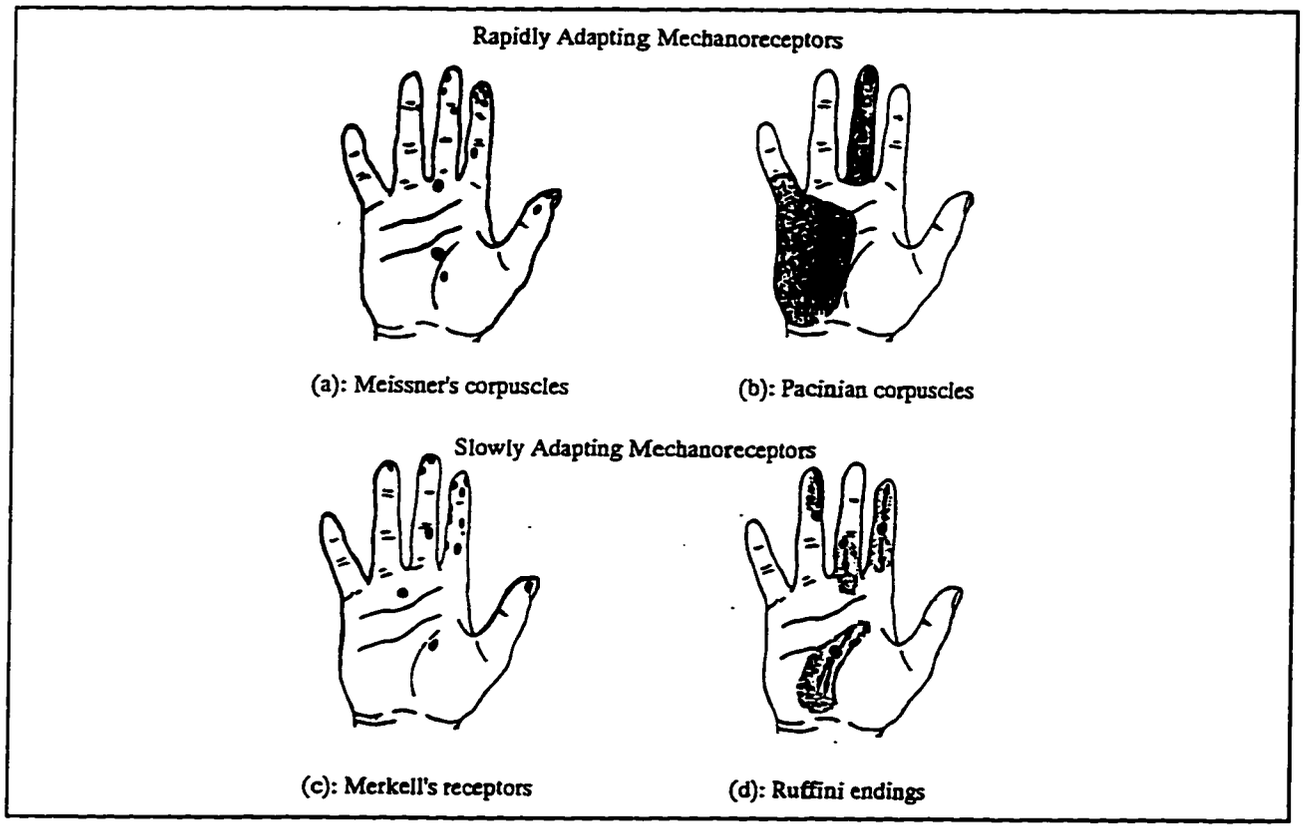


Figure A1.3 Receptive fields in the inner surface of the hand

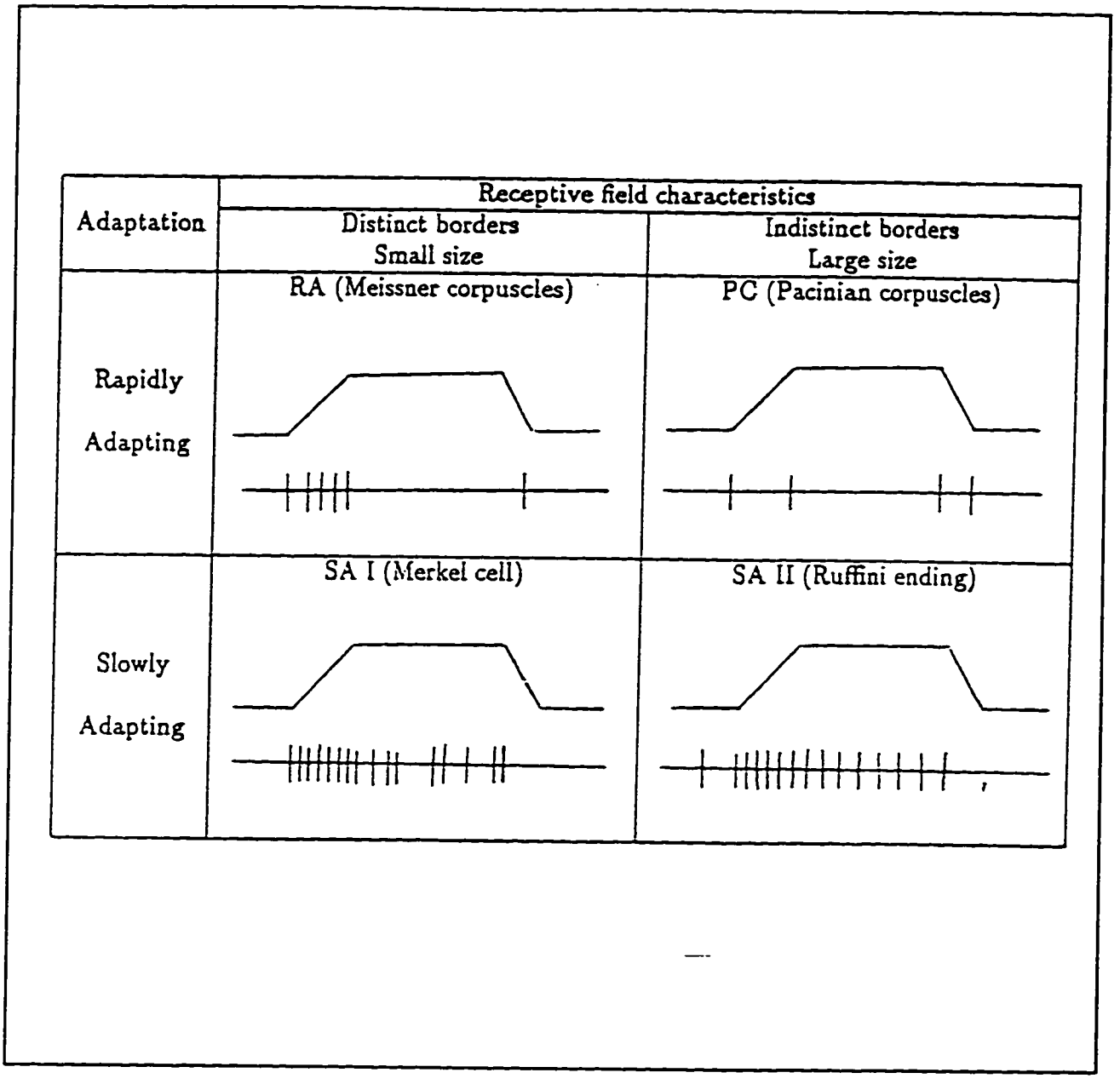


Table A1.2: Relationship Rate of Adaptation/Field Size

A1.2.3 Classification by function

As it is generally admitted, the mechanoreceptors are tuned to

be more sensitive to some kind of excitation. We will now attempt to clarify their role even though they all are involved in every sensation [A1.5].

Pressure receptors : (Intensity detectors)

In glabrous skin the Merkel's discs play this role, in hairy skin they are replaced by the Ruffini endings but they also may be found there, located in special tactile discs (about 30-50 Merkel's discs in one tactile disc). The peak response of the Merkel's discs is at about 10 Hz [A1.8].

Stretch receptors : The Ruffini endings

The stretch receptors are intensity detectors as well. The Ruffini endings are supposed to be most responsible for the skin stretch sensitivity. They respond at low frequencies.

Touch receptors : Velocity detectors

The hair follicles of hairy skin detect the hair movement and maybe the velocity of movement. In glabrous skin, Meissner's corpuscles detect velocity (with a peak response at about 30 Hz).

Vibration receptors : Acceleration detectors

In hairy and glabrous skin, this function is fulfilled by the very rapidly adapting mechanoreceptors, in other words, by the Pacinian corpuscles (with the best performance at 250-300 Hz).

Special case, the free nerve endings : Temperature and pain. As opposed to the other mechanoreceptors, they are not in corpuscular structures. Some of them are thermoreceptors, many are pain receptors (nociceptors), and a few are sensitive to low intensity touch stimuli. Observations show that two intensity levels can be discriminated so that they would be threshold detectors.

A1.2.4 Psychophysical Observations

A1.2.4.1 Thresholds for Touch

Several experiments ave been carried out in order to determine the threshold for sensation of a mechanical stimulus of the skin. The velocity of indentation is related to threshold touch sensations [A1.9]. If done slowly enough (about 0.05mm s-1), the skin may be indented as much as 1.5-2 mm without any sensation.

A1.2.4.2 Thresholds for vibration

The experiments showed that to different sizes of contactors correspond different thresholds in terms of amplitude of the skin indentation but the curve remains a U-shaped curve with a minimum value at about 250-300 Hz [A1.10]. At low frequencies in the range of 20 to 40 Hz, the threshold has been found independent of the contactor size. The absolute threshold varies inversely to the contactor area for higher frequencies. For very small contactors (0.005 cm^2 0.02 cm^2), the threshold becomes independent of the frequency of the stimulus.

The effect of the contact area on the response is referred as the spatial summation. For low frequencies, we already said that the threshold is not dependent of the size of the contact : there is no spatial summation there. Conversely, for higher frequencies, the spatial summation can be observed. When the area of the contact is up to 2.9 cm^2 with a double-size contact, the threshold is divided by two: there is a complete summation in this case.

A1.2.4.3 Perception of the Orientation of the Direction of Movement of Tactile Stimuli

In active or passive touch, our ability to detect slip is

related to the cutaneous mechanoreceptors innervating the contact area [A1.11], [A1.12]. The spatia-temporal period of a moving stimulus is well matched by the responses of the different fibers. It has been shown that human observers are able to detect 1-3 pm dot (550 µm in diameter) when it is stroked on the finger pad, although it can not be detected with a stationary contact. With or without slip, the stretch of the skin is well perceived by the observers (in [A1.12], the experiments are realized for the transversa direction).

In conclusion, some authors [A1.13] assert that the skin stretch is an important source of information for the somatosensory system. The SA II receptors (or Ruffini corpuscles) are most likely sensitive to skin stretch. However these last results are in conflict with the characteristics of the afferent fibers of those receptors. However, the enhancement of the performance is not necessarily related to stretch but to the forces generated between the superficial layers of the skin and underlying tissues [A1.13].

A1.2.4.4 Spatial factors

Point Localization:

In this experiment, the skin of the subject is touched while she or he is asked to identify the point. The results are dependent of the site of the body tested. Another factor of error is the time

elapsed between the identification of the point and its localization by the subject. Another version of this experiment is to present the subject with two stimuli successively.

The tau phenomenon is described as the decrease in distance that separate two tactile stimuli judged to be two as the time interval between their presentation increase.

The skin stretch provides the most salient cue for the discrimination of the direction of objects across the skin surface [A1.13] , [A1.14].

The table A1.3 from [A1.13] presents the thresholds for various experiments in order to discriminate the direction or the orientation of the stimulus on the skin with or without stretch.

Stimulus	Discrimination Task	N° of subjects	Thresholds (mm)
Line	orientation Lo/Tr	14	16.8
Rolled arc	orientation Lo/Tr	14	10.5
2-points simultsneously	orientation Lo/Tr	14	13.1
2-points sequentially	orientation Lo/Tr	14	8.7
5 g stroke	orientation Lo/Tr	10	4.3
stretch	direction M-L vs P-D	10	<1
0.7 g stroke	direction M-L	4	14.0
0.7 g air stream	direction M-L	4	11.3
stretch	direction M-L	4	<2

Table A1.3: Thresholds for the discrimination of the direction or orientation for various stimuli

with Lo : longitudinal M : medial D : distal

Tr : transversal L : lateral P : proximal

The thresholds to determine the skin stretch direction are dependant on the velocity of the movement:

- at 1 cm s^{-1}, the mean threshold is 0.6 mm
- at 1 mm s^{-1} the mean threshold is > 2 mm.

Experiments on monkeys showed that an interruption in the spinal cord column caused a deficit in the sensitivity to the direction. It suggested that the dorsal columns are adapted to code spatio-temporal stimulus sequences [A1.13].

The Two-Point Limen:

The difference limen (DL) or difference threshold or just noticeable difference is the smallest distance between two points allowing the subject to feel two distinct sensations of touch. This is also called the differential sensitivity [A1.3]. For smaller separations, the sensations merge into a single one. The difference with the experiments of point localization is that the point were referenced successively and they are now touched simultaneously.

The two-point limen refers to the spatial resolution of the skin. For the finger tips, the tongue and the lips this value is low: in the range of 1-3 mm.

Spatial Summation:

It refers to the ability of correlate the area of the skin excited and the response processed by the receptors. It has been already discussed in the subsection devoted to the threshold for vibration. This phenomenon occurs for high frequencies but is not observed for the low frequencies (see Figure A1.4).

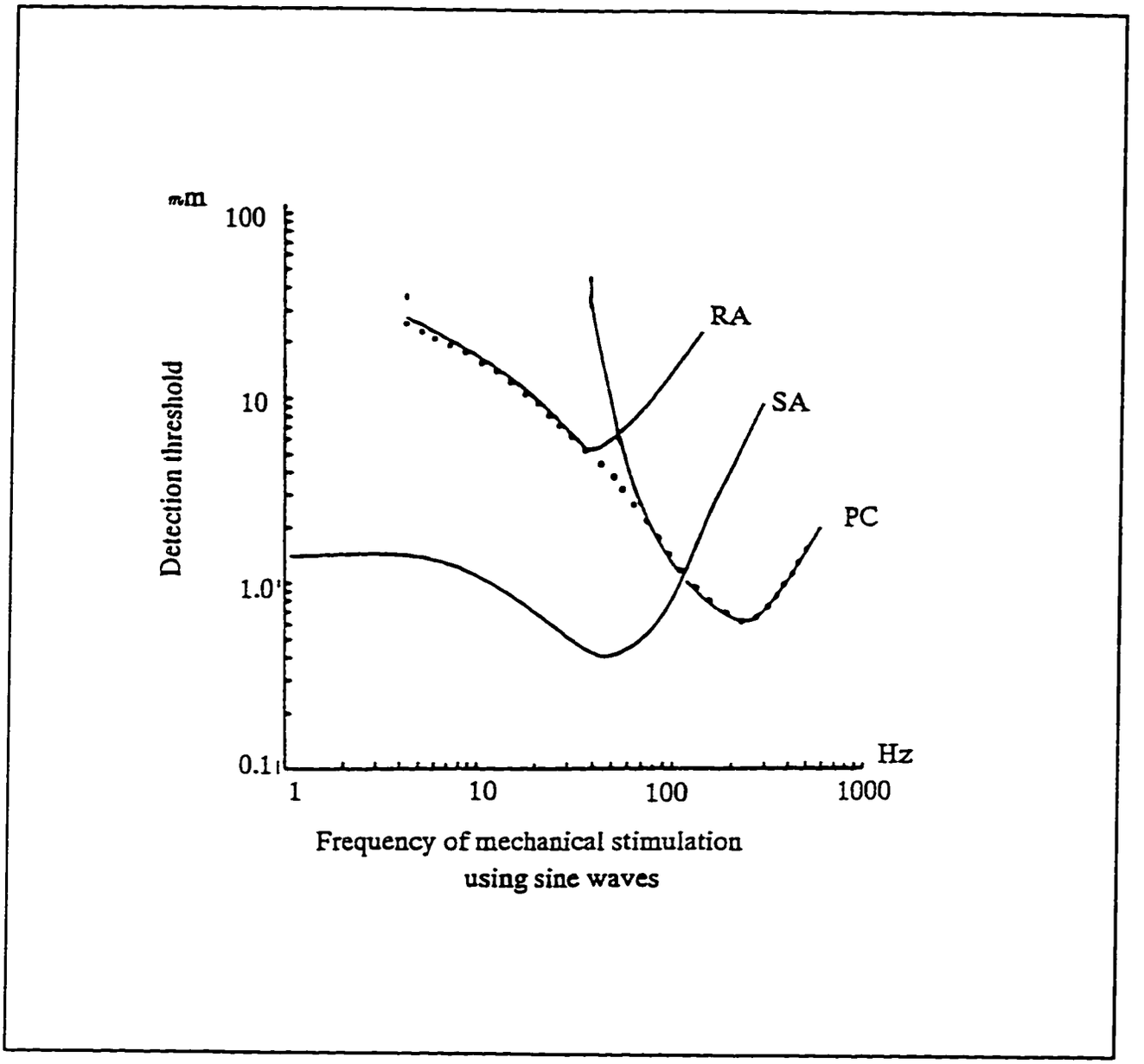


Figure A1.4 Threshold/Frequency relationship for RAs, PCs, SAs units

A1.2.4.5 Temporal factors

Temporal summation has been observed for low intensities at

about 100-250 Hz.

A1.2.4.6 Time and space interactions

The Tau Phenomenon:

Three equally loud stimuli are successively presented at different places with unequal delay times. The space between the two stimuli with the lowest time interval is judged to be smaller.

The Phi Phenomenon:

It is a tactile apparent movement. If two vibrators are 10 cm apart, one with a frequency of 40 Hz and the other with 500 Hz and a phase shifted sinusoid. Then there is a shift in localization from one source to the other. Similarly, with equal frequencies, there is a single "phantom" sensation of touch [A1.9], [A1.3]. A similar phenomenon can be observed for the auditory sense.

Saltation:

The cutaneous saltation is related (but different) to the Phi phenomenon. It gives discontinuous sensation as superficial as light touch.

Temporal Acuity:

It is the minimum time interval that must separate two tactile stimuli (at the same locus) for they being perceived as different. It is about 5.5 ms for stimuli of 1 ms. For ear, 10 ps stimuli must be separated by 10 μs.

Masking:

In the subsection on the spinal cord mechanisms, we already gave some idea of what is a masking effect. This inhibition phenomenon can have three different sources. They are referred as the surround inhibition, the complementary inhibition and the cross-modality inhibition and result from the presentation of a second stimulus respectively to an area adjacent to the first one, contralateral to it or of a different kind of stimulus.

Appendix 2

Current Tactile Sensor Technology

A2.0 Introduction:

Dextrous manipulation, the ability of a robot manipulator to perform operations on its environment which are comparable in complexity to those accomplished by human hands, can be subdivided in three main parts; grasp, micro-movement and tactile exploration. Although some of the dextrous hands proposed in recent years, such as the Salisbury three finger gripper and Utah/MIT [A2.1] four finger hand, have accomplished to some extent those goals, it is widely recognized that further improvements will be possible under condition of intensive utilization of sensory devices, especially tactile sensors.

Contact is the fundamental physical phenomenon on which manipulation relies. Contact is not only the means by which parts of the manipulator (most often its fingers) can exert forces on objects so as to control their motion, but also a source of sensorial information about manipulated objects. Therefore, robotics researchers have been devoting much attention in the recent past to replicating the capability of human to sense contact through tactile sensors.

Most attempts conducted so far used the human example not only as a performance target for their devices, but also as a model to follow in designing them. Thus, conventional contact sensing devices usually consist of many simple pressure transducers, arrayed on or in proximity of the surface of the robot finger, in much the same way that Pacini, Merkel or Meissner tactile corpuscles are located in our skin. Although several such skin-like tactile sensors have been realized showing good results, there are some limitations inherent in this approach. Besides technological difficulties in building a high number (ranging typically from several tens to hundreds) of transducing elements of a skin-like sensor is an obstacle to its use in real time tasks.

Future industrial robots will have to operate in unstructured environments for which versatile and intelligent grippers will be necessary. In humans, cutaneous touch sensations provide information for grasping and manipulating tools and other objects with dexterity. In an analogous fashion, tactile force images obtained from gripper fingers during the performance of a task, could provide at least a part of the necessary versatility and intelligence to grasp an object without slipping, to recognize the position and orientation of the grasped object, or to release an held object without dropping.

When a human subject attempts a grasping task on a small object, the compliant tissue in the fingers conforms to the object.

When the object begins to move relative to the fingers, forces acting on the fingers tend to create a resistance which gives the necessary damping or velocity feedback to stabilize the grasp. This suggests that simple contact fingertip models are not adequate to describe the process of grasping or releasing. Early work on manipulation focused interest on modelling and kinematic analysis of point contact fingers. The use of dynamics information for manipulation are affected by the dissipation of energy due to the visco-plastic nature of the fingertip.

The design of a tactile sensor is influenced by its intended use. The major applications for tactile sensors can be divided into three general categories:

(1) Simple pressure determination and slip sensing. These capabilities are necessary for the most common industrial applications of handling a workpiece without damage.

(2) Determination of object orientation and position. This is required for more complex and unstructured applications, such as picking an object from a bin, orienting it into a new position, and assembling it with other objects.

(3) Object identification or recognition. This feature is necessary for advanced applications in which a robot may be working in a totally unknown environment (such as undersea and space explorations) and may be required to classify or identify an object based solely on tactile

sensations.

Each of these applications involves a different design approach and different computational requirements. The first application is technically the simplest to implement, and industry has found several workable approaches to it. The latter two applications are the most challenging, and it is on these applications that most of the current laboratory research is focused.

A2.1 Tactile Sensors

Tactile sensing generally refers to skinlike properties where areas of force sensitive and displacement sensitive surfaces are capable of reporting graded signals and parallel patterns of touching. Tactile sensing may be viewed as two-step process: transduction and data processing. Transduction occurs when the features of an object being examined are converted into signals of some physical form (pressure, displacement, temperature, etc,...) into electrical signals. Data processing then interprets these signals to obtain useful information about the features of interest.

Since it is often stated that a robotic tactile sensor should have capabilities similar to that of human touch sensing, we should

briefly remind human tactile perception mechanisms.

As discussed in the previous chapter, cutaneous and kinesthetic responses are the two distinct components of human tactile perception. The sensitive nerve network of the fingertips carries touch, force, slip, and temperature information to the brain. This cutaneous response is what the robotic community considers as tactile sensing. Kinesthetic response conveys the limb and joint position information.

The two major forms of tactile sensors include tactile array sensors and force/torque sensors. The former consist of an array of force or deflection measuring sites, usually covered by some type of compliant, elastomeric surface. Typically, only the normal component of force or deflection is measured at each site, resulting in an array of data representing the force applied to, or the deflection of, each site. Many technologies have been applied in making the actual measurement at each point, with the predominant methods being electro-optical deflection measurement and the detection of changes in the resistance of a conductive material.

In electro-optical deflection measurement, changes in the deflection of the tactile surface are transduced into an electrical signal by measuring corresponding variations in light intensity. This enables accurate measurement of sensor surface deformation

with low hysteresis. One method employs a pin integral with the sensor surface, which partially shades a photoemitter/detector pair as the surface is deflected. A similar method uses a fibre-optic pair at each site, normal to the deflecting surface, to detect changes in retroreflected light intensity and thus measure surface deflection. Knowing the force-deflection characteristics of the surface allows applied forces to be calculated with some degree of accuracy. But with sensors having continuously deformable surfaces, the interrelation of adjacent sites can be complex. Force can be most directly inferred in sensors having isolated sites.

In the case of conductive elastomers, one design employs strips of conductive elastomer arranged in rows and columns at right angles. Each juncture forms a tactile site. Force applied to the juncture results in an increase in contact area between the strips and a decrease in electrical resistance. The resistance is measured to provide an indication of force.

Another type of resistance-measuring sensor relies on the fact that compression of the conductor-impregnated elastomer itself brings the conductive particles into closer contact, decreasing resistance. These sensors typically do not exhibit the performance of deflection measuring sensors, primarily due to the mechanical degradation of the elastomer caused by impregnating it with a conductive material. Durability and hysteresis are both affected adversely. In this respect, electro-optical methods are best in

that the transduction is separate from the elastomer, allowing mechanical properties of the elastomer to be independent of the transduction. Conductive elastomers are, however. less costly to manufacture and are well suited for applications having less stringent force/deflection measurement accuracy requirements.

Other technologies have been used with limited success in efforts to quantize spatial force distributions. These include the use of piezoelectric polymers and silicon devices.

Force/torque sensors measure minute transducer deflections from which forces and torques are calculated. Sensor systems for measuring multiple components of force and torque, that are explicitly made to be usable with robots and have the necessary integrated computational power, have only recently gained acceptance.

Load range and sensitivity are important to consider when evaluating a sensor system. The ratio of torque range to force range is also important, because in most practical applications, tooling geometry is such that sensor torque limits are reached before load limits.

A2.2.1 Optical Tactile Sensors:

An experimental tactile sensor discussed in [A2.2] consists of a matrix of stainless steel hypodermic needles, on each of which a reflective scale is partially engraved. Each needle is considered as a sensing element with a light emitting diode (LED) and a phototransistor positioned around mileometers away from it. An index grating is attached over the aperture of the phototransistor. As a pin is displaced a Moire fringe pattern is formed between the reflecting scale on the pin and the index grating in front of the phototransistor. The nodes formed are detected by the phototransistor. Both scales have a 30 micronmeter resolution, thus providing a displacement measurement accuracy of 30 micronmeters normal to the sensing plane.

The sensing plane contains a 2-D array of needles, providing a few millimetres spatial resolution along the sensing plane. This prototype sensor can be used to provide intrinsic information relating to objects of interest such as object profiles and geometrical features situated on the object surface. Information along the sensing plane can be obtained by a discrete number of parallel touched to enhance the measurement resolution. Feature heights are detectable down to thirty micrometers.

The data captured by the sensing elements are used to construct an object specific code from a unique matrix relating to the measurements taken using the Moire fringe principle. While being one of the very few tactile sensors able to measure geometric profiles, this experimental prototype is too fragile to be considered for space applications.

While the aforementioned method facilitates object recognition by reconstructing tactile information, it dose not by itself provide specific information relating to the orientation of the object with respect to the sensing plane. This means that, while an object may be discriminated from an assortment of parts stored within a knowledge base, its stable manipulation cannot be ensure unless its weight, position, and pose with respect to a gripping mechanism are determined. When such extrinsic information is ascertained from a sensory gripper comprising two such tactile sensing planes, the object can be grasped in a stable manner to eliminate all forces and moments that would otherwise exist. In the absence of eccentric forces and moments, a sufficient gripping force may be determined to act in an appropriate direction to eliminate the chance of mis-manipulation due to object slippage. In this configuration the sensing pins are required to provide force feedback information in addition to the aforementioned fine displacement data.

The distributed tactile sensing elements can acquire force feedback information at the interface between an object and the sensing planes of a sensory gripper. The advantage of the method is that the acquisition of direct contact forces provide grip force adaptability in addition to stable manipulation by determining optimum hold sites for the applied force to act through. In practice, the gripper movement is governed by a search strategy to extract the sensory data. Tactile data acquisition and interpretation for deformable objects generate a lot of researchers' interests in robotic field. For solid objects the gripper is closed until any one of the tactile elements comes into contact with the object to be sensed. This results in a change of the contacted sensor output state from zero to one. At this instant all output values from tactile elements are extracted and the first layer of tactile image is constructed in a matrix form. The gripper continues to close until the second layer is formed, and then the second layer is compared to the first layer. If all the corresponding elements of the two matrices are equal, the tactile data detection procedure ends. Otherwise the matrices of consecutive deeper layers are extracted and compared until finally any one of the tactile elements are fully displaced that ends the search. This final layer is stored as the objects profile matrix. The alternative method used is remote force sensing from the robot wrist, which requires a fair amount of additional postprocessing to

determine the gripping conditions from a remote location. In addition latter method has a certain amount of information loss and sources of inaccuracy and nonlinearity due to various coupling effects and nonlinear elastic deformation of the wrist and gripper structures.

"Lord" Tactile Sensor [A2.3]

The LTS-210 is a commercial optical tactile sensing system developed by Lord Corporation. The system incorporates two forms of tactile sensing and supporting electronics in a package appropriate for use in gripper applications. The transducer is composed of sensitive sites embedded in an elastomeric touch surface. The sites are organized as a 10 × 16 orthogonal array with 1.7 mm center-to-center spacing between each site. Each site monitors the deflection of the small portion of the elastomeric touch surface it occupies. Accordingly, the tactile array sensor conveys touch information by determining the deflection at each site in the array of sensitive sites embedded in the touch surface and communicating this information to a host computer or system controller.

The LTS-210 tactile sensor system interface electronics comprise a microprocessor-based data acquisition system which accepts simplified commands from the host and takes the necessary steps to complete the desired operation. These steps include data

acquisition, normalization, and conversion to engineering units and therefore significantly reduce the sensor's demands on the host computer or control system.

The sensor communicates with the host over a standard 3-wire RS-232 serial interface. In general, information is transferred in binary mode to reduce transmission time. Since the transducer can be damaged by loads greater than 18 Kgs at certain locations on its surface, an overload indicator is also provided. The seven conductor interface cable comprises three power lines, two communication lines, one overload line, and one ground line.

A2.2.2 Piezoresistive Tactile Sensors

To provide a flexible tactile array, both the transductive elements and their substrate must be nonrigid and durable. A piezoresistive polymer, generically called sensing film, developed by Interlink Electronics of Santa Barbara California, [A2.4], offers the properties of durability and flexibility while allowing deposition on a variety of substrates. The sensing film is an Interlink proprietary polymer, containing a suspension of organic and inorganic particles. The particles range from submicron to micron size, acting in various capacities to increase durability, promote adhesion, provide conduction, and control mechanical

properties of the polymer.

The sensing film is deposited on two separate pieces of Kapton that have been pattern with a deposition of conductive metal. The metallization forms conductive rows on one Kapton substrate, and conductive columns on the other. The sensing film is deposited in a 2 mil layer directly over and in the same pattern as the metallization. The two pieces are brought into contact with the rows and columns intersecting perpendicularly, and they are fastened together with adhesive. The rows and columns form a grid, each intersection acting as a force sensitive variable resistance.

When pressure is applied over a sensitive site, the resistance between the underlying conductive rows and columns decreases. The decrease in resistance is primarily due to changes in the surface geometry of the contacting sensing films.

Electrical connections are provided to each row and column for site addressing. Rather than solder or affix wires to the Kapton, the underlying conductors are patterned with extensions, running away from the sensor, down a narrow tail of Kapton. The tail acts as a flexible multiconductor cable. At its end the tail is flared and conductive pads are printed. Snap-on ribbon cable connectors can be attached at this point to allow access to the sensor. The

tail is covered with a dielectric to isolate the conductive traces. This allows the tails of each half of the sensor to be affixed face to face, further isolating the traces, both electrically and physically.

To provide a useful imaging capability, the flexible tactile array should have sufficient resolution to detect features on the objects it will encounter. A high resolution sensor implies a large number of rows and columns and a correspondingly large number of connecting wires to and from the device. The area to be covered by the flexible tactile array is estimated to be 0.5 × 1.0 inch on each of the famous Utah/MIT hand's fingertips, [A2.1]. The finest line that can be printed with the sensing film is 0.01 inch, which, allowing for spaces between the traces gives a potential maximum resolution of 0.02 inches. The maximum array size on a 0.5 × 1.0 sensor, therefore, would be 25 × 50. With one lead for each row and each column, 75 connecting traces would be required for a sensor of this size. This number was considered prohibitive due to the resulting width of the connecting tail and the complexity of the electronics required to address sites in the array. To reduce the number of traces on the tail and to simplify the electronics, a 16 × 16 array was built, providing a resolution of 0.032 × 0.064 inches over the rectangular array.

A2.2.3 Strain Gauge Tactile Sensors

When a force (tension or compression) is applied to a strain gauge or to an object attached with it, the physical dimensions of the gauge will be changed. This change will cause its electrical resistance to vary, and by using a simple bridge circuit, the representation of the force applied can be detected. Strain gauges are mature and proven technology, which attracts robotic researchers' attention. Since a large varieties of strain gauges with different shapes, sizes and low hysteresis are commercially available, they are mounted on mechanical structures to detect force and moment.

Some miniature strain gauges are developed from solid-state silicon, these allow for the relatively dense arrays of sensing sites which are considered to be necessary for effective tactile sensing. Each individual sensing site consists of a small box-shaped silicon element (mesa) that protrudes out of a silicon diaphragm. The mesa is capped with a protective square of hard plastic, and the whole assembly is bonded to a glass substrate housing the electrical connections. A rugged elastomer material covers the entire assembly for protection as shown in figure A2.1.

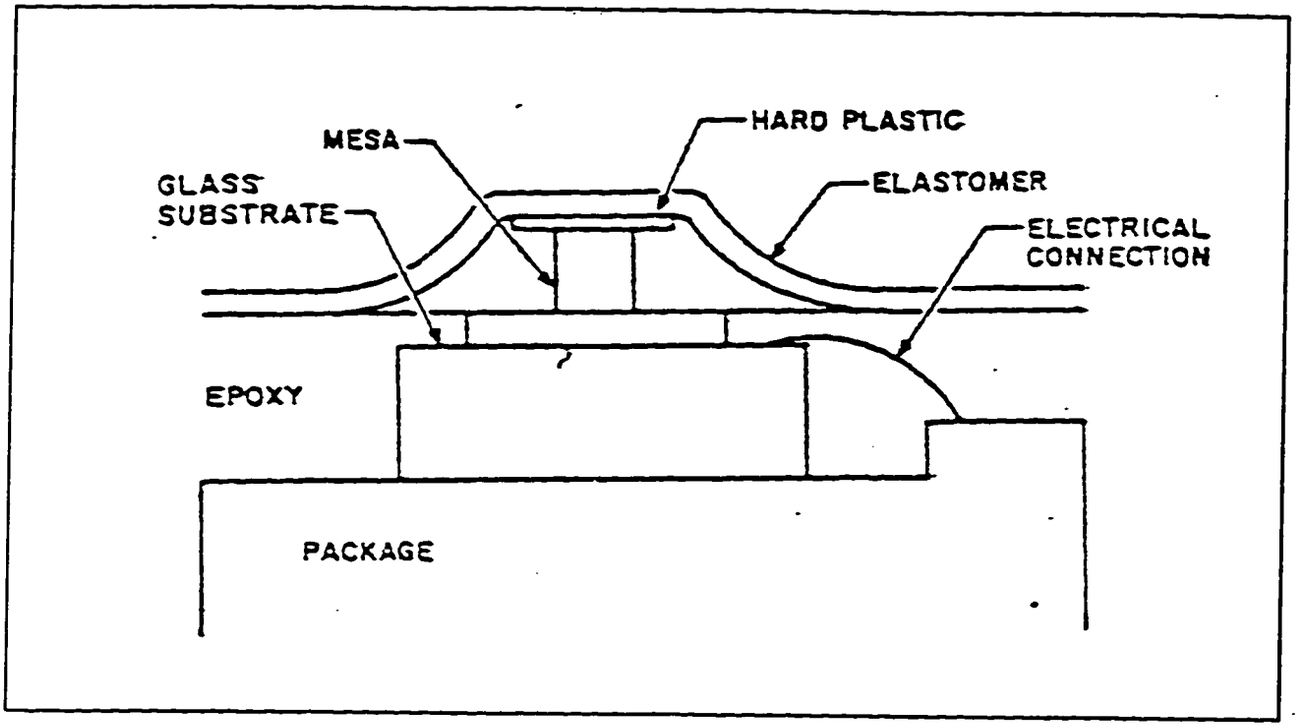


Figure A2.1 The major components of a single element tactile sensor that measures force normal to its surface

The mesa, diaphragm, and interface circuitry are all machined from a single piece of silicon wafer. In use, a 5 volt power supply provides a reference signal for the sensor, as well as power for the on-board logic circuitry. The sensor's output is an analog voltage that changes proportionally with the force applied to the sensitive area.

A 3×3 strain gauge array with sensing elements spaced at about 2 mm center to center has been developed. This sensor provide good

linearity in force sensing and are designed to measure up to one newton force. This miniature solid-state transducer offers great promise for future high-resolution sensors, but the current sensor is fragile and stiff. More development has to be done before widely applicable to industrial automation.

A2.2.4 Capacitive Tactile Sensors

Capacitive tactile sensor arrays typically consist of two layers of thin parallel conductive strips. The strips in one layer are orthogonal to those in the second layer. The two layers are separated by a dielectric medium to form an array of capacitors. Pressure applied to the surface causes the spacing between the conductive strips to change, thus varying the capacitance at the intersections. The change in capacitance can then be used to determine the applied pressure. The principle of operation of a simple capacitive force sensor is illustrated below in figure A2.2. There are minor variations to this basic principle of operation, for differing requirements.

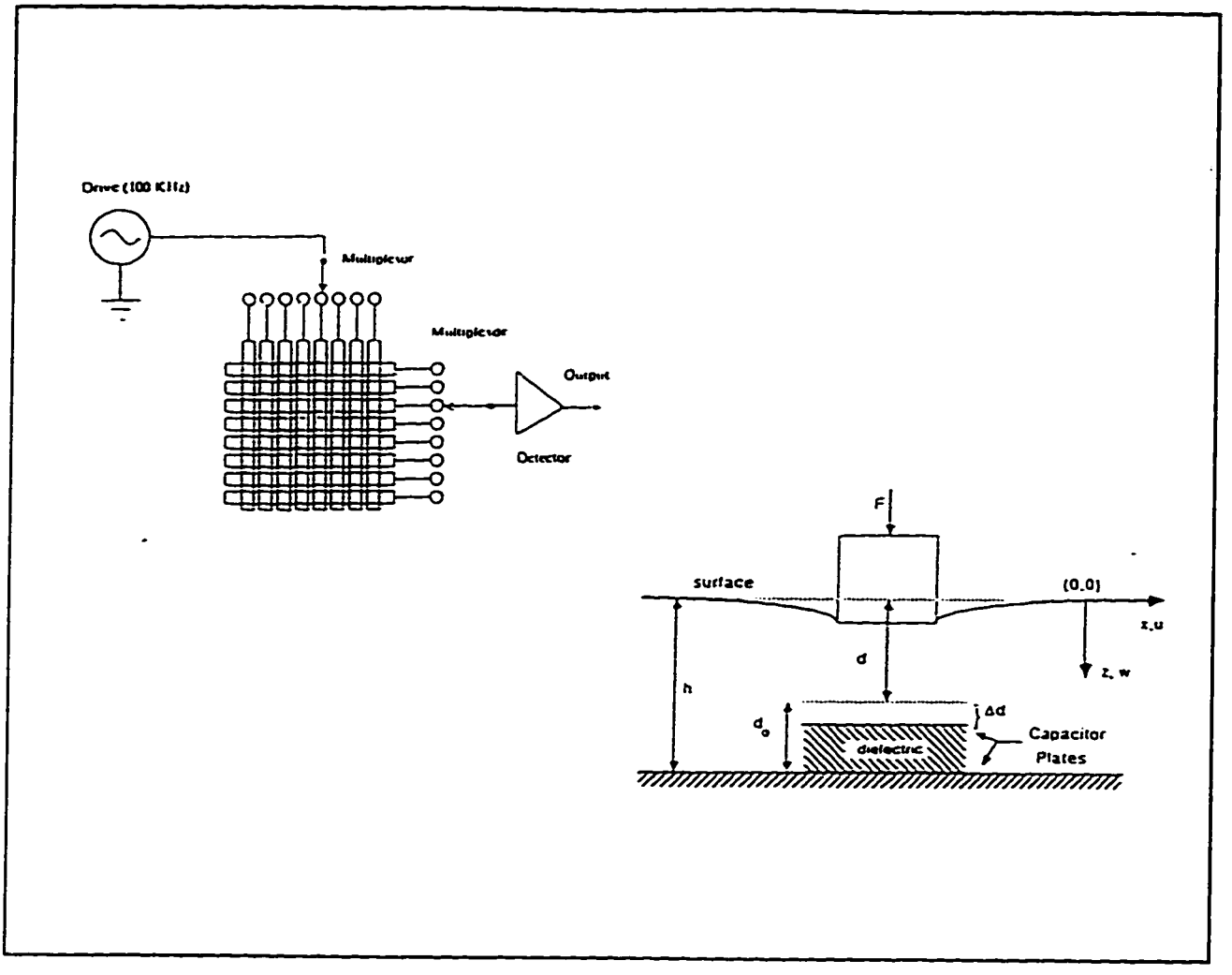


Figure A2.2 Capacitive Force Sensor

Capacitive tactile sensors, in general, can offer high sensitivity and resolution, good response times and are relatively immune to electro-magnetic interference. Also, these types of sensors are particularly simple and inexpensive to construct. However, the compression/decompression cycle of the dielectric will normally cause a significant level of non-linearity and hysteresis

in the force measurements.

Fearing constructed a finger shaped capacitive tactile sensor array using a traditional capacitive force sensor design. This sensor was installed on the Stanford/JPL robotic hand. The capacitors are formed by intersecting rows and columns of conductive strips separated by a dielectric medium. The tactile array consists of 8 × 12 sensing elements which are placed onto the cylindrical robotic finger and covered with a molded rubber skin. The sensor exhibits a poor time respponse which is attributed to the use of a highly compliant material for the skin. However, the use of a compliant skin did increase the sensitivity of the sensor by a factor of four.

A different approach to capacitive tactile sensing is presennted by Jayawant et al [A2.5]. Rather than placing the plates of the capacitor parallel to the sensing pad, the dielectric is cylindrical and is allowed to slide between concentric cylinders which form the plates of the capacitor. As pressure is applied to a sense pin, the dielectric medium is forced into the region between the plates, causing the capacitance to increase. An obvious disadvantage to this technique is that the orienntation of the dielectric and conductors makes it difficult to construct a thin sensor.

Micro-fabricated silicon diaphragm capacitive sensors are widely used as pressure sensors.[A2.6] Lu et al [A2.7] present a micro-fabrication technology for producing tactile sensor arrays of this type. The advantages of this type of sensor include the ability to fabricate very small sensors for high spatial resolution tactile sensor arrays and the ability to include pre-processing circuitry as part of the sensor. On the other hand, micro-fabricated sensors tend to be fragile and can be more easily damaged by unexpectedly large forces.

A2.2.5 Magnetoresistive

Magnetic transducer have been used by a number of researchers to design tactile sensors. Sensors based on the detection of magnetic fields will naturally be susceptible to electromagnetic interference (EMI) and must be shielded in environments where EMI is significant.

Magnetoresistive sensors typically operate on the principle that magnetic dipoles embedded in or mounted on a compliant material will change orientation when stress is applied to the material. This change in orientation can be detected as a change in the magnetic field strength at a fixed location relative to the dipole. These sensors can be designed to be most sensitive to either normal or tangential forces but it is difficult to

completely separate the two effects.

Nelson et al [A2.8] designed a magnetoresistive tactile sensor that is sensitive to both normal and tangential forces. At each sensor site, a 0.5 mm diameter magnetic rod is allowed to pivot within a hole in a sheet of stainless steel. Four magnetoresistive detectors, located below the magnetic rod, sense variations in the magnetic field. The design of a sensor site is illustrated in Figure 3.3. The output of the detectors can be used to produce differential outputs proportional to the x and y displacements of the rod and a common mode output proportional to the normal displacement. This allows for the measurement of shear and normal forces. The robustness of this device is questionable. The maximum load applied to the device was only 25 grams with no indication if the device could withstand larger loads.

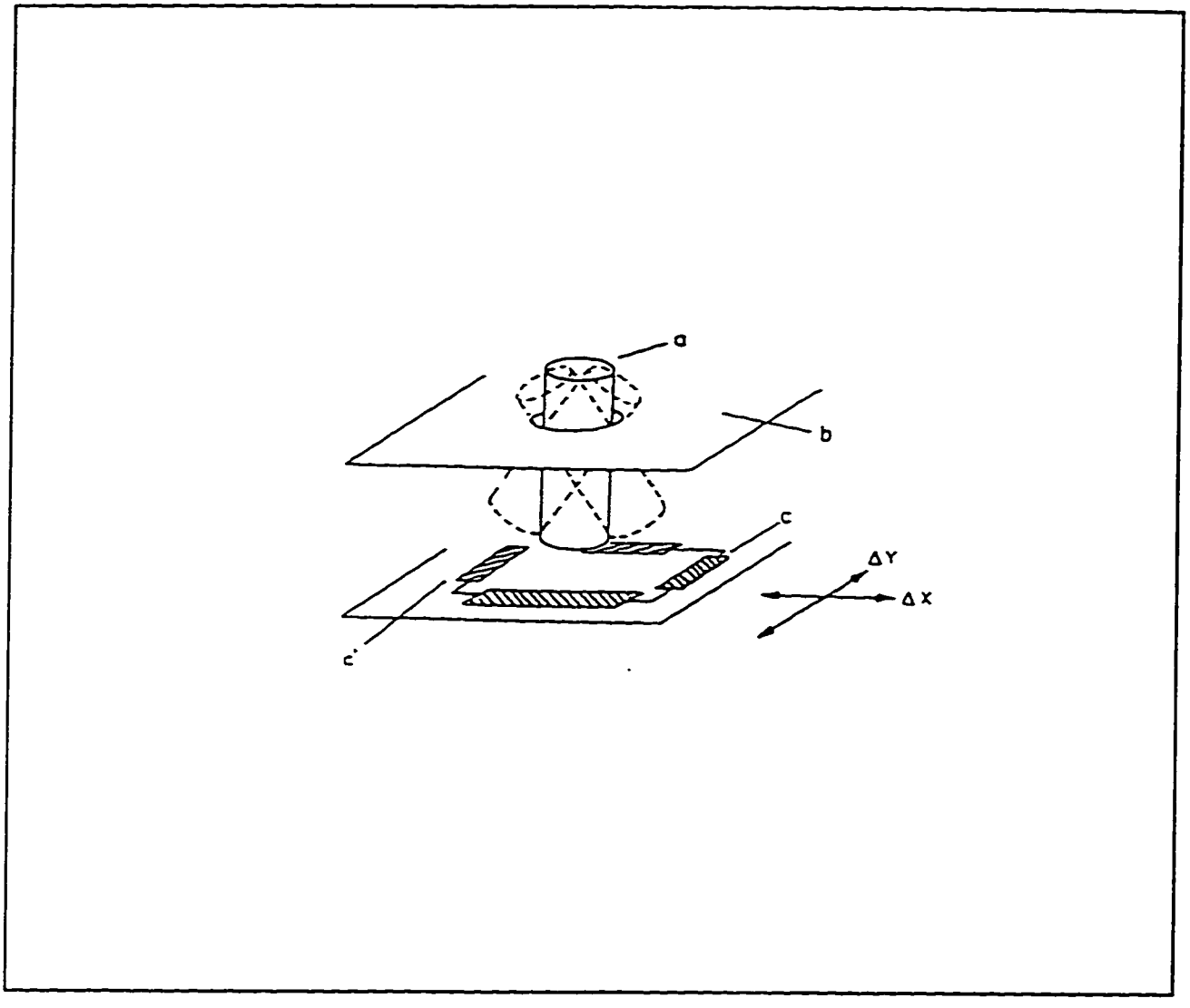


Figure A2.3 Magnetoresistive Force Sensor

Hackwood et al [A2.9] propose a design for a sensor array with a spatial resolution of less than 2 mm to measure shear force. At each site in the sensor array, a magnetic dipole is embedded in an elastic medium on a substrate containing five magnetoresistive detectors which are arrannged to optimize the measurement of translation and rotation. The sensors are relatively insensitive to

normal forces. When a force deforms the elastic medium, the magnetic dipole is displaced with respect to the detectors, therfore varying the detected magnetic field. The only data presented by the authors is a characterization of the performance of a single ideal sensor element. The proposed sensor array was not constructed.

Appendix 3

Extraction of 3D Linear Object Features Using Hough Transform

This is important in our case for the recognition of the 3-D polyhedral-shaped symbols embossed on object surfaces. To recognize these symbols by a model-based process, we need to choose a representation of the modeling primitives and abstractions. Hough Transform [A3.1] provides us the tool to fit a set of line segments extracted from sensory measurement against a polyhedral model.

Hough Transform is the technique based on the interrelation between points in 2-D space represented as (x, y) co-ordinates and then transform these points into a 'straight line space' in terms of slope (m) and intercepts (c). For points on a straight line which can be represented by the general equation

$$y = mx + c$$

If we reorganize the equation listed above and express it in terms of m and c, then we have

$$m = y/x - c/x$$

The point in the xy coordinates can be mapped into a line in mc coordinates. A set of points lying on a straght line in xy coordinates that satifies y = mx + c will generate li nes which will intersect at a common point in the mc coordinates. By observing the number of lines intersecting at a given point in mc coordinates, we can determine that the more number of lines intersecting, the higher probability of occurance of the existence of a line segment in the xy coordinates. If we keep track of the intersection points in the transformed coordinates, we can calculate the probability of the line segment in the object. The computation can be easily obtained by setting an array of elements corresponding to points spatially distributed over the coordinates defined by the m, c axes, and incrementing an array element for every crossing of a given location by a line in mc coordinates. By appying thresholding procedure, we can determine the existence of a straight line in the orginal object from the measurement sensed data.

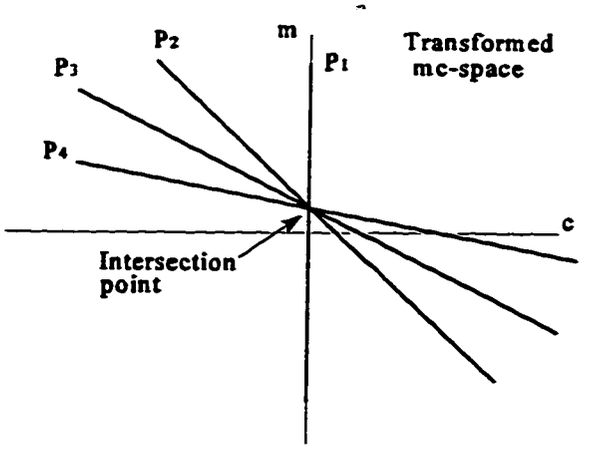


Figure A3.1 Hough Transformation mc-space

Using the simple Hough Transform to detect object edges, we are limited to sense the object in one direction. To extend Hough Transform, it is necessary to add 5 more parameters to the transformation to the model onto the image. If we assume the angle of attack to the object is scaled and known, 3 rotations and 2 translations are required to determine the orientation and position of the measured model in the image. Left-handed rule coordinate system is used to define the measured point with rotations about the x and y axes, and the rotation in the image plane is about the z-axis.

To transform a measured point coordinate from (x, y, z) on a unit sphere to (0, 0, -1), we first rotate (x, y, z) about y-axis

to bring this point onto Y-Z plane. The rotation matrix is

$$R_{\beta y} = \begin{matrix} \cos(\beta_y) & 0 & -\sin(\beta_y) \\ 0 & 1 & 0 \\ \sin(\beta_y) & 0 & \cos(\beta_y) \end{matrix}$$

Then rotate $(x_\beta, y_\beta, z_\beta)$ about X-axis through angle of β_x to bring the point to (0, 0, -1), the rotational matrix is

$$R_{\beta x} = \begin{matrix} \cos(\beta_x) & 0 & -\sin(\beta_x) \\ 0 & \cos(\beta_x) & \sin(\beta_x) \\ 0 & -\sin(\beta_x) & \cos(\beta_x) \end{matrix}$$

The final transformation will be

$$R_{\beta xy} = R_{\beta y} \times R_{\beta x}$$

$$R_{\beta xy} = \begin{matrix} \cos(\beta_y) & \sin(\beta_y)\sin(\beta_x) & -\sin(\beta_y)\cos(\beta_y) \\ 0 & \cos(\beta_x) & -\sin(\beta_x) \\ \sin(\beta_y) & -\cos(\beta_y)\sin(\beta_x) & \cos(\beta_y)\cos(\beta_x) \end{matrix}$$

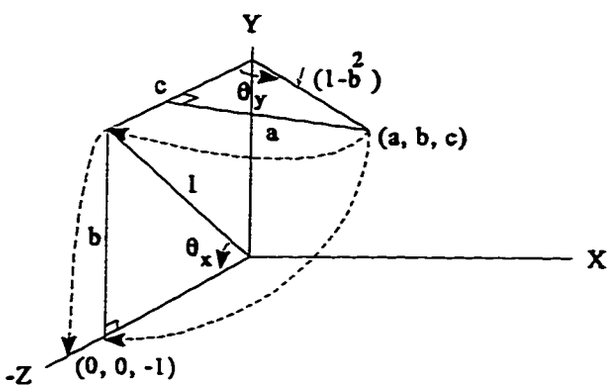


Figure A3.2 Rotation of L_{model} onto L_{image} plane

(ii) Rotation in the image plane

If L_{model} and L_{image} are in the same image plane, we can rotate L_{model} by β degrees to bring it in parallel with L_{image}. To calculate β, we have to find the slopes of L_{model} and L_{image}. In unit polar coordinate, L_{model} is at $(\cos\theta, \sin\theta)$ and L_{image} is at $(\cos\gamma, \sin\gamma)$. The rotation

will be ($\cos\beta$, $\sin\beta$), and $\beta = \theta - \gamma$.

$$\cos\beta = \cos(\theta - \gamma) = \cos\theta\cos\gamma + \sin\theta\sin\gamma$$

$$\sin\beta = \sin(\theta - \gamma) = \sin\theta\cos\gamma - \cos\theta\sin\gamma$$

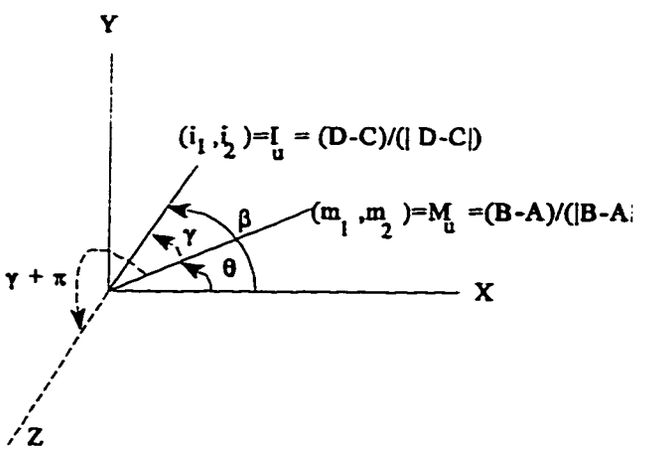


Figure A3.3 Rotation in the image plane

(iii) Translation in the image plane

To move L_{model} to cover L_{image}, we find Lmodel at (X_1, Y_1) and L_{image} at (X_2, Y_2). The displacement from L_{model} to L_{image} is $(X_1 - X_2, Y_1 - Y_2)$.

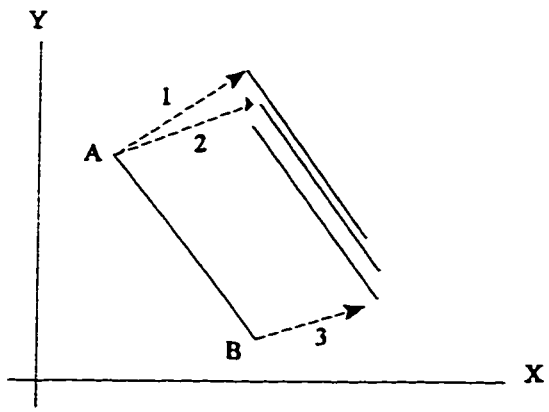


Figure A3.4 Translation in the image plane

Appendix 4

Computer Source Code in language of C++ for Tactile Recognition Data Base and Location of the milestone of Pseudo-random multi-valued array

```
/* ************************************************************************** */
/*    Program used to get data from a tactile sensor 16 by 16, display it */
/*   like a matrix 16 by 16 and allowd the user to rotate it, to zoom in   */
/*   and zoom out or to take another data.An initialization is required    */
/*   from time to time to insure that all the bias is cleaned before the   */
/*   reading.                                                              */
/*                    May.27.1992.                                         */
/*                          Written by Stephen Yeung        */
/* ************************************************************************** */

/* ************************************************************************** */
/* ----------------------HEADER FILES---------------------------------------- */
/* ************************************************************************** */
#include "graf_inc.h"
#include "graf_def.h"
#include "graf_typ.h"
#include "graf_fun.h"
#include "graf_var.h"

#include <fstream.h>
#include <strstrea.h>
#include <iomanip.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>

void SetAddress (int , int  );
void open_dialog(void );
void save_dialog(void);
void save_image(int);
void write_file_data(char *);
void read_file_data(char *);


#ifdef __cplusplus

 int max (int value1, int value2);

 int max(int value1, int value2)
 {
   return ( (value1 > value2) ? value1 : value2);
 }

#endif
```

```
/* ------------------------------------------------------------ */
/* ************************************************************ */
/* void main(void);                                             */
/*                                                              */
/*    Main function for graphic display of tactile data. This program      */
/*  starts by doing a sensor initialization and taking the bias.The grahics */
/*  mode is initialized and a right display allowed to choose from a menu   */
/*  few options like rotations, zoom in and zoom out, take new tactile data */
/*  or do a new initialization to the sensor.                   */
/* ************************************************************ */

void main(void)
   {
        graph_init();
        interface_init();
        graph_controller();
        graph_end();
        exit(0);
   }

/* ------------------------------------------------------------ */

/* ------------------------------------------------------------ */
/* ************************************************************ */
/*  void lit_angle_sonde()                                      */
/*    It is used to initialize the angles for the grahic display.         */
/* ************************************************************ */
void lit_angle_sonde()
{
       beta = PI_1_6;
       alpha = PI_1_6;
}
/* ------------------------------------------------------------ */

/* ------------------------------------------------------------ */
/* ************************************************************ */
/*  void lit_matrice_sonde()                                    */
/*    It is used to initialize the matrix first display              */
/* ************************************************************ */
void lit_matrice_sonde()
{
       int x, y, m;
       int mat2[16][16];

       float xm;
```

```
        for(x=0;x<16;x++)
           for(y=0;y<16;y++)
                 mat2[x][y]=mat1[y][x];

        for(x=0;x<16;x++)
          for(y=0;y<16;y++)
                 mat_m[16*x+y]=-mat2[x][y]/15;

}
/* ************************************************************************** */

/* ************************************************************************** */
/*  void clear_arrays();                                                      */
/*       Function used to clear all the arrays used in this program.   */
/*    It will prepare everithing for a new reading.                  */
/* ************************************************************************** */

void clear_arrays()
   {
   int i,j;

 for(i=0;i<Xdir;i++)
       for(j=0;j<Ydir;j++)
         {
         mat1[i][j]=0;
         input2[i][j]=0;
         }

 for(i=0;i<Xdir+2;i++)
       for(j=0;j<Ydir+2;j++)
         big[i][j]=0;

  for(i=0;i<Xdir*Ydir;i++)
             temp1[i]=0;

  for(i=0;i<num_max;i++)
       for(j=0;j<Xdir*Ydir;j++)
             input1[i][j]=0;
  }
/* ------------------------------------------------------------------ */

/* ------------------------------------------------------------------ */
/* ************************************************************************** */
/* void graph_init()                                                          */
/* ************************************************************************** */
```

```
void graph_init()
{
        int driver = DETECT, mode, erreur, xasp, yasp;
        unsigned user_pattern = 0;

        void graph_end();

        initgraph(&driver, &mode, "");

        if((erreur = graphresult()) != grOk)
        {
                printf("Initialization error: %s\n", grapherrormsg(erreur));
                exit(1);
        }

        cleardevice();

        setfillstyle(SOLID_FILL, BLACK);
        setlinestyle(SOLID_LINE, user_pattern, NORM_WIDTH);
        settextjustify(LEFT_TEXT, BOTTOM_TEXT);
        setusercharsize(1, 1, 1, 1);
        settextstyle(SMALL_FONT, HORIZ_DIR, USER_CHAR_SIZE);

        getaspectratio(&xasp, &yasp);
        asp = (float)yasp / (float)xasp;
        setaspectratio(yasp, yasp);

        if((mat_m=(float *)malloc(fn_dim*fn_dim*sizeof(float))) == NULL ||
          (mat_e=(position *)malloc(fn_dim*fn_dim*sizeof(position))) == NULL)
        {
                graph_end();
                exit(1);
        }
}
/* ------------------------------------------------------------ */

/* ------------------------------------------------------------ */
/* ************************************************************************ */
/* void interface_init()                                        */
/* ************************************************************************ */
void interface_init()
{
        initialization();
        ecran.dim.x  = getmaxx() + 1;
        ecran.dim.y  = getmaxy() + 1;
```

```
 ecran.debut.x = 0;
 ecran.debut.y = 0;
 ecran.fin.x   = ecran.dim.x - 1;
 ecran.fin.y   = ecran.dim.y - 1;

 graph.dim.x   = (int)(ecran.dim.x * 0.75);
 graph.dim.y   = ecran.dim.y;
 graph.debut.x = 0;
 graph.debut.y = 0;
 graph.fin.x   = graph.dim.x - 1;
 graph.fin.y   = graph.dim.y - 1;

 param.dim.x   = ecran.dim.x - graph.dim.x;
 param.dim.y   = (int)(ecran.dim.y * 0.16);
 param.debut.x = graph.fin.x + 1;
 param.debut.y = ecran.debut.y;
 param.fin.x   = ecran.fin.x;
 param.fin.y   = param.dim.y - 1;

 comma.dim.x   = ecran.dim.x - graph.dim.x;
 comma.dim.y   = ecran.dim.y - param.dim.y;
 comma.debut.x = graph.fin.x + 1;
 comma.debut.y = param.fin.y + 1;
 comma.fin.x   = ecran.fin.x;
 comma.fin.y   = ecran.fin.y;

 fn_esp = rap_fn_ecran * (float)ecran.dim.y / (float)fn_dim;


 rectangle(graph.debut.x, graph.debut.y, graph.fin.x, graph.fin.y);
 rectangle(param.debut.x, param.debut.y, param.fin.x, param.fin.y);
 rectangle(comma.debut.x, comma.debut.y, comma.fin.x, comma.fin.y);

 settextjustify(LEFT_TEXT, BOTTOM_TEXT);

 setviewport(comma.debut.x+2, comma.debut.y+2, comma.fin.x-2, comma.fin.y-2, 1);
 clearviewport();
 setusercharsize(1, 1, 1, 1);
 setusercharsize((int)(COMMA_TITRE_L * (float)comma.dim.x),
                 textwidth(COMMA_TITRE),
                 (int)(COMMA_TITRE_H * (float)comma.dim.y),
                 textheight(COMMA_TITRE));
 outtextxy((int)(COMMA_TITRE_X * (float)comma.dim.x),
                  (int)(COMMA_TITRE_Y * (float)comma.dim.y), COMMA_TITRE);
 outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)-XX,
```

```
                (int)(COMMA_SENSOR_Y * (float)comma.dim.y),SENSOR);

switch (sensor_nr)
 {
  case 0:
                {
                outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)+XX,
                                (int)(COMMA_SENSOR_Y * (float)comma.dim.y),"1");
                break;
                }

  case 1:
                {
                outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)+XX,
                                (int)(COMMA_SENSOR_Y * (float)comma.dim.y),"2");
                break;
                }

  case 2:
                {
                outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)+XX,
                                (int)(COMMA_SENSOR_Y * (float)comma.dim.y),"3");
                break;
                }

  case 3:
                {
                outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)+XX,
                                (int)(COMMA_SENSOR_Y * (float)comma.dim.y),"4");
                break;
                }

 case 4:
                {
                outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)+XX,
                                (int)(COMMA_SENSOR_Y * (float)comma.dim.y),"5");
                break;
                }

 case 5:
                {
                outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)+XX,
                                (int)(COMMA_SENSOR_Y * (float)comma.dim.y),"6");
                break;
                }
```

```
        case 6:
                        {
                        outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)+XX,
                                        (int)(COMMA_SENSOR_Y * (float)comma.dim.y),"7");
                        break;
                        }

        case 7:
                        {
                        outtextxy((int)(COMMA_SENSOR_X * (float)comma.dim.x)+XX,
                                        (int)(COMMA_SENSOR_Y * (float)comma.dim.y),"8");
                        break;
                        }
       }

       settextstyle(DEFAULT_FONT, HORIZ_DIR, 1);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                        (int)(MEDIAN_FILTERING_Y *
(float)comma.dim.y),COMMA_MEDIAN_FILTERING);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                        (int)(SOBEL_Y * (float)comma.dim.y),COMMA_SOBEL);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                        (int)(LAPLACE_Y * (float)comma.dim.y),COMMA_LAPLACE);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                        (int)(LOW_FILTERING_Y *
(float)comma.dim.y),COMMA_LOW_FILTERING);


       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM1_Y * (float)comma.dim.y), COMMA_ITEM11);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM2_Y * (float)comma.dim.y), COMMA_ITEM21);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM3_Y * (float)comma.dim.y), COMMA_ITEM31);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                        (int)(COMMA_ITEM4_Y * (float)comma.dim.y), COMMA_ITEM41);

       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM5_Y * (float)comma.dim.y), COMMA_ITEM51);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM6_Y * (float)comma.dim.y), COMMA_ITEM61);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM7_Y * (float)comma.dim.y), COMMA_ITEM71);
       outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM8_Y * (float)comma.dim.y), COMMA_ITEM81);
```

```
        outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM9_Y * (float)comma.dim.y), COMMA_ITEM91);
        outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM_OPEN_Y * (float)comma.dim.y),
COMMA_ITEM_OPEN_ASCII);
        outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM_SAVE_Y * (float)comma.dim.y),
COMMA_ITEM_SAVE_ASCII);


        outtextxy((int)(COMMA_ITEM1_X * (float)comma.dim.x)-XX,
                (int)(COMMA_ITEM10_Y * (float)comma.dim.y), COMMA_ITEM101);

        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x),
                (int)(COMMA_ITEM1_Y * (float)comma.dim.y), COMMA_ITEM12);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x),
                (int)(COMMA_ITEM2_Y * (float)comma.dim.y), COMMA_ITEM22);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x),
                (int)(COMMA_ITEM3_Y * (float)comma.dim.y), COMMA_ITEM32);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x),
                (int)(COMMA_ITEM4_Y * (float)comma.dim.y), COMMA_ITEM42);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(COMMA_ITEM5_Y * (float)comma.dim.y), COMMA_ITEM52);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(COMMA_ITEM6_Y * (float)comma.dim.y), COMMA_ITEM62);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(COMMA_ITEM7_Y * (float)comma.dim.y), COMMA_ITEM72);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(COMMA_ITEM8_Y * (float)comma.dim.y), COMMA_ITEM82);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(COMMA_ITEM9_Y * (float)comma.dim.y), INITIALIZATION);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(MEDIAN_FILTERING_Y *
(float)comma.dim.y),MEDIAN_FILTERING);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(SOBEL_Y * (float)comma.dim.y),SOBEL);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(LAPLACE_Y * (float)comma.dim.y),LAPLACE);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(LOW_FILTERING_Y * (float)comma.dim.y),LOW_FILTERING);

        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
                (int)(COMMA_ITEM_OPEN_Y * (float)comma.dim.y),
COMMA_ITEM_OPEN);
        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x)-YY,
```

```
                (int)(COMMA_ITEM_SAVE_Y * (float)comma.dim.y),
COMMA_ITEM_SAVE);


        outtextxy((int)(COMMA_ITEM2_X * (float)comma.dim.x),
                (int)(COMMA_ITEM10_Y * (float)comma.dim.y), COMMA_ITEM92);
}

/* ------------------------------------------------------------ */
/* ************************************************************************** */
void graph_ecran_param()
{
        float deg;
        char tmp[10];
        int mat[8];

        unsigned user_pattern = 0x0000;

        setviewport(param.debut.x+2, param.debut.y+2, param.fin.x-2, param.fin.y-2, 1);
        clearviewport();
        settextjustify(LEFT_TEXT, BOTTOM_TEXT);
        setusercharsize(1, 1, 1, 1);
        setusercharsize((int)(PARAM_DEG_L * (float)param.dim.x),
                        textwidth("360"),
                        (int)(PARAM_ITEM_H * (float)param.dim.y),
                        textheight(PARAM_ITEM3));
        setlinestyle(USERBIT_LINE, user_pattern, NORM_WIDTH);

        setusercharsize((int)(PARAM_TITRE_L * (float)param.dim.x),
                        textwidth(PARAM_TITRE),
                        (int)(PARAM_TITRE_H * (float)param.dim.y),
                        textheight(PARAM_TITRE));
        outtextxy((int)(PARAM_TITRE_X * (float)param.dim.x),
                (int)(PARAM_TITRE_Y * (float)param.dim.y), PARAM_TITRE);

        setusercharsize(1, 1, 1, 1);
        setusercharsize((int)(PARAM_ITEM_L * (float)param.dim.x),
                        textwidth(PARAM_ITEM3),
                        (int)(PARAM_ITEM_H * (float)param.dim.y),
                        textheight(PARAM_ITEM3));
        outtextxy((int)(PARAM_ITEM_X * (float)param.dim.x)-XX,
                (int)(PARAM_ITEM1_Y * (float)param.dim.y), PARAM_ITEM1);
        outtextxy((int)(PARAM_ITEM_X * (float)param.dim.x)-XX,
                (int)(PARAM_ITEM2_Y * (float)param.dim.y), PARAM_ITEM2);
        outtextxy((int)(PARAM_ITEM_X * (float)param.dim.x)-XX,
```

```
            (int)(PARAM_ITEM3_Y * (float)param.dim.y), PARAM_ITEM3);
        outtextxy((int)(PARAM_ITEM_X * (float)param.dim.x)-XX,
            (int)(PARAM_ITEM4_Y * (float)param.dim.y), PARAM_ITEM4);


        mat[0] = (int)((PARAM_DEG_X - 0.05) * (float)param.dim.x);
        mat[1] = (int)((PARAM_ITEM1_Y - 0.15) * (float)param.dim.y);
        mat[2] = (int)((PARAM_DEG_X + PARAM_DEG_L + 0.1) * (float)param.dim.x);
        mat[3] = mat[1];
        mat[4] = mat[2];
        mat[6] = mat[0];
        mat[7] = (int)((PARAM_ITEM4_Y + 0.1) * (float)param.dim.y);
        mat[5] = mat[7];
        fillpoly(4, mat);
        setlinestyle(SOLID_LINE, user_pattern, NORM_WIDTH);

        if(teta < 0.001)
        {
            teta = 0;
        }
        deg = teta * (float)180 / PI_1_1;
        (void)gcvt(deg, 3, tmp);
        outtextxy((int)(PARAM_DEG_X * (float)param.dim.x)+XX,
            (int)(PARAM_ITEM1_Y * (float)param.dim.y), tmp);

        if(phi < 0.001)
        {
            phi = 0;
        }
        deg = phi * (float)180 / PI_1_1;
        (void)gcvt(deg, 3, tmp);
        outtextxy((int)(PARAM_DEG_X * (float)param.dim.x)+XX,
            (int)(PARAM_ITEM2_Y * (float)param.dim.y), tmp);

        deg = alpha * (float)180 / PI_1_1;
        (void)gcvt(deg, 3, tmp);
        outtextxy((int)(PARAM_DEG_X * (float)param.dim.x)+XX,
            (int)(PARAM_ITEM3_Y * (float)param.dim.y), tmp);

        deg = beta * (float)180 / PI_1_1;
        (void)gcvt(deg, 3, tmp);
        outtextxy((int)(PARAM_DEG_X * (float)param.dim.x)+XX,
            (int)(PARAM_ITEM4_Y * (float)param.dim.y), tmp);
}
/* ---------------------------------------------------------------- */
```

```
/* ------------------------------------------------------------------- */
/* ******************************************************************** */
/* clef_touche graph_conversion()                                       */
/* ******************************************************************** */
clef_touche graph_conversion()
{

        int x, y, m, n;

        float xm, ym, zm, a, b, c, d, e, f, origine_x, origine_y;

        a = (float)(sin(alpha) * sin(beta) * cos(teta) - cos(beta) *
           sin(teta));
        b = (float)(cos(alpha) * cos(teta));
        c = (float)(sin(beta) * sin(teta) + sin(alpha) * cos(beta) *
           cos(teta));
        d = (float)(sin(alpha) * sin(phi) - cos(alpha) * sin(teta) * cos(phi));
        e = (float)(cos(beta) * cos(teta) * cos(phi) + sin(alpha) * sin(beta) *
           sin(teta) * cos(phi) + cos(alpha) * sin(beta) * sin(phi));
        f = (float)(cos(alpha) * cos(beta) * sin(phi) + sin(alpha) *
           cos(beta) * sin(teta) * cos(phi) - sin(beta) * cos(teta) *
           cos(phi));
        origine_x = (float)graph.dim.x / ((float)2 * asp);
        origine_y = (float)graph.dim.y / (float)2;

        for(y = 0; y < fn_dim; y++)
        {
                if(kbhit())
                {
                        return(TRUE);
                }

                m = y * fn_dim;
                ym = ((float)fn_dep_y + (float)y) * fn_esp;

                for(x = 0; x < fn_dim; x++)
                {
                        n = x + m;
                        xm = ((float)fn_dep_x + (float)x) * fn_esp;
                        zm = mat_m[n] * fn_esp;
                        mat_e[n].x = (int)((origine_x + xm * a + ym * b - zm *
                                c) * asp);
                        mat_e[n].y = (int)(origine_y - (ym * d - xm * e + zm *
                                f));
                }
```

```
        }
        return(FALSE);
}
/* ------------------------------------------------------------------ */

void open_dialog(void)
{
  save_image(OPEN);
}

void save_dialog(void)
{
  save_image(SAVE);
}

void save_image(int flag)
{
int i=0;
char file_name[13];
unsigned image_size;
void far *image_ptr[4];
char ch[13];


image_size=imagesize(X_DIALOG_1, Y_DIALOG_1, X_DIALOG_2, Y_DIALOG_2);
if( (image_ptr[0]=(farmalloc(image_size)))==NULL)
  {
  closegraph();
  printf("Can't allocate memory to save the input/output dialog box!");
  exit(1);
  }

getimage(X_DIALOG_1, Y_DIALOG_1, X_DIALOG_2, Y_DIALOG_2, image_ptr[0]);
putimage(X_DIALOG_1, Y_DIALOG_1, image_ptr[0], XOR_PUT);
setcolor(YELLOW);
rectangle(X_DIALOG_1, Y_DIALOG_1, X_DIALOG_2, Y_DIALOG_2);
rectangle(X_DIALOG_1+3, Y_DIALOG_1+3, X_DIALOG_2-3, Y_DIALOG_2-3);
settextstyle(BOLD_FONT, HORIZ_DIR, 1);
outtextxy(X_DIALOG_1+10, Y_DIALOG_1+70,  "Press ESC to escape.");

if(flag==OPEN)
  {
  outtextxy(X_DIALOG_1+10, Y_DIALOG_1+15, "OPEN FILE DIALOG:");
  outtextxy(X_DIALOG_1+10, Y_DIALOG_1+40, "Enter file to read data from:");
  }
```

```
 if(flag==SAVE)
   {
   outtextxy(X_DIALOG_1+10, Y_DIALOG_1+15, "SAVE FILE DIALOG:");
   outtextxy(X_DIALOG_1+10, Y_DIALOG_1+40, "Enter file to save data in:");
   }

 for(i=0;i<=13;i++)
   {
   file_name[i]='\0';
   ch[i]='\0';
   }
 i=0;

 while(1)
    {
    ch[i]=getch();
    if (ch[i] == ESC)
         goto label1;
    if (ch[i] != ENTER && ch[i] != ESC)
          {
          file_name[i]=ch[i];
          outtextxy(X_DIALOG_1+10, Y_DIALOG_1+50, file_name);
          }
    else
         goto label;
    i++;
    }

label:
if (flag==OPEN)
  {
  read_file_data(file_name);
  }
if (flag==SAVE)
   write_file_data(file_name);
label1 : setcolor(WHITE);
         putimage(X_DIALOG_1, Y_DIALOG_1, image_ptr[0], COPY_PUT);
         farfree(image_ptr[0]);
}

void put_message(void)
   {
         setfillstyle(SOLID_FILL, BLACK);
         setcolor(RED);
         bar(X_DIALOG_1+5, Y_DIALOG_1+100, X_DIALOG_2-5, Y_DIALOG_2-5);
```

```
        outtextxy(X_DIALOG_1+10, Y_DIALOG_1+90, "MESSAGE:");
        outtextxy(X_DIALOG_1+10, Y_DIALOG_1+110, "Failed to open file!.");
        setcolor(WHITE);
        getch();
    }


void write_file_data(char file[12])
    {
    int   handlew, i, j, k;
    FILE        *streamw;
    char      string[4];

        handlew=open(file, O_CREAT | O_TEXT, S_IREAD | S_IWRITE);
        streamw=fdopen(handlew,"w");
        if(streamw==NULL)
          put_message();
        else
          for(i=0;i<Xdir;i++)
            for(j=0;j<Ydir;j++)
                {
                for(k=0;k<4;k++)
                  string[k]='\0';
                itoa(mat1[i][j], string, 10);
                write(handlew, string, 4);
                }
        fclose(streamw);
}

/* ************************************************************************** */

/* ************************************************************************** */
/*     void read_file_data();                                                 */
/*          Function used to read data from a file written by the func-  */
/*          tion write_data_file.The data file is an array 16 by 16.     */
/* ************************************************************************** */

void read_file_data(char file[12])
   {

    int handle,i,j,k;
    FILE *stream;
    char string[4];

        handle=open(file, O_TEXT, O_RDONLY);
```

```
        stream=fdopen(handle, "r");
        if (stream==NULL)
          put_message();
        else
          for(i=0;i<Xdir;i++)
               for(j=0;j<Ydir;j++)
                 {
                 for(k=0;k<4;k++)
                   string[k]='\0';
                 read(handle, string, 4);
                 mat1[i][j]=atoi(string);
                 }
        fclose(stream);

   }
/* ************************************************************************** */

/* ************************************************************************** */



/* ------------------------------------------------------------ */
/* ************************************************************************** */
/* clef_touche graph_ecran()                                    */
/* ************************************************************************** */
clef_touche graph_ecran()
{
        int x, y, a, b, mat[8], c;
        float phia, phib;
        matrice debut;

        phia = angle_add(phi, alpha);
        phib = angle_sub(phi, beta);

        if(teta <= PI_1_4 || teta > PI_7_4)
        {
                if(phib > 0 && phib <= PI_1_1)
                        debut = NEG_X;
                else    debut = POS_X;
        }
        else
        if(teta > PI_1_4 && teta <= PI_3_4)
        {
                if(phia > 0 && phia <= PI_1_1)
```

```
                debut = NEG_Y;
        else    debut = POS_Y;
}
else
if(teta > PI_3_4 && teta <= PI_5_4)
{
        if(phib > 0 && phib <= PI_1_1)
                debut = POS_X;
        else    debut = NEG_X;
}
else
{
        if(phia > 0 && phia <= PI_1_1)
                debut = POS_Y;
        else    debut = NEG_Y;
}

setviewport(graph.debut.x + 1, graph.debut.y + 1, graph.fin.x - 1,
          graph.fin.y - 1, 1);
clearviewport();

switch(debut)
{
        case NEG_Y:
        {
                for(y = 0; y < fn_dim - 1; y++)
                {
                        if(kbhit())
                        {
                                return(TRUE);
                        }

                        b = (a = y * fn_dim) + fn_dim;

                        for(x = 0; x < fn_dim - 1; x++)
                        {
                                mat[0] = mat_e[c = a + x].x;
                                mat[1] = mat_e[c].y;
                                mat[2] = mat_e[++c].x;
                                mat[3] = mat_e[c].y;
                                mat[6] = mat_e[c = b + x].x;
                                mat[7] = mat_e[c].y;
                                mat[4] = mat_e[++c].x;
                                mat[5] = mat_e[c].y;
                                fillpoly(4, mat);
```

```
                }
        }
        break;
}
case POS_X:
{
        for(x = fn_dim - 1; x > 0; x--)
        {
                if(kbhit())
                {
                        return(TRUE);
                }

                for(y = 0; y < fn_dim - 1; y++)
                {
                        mat[0] = mat_e[a = x + y * fn_dim].x;
                        mat[1] = mat_e[a].y;
                        mat[2] = mat_e[b = a + fn_dim].x;
                        mat[3] = mat_e[b].y;
                        mat[4] = mat_e[--b].x;
                        mat[5] = mat_e[b].y;
                        mat[6] = mat_e[--a].x;
                        mat[7] = mat_e[a].y;
                        fillpoly(4, mat);
                }
        }
        break;
}
case POS_Y:
{
        for(y = fn_dim - 1; y > 0; y--)
        {
                if(kbhit())
                {
                        return(TRUE);
                }

                b = (a = y * fn_dim) - fn_dim;

                for(x = fn_dim - 1; x > 0; x--)
                {
                        mat[0] = mat_e[c = a + x].x;
                        mat[1] = mat_e[c].y;
                        mat[2] = mat_e[--c].x;
                        mat[3] = mat_e[c].y;
```

```
                                mat[6] = mat_e[c = b + x].x;
                                mat[7] = mat_e[c].y;
                                mat[4] = mat_e[--c].x;
                                mat[5] = mat_e[c].y;
                                fillpoly(4, mat);
                        }
                }
                break;
        }
        case NEG_X:
        {
                for(x = 0; x < fn_dim - 1; x++)
                {
                        if(kbhit())
                        {
                                return(TRUE);
                        }

                        for(y = fn_dim - 1; y > 0; y--)
                        {
                                mat[0] = mat_e[a = x + y * fn_dim].x;
                                mat[1] = mat_e[a].y;
                                mat[2] = mat_e[b = a - fn_dim].x;
                                mat[3] = mat_e[b].y;
                                mat[4] = mat_e[++b].x;
                                mat[5] = mat_e[b].y;
                                mat[6] = mat_e[++a].x;
                                mat[7] = mat_e[a].y;
                                fillpoly(4, mat);
                        }
                }
                break;
        }
    }
    return(FALSE);
}
/* ------------------------------------------------------------------ */
/* ------------------------------------------------------------------ */
/*  boolean change_sensor (int sensor)                                */
/*  It is called each time the user decide to change the sensor he wants  */
/*  to read. Itis called with the number of sensor that is chosen and     */
/*  and returns a boolean that allowed that the new display to be drawn.  */
/* ****************************************************************** */
boolean change_sensor(int sensor)
{
```

```
        sensor_nr=sensor;
        switch(sensor)
          {
          case 0: chan_nr=15;
                  break;
          case 1: chan_nr=0;
                  break;
          case 2: chan_nr=1;
                  break;
          case 3: chan_nr=2;
                  break;
          case 4: chan_nr=3;
                  break;
          case 5: chan_nr=4;
                  break;
          case 6: chan_nr=5;
                  break;
          case 7: chan_nr=6;
                  break;
          }
        interface_init();
        take_data();
        return (do_filtering(FALSE));
}
/* ------------------------------------------------------------------ */

boolean do_filtering(boolean type)
  {
  if(type==TRUE)
        ;
  else
        ;
  lit_matrice_sonde();
  lit_angle_sonde();
  return (TRUE);
  }
/* ------------------------------------------------------------------ */
/* ****************************************************************** */
/*     void graph_controller()                                        */
/* ****************************************************************** */
void graph_controller()
{
        boolean changement = TRUE;
        touche touch;
```

```
changement=do_filtering(FALSE);

while(1)
{
        if(changement)
        {
                graph_ecran_param();
                if(graph_conversion())
                {
                        changement = TRUE;
                }
                else
                {
                        if(graph_ecran())
                        {
                                changement = TRUE;
                        }
                        else    changement = FALSE;
                }
        }

        while(!kbhit());

        touch = toupper(getch());
        switch(touch)
        {
 case SORTIE: return;

 case MEDIAN_F:
                {
                make_temp(1);
                median_filtering();
                changement=do_filtering(TRUE);
                break;
                }

case SOBEL_F:
                {
                make_temp(1);
                sobel_alg();
                changement=do_filtering(TRUE);
                break;
                }
case LAPLACE_F:
                {
```

```
                conv[0][0]=1;
                conv[0][1]=2;
                conv[0][2]=1;
                conv[1][0]=2;
                conv[1][1]=4;
                conv[1][2]=2;
                conv[2][0]=1;
                conv[2][1]=2;
                conv[2][2]=1;
                make_temp(1);
                convolution();
                changement=do_filtering(TRUE);
                break;
                }
        case LOW_F:
                {
                conv[0][0]=1;
                conv[0][1]=1;
                conv[0][2]=1;
                conv[1][0]=1;
                conv[1][1]=2;
                conv[1][2]=1;
                conv[2][0]=1;
                conv[2][1]=1;
                conv[2][2]=1;
                make_temp(1);
                convolution();
                changement=do_filtering(TRUE);
                break;
                }

case SENSOR_1:
                {
                changement=change_sensor(0);
                break;
                }

case SENSOR_2:
                {
                changement=change_sensor(1);
                break;
                }

case SENSOR_3:
                {
```

```
                changement=change_sensor(2);
                break;
                }

case SENSOR_4:
                {
                changement=change_sensor(3);
                break;
                }

case SENSOR_5:
                {
                changement=change_sensor(4);
                break;
                }

case SENSOR_6:
                {
                changement=change_sensor(5);
                break;
                }

case SENSOR_7:
                {
                changement=change_sensor(6);
                break;
                }

case SENSOR_8:
                {
                changement=change_sensor(7);
                break;
                }

                case IMPRIME:
                {
                        graph_papier(0, 0, ecran.fin.x, ecran.fin.y);
                        break;
                }
                case LIT_SONDE:
                {
                        take_data();
                        changement = do_filtering(FALSE);
                        break;
                }
```

```
case INITIAL:
{
        initialization();
        changement = do_filtering(FALSE);
        break;
}

case OPEN:
{
        open_dialog();
        changement = do_filtering(TRUE);
        break;
}

case SAVE:
{
        save_dialog();
        changement = do_filtering(FALSE);
        break;
}

case SCAN_CODE:
{
        touch = getch();
        switch(touch)
        {
                case PHI_MOINS:
                {
                        phi = angle_sub(phi, PI_1_36);
                        changement = TRUE;
                        break;
                }
                case PHI_PLUS:
                {
                        phi = angle_add(phi, PI_1_36);
                        changement = TRUE;
                        break;
                }
                case TETA_MOINS:
                {
                        teta = angle_sub(teta,
                                        PI_1_36);
                        changement = TRUE;
                        break;
                }
```

```
                                        case TETA_PLUS:
                                        {
                                                teta = angle_add(teta,
                                                                PI_1_36);
                                                changement = TRUE;
                                                break;
                                        }
                                        case GROSSIR:
                                        {
                                                if(rap_fn_ecran <= 9.9)
                                                {
                                                        rap_fn_ecran += 0.1;
                                                        fn_esp = rap_fn_ecran *
                                                         (float)ecran.dim.y /
                                                         (float)fn_dim;
                                                }
                                                changement = TRUE;
                                                break;
                                        }
                                        case RAPETISSER:
                                        {
                                                if(rap_fn_ecran >= 0.2)
                                                {
                                                        rap_fn_ecran -= 0.1;
                                                        fn_esp = rap_fn_ecran *
                                                         (float)ecran.dim.y /
                                                         (float)fn_dim;
                                                }
                                                changement = TRUE;
                                                break;
                                        }
                                }
                        }
                }
        }
}
/* ------------------------------------------------------------ */

/* ------------------------------------------------------------ */
/* ************************************************************************ */
/* void initialization(void)                                    */
/*  It is used to initialized the tactile sensor. A first data is taken     */
/*  and it is used for bias (it is supposed that the sensor is anlouded     */
/*  and this is the initial values for tactile sensor. All the other        */
/*  readings are relatively to the bias (the initialization readings).      */
```

```
/* ************************************************************************** */
void initialization()
{
 int i,j;
 clear_arrays();
 for(i=0;i<num;i++)
  {
       read_sensor(i);
  }
 median(1);
}
/* ---------------------------------------------------------- */

/* ---------------------------------------------------------- */
/* ************************************************************************** */
/*  void take_data()                                          */
/* ************************************************************************** */
void take_data()
   {
   int i;
   clear_arrays();
       for (i=0;i<num;i++)
        read_sensor(i);
       median(2);
       threshold();
   }
/* ---------------------------------------------------------- */

/* ---------------------------------------------------------- */
/* ************************************************************************** */
/* void median(int a)                                         */
/*          This function calculates the median value of all the       */
/*        readings that has been done.                        */
/* ************************************************************************** */
void median(int a)
{
 int sum;
 int I,J;
for(J=0;J<Xdir*Ydir;J++)
       {
       sum=0;
       for(I=0;I<num;I++)
            sum=sum+fvalue[I][J];
       if (a==1)
          bias[J]=sum/num;
```

```
        else
          temp1[J]=bias[J]-sum/num;
        }
  }

/* ---------------------------------------------------------- */

/* ---------------------------------------------------------- */
/* ************************************************************************ */
/*  void threshold()                                          */
/* ************************************************************************ */
void threshold()
    {
    int i,j;

        for(i=0;i<Xdir;i++)
          for(j=0;j<Ydir;j++)
             mat1[i][j]=(temp1[i*16+j]);
     }
/* ---------------------------------------------------------- */

/* ---------------------------------------------------------- */
/* ************************************************************************ */
/*  float angle_add(float a, float b)                                       */
/* ************************************************************************ */

float angle_add(float a, float b)
{
        return((float)(a + b - PI_2_1 * floor((a + b) / PI_2_1)));
}

float angle_sub(float a, float b)
{
        return((float)(a - b - PI_2_1 * floor((a - b) / PI_2_1)));
}


/* ---------------------------------------------------------- */
/* ************************************************************************ */
/* void graph_papier(int x1, int y1, int x2, int y2)                        */
/* ************************************************************************ */
void graph_papier(int x1, int y1, int x2, int y2)
{
        char m;
        int i, j, k, msb, lsb;
```

```
        setviewport(x1, y1, x2, y2, 1);
        fprintf(stdprn, "\x1B\x41%c", 7);
        lsb = (y2 - y1) & 0x00FF;
        msb = (y2 - y1) >> 8;

        for(j = x1; j < (x2 - x1); j += 8)
        {
                fprintf(stdprn, "\x1B*%c%c%c", 0, lsb, msb);
                for(i = y2 - y1; i >= y1; i--)
                {
                        for(m = 0, k = 0; k < 8; k++)
                        {
                                m <<= 1;
                                if(getpixel(j + k, i)) m++;
                        }
                        fprintf(stdprn, "%c", m);
                }
                fprintf(stdprn, "\x0D\x0A");
        }
        fprintf(stdprn, "\f");
}
/* ------------------------------------------------------------------ */

/* ------------------------------------------------------------------ */
/* ****************************************************************** */
/*  void graph_end()  Function used to close the graphic mode and to release */
/*                the memeory alocated for mat_m and mat_e.          */
/* ****************************************************************** */

void graph_end()
   {
       closegraph();
       free(mat_m);
       free(mat_e);
   }
/* ------------------------------------------------------------------ */

void read_sensor(int I)
{
 int i,j;
 int value;
 float volts;

        for(i=0;i<MAX_WIDTH;i++)
              for(j=0;j<MAX_HEIGHT;j++)
```

```
                {
                SetAddress(i,j);
                volts=Execute_AI_VRead();
                PutValue(volts,i,j,I);
                }
}

void PutValue(float v, int i, int j, int I)
        {
        if(v>1.0)
                v=1.0;
        if(v<-1.0)
                v=-1.0;
        fvalue[I][i*MAX_WIDTH+j]=100*v;
        }


void SetAddress (int i, int j )
{
        int    brd,
                port;

        int     address;

        int lines[16]={224,192,96,64,0,32,128,160,232,200,104,72,8,168,136,40};
        int columns[16]={23,22,19,18,2,3,6,7,21,20,17,16,0,1,4,5};

        brd = 1;
        port = 0;
        address =(lines[i])+(columns[j]);
//      if((DIG_Out_Port (brd,port,address)) != 0)
//      {
//      closegraph();
//      printf("Unable to select the address of the I/O National Instrument board.\n");
//      exit(1);
//      }
        DIG_Out_Port(brd, port, address);
}



float Execute_AI_VRead()
        {
        int brd,
                ch,
```

```
                gain;

        double volt;

        brd=1;
        ch=chan_nr;
        gain=1;
//      if((AI_VRead(brd, ch, gain, &volt) ) != 0)
//        {
//        closegraph();
//        printf("Unable to read National Instrument I/O board.\n");
//        exit(1);
//        }
        AI_VRead(brd, ch, gain, &volt);
        return (volt);
        }

void median_filtering()
{
 int filt_9[9],filtt[9];
 int min,p,m,t;
 int I1,J1;
 int i,j;

  for(i=1;i<18;i++)
    {
    for(j=1;j<18;j++)
        {
        filt_9[0]=temp[i-1][j-1];
        filt_9[1]=temp[i-1][j];
        filt_9[2]=temp[i-1][j+1];
        filt_9[3]=temp[i][j-1];
        filt_9[4]=temp[i][j];
        filt_9[5]=temp[i][j+1];
        filt_9[6]=temp[i+1][j-1];
        filt_9[7]=temp[i+1][j];
        filt_9[8]=temp[i+1][j+1];

        for(I1=0;I1<9;I1++)
          {
          p=0;m=0;
          for(J1=0;J1<9;J1++)
              {
              if(filt_9[I1]<filt_9[J1])
                      p=p+1;
```

```
                if(filt_9[I1]==filt_9[J1])
                        m=m+1;
                }

            for(t=0;t<m;t++)
                filtt[9-p-t-1]=filt_9[I1];
             }
                mat1[i][j]=-1.8*filtt[5];
          }
        }
 }

/* ************************************************************************** */
/*  void make_temp()                                                          */
/*                                                                            */
/*          This function builts an array Xdim+2 by Ydim+2 from the    */
/*        input array Xdim by Ydim.                                           */
/* ************************************************************************** */
void make_temp(int flag)
{
 int i,j;

 temp[0][0]=mat1[0][0];
 for(j=1;j<Ydir+1;j++)
       temp[0][j]=mat1[0][j-1];
 temp[0][Ydir+1]=mat1[0][Ydir-1];

 for(i=1;i<Xdir+1;i++)
  {
      temp[i][0]=mat1[i-1][0];
  for(j=1;j<Ydir+1;j++)
        temp[i][j]=mat1[i-1][j-1];
      temp[i][Ydir+1]=mat1[i-1][Ydir-1];
  }

temp[Xdir+1][0]=mat1[Xdir-1][0];
for(j=1;j<Ydir+1;j++)
      temp[Xdir+1][j]=mat1[Xdir-1][j-1];
temp[Xdir+1][Ydir+1]=mat1[Xdir-1][Ydir-1];

if(flag==1 || flag==2)
 for(i=0;i<18;i++)
      for(j=0;j<18;j++)
        if(i==0 || i==17 || j==0 || j==17)
           temp[i][j]=0;
```

```
            else
               if(flag==1)
                    temp[i][j]=-mat1[i-1][j-1]/2;
               else
                    temp[i][j]=-mat1[i-1][j-1];
 }


 /* ************************************************************************** */
 /*  void sobel_alg();                                                         */
 /*       Takes the temporar tactile data temp[18][18] and applay the          */
 /*     Sobel algorithm to it.                                                 */
 /* ************************************************************************** */
 void sobel_alg()
 {
  int i,j;
  int a,b,c,d;

  for(i=1;i<17;i++)
    {
    for(j=1;j<17;j++)
         {
         a=abs(((temp[i-1][j-1]+temp[i-1][j]+temp[i][j-1]  )/3)-
              ((temp[i+1][j]  +temp[i][j+1]+temp[i+1][j+1])/3));

         b=abs(((temp[i-1][j-1]+temp[i][j-1]+temp[i+1][j-1]  )/3)-
              ((temp[i-1][j+1]+temp[i][j+1]+temp[i+1][j+1])/3));

         c=abs(((temp[i-1][j]+temp[i][j+1]+temp[i-1][j+1]  )/3)-
              ((temp[i][j-1]+temp[i+1][j]+temp[i+1][j-1])/3));

         d=abs(((temp[i-1][j-1]+temp[i-1][j]+temp[i-1][j+1]  )/3)-
              ((temp[i+1][j-1]+temp[i+1][j]+temp[i+1][j+1])/3));

         mat1[i-1][j-1]=-1.8*max(max(a,b),max(c,d));
         }
    }
 }
 /* ************************************************************************** */

 /* ************************************************************************** */
 /*  void convolution()                                                        */
 /*                                                                            */
 /*            This function calculates the convolution of the input           */
 /*          array input[Xdim][Ydim]. Various convolution Kernels can be       */
```

```
/*          chosen.                                          */
/* ************************************************************************** */
void convolution()
{
 int i,j;
 int max_value=0;

  for(i=1;i<Xdir+1;i++)
    for(j=1;j<Ydir+1;j++)
      {

mat1[i-1][j-1]=(conv[0][0]*temp[i-1][j-1]+conv[0][1]*temp[i-1][j]+conv[0][2]*temp[i-1][j+1]+
                        conv[1][0]*temp[i][j-1]+conv[1][1]*temp[i][j]+conv[1][2]*temp[i][j+1]+

conv[2][0]*temp[i+1][j-1]+conv[2][1]*temp[i+1][j]+conv[2][2]*temp[i+1][j+1]);
          if(mat1[i-1][j-1]<0)
                mat1[i-1][j-1]=0;
          }
  for(i=0;i<16;i++)
    for(j=0;j<16;j++)
                max_value=max(max_value, mat1[i][j]);

  for(i=0;i<16;i++)
    for(j=0;j<16;j++)
                mat1[i][j]=-mat1[i][j]/((max_value+100)/100);
}
/* ************************************************************************** */
```

```
; This program will find the milestone of a pseudo-random multi-valued sequence of length 1024
;                                                written by
;                                              Stephen Yeung
;

#include <stdio.h>
#include <math.h>
#include <alloc.h>

void main(void);

void pseudo_random_2_1_1(void);
void pseudo_random_2_1_2(void);
void pseudo_random_2_1_3(void);
void pseudo_random_2_1_4(void);
void pseudo_random_2_1_5(void);
void pseudo_random_2_1_6(void);
void pseudo_random_2_1_7(void);
void pseudo_random_2_1_8(void);
void pseudo_random_2_1_9(void);
void pseudo_random_2_1_10(void);
void pseudo_random_2_1_11(void);
void pseudo_random_2_1_12(void);
void pseudo_random_2_1_13(void);
void pseudo_random_2_1_14(void);

void pseudo_random_2_2_1(void);
void pseudo_random_2_2_2(void);
void pseudo_random_2_2_3(void);
void pseudo_random_2_2_4(void);
void pseudo_random_2_2_5(void);
void pseudo_random_2_2_6(void);
void pseudo_random_2_2_7(void);
void pseudo_random_2_2_8(void);
void pseudo_random_2_2_9(void);
void pseudo_random_2_2_10(void);
void pseudo_random_2_2_11(void);

void pseudo_random_3_1_1(void);
void pseudo_random_3_1_2(void);
void pseudo_random_3_1_3(void);
void pseudo_random_3_1_4(void);
void pseudo_random_3_1_5(void);
void pseudo_random_3_1_6(void);
void pseudo_random_3_1_7(void);
```

```
void pseudo_random_3_1_8(void);
void pseudo_random_3_1_9(void);

void main(void)
{
int i;
int p, m, n;
int prma[80];
do
 {
 label1: clrscr();
 printf("\n\n\n\t\tPseudo-random sequences and arrays generation");
 printf("\n\n   To generate a pseudo-random sequence the following parameters are required:");
 printf("\n\t -an integer number 'q' that is a prime or a power of a prime");
 printf("\n\t  to be possible to form the Galois field GF(q);");
 printf("\n\t  This number 'q' is expressed as a power 'm' of a prime number 'p'. ");
 printf("\n\t -an integer 'n' that gives the degree of the polynomial that is used ");
 printf("\n\t  to generate GF(q).");
 printf("\n\n\tChoose these three parameters:");
 printf("\n\t\t-Enter prime 'p' ( 2 or 3).....");
 scanf("%d",&p);
 if(p==2)
  {
       printf("\t\t-Enter integer 'm'( 1, 2, 3, 4)...");
   scanf("%d",&m);
   if(m != 1 && m !=2 && m !=3 && m != 4)
   goto label1;
   if(m ==1)
    {
       printf("\t\t-Enter integer 'n'(1-14).....");
    scanf("%d", &n);
    if(n != 1 && n != 2 && n != 3 && n != 4 && n != 5 && n != 6 && n != 7 &&
       n != 8 && n != 9 && n !=10 && n !=11 && n !=12 && n !=13 && n !=14 )
       goto label1;
    switch(n)
        {
        case  1: pseudo_random_2_1_1();
                          break;
        case  2: pseudo_random_2_1_2();
                          break;
        case  3: pseudo_random_2_1_3();
                          break;
        case  4: pseudo_random_2_1_4();
                          break;
        case  5: pseudo_random_2_1_5();
```

```
                                break;
        case  6: pseudo_random_2_1_6();
                                break;
        case  7: pseudo_random_2_1_7();
                                break;
        case  8: pseudo_random_2_1_8();
                                break;
        case  9: pseudo_random_2_1_9();
                                break;
        case 10: pseudo_random_2_1_10();
                                break;
        case 11: pseudo_random_2_1_11();
                                break;
        case 12: pseudo_random_2_1_12();
                                break;
        case 13: pseudo_random_2_1_13();
                                break;
        case 14: pseudo_random_2_1_14();
                                break;
      } /*end of switch */
} /* end of if (m==1) */

   if(m==2)
   {
   printf("\t\t-Enter integer 'n'(1-11).....");
   scanf("%d",&n);
   if(n != 1 && n != 2 && n != 3 && n != 4 && n != 5 && n != 6 && n != 7 &&
     n != 8 && n != 9 && n !=10 && n !=11 )
     goto label1;
   switch(n)
          {
          case 1: pseudo_random_2_2_1();
                                break;
          case 2: pseudo_random_2_2_2();
                                break;
          case 3: pseudo_random_2_2_3();
                                break;
          case 4: pseudo_random_2_2_4();
                                break;
          case 5: pseudo_random_2_2_5();
                                break;
          case 6: pseudo_random_2_2_6();
                                break;
          case 7: pseudo_random_2_2_7();
                                break;
```

```
                case 8: pseudo_random_2_2_8();
                                break;
                case 9: pseudo_random_2_2_9();
                                break;
                case 10:pseudo_random_2_2_10();
                                break;
                case 11:pseudo_random_2_2_11();
                                break;
          } /* end of switch */
        } /* end of if(m==2) */
        if(m==3)
        {
        printf("\t\t-Enter integer 'n'(1-7).....");
        scanf("%d",&n);
        if(n !=1 && n !=2 && n !=3 && n !=4 && n !=5 && n !=6 && n !=7)
          goto label1;
        }
        if(m==4)
        {
        printf("\t\t-Enter integer 'n'(1-5).....");
        scanf("%d",&n);
        if(n != 1 && n != 2 && n != 3 && n != 4 && n != 5)
          goto label1;
        }
  }
if(p==3)
  {
        printf("\t\t-Enter integer 'm'( 1 or 2).....");
  scanf("%d",&m);
  if(m != 1 && m != 2)
  goto label1;
  if(m==1)
   {
         printf("\t\t-Enter integer 'n'(1-9).....");
   scanf("%d", &n);
   if(n != 1 && n != 2 && n != 3 && n != 4 && n != 5 && n != 6 && n != 7 &&
        n != 8 && n != 9 )
        goto label1;
       switch(n)
              {
              case 1: pseudo_random_3_1_1();
                                break;
              case 2: pseudo_random_3_1_2();
                                break;
              case 3: pseudo_random_3_1_3();
```

```
                    break;
                case 4: pseudo_random_3_1_4();
                    break;
                case 5: pseudo_random_3_1_5();
                    break;
                case 6: pseudo_random_3_1_6();
                    break;
                case 7: pseudo_random_3_1_7();
                    break;
                case 8: pseudo_random_3_1_8();
                    break;
                case 9: pseudo_random_3_1_9();
                    break;
                } /* end of switch */
            }
        if(m==2)
          {
              printf("\t\t-Enter integer 'n'(1-7).....");
          scanf("%d", &n);
          if(n !=1 && n !=2 && n !=3 && n !=4 && n !=5 && n !=6 && n !=7)
              goto label1;
            }

        }
       if(getch()=='q')
         exit(1);
      }while((p !=2) && (p !=3) );
}

void pseudo_random_2_1_1(void)
{
 int prma[1];
 int i;
 clrscr();
 printf("\nYou have chosen to generate a pseudo-random sequence ");
 printf("\ncharacterized by the following parameters:");
 printf("\n\t\t -p=2, m=1, q=1, GF(2) (a pseudo-random binary sequence).");
 printf("\n\t\t -n=1 (the length of this sequence is 1).\n\n");

prma[0]=1;

for (i=0;i<1; i++)
   printf("%d ",prma[i]);
getch();
```

```
}

void pseudo_random_2_1_2(void)
{
 int prma[3];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=2 (the length of this sequence is 3).\n\n");

 prma[0]=0;
 prma[1]=1;

 for(i=2;i<3;i++)
   prma[i]=((prma[i-1]%2)+prma[i-2])%2;

 for (i=0;i<3; i++)
    printf("%d ",prma[i]);
 getch();
 }

 void pseudo_random_2_1_3(void)
 {
  int prma[7];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=3 (the length of this sequence is 7).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=1;

 for(i=3;i<7;i++)
   prma[i]=((prma[i-2]%2)+prma[i-3])%2;

 for (i=0;i<7; i++)
    printf("%d ",prma[i]);
 getch();
 }
```

```
void pseudo_random_2_1_4(void)
{

  int prma[15];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=4 (the length of this sequence is 15).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=1;

 for(i=4;i<15;i++)
   prma[i]=((prma[i-3]%2)+prma[i-4])%2;

 for (i=0;i<15; i++)
    printf("%d ",prma[i]);
 getch();
 }

 void pseudo_random_2_1_5(void)
 {

  int prma[31];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=5 (the length of this sequence is 31).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=1;

 for(i=5;i<31;i++)
   prma[i]=((prma[i-3]%2)+prma[i-5])%2;

 for (i=0;i<31; i++)
```

```
    printf("%d ",prma[i]);
 getch();
 }

 void pseudo_random_2_1_6(void)
 {

  int prma[63];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=6 (the length of this sequence is 63).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=0;
 prma[5]=1;

 for(i=6;i<63;i++)
   prma[i]=((prma[i-5]%2)+prma[i-6])%2;

 for (i=0;i<63; i++)
    printf("%d ",prma[i]);
 getch();
 }

 void pseudo_random_2_1_7(void)
 {

  int prma[127];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=7 (the length of this sequence is 127).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
```

```
prma[4]=0;
prma[5]=0;
prma[6]=1;

for(i=7;i<127;i++)
  prma[i]=((prma[i-6]%2)+prma[i-7])%2;

for (i=0;i<127; i++)
   printf("%d ",prma[i]);
getch();
}

void pseudo_random_2_1_8(void)
{
 int *prma;
 int i;
 clrscr();
 printf("\nYou have chosen to generate a pseudo-random sequence ");
 printf("\ncharacterized by the following parameters:");
 printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
 printf("\n\t\t -n=8 (the length of this sequence is 255).\n\n");


 if( ( prma=(int *)malloc(255*sizeof(int)) ) ==NULL)
   {
   printf("\nFailed to allocate memory!!!");
   exit(0);
   }
prma[0]=0;
prma[1]=0;
prma[2]=0;
prma[3]=0;
prma[4]=0;
prma[5]=0;
prma[6]=0;
prma[7]=1;

for(i=8;i<255;i++)
  prma[i]=(prma[i-2]+prma[i-3]+prma[i-7]+prma[i-8])%2;

for (i=0;i<255; i++)
   printf("%d ",prma[i]);
getch();
free(prma);
}
```

```
void pseudo_random_2_1_9(void)
{

  int prma[511];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=9 (the length of this sequence is 511).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=0;
 prma[5]=0;
 prma[6]=0;
 prma[7]=0;
 prma[8]=1;

 for(i=9;i<511;i++)
   prma[i]=(prma[i-5]+prma[i-9])%2;

 for (i=0;i<511; i++)
    printf("%d ",prma[i]);
 getch();
 }

 void pseudo_random_2_1_10(void)
 {

  int prma[1023];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=10 (the length of this sequence is 1023).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=0;
```

```
 prma[5]=0;
 prma[6]=0;
 prma[7]=0;
 prma[8]=0;
 prma[9]=1;

 for(i=10;i<1023;i++)
   prma[i]=(prma[i-7]+prma[i-10])%2;

 for (i=0;i<1023; i++)
    printf("%d",prma[i]);
 getch();
 }

 void pseudo_random_2_1_11(void)
 {

  int prma[2047];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=11 (the length of this sequence is 2047).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=0;
 prma[5]=0;
 prma[6]=0;
 prma[7]=0;
 prma[8]=0;
 prma[9]=0;
 prma[10]=1;

 for(i=11;i<2047;i++)
   prma[i]=(prma[i-9]+prma[i-11])%2;

 for (i=0;i<2047; i++)
    printf("%d",prma[i]);
 getch();
 }
```

```
void pseudo_random_2_1_12(void)
{

  int prma[4095];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=12 (the length of this sequence is 4095).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=0;
 prma[5]=0;
 prma[6]=0;
 prma[7]=0;
 prma[8]=0;
 prma[9]=0;
 prma[10]=0;
 prma[11]=1;

 for(i=12;i<4095;i++)
   prma[i]=(prma[i-5]+prma[i-8]+prma[i-9]+prma[i-12])%2;

 for (i=0;i<4095; i++)
    printf("%d",prma[i]);
 getch();
 }

 void pseudo_random_2_1_13(void)
 {

  int prma[8191];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=13 (the length of this sequence is 8191).\n\n");

prma[0]=0;
prma[1]=0;
```

```
prma[2]=0;
prma[3]=0;
prma[4]=0;
prma[5]=0;
prma[6]=0;
prma[7]=0;
prma[8]=0;
prma[9]=0;
prma[10]=0;
prma[11]=0;
prma[12]=1;

for(i=13;i<8191;i++)
  prma[i]=(prma[i-9]+prma[i-10]+prma[i-12]+prma[i-13])%2;

for (i=0;i<8191; i++)
   printf("%d",prma[i]);
getch();
}

void pseudo_random_2_1_14(void)
{

  int *prma;
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=1, q=2, GF(2) (a pseudo-random binary sequence).");
  printf("\n\t\t -n=114 (the length of this sequence is 16393).\n\n");


  if( ( prma=(int *)malloc(16393*sizeof(int)) ) ==NULL)
     {
     printf("\nFailed to allocate memory!!!");
     exit(0);
     }
 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=0;
 prma[5]=0;
 prma[6]=0;
 prma[7]=0;
```

```
 prma[8]=0;
 prma[9]=0;
 prma[10]=0;
 prma[11]=0;
 prma[12]=0;
 prma[13]=1;

 for(i=14;i<16393;i++)
   prma[i]=(prma[i-2]+prma[i-3]+prma[i-13]+prma[i-14])%2;

 for (i=0;i<16393; i++)
    printf("%d",prma[i]);
 getch();
 free(prma);
 }

 void pseudo_random_2_2_1(void)
 {
  int prma[3];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=2, q=1, GF(4)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=1 (the length of this sequence is 3).\n\n");

 prma[0]=1;

 for(i=1;i<3;i++)
   prma[i]=(2*prma[i-1])%4;

 for (i=0;i<3; i++)
    printf("%d ",prma[i]);
 getch();

 }

 void pseudo_random_2_2_2(void)
 {
  int *prma;
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=2, q=1, GF(4)(a pseudo-random four symbols sequence).");
```

```
        prma[i]=1;
 if(prma[i-1]==1 && prma[i-2]==3 && prma[i-3]==3)
        prma[i]=3;
 if(prma[i-1]==2 && prma[i-2]==0 && prma[i-3]==0)
        prma[i]=2;
 if(prma[i-1]==2 && prma[i-2]==0 && prma[i-3]==1)
        prma[i]=0;
 if(prma[i-1]==2 && prma[i-2]==0 && prma[i-3]==2)
        prma[i]=1;
 if(prma[i-1]==2 && prma[i-2]==0 && prma[i-3]==3)
        prma[i]=3;
 if(prma[i-1]==2 && prma[i-2]==1 && prma[i-3]==0)
        prma[i]=3;
 if(prma[i-1]==2 && prma[i-2]==1 && prma[i-3]==1)
        prma[i]=1;
 if(prma[i-1]==2 && prma[i-2]==1 && prma[i-3]==2)
        prma[i]=0;
 if(prma[i-1]==2 && prma[i-2]==1 && prma[i-3]==3)
        prma[i]=2;
 if(prma[i-1]==2 && prma[i-2]==2 && prma[i-3]==0)
        prma[i]=0;
 if(prma[i-1]==2 && prma[i-2]==2 && prma[i-3]==1)
        prma[i]=2;
 if(prma[i-1]==2 && prma[i-2]==2 && prma[i-3]==2)
        prma[i]=3;
 if(prma[i-1]==2 && prma[i-2]==2 && prma[i-3]==3)
        prma[i]=1;
 if(prma[i-1]==2 && prma[i-2]==3 && prma[i-3]==0)
        prma[i]=1;
 if(prma[i-1]==2 && prma[i-2]==3 && prma[i-3]==1)
        prma[i]=3;
 if(prma[i-1]==2 && prma[i-2]==3 && prma[i-3]==2)
        prma[i]=2;
 if(prma[i-1]==2 && prma[i-2]==3 && prma[i-3]==3)
        prma[i]=0;
 if(prma[i-1]==3 && prma[i-2]==0 && prma[i-3]==0)
        prma[i]=3;
 if(prma[i-1]==3 && prma[i-2]==0 && prma[i-3]==1)
        prma[i]=1;
 if(prma[i-1]==3 && prma[i-2]==0 && prma[i-3]==2)
        prma[i]=0;
 if(prma[i-1]==3 && prma[i-2]==0 && prma[i-3]==3)
        prma[i]=2;
 if(prma[i-1]==3 && prma[i-2]==1 && prma[i-3]==0)
        prma[i]=2;
```

```
    if(prma[i-1]==3 && prma[i-2]==1 && prma[i-3]==1)
          prma[i]=0;
    if(prma[i-1]==3 && prma[i-2]==1 && prma[i-3]==2)
          prma[i]=1;
    if(prma[i-1]==3 && prma[i-2]==1 && prma[i-3]==3)
          prma[i]=3;
    if(prma[i-1]==3 && prma[i-2]==2 && prma[i-3]==0)
          prma[i]=1;
    if(prma[i-1]==3 && prma[i-2]==2 && prma[i-3]==1)
          prma[i]=3;
    if(prma[i-1]==3 && prma[i-2]==2 && prma[i-3]==2)
          prma[i]=2;
    if(prma[i-1]==3 && prma[i-2]==2 && prma[i-3]==3)
          prma[i]=0;
    if(prma[i-1]==3 && prma[i-2]==3 && prma[i-3]==0)
          prma[i]=0;
    if(prma[i-1]==3 && prma[i-2]==3 && prma[i-3]==1)
          prma[i]=2;
    if(prma[i-1]==3 && prma[i-2]==3 && prma[i-3]==2)
          prma[i]=3;
    if(prma[i-1]==3 && prma[i-2]==3 && prma[i-3]==3)
          prma[i]=1;
    }
 for (i=0;i<63; i++)
    printf("%d ",prma[i]);
 getch();
 free(prma);
 }

 void pseudo_random_2_2_4(void)
 {
  int *prma;
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=2, q=4, GF(4)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=4 (the length of this sequence is 255).\n\n");

  if( ( prma=(int *)malloc(255*sizeof(int)) ) ==NULL)
    {
    printf("\nFailed to allocate memory!!!");
    exit(0);
    }
prma[0]=0;
```

```
prma[1]=0;
prma[2]=0;
prma[3]=1;

for(i=4;i<255;i++)
  {
  if(prma[i-2]==0 && prma[i-3]==0 && prma[i-4]==0)
          prma[i]=0;
  if(prma[i-2]==0 && prma[i-3]==0 && prma[i-4]==1)
          prma[i]=3;
  if(prma[i-2]==0 && prma[i-3]==0 && prma[i-4]==2)
          prma[i]=1;
  if(prma[i-2]==0 && prma[i-3]==0 && prma[i-4]==3)
          prma[i]=2;
  if(prma[i-2]==0 && prma[i-3]==1 && prma[i-4]==0)
          prma[i]=2;
  if(prma[i-2]==0 && prma[i-3]==1 && prma[i-4]==1)
          prma[i]=1;
  if(prma[i-2]==0 && prma[i-3]==1 && prma[i-4]==2)
          prma[i]=3;
  if(prma[i-2]==0 && prma[i-3]==1 && prma[i-4]==3)
          prma[i]=0;
  if(prma[i-2]==0 && prma[i-3]==2 && prma[i-4]==0)
          prma[i]=3;
  if(prma[i-2]==0 && prma[i-3]==2 && prma[i-4]==1)
          prma[i]=0;
  if(prma[i-2]==0 && prma[i-3]==2 && prma[i-4]==2)
          prma[i]=2;
  if(prma[i-2]==0 && prma[i-3]==2 && prma[i-4]==3)
          prma[i]=1;
  if(prma[i-2]==0 && prma[i-3]==3 && prma[i-4]==0)
          prma[i]=1;
  if(prma[i-2]==0 && prma[i-3]==3 && prma[i-4]==1)
          prma[i]=2;
  if(prma[i-2]==0 && prma[i-3]==3 && prma[i-4]==2)
          prma[i]=0;
  if(prma[i-2]==0 && prma[i-3]==3 && prma[i-4]==3)
          prma[i]=3;
  if(prma[i-2]==1 && prma[i-3]==0 && prma[i-4]==0)
          prma[i]=1;
  if(prma[i-2]==1 && prma[i-3]==0 && prma[i-4]==1)
          prma[i]=2;
  if(prma[i-2]==1 && prma[i-3]==0 && prma[i-4]==2)
          prma[i]=0;
  if(prma[i-2]==1 && prma[i-3]==0 && prma[i-4]==3)
```

```
        prma[i]=3;
if(prma[i-2]==1 && prma[i-3]==1 && prma[i-4]==0)
        prma[i]=3;
if(prma[i-2]==1 && prma[i-3]==1 && prma[i-4]==1)
        prma[i]=0;
if(prma[i-2]==1 && prma[i-3]==1 && prma[i-4]==2)
        prma[i]=2;
if(prma[i-2]==1 && prma[i-3]==1 && prma[i-4]==3)
        prma[i]=1;
if(prma[i-2]==1 && prma[i-3]==2 && prma[i-4]==0)
        prma[i]=2;
if(prma[i-2]==1 && prma[i-3]==2 && prma[i-4]==1)
        prma[i]=1;
if(prma[i-2]==1 && prma[i-3]==2 && prma[i-4]==2)
        prma[i]=3;
if(prma[i-2]==1 && prma[i-3]==2 && prma[i-4]==3)
        prma[i]=0;
if(prma[i-2]==1 && prma[i-3]==3 && prma[i-4]==0)
        prma[i]=0;
if(prma[i-2]==1 && prma[i-3]==3 && prma[i-4]==1)
        prma[i]=3;
if(prma[i-2]==1 && prma[i-3]==3 && prma[i-4]==2)
        prma[i]=1;
if(prma[i-2]==1 && prma[i-3]==3 && prma[i-4]==3)
        prma[i]=2;
if(prma[i-2]==2 && prma[i-3]==0 && prma[i-4]==0)
        prma[i]=2;
if(prma[i-2]==2 && prma[i-3]==0 && prma[i-4]==1)
        prma[i]=1;
if(prma[i-2]==2 && prma[i-3]==0 && prma[i-4]==2)
        prma[i]=3;
if(prma[i-2]==2 && prma[i-3]==0 && prma[i-4]==3)
        prma[i]=0;
if(prma[i-2]==2 && prma[i-3]==1 && prma[i-4]==0)
        prma[i]=0;
if(prma[i-2]==2 && prma[i-3]==1 && prma[i-4]==1)
        prma[i]=3;
if(prma[i-2]==2 && prma[i-3]==1 && prma[i-4]==2)
        prma[i]=1;
if(prma[i-2]==2 && prma[i-3]==1 && prma[i-4]==3)
        prma[i]=2;
if(prma[i-2]==2 && prma[i-3]==2 && prma[i-4]==0)
        prma[i]=1;
if(prma[i-2]==2 && prma[i-3]==2 && prma[i-4]==1)
        prma[i]=2;
```

```
 if(prma[i-2]==2 && prma[i-3]==2 && prma[i-4]==2)
        prma[i]=0;
 if(prma[i-2]==2 && prma[i-3]==2 && prma[i-4]==3)
        prma[i]=3;
 if(prma[i-2]==2 && prma[i-3]==3 && prma[i-4]==0)
        prma[i]=3;
 if(prma[i-2]==2 && prma[i-3]==3 && prma[i-4]==1)
        prma[i]=0;
 if(prma[i-2]==2 && prma[i-3]==3 && prma[i-4]==2)
        prma[i]=2;
 if(prma[i-2]==2 && prma[i-3]==3 && prma[i-4]==3)
        prma[i]=1;
 if(prma[i-2]==3 && prma[i-3]==0 && prma[i-4]==0)
        prma[i]=3;
 if(prma[i-2]==3 && prma[i-3]==0 && prma[i-4]==1)
        prma[i]=0;
 if(prma[i-2]==3 && prma[i-3]==0 && prma[i-4]==2)
        prma[i]=2;
 if(prma[i-2]==3 && prma[i-3]==0 && prma[i-4]==3)
        prma[i]=1;
 if(prma[i-2]==3 && prma[i-3]==1 && prma[i-4]==0)
        prma[i]=1;
 if(prma[i-2]==3 && prma[i-3]==1 && prma[i-4]==1)
        prma[i]=2;
 if(prma[i-2]==3 && prma[i-3]==1 && prma[i-4]==2)
        prma[i]=0;
 if(prma[i-2]==3 && prma[i-3]==1 && prma[i-4]==3)
        prma[i]=3;
 if(prma[i-2]==3 && prma[i-3]==2 && prma[i-4]==0)
        prma[i]=0;
 if(prma[i-2]==3 && prma[i-3]==2 && prma[i-4]==1)
        prma[i]=3;
 if(prma[i-2]==3 && prma[i-3]==2 && prma[i-4]==2)
        prma[i]=1;
 if(prma[i-2]==3 && prma[i-3]==2 && prma[i-4]==3)
        prma[i]=2;
 if(prma[i-2]==3 && prma[i-3]==3 && prma[i-4]==0)
        prma[i]=2;
 if(prma[i-2]==3 && prma[i-3]==3 && prma[i-4]==1)
        prma[i]=1;
 if(prma[i-2]==3 && prma[i-3]==3 && prma[i-4]==2)
        prma[i]=3;
 if(prma[i-2]==3 && prma[i-3]==3 && prma[i-4]==3)
        prma[i]=0;
}
```

```
 for (i=0;i<255; i++)
    printf("%d ",prma[i]);
 getch();
 free(prma);
 }

 void pseudo_random_2_2_5(void)
 {

  int *prma;
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=2, q=4, GF(4)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=5 (the length of this sequence is 1023).\n\n");

  if( ( prma=(int *)malloc(1023*sizeof(int)) ) ==NULL)
     {
     printf("\nFailed to allocate memory!!!");
     exit(0);
     }
 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=1;

 for(i=5;i<1023;i++)
   {

  if(prma[i-4]==0 && prma[i-5]==0)
          prma[i]=0;
  if(prma[i-4]==0 && prma[i-5]==1)
          prma[i]=2;
  if(prma[i-4]==0 && prma[i-5]==2)
          prma[i]=3;
  if(prma[i-4]==0 && prma[i-5]==3)
          prma[i]=1;
  if(prma[i-4]==1 && prma[i-5]==0)
          prma[i]=1;
  if(prma[i-4]==1 && prma[i-5]==1)
          prma[i]=3;
  if(prma[i-4]==1 && prma[i-5]==2)
          prma[i]=2;
```

```
    if(prma[i-4]==1 && prma[i-5]==3)
            prma[i]=0;
    if(prma[i-4]==2 && prma[i-5]==0)
            prma[i]=2;
    if(prma[i-4]==2 && prma[i-5]==1)
            prma[i]=0;
    if(prma[i-4]==2 && prma[i-5]==2)
            prma[i]=1;
    if(prma[i-4]==2 && prma[i-5]==3)
            prma[i]=3;
    if(prma[i-4]==3 && prma[i-5]==0)
            prma[i]=3;
    if(prma[i-4]==3 && prma[i-5]==1)
            prma[i]=1;
    if(prma[i-4]==3 && prma[i-5]==2)
            prma[i]=0;
    if(prma[i-4]==3 && prma[i-5]==3)
            prma[i]=2;
    }
 for (i=0;i<1023; i++)
    printf("%d ",prma[i]);
 getch();
 free(prma);
 }

 void pseudo_random_2_2_6(void)
 {


  int *prma;
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=2, m=2, q=4, GF(4)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=6 (the length of this sequence is 4095).\n\n");

  if( ( prma=(int *)malloc(4095*sizeof(int)) ) ==NULL)
    {
    printf("\nFailed to allocate memory!!!");
    exit(0);
    }
 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
```

```
prma[3]=0;
prma[4]=0;
prma[5]=1;

for(i=6;i<4095;i++)
  {
  if(prma[i-4]==0 && prma[i-5]==0 && prma[i-6]==0)
          prma[i]=0;
  if(prma[i-4]==0 && prma[i-5]==0 && prma[i-6]==1)
          prma[i]=2;
  if(prma[i-4]==0 && prma[i-5]==0 && prma[i-6]==2)
          prma[i]=3;
  if(prma[i-4]==0 && prma[i-5]==0 && prma[i-6]==3)
          prma[i]=1;
  if(prma[i-4]==0 && prma[i-5]==1 && prma[i-6]==0)
          prma[i]=1;
  if(prma[i-4]==0 && prma[i-5]==1 && prma[i-6]==1)
          prma[i]=3;
  if(prma[i-4]==0 && prma[i-5]==1 && prma[i-6]==2)
          prma[i]=2;
  if(prma[i-4]==0 && prma[i-5]==1 && prma[i-6]==3)
          prma[i]=0;
  if(prma[i-4]==0 && prma[i-5]==2 && prma[i-6]==0)
          prma[i]=2;
  if(prma[i-4]==0 && prma[i-5]==2 && prma[i-6]==1)
          prma[i]=0;
  if(prma[i-4]==0 && prma[i-5]==2 && prma[i-6]==2)
          prma[i]=1;
  if(prma[i-4]==0 && prma[i-5]==2 && prma[i-6]==3)
          prma[i]=3;
  if(prma[i-4]==0 && prma[i-5]==3 && prma[i-6]==0)
          prma[i]=3;
  if(prma[i-4]==0 && prma[i-5]==3 && prma[i-6]==1)
          prma[i]=1;
  if(prma[i-4]==0 && prma[i-5]==3 && prma[i-6]==2)
          prma[i]=0;
  if(prma[i-4]==0 && prma[i-5]==3 && prma[i-6]==3)
          prma[i]=2;
  if(prma[i-4]==1 && prma[i-5]==0 && prma[i-6]==0)
          prma[i]=1;
  if(prma[i-4]==1 && prma[i-5]==0 && prma[i-6]==1)
          prma[i]=3;
  if(prma[i-4]==1 && prma[i-5]==0 && prma[i-6]==2)
          prma[i]=2;
  if(prma[i-4]==1 && prma[i-5]==0 && prma[i-6]==3)
```

```
        prma[i]=0;
 if(prma[i-4]==1 && prma[i-5]==1 && prma[i-6]==0)
        prma[i]=0;
 if(prma[i-4]==1 && prma[i-5]==1 && prma[i-6]==1)
        prma[i]=2;
 if(prma[i-4]==1 && prma[i-5]==1 && prma[i-6]==2)
        prma[i]=3;
 if(prma[i-4]==1 && prma[i-5]==1 && prma[i-6]==3)
        prma[i]=1;
 if(prma[i-4]==1 && prma[i-5]==2 && prma[i-6]==0)
        prma[i]=3;
 if(prma[i-4]==1 && prma[i-5]==2 && prma[i-6]==1)
        prma[i]=1;
 if(prma[i-4]==1 && prma[i-5]==2 && prma[i-6]==2)
        prma[i]=0;
 if(prma[i-4]==1 && prma[i-5]==2 && prma[i-6]==3)
        prma[i]=2;
 if(prma[i-4]==1 && prma[i-5]==3 && prma[i-6]==0)
        prma[i]=2;
 if(prma[i-4]==1 && prma[i-5]==3 && prma[i-6]==1)
        prma[i]=0;
 if(prma[i-4]==1 && prma[i-5]==3 && prma[i-6]==2)
        prma[i]=1;
 if(prma[i-4]==1 && prma[i-5]==3 && prma[i-6]==3)
        prma[i]=3;
 if(prma[i-4]==2 && prma[i-5]==0 && prma[i-6]==0)
        prma[i]=2;
 if(prma[i-4]==2 && prma[i-5]==0 && prma[i-6]==1)
        prma[i]=0;
 if(prma[i-4]==2 && prma[i-5]==0 && prma[i-6]==2)
        prma[i]=1;
 if(prma[i-4]==2 && prma[i-5]==0 && prma[i-6]==3)
        prma[i]=3;
 if(prma[i-4]==2 && prma[i-5]==1 && prma[i-6]==0)
        prma[i]=3;
 if(prma[i-4]==2 && prma[i-5]==1 && prma[i-6]==1)
        prma[i]=1;
 if(prma[i-4]==2 && prma[i-5]==1 && prma[i-6]==2)
        prma[i]=0;
 if(prma[i-4]==2 && prma[i-5]==1 && prma[i-6]==3)
        prma[i]=2;
 if(prma[i-4]==2 && prma[i-5]==2 && prma[i-6]==0)
        prma[i]=0;
 if(prma[i-4]==2 && prma[i-5]==2 && prma[i-6]==1)
        prma[i]=2;
```

```
 if(prma[i-4]==2 && prma[i-5]==2 && prma[i-6]==2)
        prma[i]=3;
 if(prma[i-4]==2 && prma[i-5]==2 && prma[i-6]==3)
        prma[i]=1;
 if(prma[i-4]==2 && prma[i-5]==3 && prma[i-6]==0)
        prma[i]=1;
 if(prma[i-4]==2 && prma[i-5]==3 && prma[i-6]==1)
        prma[i]=3;
 if(prma[i-4]==2 && prma[i-5]==3 && prma[i-6]==2)
        prma[i]=2;
 if(prma[i-4]==2 && prma[i-5]==3 && prma[i-6]==3)
        prma[i]=0;
 if(prma[i-4]==3 && prma[i-5]==0 && prma[i-6]==0)
        prma[i]=3;
 if(prma[i-4]==3 && prma[i-5]==0 && prma[i-6]==1)
        prma[i]=1;
 if(prma[i-4]==3 && prma[i-5]==0 && prma[i-6]==2)
        prma[i]=0;
 if(prma[i-4]==3 && prma[i-5]==0 && prma[i-6]==3)
        prma[i]=2;
 if(prma[i-4]==3 && prma[i-5]==1 && prma[i-6]==0)
        prma[i]=2;
 if(prma[i-4]==3 && prma[i-5]==1 && prma[i-6]==1)
        prma[i]=0;
 if(prma[i-4]==3 && prma[i-5]==1 && prma[i-6]==2)
        prma[i]=1;
 if(prma[i-4]==3 && prma[i-5]==1 && prma[i-6]==3)
        prma[i]=3;
 if(prma[i-4]==3 && prma[i-5]==2 && prma[i-6]==0)
        prma[i]=1;
 if(prma[i-4]==3 && prma[i-5]==2 && prma[i-6]==1)
        prma[i]=3;
 if(prma[i-4]==3 && prma[i-5]==2 && prma[i-6]==2)
        prma[i]=2;
 if(prma[i-4]==3 && prma[i-5]==2 && prma[i-6]==3)
        prma[i]=0;
 if(prma[i-4]==3 && prma[i-5]==3 && prma[i-6]==0)
        prma[i]=0;
 if(prma[i-4]==3 && prma[i-5]==3 && prma[i-6]==1)
        prma[i]=2;
 if(prma[i-4]==3 && prma[i-5]==3 && prma[i-6]==2)
        prma[i]=3;
 if(prma[i-4]==3 && prma[i-5]==3 && prma[i-6]==3)
        prma[i]=1;
}
```

```
 for (i=0;i<4095; i++)
    printf("%d ",prma[i]);
 getch();
 free(prma);
 }

 void pseudo_random_2_2_7(void)
 {
 }

 void pseudo_random_2_2_8(void)
 {
 }

 void pseudo_random_2_2_9(void)
 {
 }

 void pseudo_random_2_2_10(void)
 {
 }

 void pseudo_random_2_2_11(void)
 {
 }

 void pseudo_random_3_1_1(void)
 {

  int prma[2];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=1 (the length of this sequence is 2).\n\n");

 prma[0]=1;

 for(i=1;i<2;i++)
   prma[i]=(2*prma[i-1])%4;

 for (i=0;i<2; i++)
    printf("%d ",prma[i]);
 getch();
```

```
 }

 void pseudo_random_3_1_2(void)
 {
  int prma[8];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=2 (the length of this sequence is 8).\n\n");

 prma[0]=0;
 prma[1]=1;

 for(i=2;i<8;i++)
   prma[i]=( (2*prma[i-1])%3+(prma[i-2]%3) )%3;

 for (i=0;i<8; i++)
    printf("%d ",prma[i]);
 getch();

 }
 void pseudo_random_3_1_3(void)
 {
  int prma[27];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=3 (the length of this sequence is 26).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=1;

 for(i=3;i<26;i++)
   prma[i]=( (prma[i-2])%3+(2*prma[i-3]%3) )%3;

 for (i=0;i<26; i++)
    printf("%d ",prma[i]);
 getch();
 }
```

```
 void pseudo_random_3_1_4(void)
 {
  int prma[80];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=4 (the length of this sequence is 80).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=1;

 for(i=4;i<80;i++)
   prma[i]=( (2*prma[i-3])%3+(prma[i-4]%3) )%3;

 for (i=0;i<80; i++)
    printf("%d ",prma[i]);
 getch();
 }

 void pseudo_random_3_1_5(void)
 {
  int prma[242];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=5 (the length of this sequence is 242).\n\n");

prma[0]=0;
prma[1]=0;
prma[2]=0;
prma[3]=0;
prma[4]=1;

for(i=5;i<242;i++)
  prma[i]=( (prma[i-4])%3+2*(prma[i-5]%3) )%3;

for (i=0;i<242; i++)
   printf("%d ",prma[i]);
getch();
```

```
}

void pseudo_random_3_1_6(void)
{
 int prma[728];
 int i;
 clrscr();
 printf("\nYou have chosen to generate a pseudo-random sequence ");
 printf("\ncharacterized by the following parameters:");
 printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
 printf("\n\t\t -n=6 (the length of this sequence is 728).\n\n");

prma[0]=0;
prma[1]=0;
prma[2]=0;
prma[3]=0;
prma[4]=0;
prma[5]=1;

for(i=6;i<728;i++)
  prma[i]=( (2*prma[i-5])%3+(prma[i-6]%3) )%3;

for (i=0;i<728; i++)
   printf("%d ",prma[i]);
getch();
}

void pseudo_random_3_1_7(void)
{
 int prma[2186];
 int i;
 clrscr();
 printf("\nYou have chosen to generate a pseudo-random sequence ");
 printf("\ncharacterized by the following parameters:");
 printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
 printf("\n\t\t -n=7 (the length of this sequence is 2186).\n\n");

prma[0]=0;
prma[1]=0;
prma[2]=0;
prma[3]=0;
prma[4]=0;
prma[5]=0;
prma[6]=1;
```

```
 for(i=7;i<2186;i++)
   prma[i]=( (2*prma[i-1])%3+2*(prma[i-3]%3)+2*prma[i-7]%3 )%3;

 for (i=0;i<2186; i++)
    printf("%d ",prma[i]);
 getch();
 }

 void pseudo_random_3_1_8(void)
 {
  int prma[6560];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
  printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=8 (the length of this sequence is 6560).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=0;
 prma[5]=0;
 prma[6]=0;
 prma[7]=1;

 for(i=8;i<6560;i++)
   prma[i]=( (2*prma[i-3])%3+prma[i-8]%3 )%3;

 for (i=0;i<6560; i++)
    printf("%d ",prma[i]);
 getch();
 }


 void pseudo_random_3_1_9(void)
 {
 }
 /*
  int prma[13682];
  int i;
  clrscr();
  printf("\nYou have chosen to generate a pseudo-random sequence ");
  printf("\ncharacterized by the following parameters:");
```

```
  printf("\n\t\t -p=3, m=1, q=3, GF(3)(a pseudo-random four symbols sequence).");
  printf("\n\t\t -n=9 (the length of this sequence is 13682).\n\n");

 prma[0]=0;
 prma[1]=0;
 prma[2]=0;
 prma[3]=0;
 prma[4]=0;
 prma[5]=0;
 prma[6]=0;
 prma[7]=0;
 prma[8]=1;

 for(i=9;i<13682;i++)
   prma[i]=( (2*prma[i-2])%3+2*prma[i-4]%3+(2*prma[i-9])%3 )%3;

 for (i=0;i<13682; i++)
    printf("%d ",prma[i]);
 getch();
 }
 */
```

Appendix 5

A5.1 Overview of Chinese Remainder Theorem

According to the Chinese Remainder Theorem [A5.1] for integers, a given x, $0 \le X < M$, X can be expressed as

$$X \equiv \sum_{i=1}^{k} x_i L_i M_i \pmod{M}$$

where x_i is the ith residue integer corresponding to the ith modulus m_i and $0 \le x_i \le m_i$,

$M_i = M/m_i$,

the unique integers L_i , i = 1,2, ..., k are determined previously using

$$L_i M_i \equiv 1 \pmod{m_i}$$

and M is a composite number, it can be uniquely expressed as $M = p_1^{\alpha 1} p_2^{\alpha 2} \ldots p_t^{\alpha n}$ with all distinct prime numbers p_i and positive integer exponents, α_i.

The ring Z(M) can be written as a direct sum of subrings $Z(p_t^{\alpha n})$ under moduli operation, $Z(M) = Z(p_t^{\alpha 1}) \oplus Z(p_t^{\alpha 2}) \oplus \ldots \oplus Z(p_t^{\alpha n})$, for i = 1,2, ..., n.

Similar to the integer, a polynomial rings over a field exists using the Chinese Remainder Theorem for polynomials. If a polynomial defined over a field, $P(y)$, has K relatively pairwise prime factors, $P_1(y)$, $P_2(y)$, ..., $P_K(y)$. Every polynomial $X(y)$ with degree less than the degree of $P(y)$ can equivalently be represented by a residue polynomial vector $x(u)$ having K components $X_1(y)$, $X_2(y)$,..., $X_K(y)$, $X(y) \sim [X_i(y)]^K_{i=1}$ where $X_i(y) \equiv X(y) \pmod{P_i(y)}$. $X_i(y)$ is the *ith* residue polynomial corresponding to $P_i(y)$. Given the vector $X(y) = [X_1(y), X_2(y),...X_K(y)]$, the corresponding polynomial $X(y)$ can be uniquely determined by solving K cogruences.

$$X(y) \equiv \sum^K_{i=1} X_i(y)\, L_i(y)\, P_i'(y) \pmod{P(y)} \quad \text{where } P_i'(y) = P(y)/P_i(y)$$

and $L_i(y)$, $i = 1,2,...K$ are determined uniquely by using $L_i(u)Pi'(y) \equiv 1 \pmod{P_i(y)}$ or expressed as $X(y) \sim X_1(y) \oplus X_2(y) \oplus ... \oplus X_K(y)$.

A5.2 Method For Finding Pseudo-Random Window Index

To illustrate GF(p) containing pseudo-random sequence, we take the set $\beta = \{0, 1, 2, 3, 4\}$ with the operation $\oplus$ and $\odot$ of mod (5), we can see how GF(p) work with the integers

$$3 + 4 = 7 = 2 \bmod (5)$$

$$3 \cdot 4 = 12 = 2 \bmod (5)$$

Both operation of the elements in the GF(p) will produce an element

in GF(p). This satifies the GF(p) operations.

Let us look into the polynomials. For GF($q = p^m$), taking $q = 8$, $p = 2$, $m = 3$, mod (2) we have polynomials

$P_0 = 0$

$P_1 = 1$

$P_2 = x$

$P_3 = x + 1$

$P_4 = x^2$

$P_5 = x^2 + 1$

$P_6 = x^2 + x$

$P_7 = x^2 + x + 1$

The operation of $P_3 \oplus P_6$ will be $x + 1 + x^2 + x = x^2 + 1$, mod (2). $x^2 + 1$ is P_5 which is an element in GF($q = p^m$). If we can find h(x) which is prime to all the polynomials in GF($q = p^m$), then GF($q = p^m$) will become a cyclic group.

Let $h(x) = x^3 + x + 1$. In mod(2), $P_4 \otimes P_7 / h(x)$ mod 2 = x which is P_2. and $P_2 \otimes P_4/h(x)$ mod 2 = x + 1 which is P_3. continuing the operation, the result can be found inside GF($q = p^m$) in cyclic order but not in systematic order such as P_1, then P_2 ... then P_7.

We have already known that a pseudo-random window [5.1] is a polynomial and is the remainder of the array and is denoted by

$r(t) = r_0\, t^{m-1} + r_1\, t^{m-2} + \ldots + r_{m-1}$ where m is the power of the GF(p)

Let t^i be the pseudo-random polynomial.

$t^i = h(t)\, q(t) + r(t)$ where i has maxium value $p^{n-1} - m$.

h(t) has degree of m with coefficients from GF(p);
q(t) has degree of n-m with coefficients from GF(p);
r(t) has degree of m-1 with coefficients from GF(p).

Since r(t) is known from the pseudo-random window, h(t) is also known from the generating polynomials for the array, the degree of q(t) needs to be found.

If the polynomials are expended,

$$h(t) = h_0 t^m + h_1 t^{m-1} + \ldots + h_m$$
$$q(t) = q_0 t^{n-m} + q_1\, t^{n-m+1} + \ldots + q_{n-m} \qquad \text{.....eq. (A5.1)}$$
$$r(t) = r_0 t^{m-1} + r_1 t^{m-2} + \ldots + r_{m-1}$$

Subtituting these equation into equation A5.1, we have

$t^i = h_0 q_0 t^i + h_1 q_0 + h_0 q_1 t^{n-1} + \ldots$

Only the coefficient with t^i term is equal to unity but the rest is zero. We can sseatablish system equations to find the value of i.

To demonstrate the method, let us look into an example below;

If $GF(2^3)$, $p = 2$, $m = 3$, $q = p^m = 8$ and $m = p^m - 1$

The sequence wiht length of $p^m - 1 = 7$, is expressed as
0 0 1 0 1 1 1

$r(t) = t^2 + 1$
$h(x) = x^3 + x + 1$ or $h(t) = t^3 + t + 1$, the generating polynomial.

To decode the pseudo-random window, it is equivalent to find the index of the window of the sequence. As we know the element of a Galois field, it can be expressed as a polynomial or as a power of a generating element (a root of the generating polynomial).

$q(t) = q_0 t^4 + q_1 t^3 + q_2 t^2 + q_3 t + q_4$

Therefore
$h(t)\ q(t) + r(t)$

$= (t^3 + t + 1)(q_0 t^4 + q_1 t^3 + q_2 t^2 + q_3 t + q_4) + (t^2 + 1)$

$$= q_0\ t^7 + q_1\ t^6 + (q_0+q_2)t^5 + (q_0+q_1+q_3)t^4 + (q_1+q_2+q_4)t^3 + (q_2+q_3+1)\ t^2 + (q_3 + q_4)\ t + (q_4 +1)$$

or

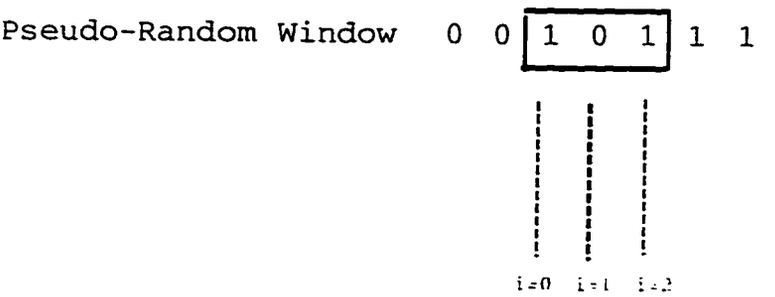
$$t^i = q_0t^7 + q_1t^6 + (q_0+q_2)t^5 + (q_0+q_1+q_3)t^4 + (q_1+q_2+q_4)t^3 + (q_2+q_3+1)\ t^2 + (q_3 + q_4)\ t + (q_4 +1)$$

From the polynomial h(t) q(t) + r(t), should be by definition = t^i, that is the only term in "t". This will result in only one "q" coefficient in the polynomial that will have a "1" and the rest is "0". The system of equations will be solved by trial and error for all possible values of "i", hence the index of the window will be found.

There will be p^m equations for (p^m - 2) unknowns. We can list them as follows:

t^7	q_0
t^6	q_1
t^5	$q_0 + q_2$
t^4	$q_0 + q_1 + q_3$
t^3	$q_1 + q_2 + q_4$
t^2	$q_2 + q_3 + 1$
t^1	$q_3 + q_4$
t^0	$q_4 + 1$

By definition, only t^7 has a coefficient of one while the rest is zero. We start the iteration of i = 0, 1, 2.. We find i =2 will satisfy the sets of equations. In the following diagram will show the window and the sequence.



A5.3 Algorithm for Identifying the Location of A Pseudo-Random Window

Steps to find a Pseudo-Random Window Index:

(1) From the window, we form a reminder polynomial, r(t).

(2) We get the generating polynomial of the Pseudo-Random Array, h(t). [5.1]

(3) Formulate the unknown gquotient polynomial, q(t) or the root of the generating polynomial, h(t).

(4) Expand h(t)q(t) + r(t) into a polynomial g(t) with i degree.

(5) Formulate the coefficient of the g(t) as follows:

(a) The coefficient of the highest degree will be set to "1"

(b) The coefficients of the degree less than i will be set to "0".

(6) Start to solve the set of equations with iteration i = 0, 1, 2, ... n, till we find a value which will satisfy $g(t) = t^i$.

References

Chapter 1

[1.1] L.D. Harmon, *Touch-Sensing Technology: A Review*, SME Report MSR8O-83, 1980.

[1.2] K.E. Pennywitt, *Robotic Tactile Sensing, Byte*, Jan. 1986, pp.177-200.

[1.3] P. Dario et al., *A Sensorised Scenario for Basic Investigation on Active Touch, in Robot Sensors*, (A. Pugh, Editor), Vol. 2, pp. 237-245, IFS (Publication) Ltd. and Springer-Verlag, 1986.

[1.4] Thomas J. Moore., "A survey of the mechanical characteristics of skin and tissue in response to vibratory stimulation," *IEEE Transactions on Man-Machine Systems*, Vol. MMS-11 No. (1): pp79-84, March 1970.

[1.5] M. Ogorek, *Tactile Sensors. Manufacturing Engineering*, Feb. 1985, pp 70-77.

[1.6] *** *LTS-210 Tactile Sensor: Installation and Operation Manual*, Lord Corporation, Industrial Automation Division, Cary, NC, USA.

[1.7] *** *Force Sensing Resistor (FSR) Apolication Notes*, Interlink Electronics, Santa Barbara, CA, USA.

Chapter 2

[2.1] Paul J. Besl; Ramesh C. Jain, "Three-Dimensional Object Recognition," *Computing Surveys*, Vol. 17, N0.1 pp.75-145, March 1985.

[2.2] B. Bhanu, "Representation and shape matching of 3-D objects," *IEEE Trans. Pattern Anal., Machine Intell.* PAMI-6, pp 340-350., 3 May 1984.

[2.3] H. C. Lee and K. S. Fu, "Generating object description for model retrieval," *IEEE Trans Pattern Anal. Machine Intell.* PAMI-5, pp 462-471, 5 Sept, 1983.

[2.4] M. O. Shneier, "Models and strategies for matching in industrial vision", *Computer Science Tech. Rep. TR-1073*, University of Maryland, College Park, Md., July, 1981.

[2.5] E. M. Petriu, N. Trif, **S. K. Yeung**, W. S. McMath and I. Nicolescu, "Model-based object recognition using pseudo-random encoding", *Applied Machine Vision '94, MS94-175*, June 6-9 1994.

[2.6] N. Trif, E. M. Petriu, , **S. K. Yeung**, W. S. McMath, "Model-based Visual recognition of 3-D objects using pseudo-random grid encoding," *Dartment of National Defence, Canada Workshop on Intelligent Robotics and Vehicles, DND Workshop 93* , 1993.

Chapter 3

[3.1] S. C. Jacobsen et al, *The Utah/MIT Dextrous Hand: Work in Progress International Journal of Robotics Research.* Vol 3(4), 1984.

[3.2] K. Pribadi et al, *Exploration and Synamic Shape Estimation by a Robotic Probe, IEEE Trans Syst. Man Cyber.*, vol. 19, no. 4, pp.840-846, July/August 1989.

[3.3] *** *LTS-210 Tactile Sensor: Installation and Operation Manual,* Lord Corporation, Industrial Automation Division, Cary, NC, USA.

[3.4] *** *Force Sensinq Resistor (FSR) Apolication Notes,* Interlink Electronics, Santa Barbara, CA, USA.

[3.5] Patrick Fisher, Ron Daniel, and K.V. Siva, "Specification and design of input devices for teleoperation," *IEEE International Conference on Robotics and Automation,* pages 540-545, 1990, Cincinnati, Ohio.

[3.6] William R. Gould, Charles J. Vierck, and Mary Margaret Luck, "Cues supporting recognition of the orientation or direction of movement of tactile stimuli," Dan R. Kenshalo, editor, *Sensory Functions of the Skin of Humans,* pages 63-78, Florida State University, Tallahassee, Florida, June 1978. Proceedings of the Second International Symposium on Skin Senses, Plenum Press.

[3.7] Robert J. Gregor, "Neurology, kinesthesia, and servomotor control," Phillip /J. Rasch, editor, *Kinesiology and Applied Anatomy,* Chapter 5, pages 62-77, Lea & Febiger, 1989.

[3.8] *** *Astek FS6-120a-200 6-Axis Force Sensor Hardware & Software Orieation Manual,* Barry Wright Corp, Watertown, MA., USA.

[3.9] G. Gleason and G. J. Agin, "A modular vision system for sensor-controlled manipulation and inspection," in *Proc. 9th ISIR,* Washington, DC, Mar. 1979, pp. 57-70.

[3.10] M. Briot, M. Renaud, and Z. Stojikovic, "An approach to spatial pattern recognition of solid objects," *IEEE Trans. Syst., Man. Cybern.*, vol. SMC-8, pp.690-694, Sept. 1978.

[3.11] V. Marik, "Algorithms of the complex tactile information processing," in *Proc. 7th Int. Joint Conf. Artificial Intell.*, 1981, pp. 773-774.

[3.12] Z. Stojikovic and D. Saletic, "Learning to recognize patterns by Belgrade hand prosthesis,"

in *Proc. 5th ISIR*, IIT Res., Inst., Chicago, IL, 1975, pp. 407-413.

[3.13] G. Kinoshita, S. Aida, and M. Mori, "A pattern classification by dynamic tactile sense information processing," *Pattern Recognition*, vol. 7, pp. 243, 1975.

[3.14] C. J. Page, A. Pugh, and W.B. Heginbotham, "Novel techniques for tactile sensing in a three-dimensional environment," in *Proc. 6th ISIR*, Univ. Nottingham, Mar. 1976.

[3.15] S. Takeda, "Study of artificial tactile sensors for shape recognition: Algorithm for tactile datainput," in *Proc. 4th ISIR*, 1974, pp. 199-208.

[3.16] H. Ozaki, S. Waku, A. Mohri, and M. Takata, "Pattern recognition of a grasped object by unit-vector distribution," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-12, no. 3,pp.315-324, May/June 1982.

[3.17] W.E.L. Grimson, *From Images to Surfaces: A Computational Study of the Human Early Vision System.*, Cambridge, MA: MIT Press, 1981.

[3.18] G. Gleason and G.J. Agin, "A modular vision system for sensor-controlled manipulation and inspection." in *Proc. 9th ISIR*, Washington, DC, Mar. 1979, pp. 57-70.

Chapter 4

[4.1] R. J. Schilling, *Fundamentals of Robotics: Analysis and Contro*l, New York, Prentice-Hall, 1990.

[4.2] E. M. Petriu, **S. K. Yeung**, W. S. Math and N. Trif, "*Active Tactile Perception of Object Surface Geometric Profiles*" IEEE Transactions on Instru and Meas., Vol 41 No1, Feb 1992.

[4.3] T.H. Speeter, "A Tactile Sensing System for Robotic Manipulation," *The Int. Journal of Robotics Research*, vol. 9 no. 6, pp. 25-36, 1990.

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

[4.5] D.M. Siegel, S.M. Drucker, and I. Garabieta, "Performance Analysis of a Tactile Sensor," *Proc. 1987 IEEE Int. Conf. Robotics and Automat.*, pp. 1493-1499, 1987.

[4.6] **S. K. Yeung**, E. M. Petriu, W. S. McMath and D. C. Petriu, "High Sampling Resolution Tactile Sensor for Object Recognition" *Trans. on Instru and Meas.*, Vol 43, No. 2, April, 1994.

[4.7] E. M. Petriu, N. Trif, **S. K. Yeung**, W. S. McMath and I. Nicolescu, "Model-based object recognition using pseudo-random encoding", *Applied Machine Vision '94*, MS94-175, June 6-9 1994.

[4.8] R.Q. Yang and M.W. Siegel, "A Finger-tip Optical Sensor Array Sorting System," *Proc. Sensors '85 Conf., CASA/SME Technical Paper MS85-991*, pp. MS85-991-11, 1985.

[4.9] A.W. De Groot, "Effect of Sensor Size in Robotic Tactile Sensor Arrays," *Robotica*, vol. 6, pp. 285-287, 1988.

[4.10] R. A. Russell, "Tactile Sensing of 3-Dimensional Surface Features," *Robotics (1990)*, vol.8, pp.111-115.

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

[4.12] A. K. Jain, *Fundamentals of Digital Image Processing*, New York: Prentice Hall, 1989.

[4.13] C. Muthukrishnan, D. Smith, D. Myers, J. Rebman, and A. Koivo, "Edge detection in tactile images," in *Proc. 1987 IEEE Int. Conf. Robotics and Automat*, pp. 1500-1505.

[4.14] A. A. Berger and P. K. Khosla, "Using tactile data for real-time feedback," *Int. J. Robotics Res.*, vol 10, no.2 pp. 88-102, 1991.

[4.15] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision, and Intelligence*, New York: McGraw-Hill, 1987, ch. 7.

[4.16] A. A. Berger and P. K. Khosla, "The modified adaptive Hough transform (MAHT)," *J. Robotic Syst.*,vol. 7, no. 2, pp. 277-290, 1990.

[4.17] F. Ali, T. Pavlidis, "A hierarchical syntactic shape analyzer," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 1, no. 1, pp. 2-9, 1979.

[4.18] H. Khalfallah, E. M. Petriu, and F. C. A. Groen, "Visual poition recovery for an auttomated guided vehicle," *IEEE Trans.Instruum.Meas.*,vol41, no. 6, pp.906-910, 1992.

Chapter 5

[5.1] S. W. Golomb, *Shift Register Sequences*, Holden-Day, Inc., San Francisco, 1967.

[5.2] F. J. Macwilliams and N. J. A, Sloane, "Pseudorandom Sequences and Arrays", *Proc. IEEE*, Vol. 64, No.12, pp.1715-1729, Dec. 1976.

[5.3] D. V. Sarwate and M. B. Pursley, "Cross-correlation Properties of Pseudo-Random and Related Sequences", *Proc. IEEE*, Vol. 68, No.5, pp.593-619, May 1980

[5.4] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, Ch. 14, North Holland, Nwe-York, 1992.

[5.5] S. W. Golomb, (Ed.), *Digital Communications with Space Applications*, Prentice-Hall, Englewood Cliffs, N.J., 1964.

[5.6] M. B. Pursley and H. F. Roefs, "*Numerical Evaluation of Correlation Parameters for Optimal Phases of Binary Shift-Register Sequences*," IEEE Trans. Commun., Vol. COM-27, pp. 1597-1604, 1979.

[5.7] U. Cheng, "Performance of A Class of Parallel Spread-Spectrum Code Acquisition Schemes in the Presence of Data Modulation", *IEEE Trans. Commun.*, Vol. 36, No. 5, pp. 596-604, 1988.

[5.8] Yong-Hwan Lee and Sawasd Tantaratana, "Sequential Acquisition of PN Sequences for DS/SS Communication, Design and Performance", *IEEE Journal on Selected Areas in Communications*, Vol. 10, No. 4, pp. 750-759, 1982.

[5.9] P. Vijay Kumar and Victor K. Wei, "Minimum Distance of Logarithmic and Fractional Partial m-Sequences", *IEEE Inform. Theory*, Vol. 38, No. 5, pp. 1474-1482, 1992.

[5.10] K. K. Chawls and D. V. Sorwate, "Acquisition of PN Sequences in Chip Synchronous DS/SS System Using A Random Sequence Model and The SPRT", *IEEE Trans. Commun.*, Vol. 42, No. 6, pp. 2325-2334, 1994.

[5.11] J. Li and S. Tantaratans, "Optimal and Suboptimal Coherent Acquisition Schemes for PN Sequences with Data Modulation", IEEE Trans. Commun., Vol. 43, No.2/3/4, pp 554-564, 1995.

[5.12] Y. Funahashi and K. Nakamura, "Estimation of Discrete-time Systems Using Pseudorandom Sequences", *Electronics Letters*, Vol. 9, No. 10, pp. 208-209, 1973.

[5.13] Ta-Hsin Li, "Blind Identification and Deconvolution of Linear Systems Driven By Binary

Random Sequences", *IEEE Trans. Inform. Theory*, Vol. 38, No. 1, pp. 26-38, 1992.

[5.14] M. C. Greest and M. O. J. Hawksford, "Nonlinear Distortion Analysis Using Maximum-Length Sequences", *Electronics Letters*, Vol. 30, No. 13, pp. 1033-1035, 1994.

[5.15] B. Arazi, "Position Recovery Using Binary Sequences", *Electronics Letters*, Vol. 20, No. 2, pp. 61-62, 1984.

[5.16] E. Petriu, "Absolute-Type Pseudo-Random Shaft Encoder with Any Desired Resolution", *Electronics Letters*, Vol. 21, No. 5, pp. 215-216, 1985.

[5.17] G. H. Tomlinson, "Absolute-Type Shaft Encoder Using Shift Register Sequences", *Electronics Letters*, Vol. 23, No. 8, pp. 398-400, 1987.

[5.18] E. Petriu, "Absolute Type Position Transducers Using a Pseudorandom Encoding", *IEEE Trans Instrum. Meas.*, Vol. IM-36, No. 4, pp. 950-955, 1987.

[5.19] E. M. Petriu, "New pseudorandom/natural code conversion method," *Inst. Elec. Eng. Electron. Lett.*, vol. 24 no.22, pp1358-1359, 1988.

[5.20] Petriu E.M., Basran J.S., Groen F.C.A., "Automated Guided Vehicle Position Recovery," *IEEE Trans Instrum Meas.*, vol. 39, no. 1, pp. 245-258, 1990.

[5.21] E. Petriu, W. S. McMaath, S. K. Yeung, N. Trif, T. NBieseman, "Two-Dimensional Position Recovery for a Free-Ranging Automated Guided Vehicle", IEEE Trans. Instrum. Meas., Vol. 42, No. 3, pp. 701-706, 1993.

[5.22] M. Arsic and D. Denic, "New Pseudorandom Code Reading Method Applied to Position Encoders", *Electronics Letters*, Vol. 29, No. 10, pp. 893-894, 1993.

[5.23] E. M. Petriu, N. Trif, W. S. McMath, **S. K. Yeung**, "Automated Guided Vehicle Position Recovery using Pseudo-random Grid encoding", *IAS-3: Intelligent Autonomous Systems Conference*, Pittsburgh, PA., Feb. 1993.

[5.24] S. K. Yeung, W. S. McMath, E. Petriu, N. Trif, "Three Dimensional Object Recognition Using Integrated Robotic Vision and Tactile Sensing", *Proc. IEEE&RSJ Int. Workshop on Intell. Robots and Systems IROS'91*, pp. 1370-1373, Osaka, Japan, 1991.

[5.25] S. K. Yeung, W. S. McMath, D. S. Necsulescu, E. Petriu, "Sensing and Modelling Concepts for Autonomous and Remote Mobile Robot Control", *in Missions Technol. and Design of Planetary Mobile Vehicles*, pp. 473-482, Centre National d'Etudes Spatiales, Toulouse, France, 1992.

[5.26] E. Petriu, N. Trif, S. K. Yeung, W. S. McMath, I. Nicolescu, "Model-Based Object

Recognition Using Pseudo-Random Encoding" (SME paper #MS94-175), *Applied Machine Vision '94 Conf.*, pp. 94.175/1-94.175/16, Minnesota, 1994.

[5.27] Senya Kiyasn, Hiroshi Hoshino, Koichi Yano, and Sadao Fujimura, "Measurement of the 3-D Shape of Specular Polyhedrons Using an M-Array Coded Light Source", *IEEE Trans Instru. Meas.*, Vol. 44, No.3, pp. 775-778, 1995.

[5.28] E. Petriu, D. Ionescu, D.C. Petriu, S.K. Yeung, Ph. Lavoie, N. Trif, "Absolute Position Measurement Applications of Pseudo-Random Encoding " *Proc. ETIM'96 IEEE Intl. Workshop on Emergent Technol. for Instrum. Meas.*, pp. 119-126, Como, Italy, 1996.

[5.29] S.K. Yeung, W.S. McMath, E. Petriu, S.J. Pilon, "Model-Based Tactile Object Recognition," *Proc. IMTC/94, IEEE Instrum. Meas. Technol. Conf.*, pp. 1349-1352, Hamamatsu, Japan, 1994.

[5.30] Wil J. vanGils, "Two-dimensional Dot Codes for Product Identification," *IEEE Trans Inform.Theory*, vol. IT-33, No. 5, pp. 620-631, 1986.

[5.31] Ramon Pallas-Areng and John G. Webster, "Sensors and Signal Conditioning," *John Wiley & Sons, Inc.*, chapter 8, pp. 309, 1991.

[5.32] D. H. Green, I. S. Taylor, "Irreducible polynomials over composite Galois Fields and their applications in coding techniques", *Proc. IEEE*, vol 121, No. 9, pp.935-939, Sept. 1974.

[5.33] John J. Komo & Maurice S. Lam, "Primitive Polynomials and M-Sequence Over GF (q^m) ," *IEEE Trans Inform. Theory*, vol. 39, No. 2, pp. 643-647, 1992.

Chapter 6

[6.1] **S.K. Yeung**, E.M. Petriu, W.S. McMath, D.C. Petriu, "High Sampling Resolution Tactile Sensor for Object Recognition" *IEEE Special issue on Selected Papers Trans. Instrum. Meas.*, Vol. 43, No.2, pp277-282, April 1994.

[6.2] D. H. Ballard and C. M. Brown, *Computer Vision*. Englewood Cliffs, NJ: Prentice-Hall, 198, ch.9.

Chapter 7

[7.1] W.S. McMath, M.D. Colven, **S.K. Yeung**, E.M. Petriu, "Tactile Pattern Recognition Using Neural Networks", *Proc. IEEE & SICE IECON'93 Conf*, (in press), Hawaii, Nov. 1993.

Appendix 1

[A1.1] A. Iggo, "Cutaneous sensory mechanisms," H. B. Barlow and J. D. Mollon, editors, *The Senses*, Chapter 17, pages 369-408. Cambridge University Press, 1982.

[A1.2] John H. Martin, "Receptor physiology and submodality coding in the somatic sensory system," Eric C. Kandel and James H. Schwartz, editors, *Principles of Neural Science*, Chapter 23, pages 287-300. Elsevier, 1985.

[A1.3] Carl E. Sherrick and Roger W. Cholewiak, "Cutaneous sensitivity," Kenneth R. Boff, Lloyd Kaufman, and James P. Thomas, editors, *Handbook of Perception and Human Performance: Sensory Processes and Perception*, volume 1, Chapter 12. Wiley-Interscience, 1986.

[A1.4] Patrick D. Wall, "Sensory role of impulses travelling in the dorsal columns," *IEEE Transactions on Man-Machine Systems*, Vol. MMS-1 No.1, pp39-44, March 1970.

[A1.5] Robert F. Schmidt, "Somatovisceral sensibility," Robert F. Schmidt, editor, *Fundamentals of Sensory Physiology*, Chapter 3, pages 81-101. Springer Verlag, 1978.

[A1.6] A. Iggo, "Cutaneous sensory systems," M. S. Laverack and D. J. Cosens, editors, *Sense Organs*, Chapter 17, pages 310-327. Blackie, 1982.

[A1.7] Roland S. Johansson, "Tactile afferent units with small and well demarcated receptive fields in the glabrous skin area of the human hand," Dan R. Kenshalo, editor, *Sensory Functions of the Skin of Humans*, pages 129-152, Florida State University, Florida, June 1978. Proceedings of the second International Symposium on Skin Senses, Plenum Press.

[A1.8] Thurston L. Brooks, "Telerobot response requirements" *1990 IEEE Conference on Systems, Man, and Cybernetics*, pp. 113-120, November 1990. Los Angeles, California.

[A1.9] Dan R. Kenshalo, "Tactile sensitivity," Edward C. Carterette and Morton P. Friedman, editors, *Handbook of Perception: Feeling and Hurting (VI-B)*, pp. 30-49. Academic Press, 1978.

[A1.10] Ronald T. Verillo and George A. Geischeider, "Psychophysical measurements of enhancement, suppression, and surface gradient effects in vibrotaction," Dan R. Kenshalo, editor, *Sensory Functions of the Skin of Humans* pages 153-181 , Florida State University, Florida, June 1978. *Proceedings of the second International Symposium* on Skin Senses, Plenum Press.

[A1.11] Ian Darian-Smith and Linda E. Oke, "Peripheral neural representation of the spatial frequency of a grating moving across the monkey's finger pad," *Journal of Physiology*, 309 pp. 117- 133, 1980.

[A1.12] M. A. Srinivasan, J. M. Whitehouse, and R. H. Lamotte, "Tactile detection of slip : surface microgeometry and peripheral neural codes," *Journal of Neurophysiology*, Vol. 63 No.6, pp. 1323-1332, June 1990.

[A1.13] William R. Gould, Charles J. Vierck, and Mary Margaret Luck, "Cues supporting recognition of the orientation or direction of movement of tactile stimuli," Dan R. Kenshalo, editor, *Sensory Functions of the Skin of Humans*, pp. 63-78, Florida State University, Tallahassee, Florida, June 1978. *Proceedings of the Second International Symposium on Skin Senses*, Plenum Press.

[A1.14] B. L. Whitsel, D. A. Dreyer, M. Collins, and M. G. Young, "The coding of direction of tactile stimulus movement : correlative psychophysical and electrophysiological data," Dan R. Kenshalo, editor, *Sensory Functions of the Skin of Humans*, pages 79-107, Florida State University, Ta!lahassee, Florida, June 1978. *Proceedings of the Second International Symposium on Skin Senses*, Plenum Press.

Appendix 2

[A2.1] S. C. Jacobsen et al, *The Utah/MIT Dextrous Hand: Work in Progress International Journal of Robotics Research.* Vol 3(4), 1984.

[A2.2] K. Pribadi et al, *Exploration and Synamic Shape Estimation by a Robotic Probe, IEEE Trans Syst. Man Cyber.*, vol. 19, no. 4, pp.840-846, July/August 1989.

[A2.3] *** *LTS-210 Tactile Sensor: Installation and Operation Manual*, Lord Corporation, Industrial Automation Division, Cary, NC, USA.

[A2.4] *** *Force Sensinq Resistor (FSR) Apolication Notes*, Interlink Electronics, Santa Barbara, CA, USA.

[A2.5] Patrick Fisher, Ron Daniel, and K.V. Siva, "Specification and design of input devices for teleoperation," *IEEE International Conference on Robotics and Automation*, pp. 540-545, 1990. Cincinnati, Ohio.

[A2.6] William R. Gould, Charles J. Vierck, and Mary Margaret Luck, "Cues supporting recognition of the orientation or direction of movement of tactile stimuli," Dan R. Kenshalo, editor, *Sensory Functions of the Skin of Humans*, pages 63-78, Florida State University, Tallahassee, Florida, June 1978. *Proceedings of the Second International Symposium on Skin Senses*, Plenum Press.

[A2.7] Robert J. Gregor, "Neurology, kinesthesia, and servomotor control," Phillip /J. Rasch, editor, *Kinesiology and Applied Anatomy*, Chapter 5, pp. 62-77. Lea & Febiger, 1989.

[A2.8] *** *Astek FS6-120a-200 6-Axis Force Sensor Hardware & Software Orieation Manual,* Barry Wright Corp, Watertown, MA., USA.

[A2.9] *** *7151 Solartron Multimeter Operation Manual,* Schlumberger, USA.

Appendix 3

[A3.1] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognition 13,* pp111, 1981.

Appendix 5

[A5.1] H. J. Nussabaumer, *Fast Fourier Transform and Convolution Algorithms,* 2nd Ed. New York: Springer-Veriag, 1982.