A Multi-Level Invariant Representation of Digital Images

by

Ismael Mohmed Limalia

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

May, 1991

A MULTI - LEVEL INVARIANT REPRESENTATION OF DIGITAL IMAGES

BY

ISMAEL MOHMED LIMALIA

A Thesis Presented to the

FACULTY OF THE COLLEGE OF GRADUATE STUDIES

KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

DHAHRAN, SAUDI ARABIA

In Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

In

COMPUTER SCIENCE

MAY 1991

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS
DHAHRAN 31261, SAUDI ARABIA

COLLEGE OF GRADUATE STUDIES

This thesis, written by LIMALIA ISMAEL MOHMED under the direction of his Thesis Advisor and approved by his Thesis Committee, has been presented to and accepted by the Dean of the College of Graduate studies, in partial fulfillment of the requirements for the degree of MASTER OF SCIENCE in INFORMATION AND COMPUTER SCIENCE.

SPCC
A
1
LSS
C-2
1069325/1069369

THESIS COMMITTEE

[illegible]
M. AL-MOUHAMED 23/4/1991
Thesis Advisor

[illegible]
Bassel Arafeh 29/4/91
Member

[illegible]
Tarik Ozkul 4/5/91
Member

Muhammad Shafiq
MUHAMMAD SHAFIQUE 1/5/91
Member

[illegible]
Department Chairman

[illegible]
Dean, College of Graduate Studies

11-5-91
Date

TO WHOM I CHERISH INFINITELY AND MORE

ACKNOWLEDGMENT

Acknowledgment is due to King Fahd University of Petroleum and Minerals for providing the opportunity to carry out this research work. and to the ICS Department and its chairman Dr. M. Al-Tayyeb for his patient support.

I gratefully acknowledge the guidance of Dr. M. Al-Mouhamed who served as my committee chairman and the help of the thesis committee members Dr. B. Arafeh, Dr. T. Ozkul and Dr. M. Shafique.

Lastly but not least let me mention my sincere thanks and appreciation to the friends who helped and encouraged me throughout : Ali, Adel, Billal, Husni, Ismail, Jaweed, Khalid, Mahmood, Naji, Nassir, Nazzal & Nazzal, Purushothaman, Rashed, Said, Salah, Salek, Sam & Sam, Shaikhnag, Shawkat, Yahya and Yahya ; they have been a constant source of support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

THESIS ABSTRACT

NAME OF STUDENT: LIMALIA ISMAEL MOHMED

TITLE OF STUDY : MULTI_LEVEL INVARIANT REPRESENTATION OF DIGITAL IMAGES

MAJOR FIELD : INFORMATION AND COMPUTER SCIENCE

DATE OF DEGREE : MAY 1991

Recognizing a digital image from an object dictionary is a slow and memory consuming process. In this work the process was accelerated by using a compact geometrical model and a multi-level representation in conjunction with a Database.

An invariant representation of the image is used which is constant under object's position, orientation or size. The Recognition system is based on a hierarchical level to accelerate the Identification process; one level divides the image into a Coarse and Detailed model. The Coarse one generates the overall feature Descriptors used to minimize identification of unknown image through clustering, while the Detailed model is used to precisely store the image such that an accurate template match is made possible.

Another level of hierarchy is achieved by grouping objects of similar silhouettes into classes; once an unknown object is matched to a known class, it is further matched to a specific object within that class depending on some delineating physical dimension.

MASTER OF SCIENCE DEGREE

KING FAHD UNIVERSITY OF PETROLEUM AND MINERALS

Dhahran, Saudi Arabia

May 1991

خلاصة الرسالة

اسم الطالـــب : اسماعيل محمد ليماليا .
عنوان الرسالـة : التمثيل الثابت المتعدد المستويات للصور الرقمية .
التخصــــص : علوم الحاسب الآلي والمعلومات .
تاريخ الشهادة : ايار (مايو) ١٩٩١ م .

ان عملية التعرف على صورة رقمية من خلال قاموس للصور هي عملية بطيئة ومستهلكة لذاكرة الحاسب . وفي هذا البحث تم تسريع العملية باستخدام نموذج هندسي وتمثيل متعدد المستويات اضافة الى قاعدة بيانات .

وقد تم استخدام تمثيل ثابت للصورة بحيث نفرض ثبات الصورة واتجاها وحجها . وبني نظام التعرف على الصور على اساس المستويات الشجرية لتسريع عملية التعرف .

وينقسم احد المستويات الى قسمين ، احدهما يستخدم لتقليل الوقت اللازم للتعرف علي الصور غير المعروفة م خلال التجربة والاخر يستخدم لتخزين الصورة بطريقة دقيقة بحيث يمكن التعرف عليها لاحقًا .

ويمثل احد المستويات الشجرية مجمعًا للصور المتشابهة في اشكل الخارج في صفوف مختلفة بحيث انه اذا تم التعرف على الصورة في صف ما فانه يتم البحث عنها خلال الصف بتفصيل اكثر .

درجة الماجستير في العلوم
جامعة الملك فهد للبترول والمعادن
الظهران - المملكة العربية السعودية
ايار (مايو) ١٩٩١ م

CHAPTER 1

INTRODUCTION

Image Recognition is experiencing fast progress in the world of Automated Assembly in situations where parts to be handled appear in an unpredictable sequence. For the robot to execute a correct action, the part must be identified by ' non - contact ' measurements : i.e. through a camera. The task of part identification is rendered more complicated by some factors, namely that the part's orientation or position or size is not exactly available, the whole process should be done in real time and that the underlying model used by the algorithms represents the part such as to make available positional and geometrical parameters to the manipulator.

This thesis will attempt to show that the Recognition System can be implemented on a hierarchical level to acccelerate the identification process.

One kind of hierarchy will be modelling the object's shape on two levels, one Coarse, the other Detailed; the first is used to extract the overall features of an object and helps in limiting the search of an unknown object to a subset of the object dictionary while the latter stores the shape in a precise manner and is used to ascertain absolutely that an unknown object is identified.

The second kind of hierarchy implemented is dividing the objects into classes, where all objects with the same contour belong to one class irrespective of their sizes.

The object dictionary will be kept in a DataBase, which will also be implemented as a hierarchy of relations. The master files will contain the

Descriptor array of objects (features) and objects within classes and the Detailed Model. The indexed files will be sorted on each feature such that isolating all classes satisfying the features of an unknown object is achieved in a direct access fashion.

Chapter 2 will review the Image Recognition field, the requirements of such a system, the subprocesses involved, an analysis of the different methods in use today, the suitability of these different techniques and finally the motivations for tapering down to one particular method.
Chapter 3 presents the general overview of this research, which method of image segmentation is used, which model (Polar) is used, the needs for hierarchical models, how a Descriptor array is used to accelerate the Recognition phase in conjunction with a DataBase.
Chapter 4 explains the implementation of all the Image Acquisition modules, all programmed at Assembly level, the process converting a grey level image to a binary one and further to a chain link of the contour. Some utilities for saving and loading images are also given.
Chapter 5 describes the invariant Polar representations used for modelling objects, how the contour of an object is segmented to obtain the polygonal representation, how two hierarchies of representation are extracted and finally how the Descriptor array is found as well as normalizing the models to make them truely invariant. All the modules implemented were in Pascal.
Chapter 6 describes the DataBase, its interface with the other Pascal modules, its organization to allow efficient and fast retrieval for the Recognition module. Some utilities available to the operator during the Learning stage is presented.

The insertion of objects to the DataBase is explained, the techniques used in the Recognition phase, namely decision searches, clustering class and minimum distance test is explained in details.
Chapter 7 lists out the results of all the algorithms implemented, how far the different stages were successful in meeting the requirements, the statistics of how long each step took in the experimental testing of the system.
Finally, in Chapter 8, a summary of the results is presented as a conclusion and some improvements which can be carried out as future work is proposed.

CHAPTER 2

IMAGE PROCESSING AND RECOGNITION : LITERATURE REVIEW.

2.1 Introduction

Over the past decades, much of the work on image processing has dealt with the analysis and interpretation of images but were restrained to images which needed only storage and processing of the raw image data [ROS84]. The only processing needed for such applications like microscopy, radiology, satellite reconnaissance involves obtaining digitized images, cleaning up the image from random noises, increasing contrast and enhancing the edges to delineate parts of the image from the background and finally analysis of color, shading or texture.Most of the algorithms developed in these areas are still useful.

With the advent of document processing and character recognition, the raw image data had to be processed further to obtain clear-cut edges of objects of interest in the image and representing those linked edges in some mathematical form for the purposes of recognition [ROS84].

More recently, automatic assembly of parts in industries have prompted the development of the field known variously as computer vision, image recognition or robot vision. Demands on automation have begun to change. One important tool for flexible automation is the visual sensor which can determine the position and orientation of workpieces [RUM84].

A robot that can 'see' and 'feel' should be easier to train in the performance of complex tasks while at the same time should require less stringent control mechanisms than those required by preprogrammed machines [GON83]. The latter refer to the 'dumb' non-autonomous robots widely used in industrial inspection and assembly whose tasks are fixed and limited; sensory trainable systems are thus more adaptable to a much larger variety of tasks.

In general, robot vision is the collection of techniques, software and hardware required for the location, recognition and manipulation of objects [TIO82]. The algorithms to solve visual inspection problems can be classified as one of two categories : pattern recognition and scene analysis [SUE86]. The latter is usually applied to complex scenes of 3-dimensional nature and is not the subject of this thesis. Pattern or image recognition assumes that a 2-dimensional or 3-dimensional scene or image can be represented as a set of numbers, each number representing one feature of the object. This representation is a structural one in that the model reflects the object's geometry and dimensions, invaluable information for robot grasping [RUM84]. Recognition or interpretation of unknown objects is then performed by calculating its characteristic features and assigning it to one of the known objects with similar features [SUE86].

2.2 Robot Vision Setup & Requirements

An efficient implementation of an image recognition system requires a computer vision with special characteristics [SUE86] namely :

1) Part handling system.
2) Image acquisition and storage system.
3) Special hardware for rapid image processing.
4) Communication of vision system with robot's actuators [GON83].

The setup, illustrated in Fig 2.1, shows the essential components of a robot vision system, the details of which are now explained.

2.2.1 The Part Handling System

The part handling system of a feeding system for transportation of objects under the visual sensor (camera), usually through a conveyor belt. Most machine vision systems assume that the part is separated and unobstructed and appears stably on the belt [SUE86] for the visual sensor to operate correctly. The other part of the system is the classification or separation system which sorts the inspected part according to some recognition responses; usually the part is accepted or rejected as bad or rejected for reinspection [HEN89].

2.2.2 The Visual Sensor

The visual sensor is usually a camera of some sort, linear or array, where the optical image of the part is converted to electric signals (video signal) [GON83]. Solid state area cameras (CCD) contain area arrays of regularly spaced photosensitive elements; they are very popular because of their sensitivity over a wide light spectrum, low power consumption, small size and reliability [PAV77].

2.2.3 The Video Processor

The video processor converts the video signal (analog) to digital values (numeric) which represent the image in a numeric format suitable for computer processing. This digital image is usually stored in a frame buffer and some video processors may have more than one frame.

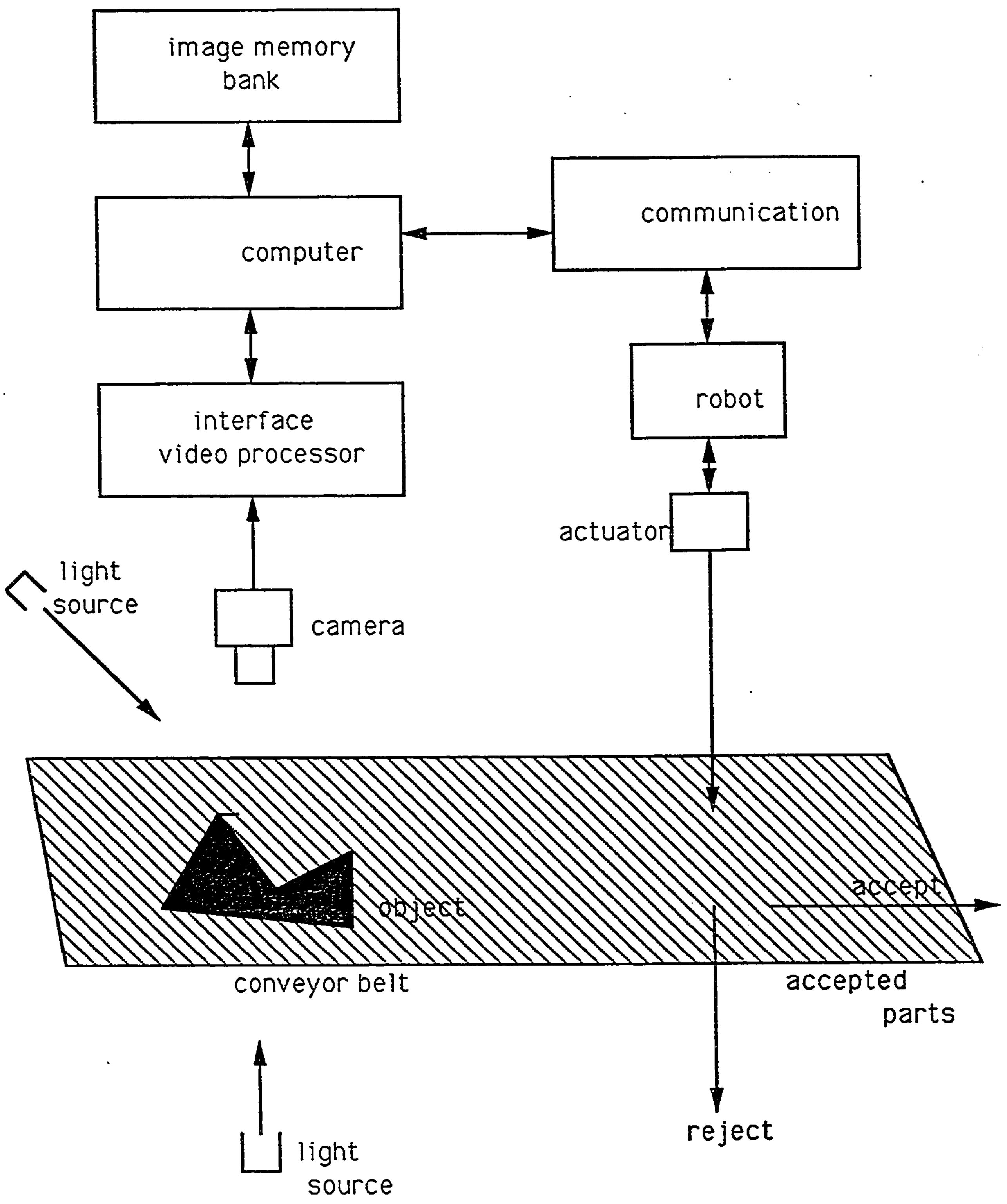


Fig 2.1 : Robot Vision Setup

2.2.4 Lighting Conditions

A lot of applications, which at first glance, require complex vision algorithms, become much simpler by proper lighting and viewing conditions [SUE86]. The importance of good lighting cannot be overemphasized; backlighting, interception of collimated light beams and careful choice of background colors are keys to effective use of computer vision [COR83]. Robotic vision systems use various lighting systems, some very simple to set up , some using complex X-rays or laser lighting; which type of light is to be used or is most effective is highly dependent on the application.

Backlighting, (Fig 2.2-a) where the light source is situated directly under the part being viewed, produces high contrast images of solid object's contour and is the most widely used in industry. The contour can then be easily extracted from the background without the use of computationally costly algorithms.

Sidelighting (Fig 2.2-b) is used for application where defects like cracks and scratches are to be detected; the camera and light sources are directed perpendicular to the object's surface making the cracks appear dark on a bright background [SUE86].

Structured lighting (Fig 2.2-c) refers to a light stripe projected from a long tube onto the part such that the line of lights delineates the part from the background. A linear array camera records this line and the steps are repeated as the part moves on the belt [KLA89]. The image consists of parallel strips corresponding to the part and is reconstructed to obtain a 2-dimensional digitized image.

More specialized lighting such as X-rays are used for specific applications such as when the interior holes of some solid part must be detected [HEn89].

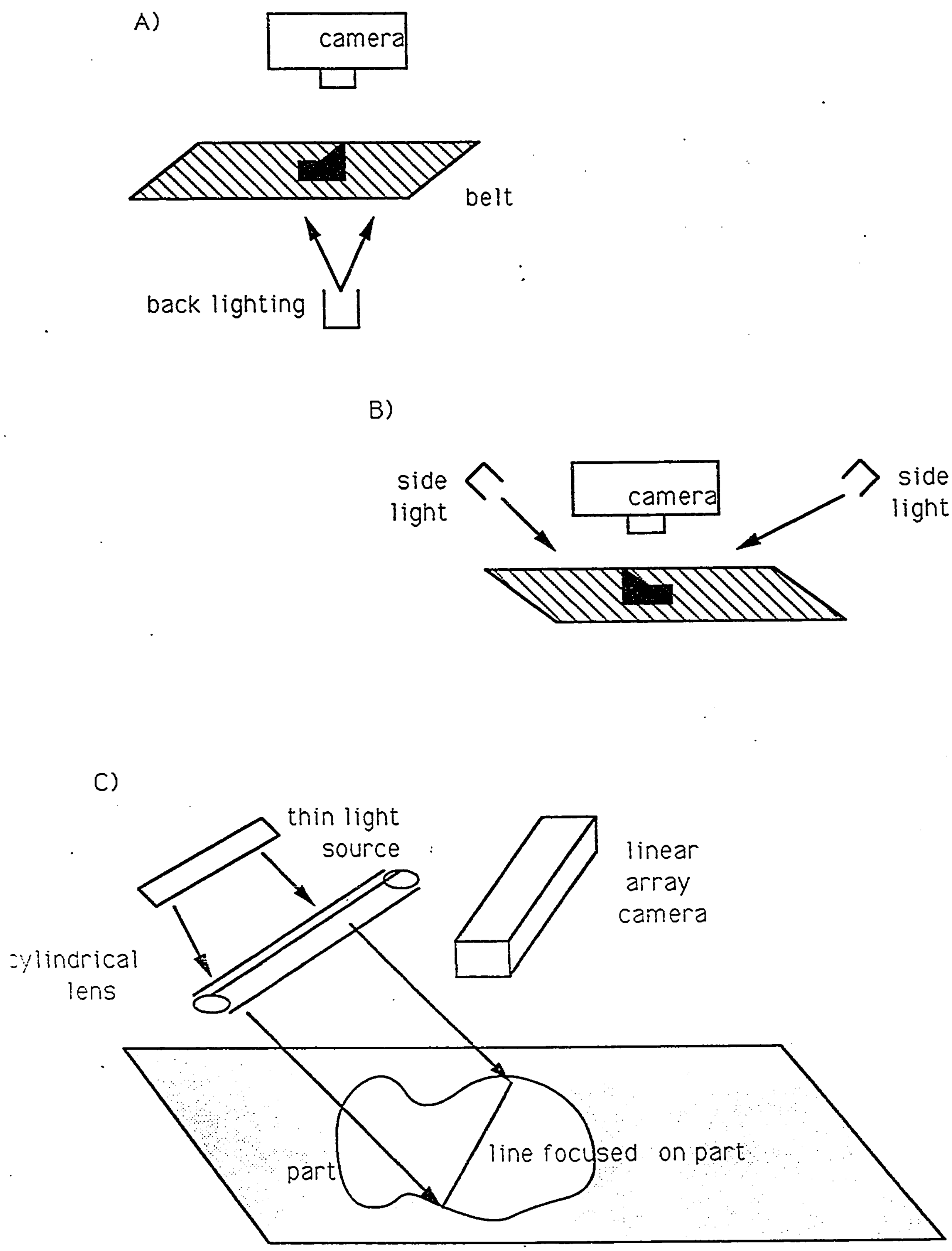


Fig 2.2 : Types of lighting

2.2.5 Robotic Vision Requirements

An image recognition system for use in conjunction with robots imposes some requirements :

a) Image sensing of object and its extraction from the background.

b) Segmentation of the binary image of the object into a model linking the contour edges [GON83]. This structural model should be invariant with respect to translation,

c) Feature extraction which identifies global characteristics of outlined object as n-dimension vector [COR83].

d) Classification / Recognition which could use a notion of minimum distance or a heuristic method of search or some other method where an unknown object is assigned to the closest class of known objects [SUE86].

These requirements will be explained further in the next section. In a general application, a typical sequence of operations required from a robot endowed with vision are :

* Locate an object on conveyor belt.

* Segment and determine object's orientation.

* Recognize object.

* Move manipulator and grasp object at specific point using orientation and recognition information.

* Grasp object and check visually if successful.

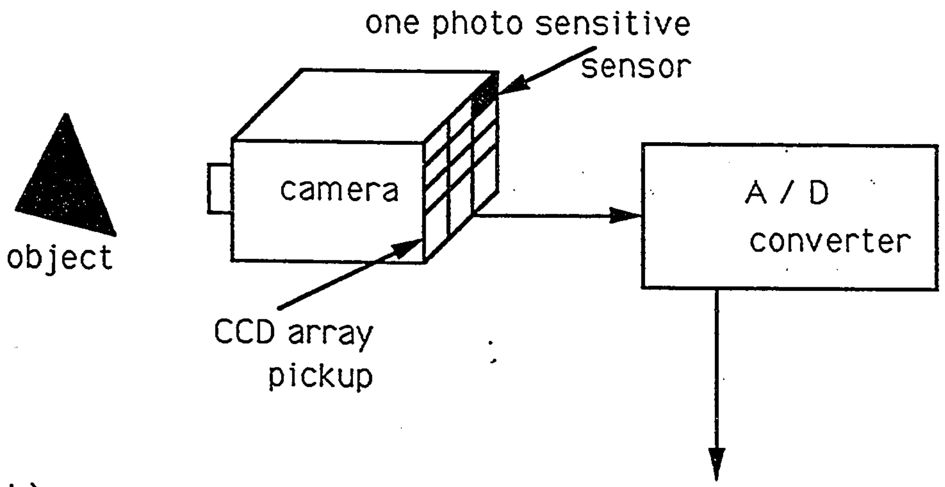
All these typical operations must be done quickly so as to appear 'natural' and the computation speeds must produce responses comparable to those of humans [TIO82]. The computational requirements are not trivial. A robot must already have the ability to perform complex coordinate transformations and straight-line coordinated motions; these are significant computational tasks involving lengthy matrix operations. Adding a vision system operating in real time significantly burdens the computation time and sometimes the only way to solve it is by designing special purpose hardware with built-in sophisticated functionality. The extra processing time due to vision must be no more than a few machine cycles; most robotic applications have cycle times of 1 to 10 seconds and an additional time for vision processing of around 200 ms is considered acceptable [KLA89].

The four requirements : sensing, segmentation, feature extraction and recognition are now discussed in greater details in the following sections.

2.3 Image Segmentation

2.3.1 Image Sensing

In order to process any type of optimal image by a computer, one must quantitize it. The picture can be divided into sufficiently small regions where each area measures the intensity of light through a photosensitive sensor [PAV77]. A CCD array camera will have a finite number of such sensors, the numbers in horizontal and vertical directions being fixed, which measure the light intensity hitting them as an anlaog electric charge (Fig 2.3-a). These charges for each picture area (pixel) must then be converted to a digital value. The size of the image or resolution refers to the number of horizontal and vertical pixels and can range from 64×64 to 512×512 depending on resolution requirements of a given application [GON83]. The higher the resolution, the more detailed is the image and the more the processing times [KLA89].



b)

Gray scale image

20	20	20	25	30	25	30	.
30	32	35	20	40	40	30	.
31	30	29	40	150	50	48	.
32	38	40	175	180	175	55	.
21	20	150	160	190	185	190	.
20	175	185	174	200	190	188	.
.	.	.	.	.	.	.	.

c) Binary image

0	0	0	0	0	0	0	.
0	0	0	0	0	0	0	.
0	0	0	0	1	0	0	.
0	0	0	1	1	1	0	.
0	0	1	1	1	1	1	.
0	1	1	1	1	1	1	.
.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.

Fig 2.3 : Image Acquisition matrix

This array of digitized pixels of an image is referred to as the gray-scale matrix. Each individual pixel can have a range of values e.g. for an 8-bit A/D converter, a pixel value of 0 indicates a very white pixel while a 255 value indicates a very dark region with values in between representing gradations of grey (Fig 2.3-b) This gray-scale image is stored in memory in a frame buffer.

2.3.2 Global Thresholding

Handling the raw information of gray-scale images without further processing, imposes problems in storage and computations [PAV77]. The next logical step which is widely used in any vision system is converting the gray-scale matrix to a binary image [COR83]. where each pixel takes only two values of 0 : white or 1 : black (Fig 2.3-c). This step can be a simple one like global thresholding or a complicated one like gradient thresholding depending on the quality of lighting, the object's color and background color and the presence of noise in the image. Thresholding thus outlines objects in the image for further processing [SUE86].

Global thresholding is suitable if the object to be extracted varies markedly from the background [PAV77] or if the lighting environment is controllable [KLA89] and involves the comparison of each pixel value of gray-scale image, G(i,j) to a global threshold constant, T , and returning a binary value, B(i,j) of 0 or 1 [COR83,PAV77].

$$B(i,j) = \begin{cases} 0 & \text{if } G(i,j) < T \\ 1 & \text{if } G(i,j) >= T \end{cases}$$

The threshold value T can be chosen interactively by an operator or can be set automatically from an analysis of the gray-level histogram; referring to the histogram of Fig 2.4, T can be chosen as any value in the zone A-B or the sharpest break-point zone [WAT84]. Global thresholding is quick, visits each pixel only once but results in a binary image with the whole object, including the inside pixels, as black.

2.3.3 Local Thresholding

Other methods of thresholding set only the pixels on or near the edge of the object to 1 while all the background pixels as well as those in the object's interior are set to 0. The advantage is that less pixels are stored; however the operators involve more arithmetic and the edge is not usually 1-pixel thin.

Edge-detection algorithms fall into 3 categories : [KLA89].

a) First Difference / 1-dimensional

b) Sobel Operators / 2-dimensional

a) Contrast Operators

All edge operators involve comparing a central pixel, E, with its 8 neighbours A, B, C, D, F, G, H, I (Fig 2:5). These 9 pixels, the 3×3 mask, is moved over each pixel in the image one at a time and measures the discontinuity in the image The first difference operators are :

$\text{Edge1} = | I - E | + | E - A |$

$\text{Edge2} = | F - E | + | E - D |$

$\text{Edge3} = | H - E | + | E - B |$

$\text{Edge4} = | G - E | + | E - C |$

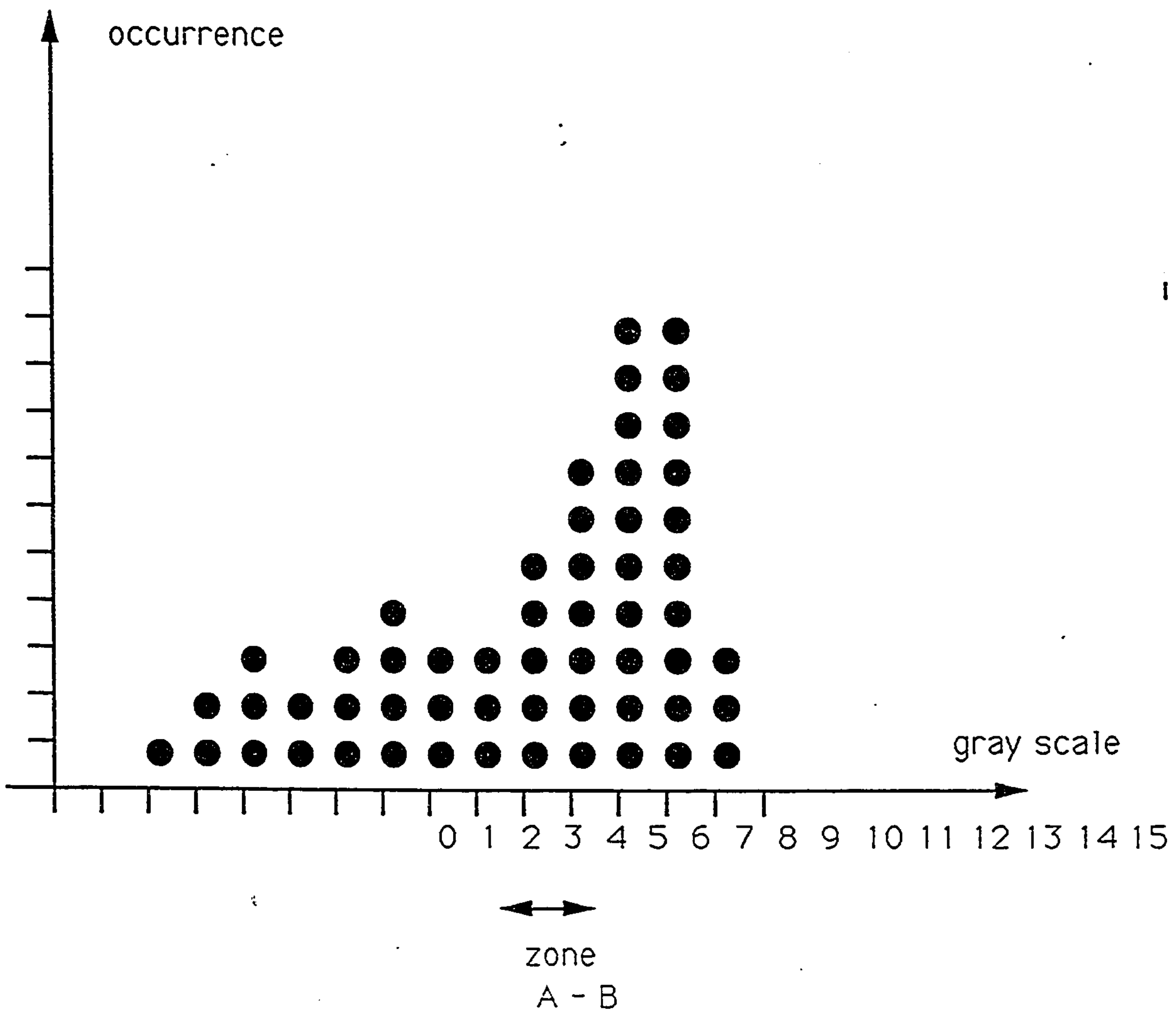


Fig 2.4 : Gray level histogram (4 - bit)

Edgel computes a first difference function in the 'north west - south east' direction. This is most sensitive to edges perpendicular to the direction or orientation of computation and returns a magnitude of edge in only one spatial direction. Similarly the other functions generate similar types of one dimensional edge property [KLA89]. More powerful approaches are based on neighborhood pixels involving a range of sizes e.g. 3×3, 4×4 masks and combining their outputs [ROS84].

The contrast operators is of the following type :

$$\text{Edge} = E - \frac{A+B+C+D+F+G+H+I}{8}$$

This operator compares the absolute intensity of the next neighborhoods only and can be computed sequentially as the image is scanned. The disadvantage is that it characterizes only edge points but not edges; i.e. there is no explicit or implicit connection between the edge points.

The Sobel operators, the most popular of the edge detection schemes and easily implemented, compute two edge measurements perpendicular to each other and generally enhances edges more than other 1-dimensional methods and takes the form of :

Gradients in X- and Y- directions are :

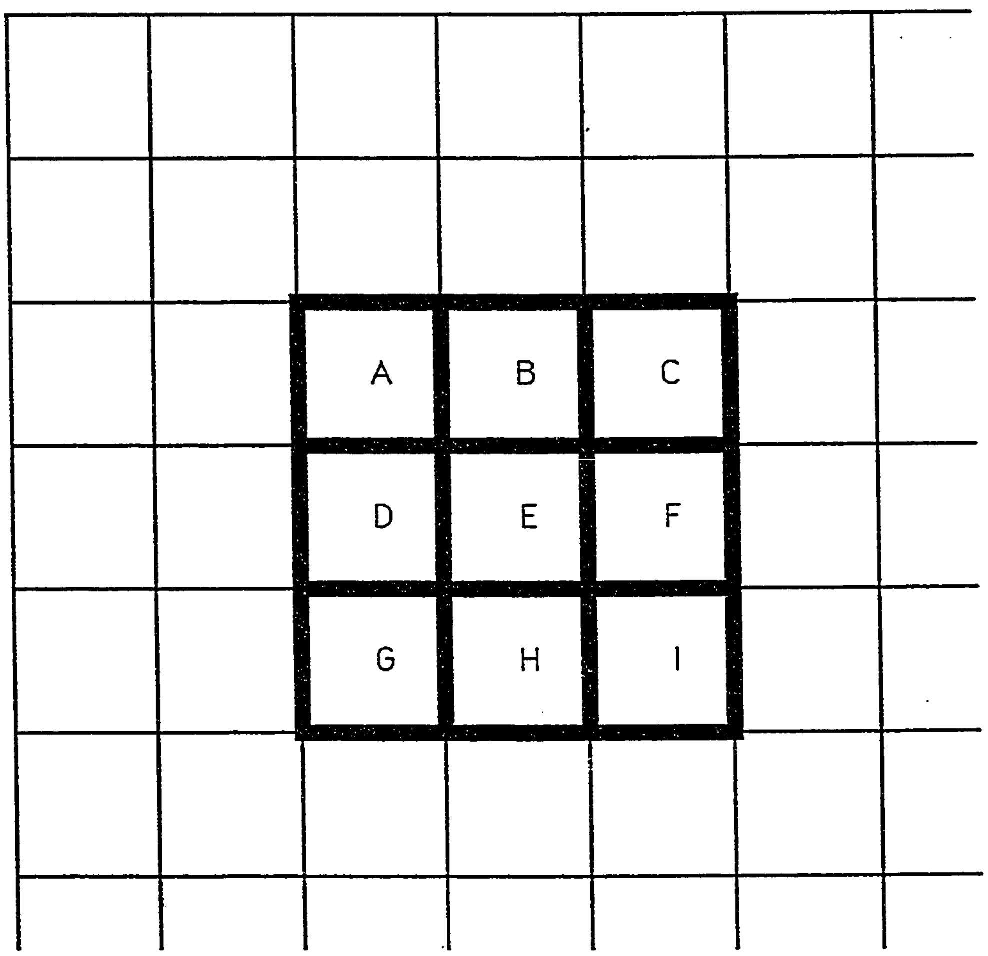
$$G_x = (G + 2H + I) - (A + 2B + C)$$

$$G_y = (C + 2F + I) - (A + 2D + G)$$

The basic Sobel operator masks are :

$$G = | G_x^2 + G_y^2 |^{1/2}$$

Digitized Image Matrix



A, B, C, D, F, G, H, I, : intensity values for neighbor pixels

E : intensity of pixel ubder test

Fig 2.5 : 3 * 3 Convolution mask

The center pixel value E is then replaced by G. A common variation consists of replacing G only if it is greater than some specified threshold [GON83]. In practice the squares and square root in calculating G, being too time consuming, will be eliminated such that

$G = |G_x| + |G_y|$

2.3.4 Contour Following

Once the object has been isolated from the background by any of the previous algorithms, all the edge pixels will be isolated; usually due to lighting conditions and shadows, the edge will be a few pixels thick and need to be trimmed down to a 1-pixel thin contour. The next step is linking the pixels on the contour as a chain whereby starting from one edge pixel the relative direction or coordinates of the next pixel on the edge is found; this is repeated for all pixels on the contour [KLA89].

2.3.5 Feature Extraction

Analysis of an image usually starts at low levels (sensing of gray-scale image) and generates from it lines and edges (segmentation [MAR82] and then converting edges to a list of parameters describing corners, straight lines or circular arcs to get the image data into a reduced and organized form [RUM84] , a form which represents the geometry of the object. Some global features often used are area, perimeter, number of sides, centroid, minimum and maximum radii, bounding box size [SUE86]. Often features relating to shape or measures based on the silhouette are powerful characteristics of an object and are sufficient to identify it in the recognition tasks [COR83], These shape related features form the primitive description and ideally they are invariant under rotation, translation or scaling.

2.4 Non-Structured Image Representation (region based)

Some fields of image processing successfully employ only non-structured representations of images in the recognition process; these representations are most often based on an object as a whole, hence region based, rather than on geometrical measures. Methods like gray-level histograms, pixel template matching or run-length transitions use statistical decision techniques which rely on scalar measurements and all structure of the image is destroyed; they have been implemented on many high speed systems and succeed because the object's contour is fixed in orientation and rotation [COR83] ; their applications are not expandable to situations where no 'a priori' information about the positioning of the object is available [TOU74]. Their very simple data structures make them efficient for some limited fields but in the field of event driven recognition, the amount of computing time required for implementation is too massive to be practical.

2.4.1 Color Coding

Frequently color of an object is useful in separating it from other objects of from a background image and in robotic situations part identification is often based on color. Color coded parts such as electronic components on a board must be recognized or their presence verified and often in assembly, a needed part is identified and differentiated solely on color [COR83].

For color coding to be effective, naturally proper lighting (certain colored lights will pick up only parts of same color) and proper filters in the optical path of the camera (to highlight specific colored parts) are used [KLA89].

2.4.2 Gray-Level Histograms

Gray-level histogram is a statistical method which obtains a 1-dimensional array containing the distribution of gray scale intensities. Fig 2.4 shows such a histogram for a 4-bit gray scale. The image transformation is of a very simple type but is often very useful because of enormous data reduction, typically over 95 % [KLA89]. The matching is a simple process where each element of the distribution array is compared and checked if it falls within a tolerance match; prior to this some parameters such as average gray level or variance spread of distribution are often used.

Although the gray level histogram method is rotation and translation invariant, it is not useful for robot vision because

* the dimensionality of the array in terms of storage and processing is too large to be effective.
* effects of changing light conditions or shadows drastically modifies the histogram and renders recognition difficult.
* all geometrical or structural information about the object is destroyed; this information is extremely helpful for the robot.

2.4.3 Differential Delta Coding

This method of coding is used to store gray scale images more efficiently; the difference between the intensities of a pixel and the previous one is stored and used. In ordinary uncluttered scenes, differences between successive pixels are usually small except near border pixels; so the number of bits required to store the difference in pixel intensities compared to storing their absolute intensities is relatively smaller.

2.4.4 Run-Length Encoding

It is clear from object images that there exists long horizontal strips of pixels of similar gray scale value (Fig 2.6); this is particularly true if the image was acquired from a linear array camera i conjunction with structured line-striped lighting (section 2.2.4). For these cases a separate entry for each pixel is a waste of computer memory. For each line in the image, the line number, transition point of each background to object and string length (number of consecutive pixels of similar value or residing in object) are stored, resulting in a very efficient model in terms of storage. This method is successfully used in VS-100 of Machine Intelligence Corporation [SUE86] and in the GM Consight System, a robotic vision and assembly system still operational today [KLA89 , HOL79] .

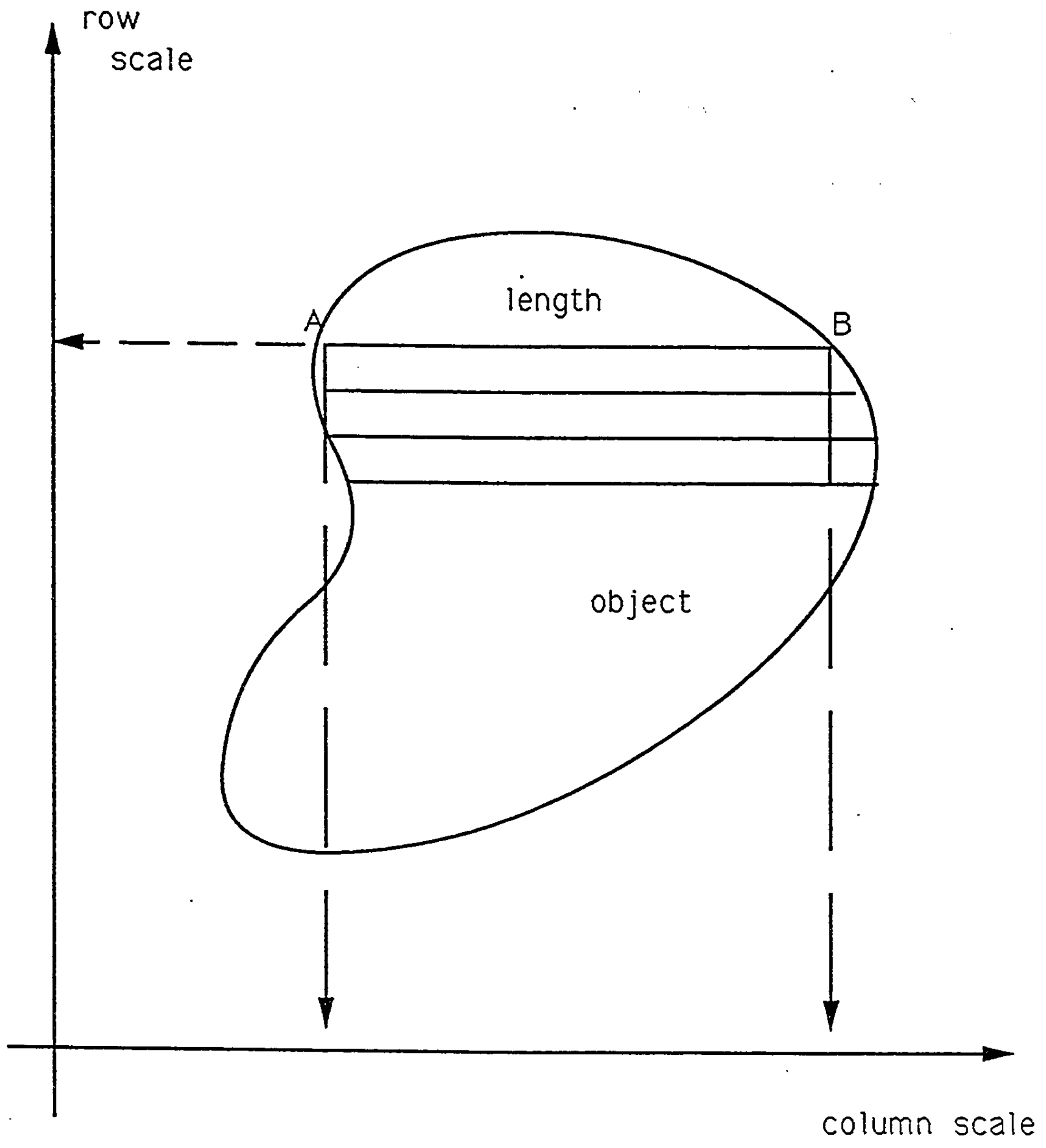


Fig 2.6 : Run Length Encoding

2.4.4 Freeman Chain Coding

The Freeman chain code is not a region based technique but is rather edge based; it is considered here however because as such and without any further processing, it also has the disadvantages of other region based methods. The chain code basically links up all the border pixels on the contour of an object by considering the direction of a next pixel relative to the one considered along any of the 8 possible directions (Fig 2.7-b). [TOU74]. The steps from pixel to next pixel is uni-step and all the pixels on the contour are thus linked until the starting pixel is reached once again (Fig 2.7-c).

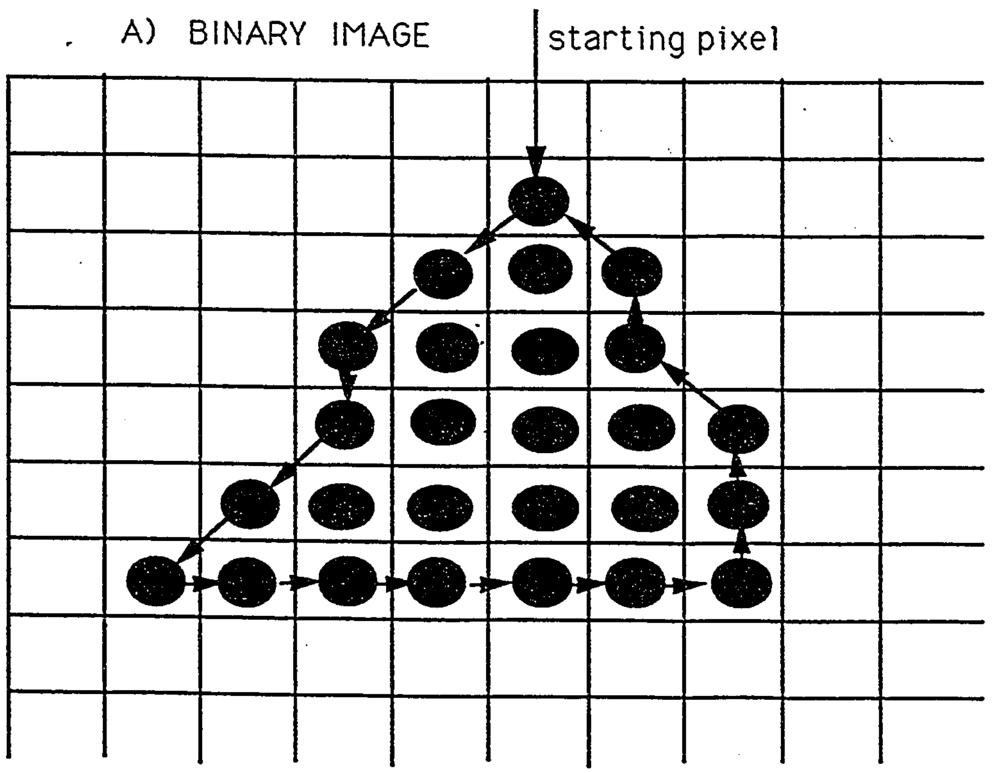
Algorithms based on Freeman chain code are useful as such especially for data compression but are very sensitive to digitization noise or image resolution [WIE86]. However they can be further processed if the Freeman chain is unrolled as segments and each segment tested to see if it belongs to a straight line or not; the result is a polygon based contour [JUV86].

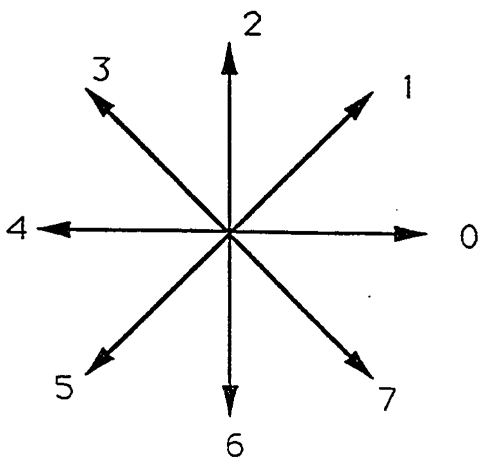
2.5 Structured Image Representation (contour based)

The non structured coding of images as such present many disadvantages for a real time robot vision system :

a) the storage requirements are large, from 8 to 65 kilobytes, especially for high resolution images in addition to the slowness of processing [KLA89 , PAV77].

b) all codings which store gray scale values of pixels as such are highly dependent on lighting conditions.





B) FREEMAN DIRECTION (8)

C) FREEMAN CHAIN CODE = 5 5 6 5 5 0 0 0 0 0 0 2 2 3 2 3

Fig 2.7 : Freeman Chain Code

c) some are not invariant under rotation, translation or scaling : run-length and differential encoding are examples.

d) they do not tolerate noises e.g. the Freeman chain contour changes significantly with digitization errors, a common problem in any image sensing device [WIE86].

e) the complexity of interpretation increases if objects are touching or partly occluded leading to failure of recognition; edge based techniques are more powerful in that only fragments of the object's boundary can be used to match it against a template [STO84].

f) no information about the geometry of the object is explicitly known. This is perhaps the most serious drawback of non structured models because in a robotic environment interaction of the manipulator on the object necessitates information about position, orientation dimension and grasping points; e.g. a robot needs details such as part is located at (X_o, Y_o), oriented at angle θ, grasping point is $(X_o + X_d, Y_o + Y_d)$.

What is needed essentially is to reduce the 'raw' data of region based scalars to a graph which describes the relative positions of regions of 'simple' shapes which together form a region of 'complex' shape [TOU74]. The boundary of a region is a natural choice to obtain geometrical characteristics; a sequence of boundary points are treated as data from which mathematical invariant features are to be extracted [COR83].

These mathematical models are to be obtained in a rather simple way and should be more compact representations, yet without losing any of the essential information inherent in the previous representation [PAV77]. Mathematical models ease the task of recognition considerably to a mapping of two relational structures [CHE82]. with effective accuracy and hit ratio. Some of these models, widely used for varying applications, are presented next

2.5.1 Mass Center Method

The mass center method transforms a binary image into a set of 1-dimensional sequences of vector lengths which have their origin in the mass center and their ends on the contour [WIE86].

Let (X_i , Y_i) for i = 1, 2, ... n denote the integer values of contour coordinates (Fig 2.8) in Cartesian system. The center of mass is then expressed as

$$X_s = \frac{\sum_{i=1}^{n} X_i}{n}$$

$$Y_s = \frac{\sum_{i=1}^{n} Y_i}{n}$$

and the sequence of vector length $\{S_i\}$ is $S_i = \sqrt{(X_i - X_s)^2 + (Y_i - Y_s)^2}$

The mass center method, besides being rotational and positional invariant, can be used to obtain other useful descriptors such as the size of the image which is the mean value of $\{S_i\}$.

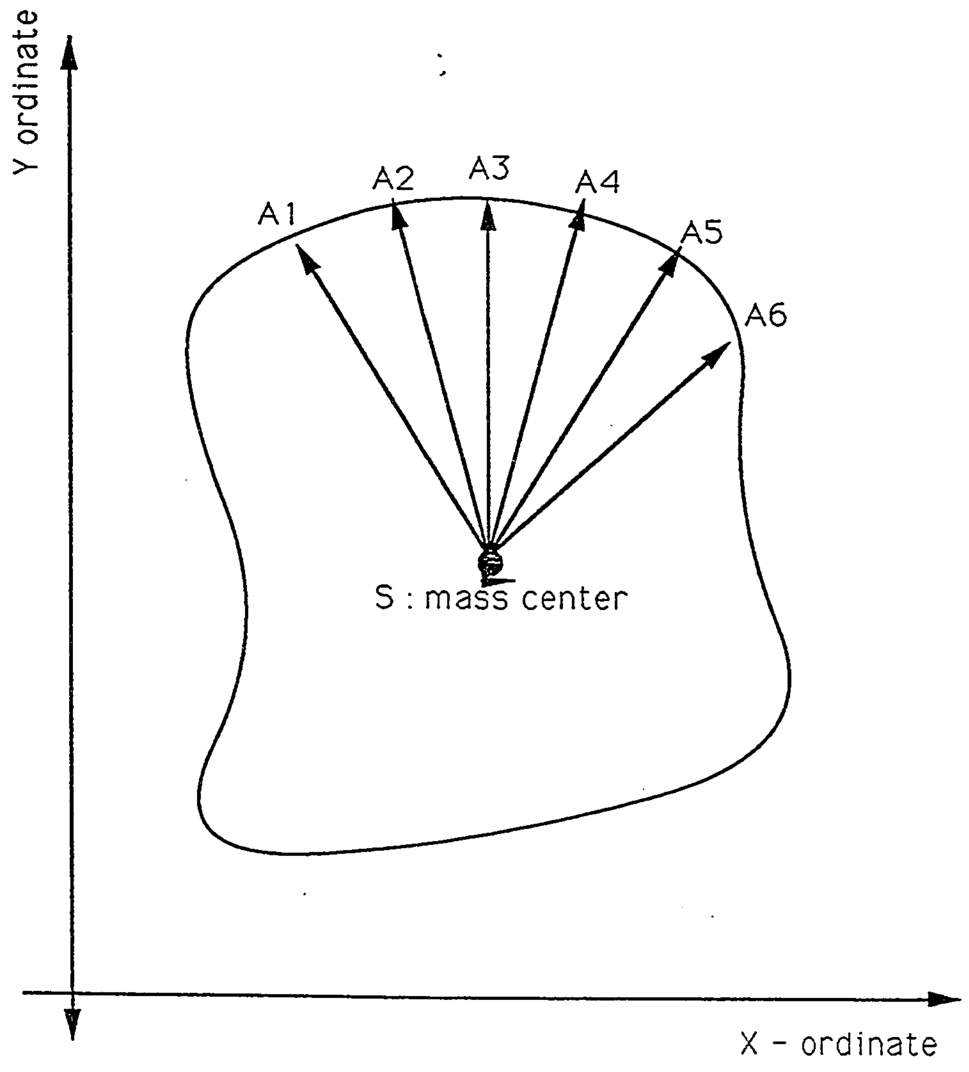


Fig 2.8 : Mass Center Method

2.5.2 Fourrier Transform

Fourrier descriptors are a method of extracting information about an object using its boundary points in terms of tangent angles versus arc length or a parametric representation of boundary coordinates [TOU74].

Let (X_i , Y_i) for i = 1, 2, ... n be the Cartesian ordinates (Fig 2.9) of boundary points with X_i being mapped to the real component and Y_i to the imaginary component of complex numbers.

Fourrier analysis often uses amplitude and phase parameters in the transforms to distinguish boundary shapes in object mapping and recognition [SAF89].

The boundary $\{f_i\}$ being a series of complex numbers :

$f(i) = (X_i , Y_i) , i = 0, 1, 2, \ldots n\text{-}1$

The Fourrier transform of the complex sequence :

$$F(u) = \sum_{k=0}^{n-1} f(k) \ \exp\left(-j\, 2\, \pi \left(\frac{k\, u}{n} \right) \right)$$

and the amplitude spectrum as

$$| F(u) | = | \text{Real } F(u)^{2} + \text{Imaginary } F(u)^{2} |^{1/2}$$

and the phase as

$$\Phi (u) = \arctan \left[\frac{\text{Imaginary } F(u)}{\text{Real } F(u)} \right]$$

Fourrier descriptors are invariant without requiring any further computations. They are useful because of certain properties in the Fourrier domain : easy movement between the Spatial and Fourrier domains; size changes generated by multiplying by a constant, angle rotations by a simple multiplication and translation by simple addition [SAF89].

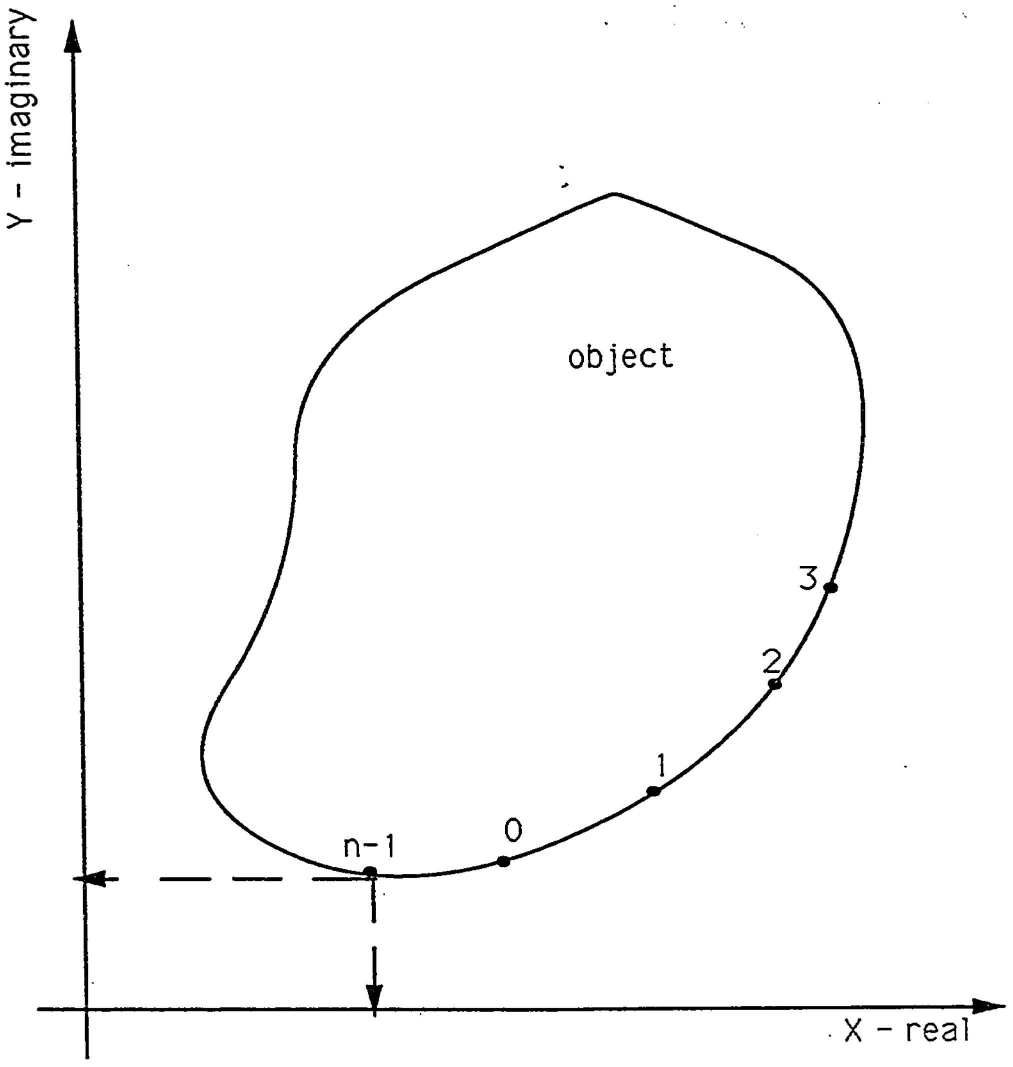


Fig 2.9 : Fourrier Transform

2.5.3 Spline Fitting

The domain of input data is divided into a finite number of intervals (such as corner points of boundary) and a different approximation is applied to a subset of intervals; i.e. every 3 or 4 consecutive corner points are fitted by a polynomial and the sum total of all these polynomials identify the contour [PAV77].

An example is a partition $\underline{X}$ with a sequence of 4 boundary points X_i, ($i = 0, 1, 2, 3$) such that $X_0 < X_1 < X_2 < X_3$; a cubic B-spline can then be interpolated over these 4 points and they are cubic polynomials such that the spline smoothly links the partition $\underline{X}$ [HAR83].

2.5.4 'Fast' Structured Methods

The structured representations mentioned previously, which will be referred to as 'mathematical' models have many drawbacks which prevent their use in real time object recognition and they are :

a) the majority of 'mathematical algorithms like Integral Square, Least Square or Spline Fitting originated from the field of character recognition; unfortunately many of them require determined positions of the image or claim to solve problems only if the data have a certain regularity [WIE86].

b) one difficulty in mathematical methods is that the iterative functions do converge in the continuous case but tend to cycle in discrete cases of digitized images [PAV77].

c) they are too costly, computation wise to be efficiently implemented as too many mathematical operations are involved; this is specially true for images of high resolution.

d) special problems for the otherwise excellent Fourrier method are that

1) it is invariant for continuous object contours but problems arise for discretized images on pixel arrays [AND86],

2) there is a lack of closure of boundary contour such that image reconstruction is inaccurate [AND86, TOU74], and

3) it often fails in distinguishing objects if significant variations exist over contours of similar class [STA86].

e) one serious disadvantage to Mass Center, Fourrier or Spline Fitting methods is in choosing which pixel points on the contour are selected for representing the silhouette accurately enough. Choose too much points and the method becomes costly and redundant; choose less points and the representation will not be accurate enough to identify the silhouette uniquely. Detecting these corner pixel points is by itself a problem : if a contour tracking algorithm is used to detect points of inflexion on the border, it could be argued that if the coordinates of break points are available, why not use them in a simpler polygonal representation.

f) perhaps the most serious drawback of mathematical representations is that they do not make available geometrical information as such without further processing. Robotic manipulation implies precise knowledge of absolute orientation and absolute grasping point positions; this is only possible if the absolute dimension and orientation of the object relative to a distinctive local feature is known such as a unique corner with specific large angular change, a unique maximum segment length or a sequence of distinctive characteristics. This cannot be obtained readily from mathematical methods without further feature extraction procedures

Therefore what is required for practical applications is a 'simplified' structural representation which is fast enough to be efficient in real time. The shape of most technical workpieces can be adequately described by simple geometrical forms such as corners, straight lines and circular arcs [RUM84]. Polygonal approximation or incremental angular change of successive boundary segments is a powerful method because :

a) a contour following algorithm (which is a must even for 'mathematical' methods) quickly identifies corners and parameters of the polygonal model with minimum additional processing.

b) rotation and translation do not affect the model which is invariant as such; scaling preserves all angles and the lengths need be scaled by only one factor.

c) a straight line is a computationally simpler curve to fit a group of data than any higher order curve [TSU82].

d) it offers the richest description capabilities and hence accelerates any recognition algorithm considerably [COR83].

e) the geometry of the contour or how the object is positioned relative to the environment is readily available to aid robotic manipulation [COR83 , TSU82].

Two such 'simple' and 'fast' methods are now presented.

2.5.5 Circular Scanning

Circular scanning is an edge-based technique for rapid acquisition o 2-dimensional objects where the object is scanned in a circle (Fig 2.10-a) centered at an arbitrary boundary point of the object [STO84]. A boundary point P on the object is sought and scanning proceeds in a circular path (e.g. anti clockwise) of a certain radius R_1 yielding a scan which detects the 0-to-1, background to object transition and the length of the arc crossing the object or boundary (Fig 2.10-b). The resulting scan is rotation and translation invariant provided the scan center and radius are preserved. The method is simple and fast but has some drawbacks namely :

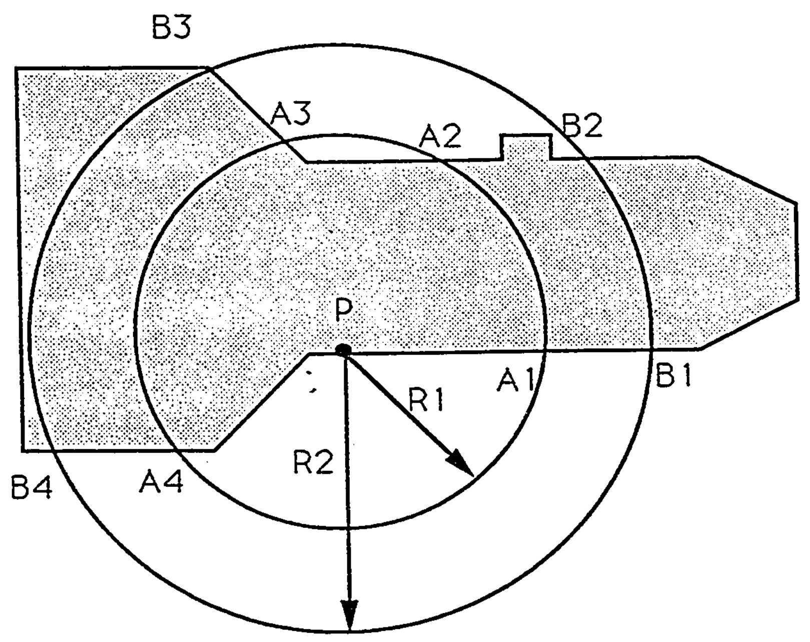
a) choice of center of scan, P, must always be selected and positioned on a distinctive contour point for the scans of unknown object to be matched to a known object. The choice of P obviously depends on a unique shape feature obtained only through a contour following algorithm.

b) one circular scan cannot uniquely map an object, so a series of scans are needed with different concentric radii. The more the scans, the more finely detailed representation is obtained but more bytes are needed for storage; if fewer scans are used, distinguishing features on the contour located between any two successive scans will be overlooked.

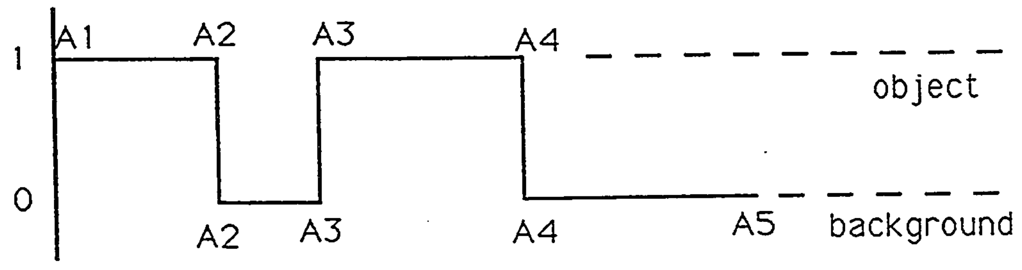
c) detecting distances along circular arcs crossing the object is not a straight forward linear calculation.

d) special hardware equipment is often used like a circular scanning camera centered on P with photodiodes (e.g. 720) arranged in concentric circles.

A)



B)



C)

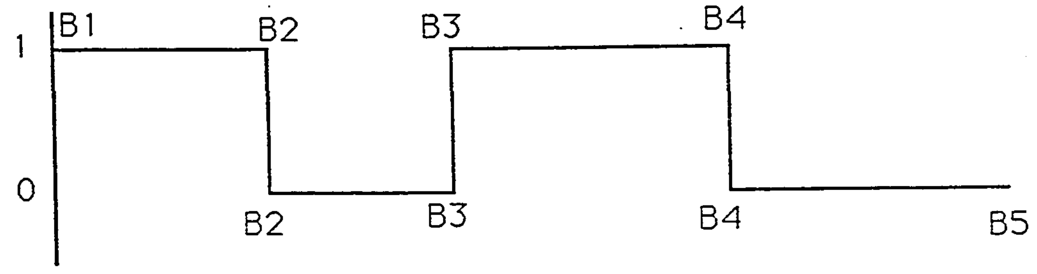


Fig 2.10 : Circular Scanning

2.5.6 Polar Representation

Polar representation is a simple yet powerful technique for representing and recognizing objects and has been applied widely. Consider an object (Fig 2.11-a) whose contour corners have been identified as

$$\{X_i, Y_i\} = \begin{bmatrix} X_1 & Y_1 \\ X_2 & Y_2 \\ X_3 & Y_3 \\ X_4 & Y_4 \\ X_5 & Y_5 \end{bmatrix}$$

where (X_i , Y_i) refers to the Cartesian coordinates of corner i of contour. The relative polar representation converts the polygonal contour as an ordered array of segment lengths (ρ_i) and change in angular direction between successive segments (θ_i) .

$$\{\rho_i, \theta_i\} = \begin{bmatrix} \rho_1 & \theta_1 \\ \rho_2 & \theta_2 \\ \rho_3 & \theta_3 \\ \rho_4 & \theta_4 \\ \rho_5 & \theta_5 \end{bmatrix}$$

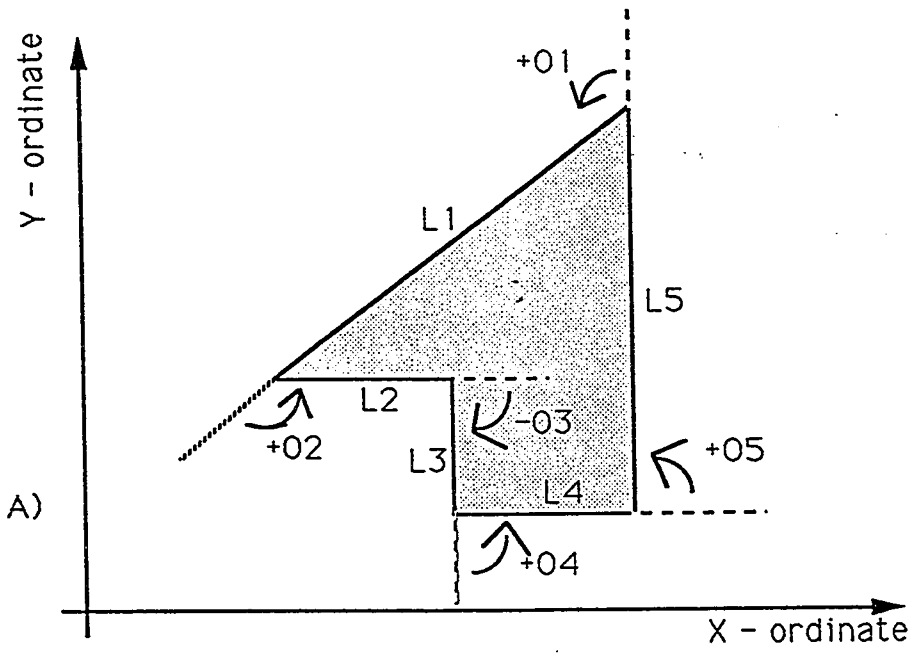
Besides the advantage of being a fast structured method (already covered in Section 2.5.5) the polar representation possesses a few more namely :

a) a translation of the figure is invariant

b) rotation of the object leaves the polar representation invariant except that the ordering of the set { ρ_i , θ_i } is circularly rotated.

c) scaling of the figure multiplies each ρ_i by one constant magnification factor.

A)



B)

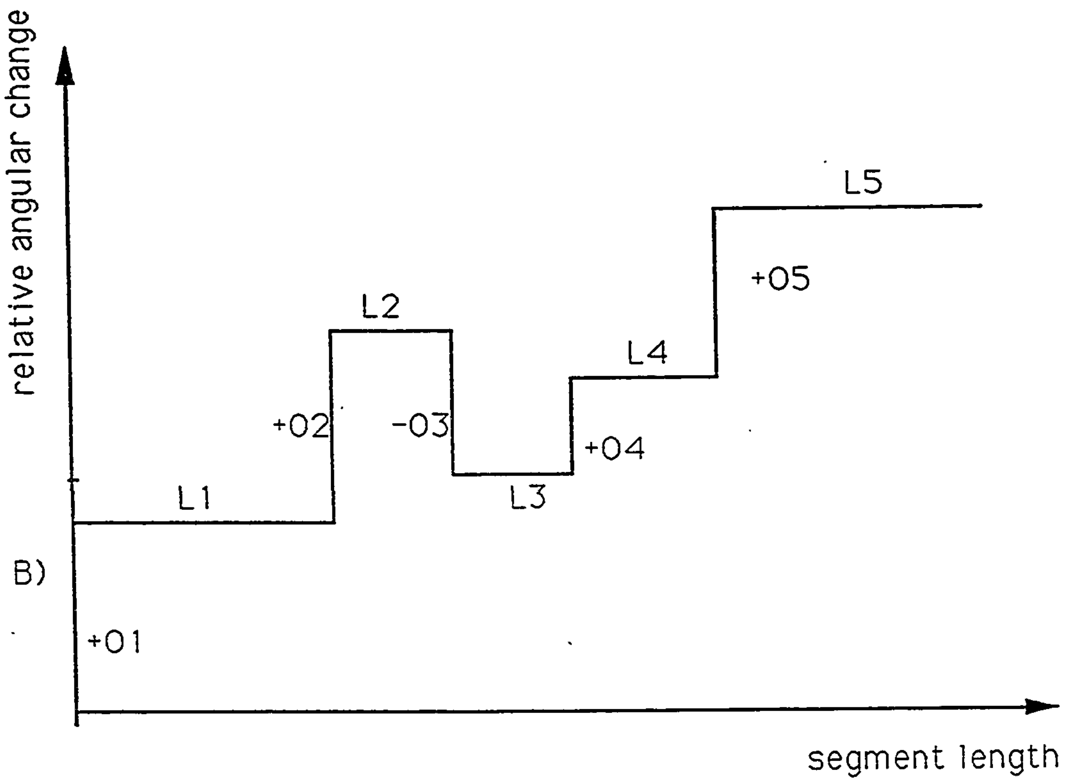


Fig 2.11 .: pPolar representation and 'graph'

d) whereas in mathematical methods or even circular scanning, the algorithm which follows the contour to obtain contour points (or distinctive center for circular scanning) is needed just to start the process and the bulk of the computation is done later, in polar representation, the tracking algorithm itself is the major task and obtaining the ρ_i , θ_i from the Cartesian coordinates outputted is pretty straight forward.

e) only one polar encoding or graph (a graph drawn on Cartesian axes of relative orientations of segments versus their linear distances as in Fig 2.11-b) is required to uniquely identify objects; compare this with the multiple scans required for circular scanning.

f) robotic manipulation require knowledge of grasping points relative to a certain distinctive or easily identified shape feature usually a segment or/and a corner and this information is readily available form the polar model.

g) the richest description features (like longest segment, largest positive and negative angles) which are very important in recognition are immediately obtained.

For all these reasons, the polar representation was chosen to be implemented for this thesis work. Two levels of the polar model will be used for an object namely :

i) coarse model - used to obtain overall description features of an object and helps in accelerating recognition.

ii) detailed model - used for accurate and 'perfect' matching of unknown object to a known one.

2.6 Recognition / Classification

An effective recognition system for robots must inherently represent objects in a data model that is invariant under translation, rotation or ideally even independent of the camera-object distance; this is essential for unplanned situations in the robot work space where no 'a priori' information about the orientation or dimension exists.

Some recognition algorithms, the so called region based, compare the image of unknown object to defect free images of known objects pixel by pixel or through gray level analysis. These have been widely used in industrial inspection systems because they are fast, simple and reliable; however they require storage of complete error free images, are not flexible enough, require perfect alignment of and are not particularly suited for robot vision [GON83].

Many excellent mathematical recognition techniques have been developed and implemented with a high hit-ratio but are ill suited for real time robotic systems due to their complexity in computations.

A powerful and flexible recognition system begins with image analysis in a data driven mode (region based) but when sufficient information is available it switches to a model driven mode [PER86] where analysis is based on geometry and structure [GON83]. The shape and geometry features clearly reduce the dimensionality required to characterize objects and will greatly reduce identification time [KLA89]. A feature vector of an object is the global features (shape based) which in totality characterizes an object or class; recognition of an unknown object is reduce to matching its feature vector with those of all previously known or tagged objects. Clearly it is impractical to use exhaustive 1-to-1 search to find the best distance two vectors when vector dimensions are near typical value of 15 or so [WAL81]. Assigning an unknown object to an object class in the dictionary involves the 'minimum distance' or 'nearest neighbour' technique where the object is assigned to the closest class i.e. the distance or difference in features is minimal) [SUE86].

Recently there has been interest in the use of hierarchically structured representations that incorporate both coarse and detailed information about an object so that both gross geometry and important local features are easily available while topological and locational information is used only when needed [ROS84]. The coarse description is kept in active memory for quick access whilst the massive detailed data can be kept on auxiliary storage to be paged in when required. Recognition always starts from the coarse level to identify a subset of the object dictionary on which to pursue the search; thus results obtained from the coarse level search speed up and improve the accuracy of the detailed level searches [BHA82].

The requirements of such a hierarchical system are :

i) partitions allow recognition of object to nearest class in minimal time.

ii) the first level search involves only the coarse model and reduces the lower level search.

iii) the recognition behaves unambiguously or consistently by not assigning an object to more than one class.

iv) the detailed features are kept only at the lowest nodes of the hierarchy and on secondary storage; if recognition can be achieved without the detailed level, so much the better but if not, the detailed data will be paged in active memory only if absolutely needed.

CHAPTER 3

OVERALL DESIGN OF THESIS WORK

3.1 Introduction

The objective of this thesis is to demonstrate the efficiency of a hierarchical model of image representation. For reasons stated in the previous chapter an edge based polar representation will be used with modification to allow for a 2-level model. The recognition process will involve a feature vector based on the coarse model, a set of decision heuristics to accelerate the search and a minimum distance match at the detailed model level. The main thrust of the implementation will be on a hierarchical model and the recognition task, not on designing a complete robot vision system; this thesis will thus be a pilot vision system on which other features will be added and expanded in the future.

The implementation of this thesis work will depend on many factors, some of which forced us to the use of already available hardware equipment and some which guided us to the use of software. The main choices of software and hardware were :

a) the choice of hardware equipment was limited to what was available in the Robotics Laboratory of the Computer Engineering Department; it includes a personal computer IBM-AT, a FG-100 video interface which digitizes a video image to a 512 × 512 pixel image of 8-bit gray scale (monotone), a SONY CCD camera, a television monitor and a line printer.

b) assembly 8086 / 8088 again was the only choice for implementing the image acquisition and thresholding for two reasons :

i) the FG-100 video processor board (gray scale image frame) can be accessed and processed only through assembly and

ii) the processing required for pixel by pixel analysis or contour extraction from a high resolution 512×512 image imposes the use of the fastest possible language.

c) a programming language for the recognition task had to have the following properties :

i) it must be able to interface easily with Assembly 8086/88 with ease and minimum passing of parameters.

ii) it must support graphics capabilities because invariably the object's contour has to be displayed visually in a robot vision system specially in the learning phase.

iii) it must be powerful in programing features. It should allow modular design and coding of software and should be reasonably fast.

iv) it must interface with some database which would store the object's models in indexed files for direct access at recognition time. If the language has no interface with an existing database, then all the basic storage and retrieval facilities needed will have to be designed from scratch.

Turbo Pascal ver 5.0 was the only reasonable choice available and satisfied all the above requirements.

d) designing a separate data base for the system would involve managing sequential and direct files, implementing efficient sorting, taking care of storage and swapping in pages from auxiliary memory to active memory; all this would burden the speed of the system because of the overhead.

Pascal Database Toolbox is a small data base package (implying minimum overhead) which interfaces easily with Turbo Pascal and yet is powerful enough to do the basic duties asked from a database. It efficiently locates, inserts or deletes tuples in large data files (either randomly or in sorted sequence) and indexing is achieved through B^+ trees, the fastest method for finding and retrieving data base information; in addition it supports relations with a maximum number of 2 billion records.

3.1.1 Binary Image Acquisition

This will involve the interface to the FG-100 board and issuing it commands to snap an image of an object which is digitized as a 512×512 pixel frame with an 8-bit gray level intensity.

Let this image be $G[512,512]$ (Fig 2.3-b). The image consists of 512 rows and 512 columns where the first row and first column is at the top left corner. $G(i,j)$ represents the gray level value (between 0 - 255) of the pixel at row **i** and column **j**. The $G[512,512]$ will be transformed to a binary image $B[512,512]$ where $B(i,j)$ is a pixel value of 0 or 1 (Fig 2.3-c).

$B(i,j) = 1$ if pixel is located on object region.

$B(i,j) = 0$ if pixel is outside object, on background.

The mapping of $G[512,512] \rightarrow B[512,512]$ will be implemented as a simple 'interactive' global thresholding as

$$B(i,j) = \begin{cases} 0 & \text{if } G(i,j) < T \\ 1 & \text{if } G(i,j) \geq T \end{cases}$$

where T is a threshold gray level intensity which clearly delineates object from background. The operator will be given the choice to interactively set this value of T depending on lighting or object / background color contrast.

3.1.2 Primary Representation

The binary matrix B[512, 512] is further processed to isolate all pixels lying on the object's boundary.

Let $C(i,j)_k$ be a contour pixel if at row i and column j of the image, there is a $0 \rightarrow 1$ or background $\rightarrow$ object transition. All the border pixels $C(i,j)_k$ will be kept as a chain code using Freeman's directions (Fig 2.6).

Let $FMC[C_k]$ be the chain code of all the contour pixels linked as an unbroken chain where the contour is scanned from top to bottom and in anti clockwise direction and

(X_0, Y_0) : Cartesian position of top most border pixel.

C_1 : first successor border pixel of (X_0, Y_0).

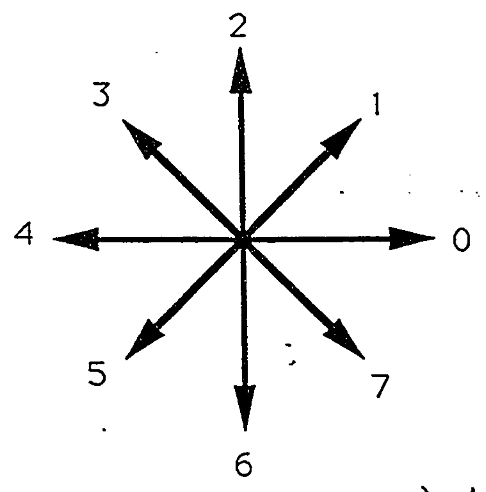
C_k : the next consecutive pixel on the contour from C_{k-1}.

The Freeman Contour coding

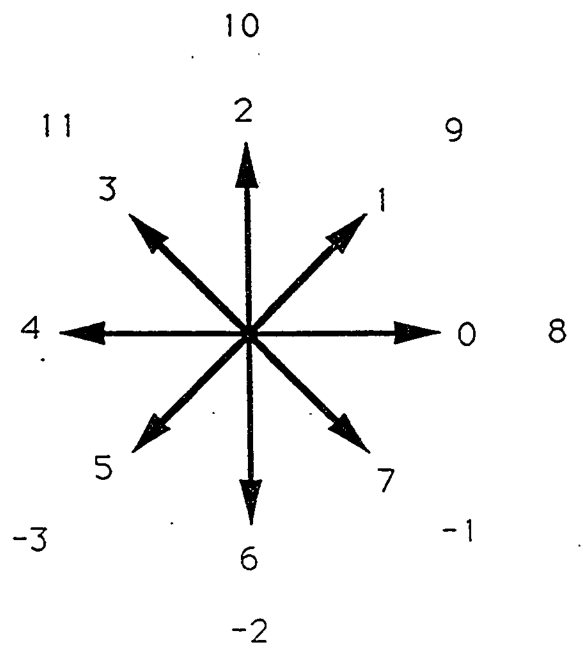
$FMC(C_k) = \{0, 1, 2, 3, 4, 5, 6, 7\}$ indicates the direction of orientation of pixel C_k with respect to pixel C_{k-1}. (Fig 3.1)

Only the FMC is saved and processed later, the (X_0, Y_0) or start pixel coordinates are also saved as well as the number of pixels lying on the contour.

A useful next step would be to eliminate the big difference between DIR0 (east) and DIR7 (south east) which does not reflect the closeness of these two directions, e.g. the difference of 1 between DIR1, DIR2 reflects an angle change of 45° while a difference of 7 between DIR0, DIR7 also reflects the same angle change; this will prove problematic in the segmentation stage especially if modulo arithmetic is to be used constantly for each $FMC(C_k)$.



a) Absolute FMC



b) Relative FMC

Fig 3.1 : FMC & RFMC

It is proposed to change the FMC (C_k) to a Relative Freeman Contour Coding (RFMC) :

RFMC (C_k) = { 0 , 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 , . . . }

For example a DIR0 will be changed to a DIR8 if the previous pixel directions were 5, 6 or 7 .

$$\mathrm{RFMC}(C_k) = 8 \begin{cases} \text{if FMC}(C_k) = 0 \\ \text{\& if FMC}(C_{k-1}) = \{5, 6, 7\} \end{cases}$$

Now the new direction DIR7, DIR8 yield a difference of 1 (not 7) which correctly reflects an angle change of 45° .

3.1.3 Invariant Representation (polar)

The next step is to convert the RFMC of the object to a polar model, an intermediate requisite is to calculate all the break points coordinates. One method to obtain the corner points from the RFMC is to unroll the chain and break it into regular group of pixels and checking if the chain segment belongs to a direction being considered.

e.g. The FMC for segment $C_5 - C_6$ (Fig 3.2) might be :

4 4 4 5 3 4 4 4 4 4 3 4 4 4 4 5 4 4 4 4 ... etc

Obviously considering individual C_k 's and checking the direction difference is not possible because of

i) effect of digitization noise is very local.

ii) comparison of FMC has to be done n times if the perimeter is n-pixel long.

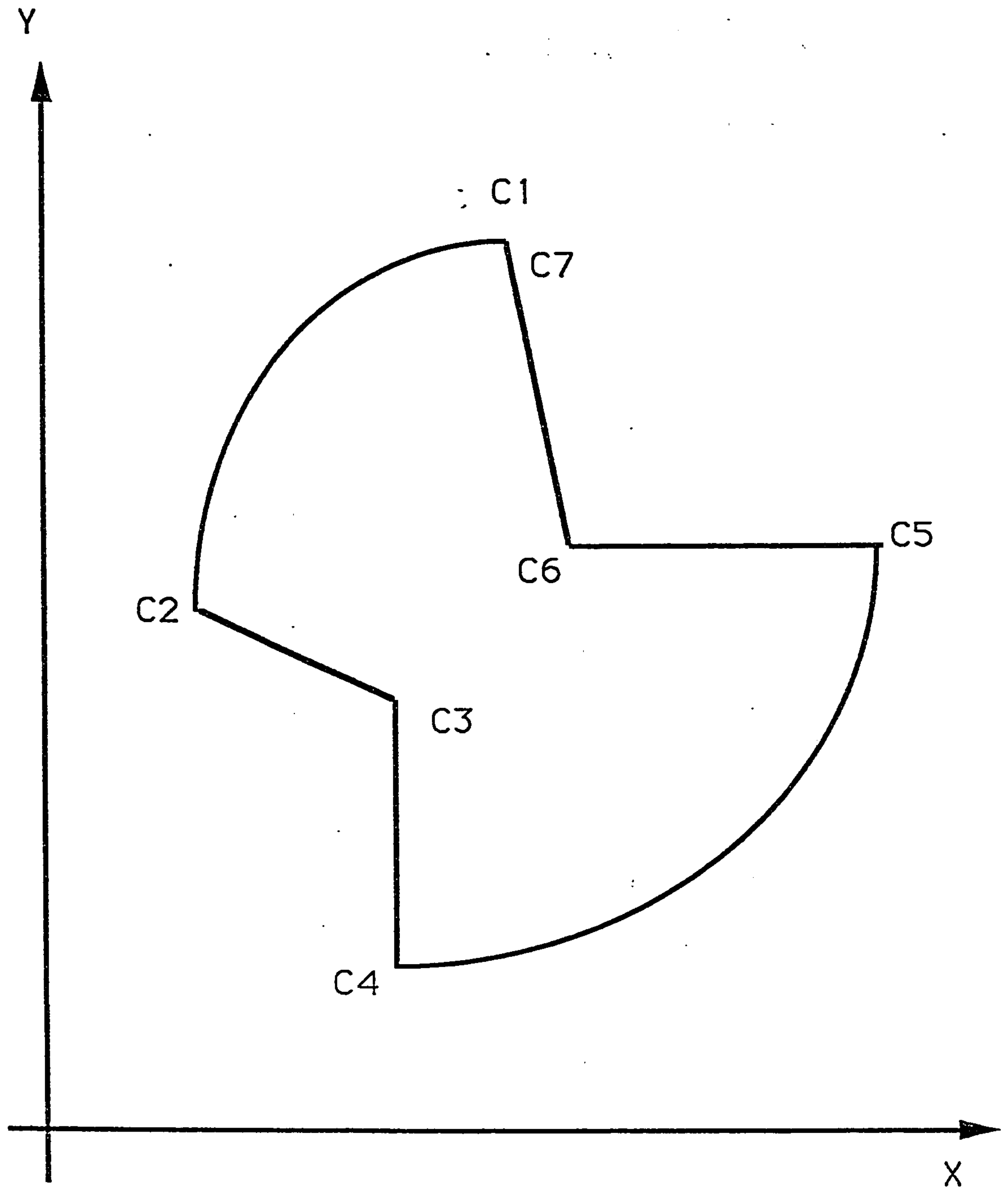


Fig 3.2 : Part to be segmented

To overlook the noise effects, a group of C_k's will be taken (e.g 4, 5, or 6 will be a good grouping) and their sums calculated ; e.g. :

4-pixel	4 4 4 5	3 4 4 4	4 4 3 4	4 4 4 5	4 4 4 4
$\sum$:	17	15	15	17	16

Let S_k denote the sum of a group of 4 consecutive C_k's

and $\overline{S_k}$ denote $\frac{S_k}{4}$ or average direction of S_k.

The sum of any 4-pixel directions is

$$S_k = \sum_{i=k}^{i=k+3} f_i = \{ 14, 15, 16, 17, 18 \}$$

$\overline{S_k}$ will always take a value which reflects the direction of segment $C_5 - C_6$, the overall direction of the segment $\overline{S_k}$ is given as :

$$3.5 \leq \frac{S_k}{4} \leq 4.5$$

For the segment $C_6 - C_7$, the FMC might look like :

2 2 2 1 3 2 2 2 2 1 2 2 2 2 2 3 2 3 2 2 2 2 2 1

where $S_k = \{ 6, 6, 8, 9, 10 \}$

and $1.5 \leq \overline{S_k} \leq 2.5$

This property of $\overline{S_k}$, which shows a marked difference as the direction of a segment is changed, will be used to detect corner points; e.g. examining the continuous FMC of the object's contour for segments $C_5 - C_6 - C_7$, and specifically at corner C_6, the value of $\overline{S_k}$ at segment $C_6 - C_7$, will differ markedly from $\overline{S_k}$ of segment $C_5 - C_6$.

A corner point is detected along $\{ D_i \}$ by examining all

$$\overline{S_k} = \sum_{i=k}^{i=k+3} f_i \text{ for } k = 0, 4, 8, 12 \ldots \frac{n}{4}$$

Let $\overline{S_k}(\text{ref})$ denote the overall direction of segment under consideration, a corner point is located if

$| \overline{S_k} - S_k(\text{ref}) | \geq \varepsilon$ or if the direction of a 4-pixel groups compared to the considered direction reflects a large change in direction. The value of ε will determine to what degree the algorithm tolerates deviation in a straight line segment. A large value of ε will detect break points only at places where the segments show a large deflection while a smaller ε will overlook small defections or fluctuations due to noises and break up the segments into many small segments. An optimal value of ε will be decided upon.

For this thesis two values of ε namely ε_c and ε_d will be used to generate a coarse and detailed breakdown of the contour (Fig 3.3) and two polar representations of the same object. The coarse model will be used to generate the overall feature vector of the object because it conveniently ignores digitization noises while the detailed model is needed for precise object matching in the last step of recognition. The two models referred to as :

Coarse(obj) $= \{ C_i \}$ where $C_i = (X_i , Y_i) , i = 1, 2, \ldots , p$

Detail(obj) $= \{ D_i \}$ where $D_i = (X_i , Y_i) , i = 1, 2, \ldots , q$

Obviously $q \geq p$ and many corners will occur simultaneously in both $\{ C_i \}$ and $\{ D_i \}$ but at different indices, e.g $C_1 \equiv D_1$, $C_2 \equiv D_3, \ldots,$ $C_5 \equiv D_8$,

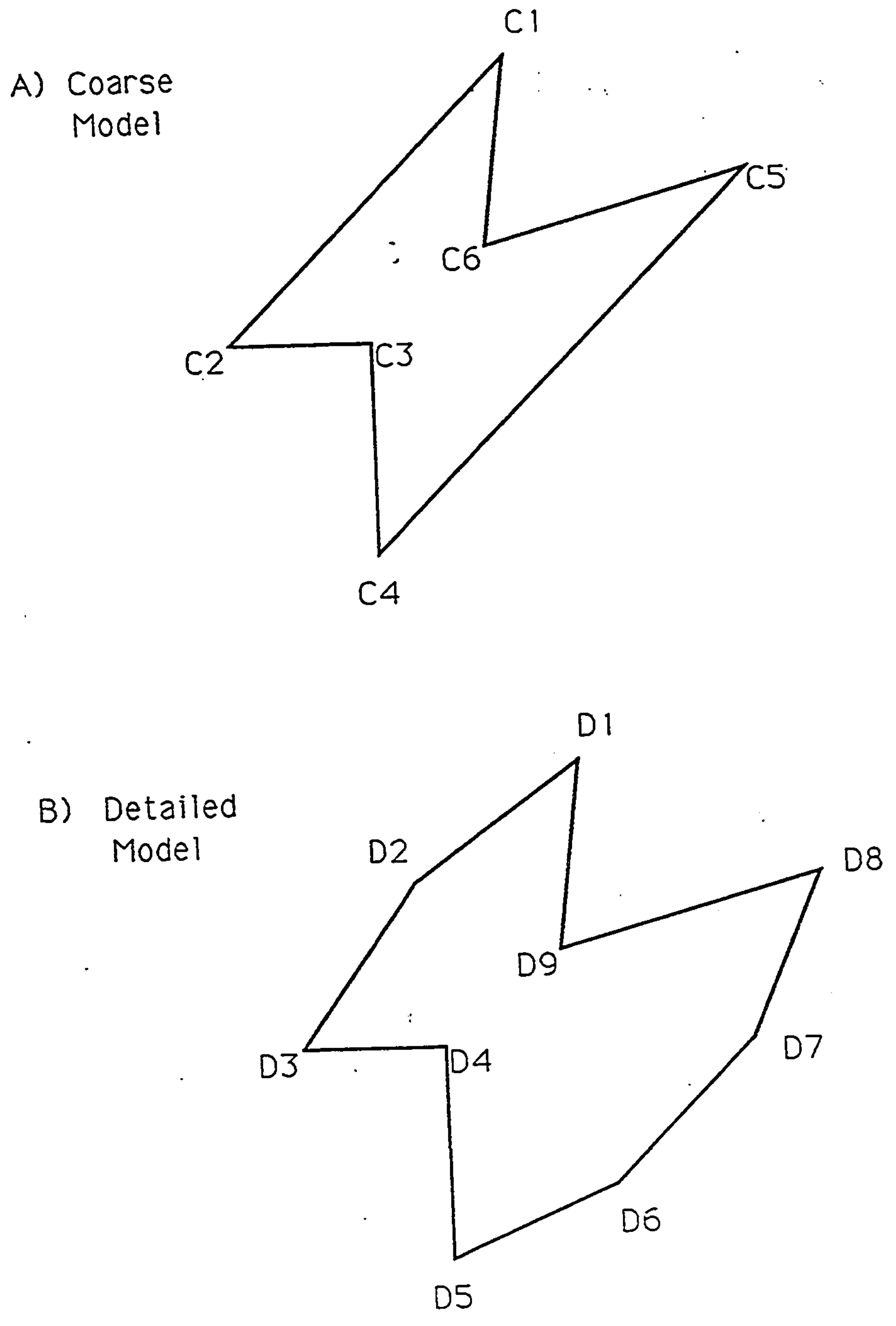


Fig 3.3 : 2-hierarchy models

Obtaining the polar representation from the Cartesian coordinates of the breakpoints $\{ C_i \}$ or $\{ D_i \}$ is straight forward but the use of cosine, tangent, slope or vector method will be examined and the most efficient method will be implemented. The method naturally will calculate the angular change of segments but the absolute value will not help; the direction of angle change (clockwise or anti clockwise) is also needed. Basically the output of this algorithm will be to generate a

Polar(obj) = $\{ P_i \}$ where $P_i = (\rho_i , \theta_i)$,

ρ_i = length of segment i

θ_i = angular change between segments ρ_i , ρ_{i-1} .

example for :

$\{ C_i \} \rightarrow \{ P_i \}$ of Fig 3.3-a where

$P_1 = (\rho_1 , \theta_1) , P_2 = (\rho_2 , \theta_2) , \ldots\ldots P_6 = (\rho_6 , \theta_6)$,

Again for this thesis the coarse and detailed model $\{ C_i \}$ or $\{ D_i \}$ will result into two polar representations of the same object as $\{ P_{i(c)} \}$ or $\{ P_{i(d)} \}$ respectively.

A graph of the polar model helps us visualize the polygonal representation of an object and also helps in understanding how objects differ in shape and to what degree they differ (minimum distance test); the 'graph' is a plot of θ_i versus ρ_i for each segment of the object (Fig 3.4). We notice that :

* the representation is invariant under translation.

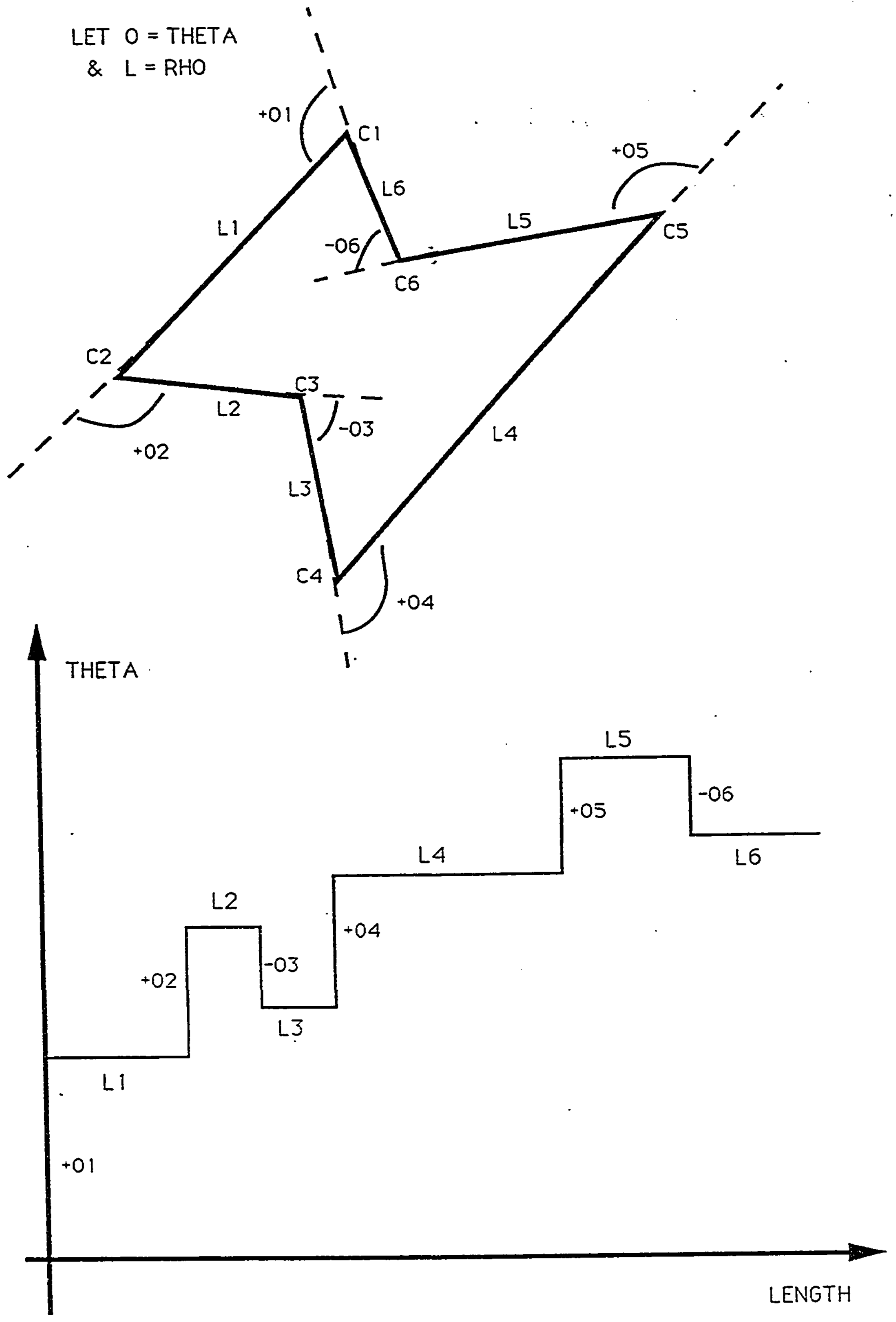


Fig 3.4 : Rho - Theta graph

* rotation leaves the absolute values and ordering of θ_i and ρ_i invariant (considering $\{ P_i \}$ as circularly linked)·

e.g if corner C_1 is the 'top most' corner of the object, then

$\{ P_{i(c)} \} = \{ P_1 , P_2 , P_3 , P_4 , P_5 , P_6 \}$

now if the object is rotated such that C_5 is the top most corner, its

$\{ P_{i(c)} \} = \{ P_5 , P_6 , P_1 , P_2 , P_3 , P_4 \}$

* under scaling, i.e. the same object viewed at different camera distances or different objects with exactly the same outline, all θ_i are left invariant and all ρ_i of the model will be multiplied by a single factor.

example : Let $\{ P_{i(c)} \}$ be the original polar model and let $\{ \overline{P_{i(c)}} \}$ be the model of the same object but a different camera's viewing distance resulting in a linear magnification factor , **m** , on each segment; then $\{ \overline{P_{i(c)}} \}$ is obtained from $\{ P_{i(c)} \}$ by multiplying each segment length by m :

$\overline{\theta_{i(c)}} = \overline{\theta_{i(c)}}$

and

$\overline{\rho_{i(c)}} = \overline{\rho_{i(c)}} \times m$

* to overlook the effect of scaling the $\{ P_{i(c)} \}$ and $\{ P_{i(d)} \}$ will be normalized by standardizing the perimeter of the contour to a fixed value in pixels.

* lastly we observe how the two graphs (Fig 3.5) of $\{ P_{i(c)} \}$ and $\{ P_{i(d)} \}$ correlate.

LET O = THETA
& L = RHO

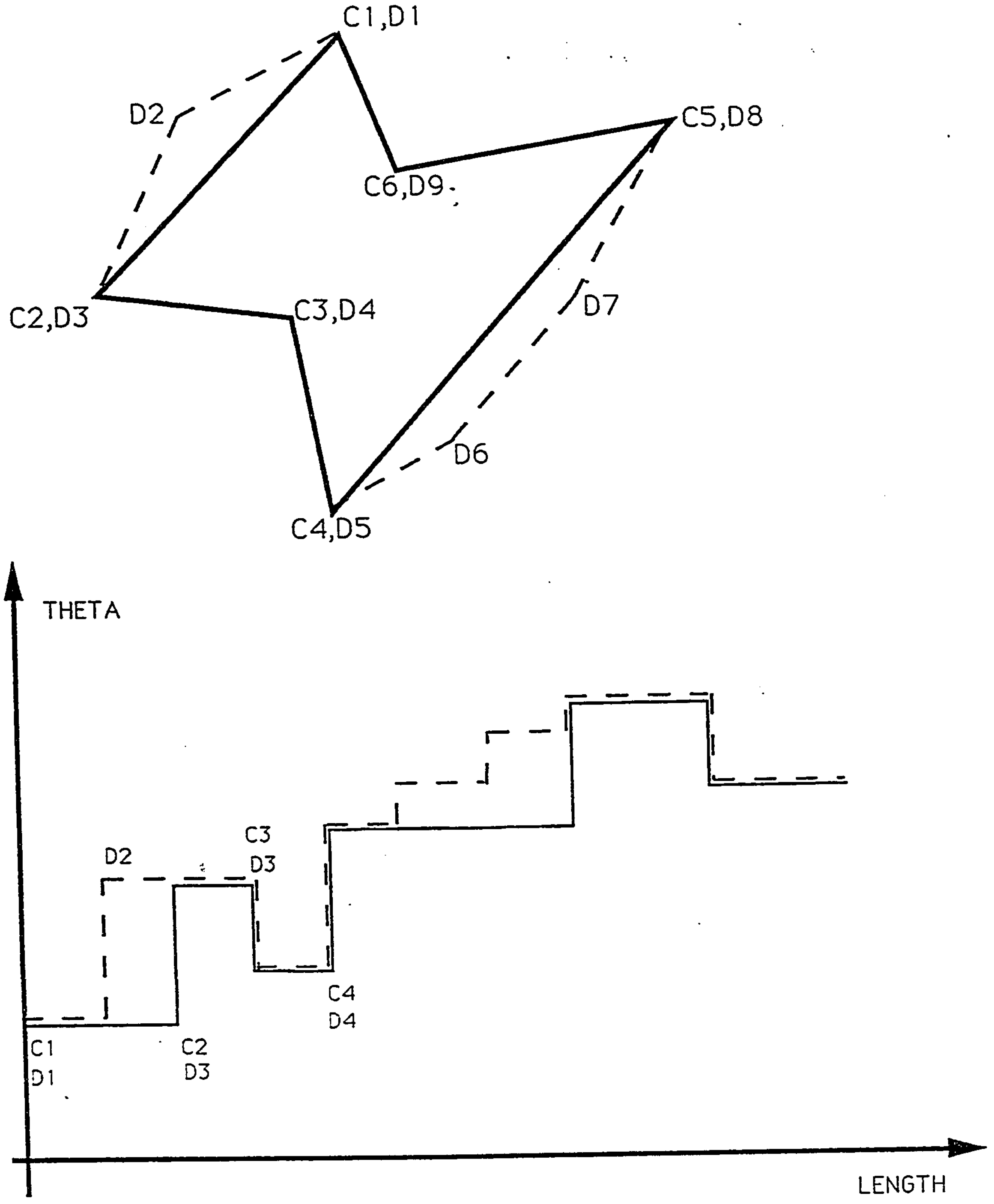


Fig 3.5 : Coarse & Detailed 'Graphs'

As already noted, C_1 and D_1, C_3 and D_4 match to exactly the same value of the ρ axis of the graph; the segment $C_1 - C_2$ of the coarse graph corresponds to segments $D_1 - D_2 - D_3$ of the detailed graph and segment $C_4 - C_5$ corresponds to segments $D_5 - D_6 - D_7$.

The coarse graph shows a long segment, especially a 'curved one', as one large segment and one large jump in the graph while the detailed graph breaks it into smaller segments such that the jump is smoother and hence more accurate.

3.1.4 Descriptor Array

The descriptor array will be an ordered set (array) of global features of the object which taken as a whole, uniquely or as closely as possible, identifies the object. The features will be extracted from the coarse model $\{ P_{i(c)} \}$ because the features should be invariant under digitization errors.

From $\{ P_{i(c)} \}$ obtain Des(obj)

where Des(obj) $= \{ Des_i \}$ for $i = 1, 2, \ldots S$

The features which will be chosen to be included in $\{ Des_i \}$ must :

* be relatively easy to extract from $\{ P_{i(c)} \}$ thus features such as center of mass, area of object though being powerful descriptors will not be included because of their lengthy computations.

* be powerful in the acceleration of the recognition algorithm

in general shape features which will act as separators of objects are needed.

* The set of descriptors chosen for implementation were :

  - largest segment length.
  - largest angular change of segments.
  - number of 'large' segments.
  - number of 'large' positive angles.
  - number of 'large' negative angles.
  - largest positive angle change.
  - largest negative angle change.
  - number of segments in coarse model.
  - number of segments in detailed model.

3.1.5 Database

The database will reflect the hierarchical structure of the design one level of hierarchy is already achieved by the two polar representation (coarse and detailed). The hierarchy in the database involves the concept of classes.

An object class is a set of objects which have the same geometrical 'shape' in that all the angles θ_i in their coarse polar model are approximately equal in magnitude and order but differ in their ρ_i either because the objects differ physically in size or they are positioned at varying camera distances.

Let the class :

$$CL_i = \{ obj_{i1}, obj_{i2}, \ldots, obj_{it} \}$$

Obviously an absolute physical dimension is required to differentiate between the objects in a class. A convenient one is the actual perimeter of the digitized object's contour, a value which is already available in the FMC generation. This aspect, i.e. the absolute differentiator of objects within a class, will not be implemented in details in this thesis.

The database will comprise of :

* classes of objects, each class consisting of similar objects

* descriptor array for each class.

* detailed polar representation for each class.

the database **db** is :

$\{ CL_i \}$, for i = 1, 2, . . . # of classes

$\{ Des_{ij} \}$, for j = 1, 2, . . . # of descriptors in class i

$\{ P_{ik(d)} \}$, for k = 1, 2, . . . # of segments in detailed polar model of class i

3.1.6 Supervised Learning

The supervised learning stage will allow the operator, given that the coarse and detailed models are available, to

- view the object at the coarse & detailed level graphically
- normalize the polar models
- examine the feature vector (descriptors)
- insert the object in the database within a class or as a new class
- view the database

The learning stages will be implemented as user friendly menu driven utilities.

3.1.7 Recognition via 'Clustering'

Given an unknown object and having the task of identifying it amongst a substantial set of possible object classes, a direct 1-on-1 exhaustive matching of unknown object's descriptors to each set of a class is too tedious. What is needed is to reduce the set to be matched to a subset of classes such that the unknown object is 'most likely' to belong to one in the subset.

For $\{ CL_i \}$, for i = 1, 2, . . . # of classes

A sequence of decisions or heuristics H_i is applied to $\{ CL_i \}$ such that a subset of 'most likely' results. e.g. Let H_1 represent 'pick classes with of segments = 8'; applying H_1 will output a cluster of classes $CLUS_1$ satisfying the decision search. In general a decision H_i will pick out a cluster of classes $CLUS_i$ satisfying :

descriptor Des_i = search value χ

or more generally a larger cluster satisfying :

$Des_i = \chi \pm \delta\chi$

Given an unknown object $Des_{(obju)}$, each decision is applied to the database of the dictionary

$$H_i \{ db , Des_{(obj)} \} \rightarrow CLUS_i : \{ Cl_k , Cl_l , \ldots , Cl_p \}$$

$H_1 \rightarrow CLUS_1$: all classes satisfying decision 1

$H_2 \rightarrow CLUS_2$: all classes satisfying decision 2

$H_s \rightarrow CLUS_s$: all classes satisfying decision s

Implementing rapid decision searches will imply the use of indexed database files for each feature of the descriptor array.

Let s = # of decisions applied; then the result of all the s searches results in s clusters

$CLUS_1, CLUS_2, \ldots, CLUS_s$

A resulting cluster $CLUS_r$ is calculated which is the union of all $CLUS_i$ and a frequency count $FREQ_i$ is kept for how many times $CLUS_i$ was found in the union.

$CLUS_r : \{ (Cl_k, F_k), (Cl_l, F_l), \ldots, (Cl_q, F_q) \}$

Next $CLUS_r$ is sorted in an ordered list called $CLUS_{rs}$ with highest value of $FREQ_i$ first.

$CLUS_{rs} : \{ (Cl_i, F_i), (Cl_j, F_j), \ldots, (Cl_m, F_m) \}$

such that $F_i \geq F_j \geq \ldots \geq F_m$

The higher the frequency count for a class, the more likely it is that the unknown object belongs to that class.

3.1.8 Identification via 'Minimum Distance'

Once the resulting class cluster and their frequency of hit are available, a precise matching is needed to verify accurately that the unknown object does belong to that class. If the object does not belong to the class with highest hit count, the class with next highest hit count is examined.

The precise identification is achieved through a 'minimum distance' or 'closest neighbor' matching of the polar model 'graphs' (detailed one) of the unknown object and class. The area between the graphs is an excellent minimum distance choice (Fig 3.6)

The area between two graphs can be given as :

$$\text{dis}(\{P_{i(d)}\},\{P_{i(u)}\}) = \sum_{rho=0}^{perimeter} |\theta_i(\rho) - \theta_j(\rho)| \; \delta_p$$

An algorithm to accelerate this 'area' matching process will be presented such that the value of δ_p will vary adaptively according to the segment lengths of the two graphs.

Two graphs which match exactly will obviously have a distance of zero because the area between them is nil; however to say that two graphs match only if the distance between them is zero is unrealistic in practical applications; instead object **i** and object **j** will be considered identical if :

$\text{dis}(\{P_{i(d)}\},\{P_{i(u)}\}) \le \text{dis}_{min}$

The value of dis_{min} will depend mainly on the scale of the graphs , e.g. how θ_i is represented (angles , degree) and the value of the normalized perimeter. It will be determined experimentally.

3.2 modules in design

The overall modules in this thesis will comprise of :

a) Supervised learning module.

b) Database & interface module.

c) Recognition module.

3.2.1 Supervised Learning Module.

The supervised learning module will consist mainly of 3 sub modules namely :

- get object's descriptors
- normalize polar model
- update object in database

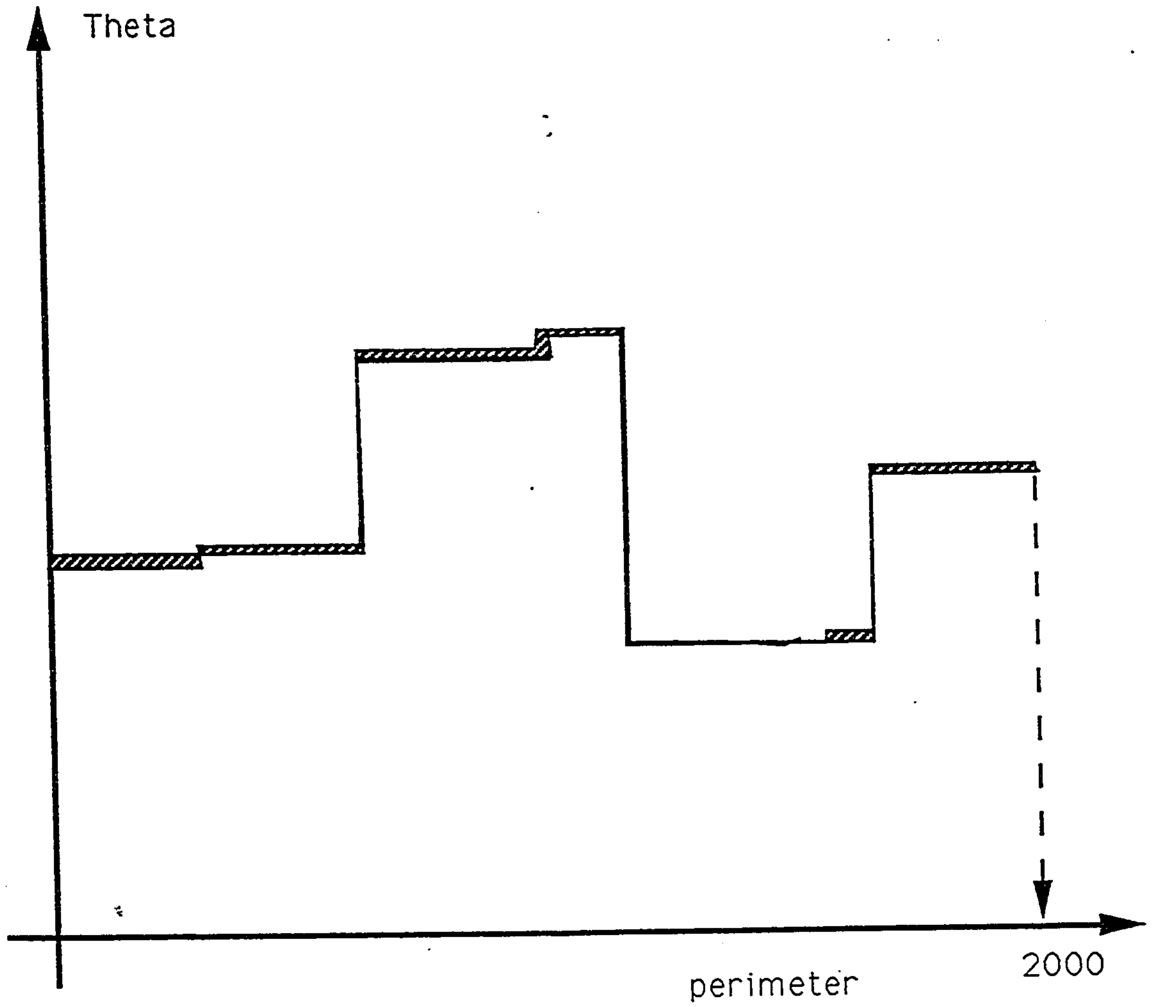


Fig 3.6 : Area between 2 graphs

Getting object's descriptors involves mainly acquiring the image, converting the gray scale image to a binary one , transferring the image to host memory and obtaining the FMC contour and then extracting the two polar models of the object or class.

Normalizing polar model will include converting the absolute model to a normalized one such that similar objects with different scale maps to the same model.

Updating an object to database is an interactive process where the operator, after viewing the object's contours graphically and examining the descriptors, will decide to insert the object in the dictionary; the object can be the first one of a new class or appended to an existing one. This step will not be automatic in the pilot system.

3.2.2 Database & Interface Module.

The database module will take care of the interface of the control module and the database. The database will be defined by three main relations and they are :

1. GENREC : the descriptor arrays of classes.
2. OBJREC : objects within classes and differentiator.
3. DETREC : detailed polar models of classes.

In addition, one more relation CLUSREC will be used in the recognition step.

3.2.3 Recognition Module.

The recognition module will include the steps :

- get object's descriptors & normalize
- obtain cluster for each decision search
- sorted join or union of all clusters
- apply minimum distance test for resulting cluster

The first two modules are the same employed in the learning module once the feature array of an unknown object is obtained, each descriptor will be used to extract all classes with similar or close values. This step is repeated as many times as there are descriptors and the resulting subset of classes only is used for further recognition. The cluster is sorted with the classes of highest occurrence and the minimum distance test applied to each sequentially until a match is found or the object is rejected.

3.3 System Interactive Menus & Utilities

The pilot system implemented in this thesis will be menu driven so that interaction of the operator is user friendly. The system is not complete; only the basic utilities will be implemented.

All the menus are driven by cursor keys : e.g. the user moves the 4 cursor keys to highlight the option he wants and presses 'RETURN' to execute that command. The utilities are presented next.

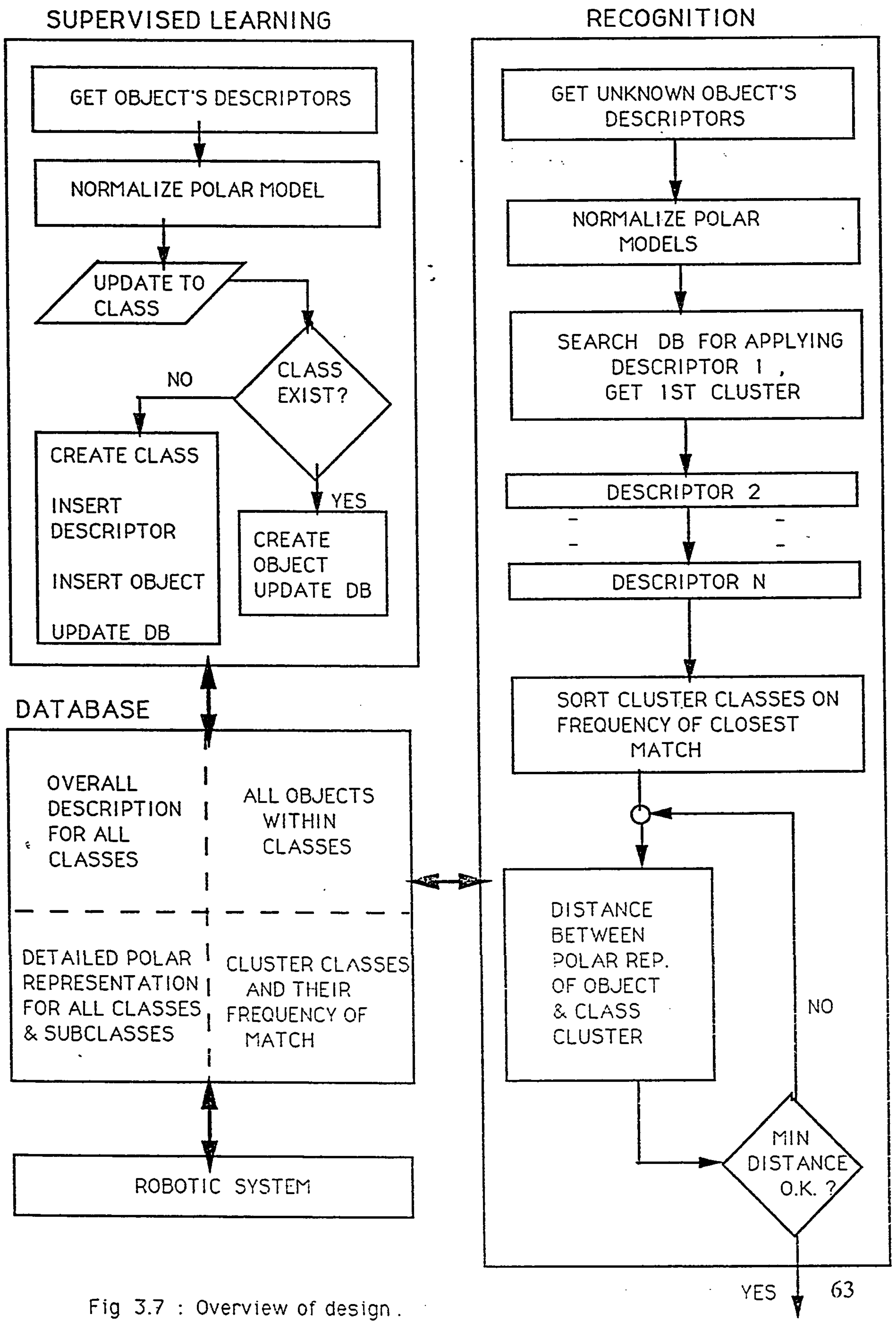

SUPERVISED LEARNING
GET OBJECT'S DESCRIPTORS
NORMALIZE POLAR MODEL
UPDATE TO CLASS
CLASS EXIST?
NO
CREATE CLASS
INSERT DESCRIPTOR
INSERT OBJECT
UPDATE DB
YES
CREATE OBJECT
UPDATE DB
DATABASE
OVERALL DESCRIPTION FOR ALL CLASSES
ALL OBJECTS WITHIN CLASSES
DETAILED POLAR REPRESENTATION FOR ALL CLASSES & SUBCLASSES
CLUSTER CLASSES AND THEIR FREQUENCY OF MATCH
ROBOTIC SYSTEM
RECOGNITION
GET UNKNOWN OBJECT'S DESCRIPTORS
NORMALIZE POLAR MODELS
SEARCH DB FOR APPLYING DESCRIPTOR 1, GET 1ST CLUSTER
DESCRIPTOR 2
DESCRIPTOR N
SORT CLUSTER CLASSES ON FREQUENCY OF CLOSEST MATCH
DISTANCE BETWEEN POLAR REP. OF OBJECT & CLASS CLUSTER
NO
MIN DISTANCE O.K. ?
YES

Fig 3.7 : Overview of design .

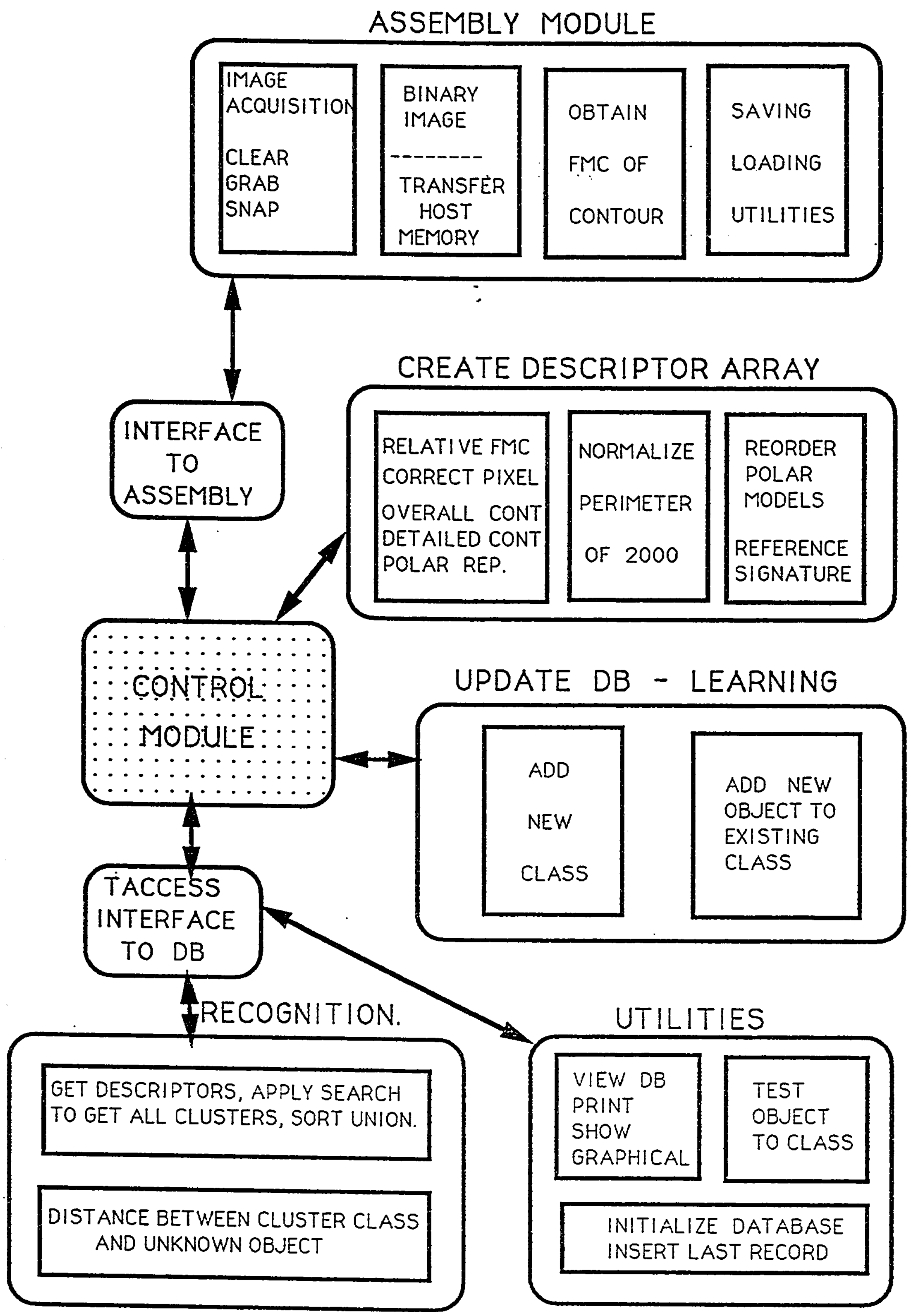


Fig 3.8 : Programming Modules.

3.3.1 Image Acquisition Utilities

The image acquisition menu will be of the following format :

```
Frame : from TV / Floppy
Image : Binary  / Not Binary
 FMC  : Present / Not Present
```

CURRENT MENU OPTIONS **:**

INIt	: set up new diskette
CLEar	: blanks out image
SNAp	: snap a new image
GRAb	: continuous acquisition
BINary	: convert to binary image
NEGate	: get negative of image
LOAd	: load image from floppy
SAVe	: save image to floppy
TV/Dsk	: image from FG-100 or floppy
FSD	: get FMC, save it & display edge
FDIsp	: get FMC & display edge
FSAve	: get FMC & save on floppy
FPUt	: save FMC on floppy
FGEt	: load & show FMC from floppy
Quit	: return to main menu

option = > ___

This menu is concerned mainly with obtaining an image and its FMC and saving it on a diskette in drive A. Both the binary image and its FMC can be saved or restored (a total of 8 of each on a non high density diskette).

INIT sets up a new formatted floppy in drive A and makes it ready for storage. It creates 8 files F1.bin, F2.bin, . . ., F8.bin which can store a binary image of 512×512 and the size of each file is 32 Kbytes. 8 files to store the FMC files are also created each of size 8 Kbytes where 8 K is the maximum possible contour of any object. These 16 files, all initialized to 0's, are created at specific sectors and their starting sectors are loaded in the FAT (file allocation table).

CLEAR will blank out the image frame of the FG-100 board, to prepare the unit for obtaining a new image.

SNAP takes a new image of an object is taken and the gray scale matrix is stored in frame buffer.

GRAB allows continuous acquisition of an object in real time; it is used mainly to allow positioning of the object, adjust lighting and contrast, change the camera lens' aperture and focus to ensure that object stands out clearly from the background.

BINARY converts a snapped image to its equivalent binary matrix using a user prompted threshold value and transferring the image from the FG-100 board to the host memory.

NEGATE obtains the negative of the binary image where each 0 pixel is turned to 1 and vice versa; this feature is useful if the object is white and the background is black since the contour extraction algorithm will assume that the image is always of a black object against a white backdrop.

SAVE will store the binary image from host memory to a user specified file on floppy in drive A.

LOAD will display a previously stored binary image from floppy and transfer it to FG-100 frame and display it on the TV monitor.

TV/DSk will allow processing (e.g extracting FMC) from either the real time image snapped from the FG-100 board or from a previously stored image on floppy.

FSD, FDISP, FSAVE processes the binary image to obtain the FMC of the contour of the object and keep the FMC temporarily on host memory, then either save it on floppy and / or display the contour only.

FGET will load a previously saved FMC of an object and display its contour only on the TV monitor. Other submenus will be used in the image acquisition step and they are concerned mainly with obtaining the threshold value and asking the user which file of floppy is needed; they are :

```
which file of diskette :
 01 : F1.bin (fmc)      05 : F5.bin (fmc)
 02 : F2.bin (fmc)      06 : F6.bin (fmc)
 03 : F3.bin (fmc)      07 : F7.bin (fmc)
 04 : F4.bin (fmc)      08 : F8.bin (fmc)
                 choice = > ___
enter threshold value = > ____ ( hex )
```

3.3.2 Graphical Utilities Menu

The graphical utilities allow the operator to view the object's contour graphically on the PC's monitor (he can already view it on the TV monitor but the scale is fixed and processing is slower); the operator can visually check if the contour was correctly extracted and if satisfied can decide to include it in the data base dictionary. The resolution of the PC's monitor is only 200 rows × 640 columns so that some scaling is required if the original object was 'big'. The menu will be :

- Show main image
- Show detailed image
- Move center of image
- Change scale of image
- Display the 'rho-theta' graphs
- Clear the PC's monitor
- Return to caller menu

Since two polar models of an object are kept, they can both be viewed on the monitor; the image can be shifted in any direction by the operator through the center option and can be enlarged or decreased through the scale option; the 'rho-theta' option shows the 'polar graph' of the 2 models of the object.

3.3.3 Descriptor Array

The operator is allowed to view and print the polar model representations of an object and print it; the menu is :

- Show main model
- Show detailed model
- Show 'rho-theta' graphs
- Print main model
- Print detailed model
- Show feature error
- Return to caller menu

Show main will display the coarse model of the object with the coordinates of all corners, lengths of each segment and angular changes between each two successive segments.

Show detailed does the same job as Show main except it involves the detailed model.

Show 'rho-theta' displays graphically the 'polar graph' of both models on the same screen to provide the operator with some insight to their difference and similarities.

Print main, detailed options allows a hard copy output of the representations on a line printer. The graphical screens can also be printed out by the use of the 'print screen' function available through DOS.

Show feature array list the feature array calculated for an object to the operator before he decides to include the object in the dictionary; the descriptors listed are the same as those given in Section 3.1.4

3.3.4 Supervised Learning

This is a control menu allowing the operator to access all the previous menus as well as adding an object to database and viewing the database; the options are :

- Process new object
- Graphical utilities
- Descriptor array
- Normalize & preview
- Update to database
- View database
- Exit from control menu

Process new object allows the operator to load the FMC of an object, convert to RFMC and obtain the polar models and descriptors.

Graphical utilities and descriptor array have already been discussed in the previous subsections.

Normalize allows the models to be normalized to a fixed relative perimeter i.e. 2000 pixels.

Update allows the operator to insert the new object to the database in an existing class or in a new class.

View permits viewing of all database relations; a sub menu is :

- Specific class - feature array & detailed model
- all objects in database - to which class
- all classes in database - all subobjects
- all detailed models
- all indexed files - sorted on each descriptor

3.3.5 Automatic Recognition & Timing

Given the FMC an an unknown object, the automatic recognition step will go through all the identification steps and come up with the matching class. The operator can see the results of each decision rule (which class was found for that descriptor), the final sorted cluster of classes (class and hit frequency). Further, timing statistics are also available about how long it takes to obtain the feature array, to normalize, to apply the decision rules, to sort the final cluster, to make the minimum distance match , etc.

CHAPTER 4

IMAGE ACQUISITION

4.1 FG-100 Digitizing Unit

The series 100 video source interface was set up with the IBM-AT as shown in the pictorial sketch of Fig 4.1 . The output of the FG-100 was not in color but monotone (green).

The video interface performs 3 main functions :

a) signal conditioning : it samples the black level of the input video signal (from camera) and corrects it for any DC offsets resulting in good picture quality.

b) digitization : an 8-bit flash analog to digital converter (A/D) samples the analog video signal at discrete time intervals and converts each sample to a digital value (pixel). The incoming signal (from 0 to 714 milli volts) is sampled at 10 MHz and produces an array of 512×512 pixels each with a value between 0 and 255.

c) synchronization and timing.

4.1.1 FG-100 Interface

The overall interface of the FG-100 board with the host IBM-AT is shown in Fig 4.2; note that :

a) all registers of FG-100 are I/O mapped into the host computer, occupying sixteen words within its 512 words I/O channel. The register base address is set at 0300 (hexadecimal).

b) the memory and look up table (LUT) are memory mapped occupying either 64 K or 512 K.

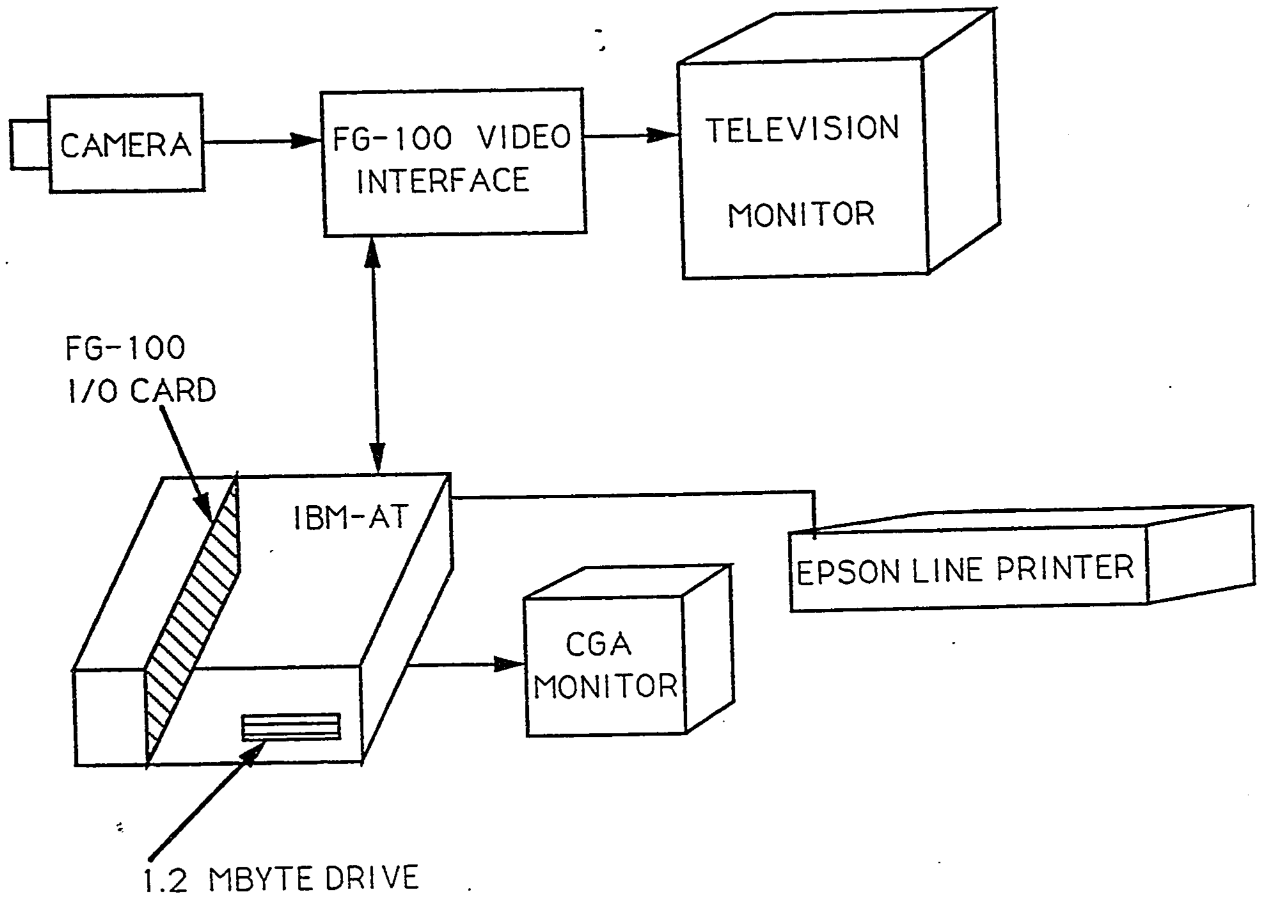


Fig 4.1 : The Setup

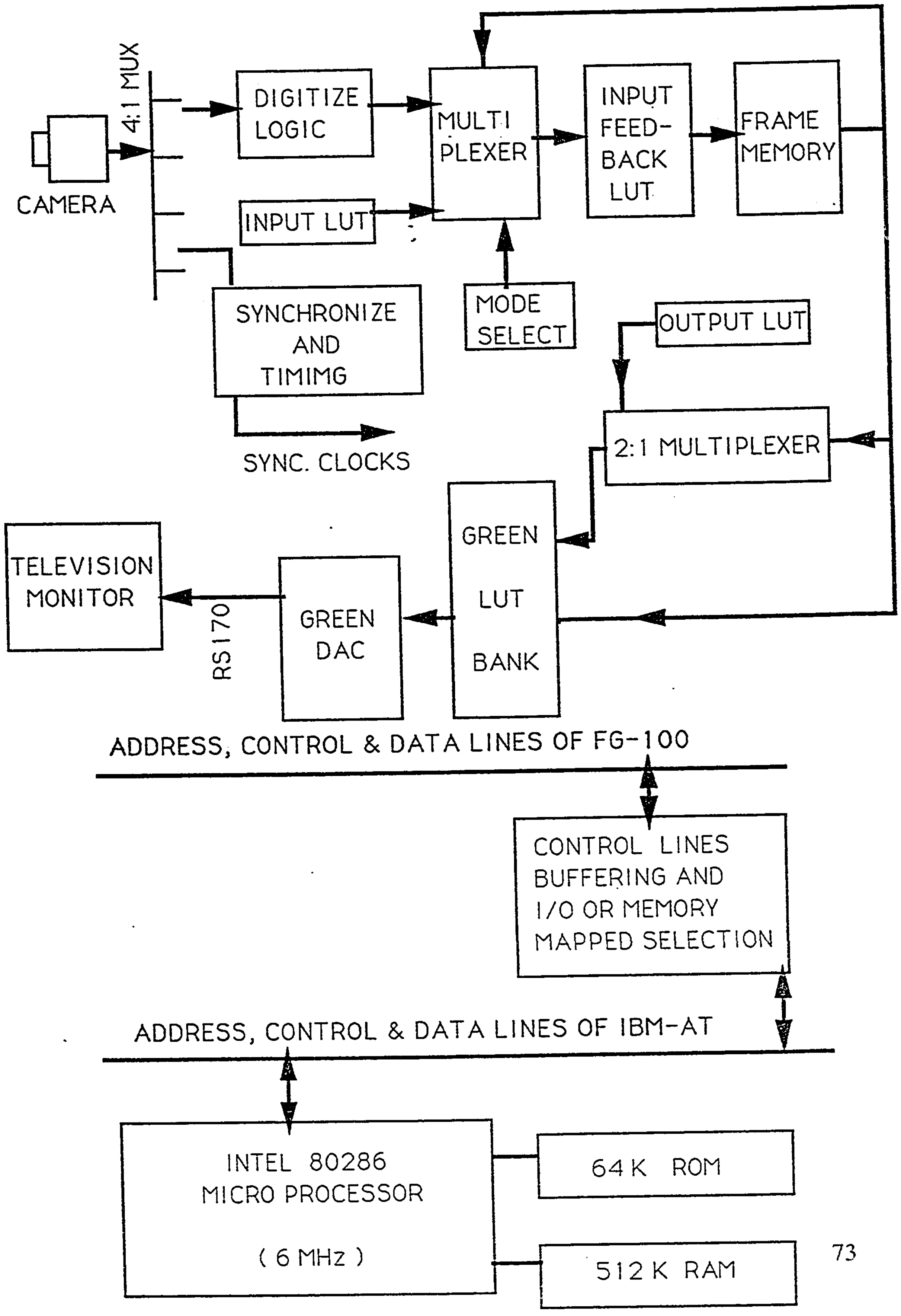


Fig 4.2 : Hardware Interface

Both the memory base address and the size of the address space are selectable; the board base address was set to A0000 (hex) and the block size accessible is 64 K. (since DOS versions 3.1 and earlier only allow addressing of 64 K blocks only at a time).

4.1.2 Access of FG-100 Registers

The 16 registers of FG-100 are I/O mapped to the host PC. The FG-100 can store up to 4 frames of images; the Active Video Window (AVW) is that frame memory which is displayed. The registers are shown in Fig 4.3 . All the registers are initialized to the correct settings (Appendix A-1) to allow access of host to the FG-100 board. A register can be set to a value by using the OUT command as follows :

```
MOV  DX,0300 H
MOV  AX,4040 H
OUT  DX,AX
```

The address of the register to be set is loaded in DX, the contents to be set is loaded in AX and the OUT command set the value directly through the output port.

The basic functions and settings of each register are briefly explained next, only the functions which need resetting will be discussed in further details.

<u>Memory Access Control Register</u> specifies various functions used to access frame memory from the host PC.

Bits 4,12 : 0 (host computer can access in dual scan mode)

Bits 6,14 : 1 (host can write into pixel buffer area)

Register set to 4040(h)

REGISTER BASE ADDRESS : 0300 (H)

OFFSET

0	Memory Access Control
2	Host Mask
4	Video Acquisition Mask
6	Pixel Buffer Register
8	X Pointer
A	Y Pointer
C	Pointer Control
E	CPU Address Control
10	X Spin
12	Y Spin
14	Pan A
16	Look Up Tables

Fig 4.3 : FG-100 Registers

Host Mask Register

Bits 0-15 : 0 (bit plane can be modified by host write and clear operations)

Register set to 0000(h)

Video Acquisition Mask Register

Bits 5-15 : 1 (bit plane not modified during image acquisition)

Register set to 07FF(h)

X Pointer Register is used in indirect addressing scheme to access frame memory and holds the horizontal address of pixel; the X Pointer Register holds an offset used to move a window around in frame memory.

Bits 0-7 : 0 (direct addressing used, no offset required)

Register set to 0000(h)

Y Pointer Register , as above, holds the vertical address of pixel.

Bits 0-7 : 0 (direct addressing scheme)

Register set to 0000(h)

Pointer Control Register

Bit 1 : 0 (auto increment of pointer address)

Bits 8-10 : 100 (increment X pointer by 16)

Bits 13-15 : 100 (increment Y pointer by 16)

Register set to 0044(h)

CPU Address Control Register

Bit 0 : 1 (host computer address mapped into frame memory)

Bits 1-4 : 0011 (window size of 512 by 512 accessed)

Bits 9,10 : 10 (host memory address scaled by 1 to/from memory address)

Register set to 0407(h)

X Spin Constant Register is used to set the zoom factor of 1 (image is not magnified)

Register set to 0010(h)

Y Spin Constant Register

Register set to 0010(h)

Pan-A Register

Bit 12 : 1 (frame memory selected for access)

Bit 13 : 1 (board's memory enabled onto host computer)

Bit 15 : 0 (one active video window)

Register set to 3000(h)

4.1.3 Program Listing Convention

Some problems are encountered when programs written in Pascal are printed with this word processor due to the fact that a ; has a special editing function; so instead the usual ; of Pascal will be replaced by a : in this manuscript for all Pascal listings. Similarly all comment messages which should utilize { and } will be substituted by [and] .

4.2 Frame Buffer Acquisition

Once all the registers are set such that the host can directly access the image frame of the FG-100, different operations can be carried out; as previously noted, the AVW can be accessed directly from the host's address lines.

The next three operations make use of the Board Status / Control / Scroll B Register; the bits which are crucial are :

ACQUIRE Bits 13,12 :   ACQ

0 0 :	a clear or snap operation complete
0 1 :	initialize a clear frame
1 0 :	acquire a single frame
1 1 :	grab image continuously

Once a command is initiated by setting the corresponding ACQ bits of the register, the ACQ bits have to reset to 0 0 ; this takes some time and a delay is required. This is achieved by a do-nothing loop :

```
        MOV  CX, 0FFFFH    : set delay counter
L1:     XOR  BX, BX        : do 'nothing'
        LOOP L1            : decrement counter to 0
```

4.2.1 Clear, Snap, Grab Operations

CLEAR : operation clears the contents of the AVW and sets each pixel value to 0 to prepare for the acquisition of a new image. This is done by setting the ACQ bits to 0 1 and waiting until they go to 0 0 and checking if the image frame has been cleared (see Appendix A2-a).

SNAP : Operation takes a 'freezed' image of an object. This is initialized by setting the ACQ bits and waiting for them to reset and then setting them to 1 0

GRAB : is issued directly without delay for ACQ bits to reset by setting the bits to 1 1 .

These operations are very fast unlike other operations which process the frame pixel by pixel rather than in a 'flash' operation. An understanding of the image frame and addressing it is necessary.

4.2.2 Buffer Frame of FG-100

The 512 rows by 512 columns of the frame is mapped to the memory space of the host computer at address A0000(h) and each pixel is stored in 2 consecutive bytes where the first byte keeps the digitized gray level value and the second keeps some information about color if an extended FG-100 board is used; the board used in this thesis was a monocolor one.

The main problem for the host in accessing the frame is the limit imposed by DOS versions 3.1 or earlier, which allow access of only 64 Kbytes. Referring to Fig 4.4, the frame is divided into 8 'strips' for explanatory reasons; the size of one strip is 64 rows by 512 columns and occupy a memory space of $64 \times 512 \times 2$ bytes or 65,536 bytes, conveniently the same 64 K size block accessible by DOS.

The memory base address is A0000(h) and is set as such by the Data Segment Register (DS) in Assembly while which one strip is accessed has to be set to 0000h, 0040h, . . , 01C0h for strip1, strip2, . . , strip 8 respectively; this is set by changing the Y Pointer Register of the FG-100 .

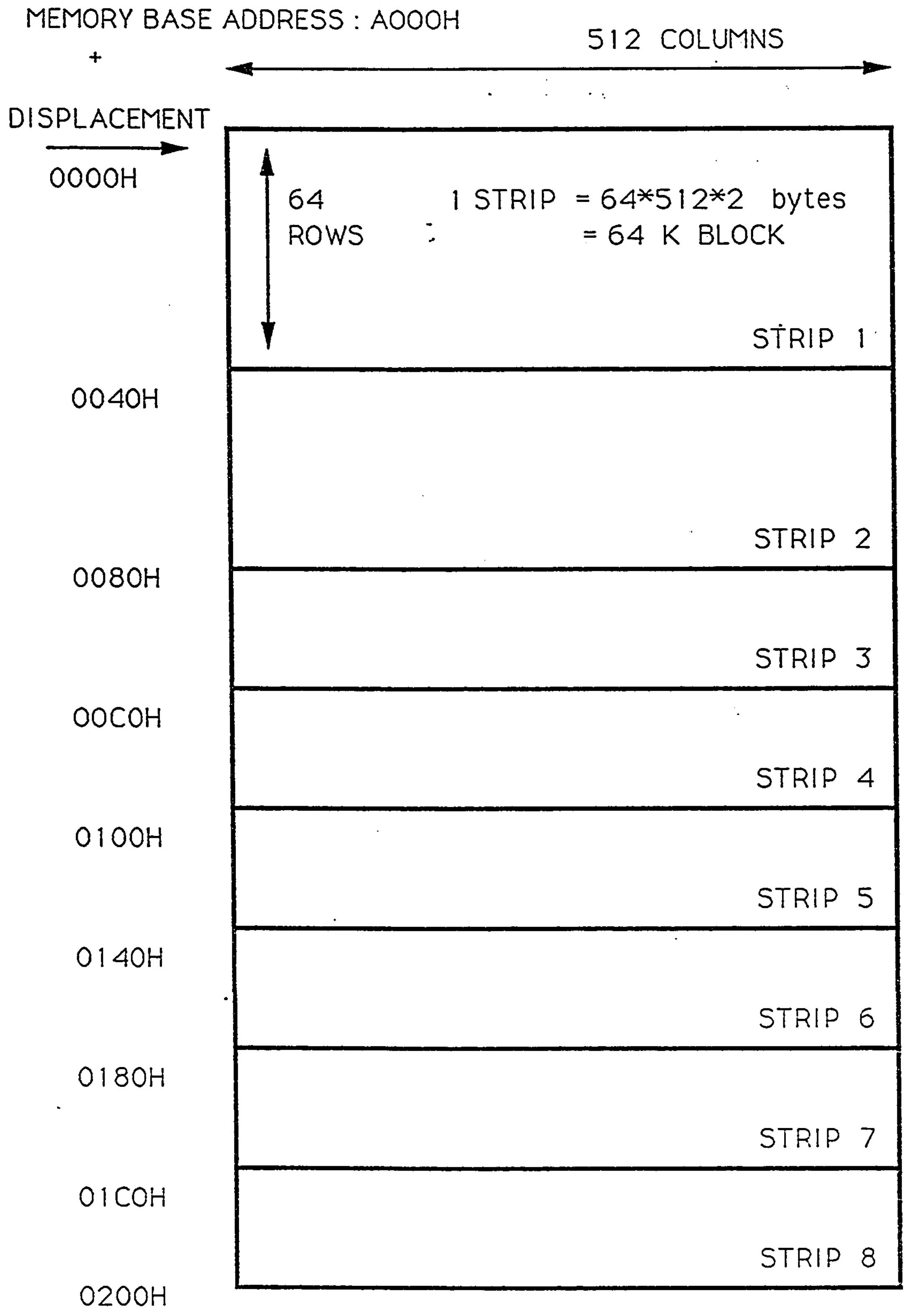


Fig 4.4 : frame Image

4.2.3 Pixel by Pixel Access of frame

Many of the operations will involve visiting each pixel in the frame going from row 1 to row 512 and for each row going from column 1 to column 512 where row 1, column 1 correspond to the top left most corner pixel of the image (Fig 4.5). 'Visiting' each pixel (either read or write) involves setting the memory base address to A0000h, selecting which of the 8 strips is currently used and which individual pixel within that strip is being pointed to. Note that consecutive rows of pixels are stored in frame memory in contiguous 2-byte locations.

All operations which need to visit the image on a pixel by pixel basis have the same programming structure as the one given in the template of Fig 4.6.

Let BX : pointer to one of the 8 strips

& DI : pointer to individual pixel within a strip

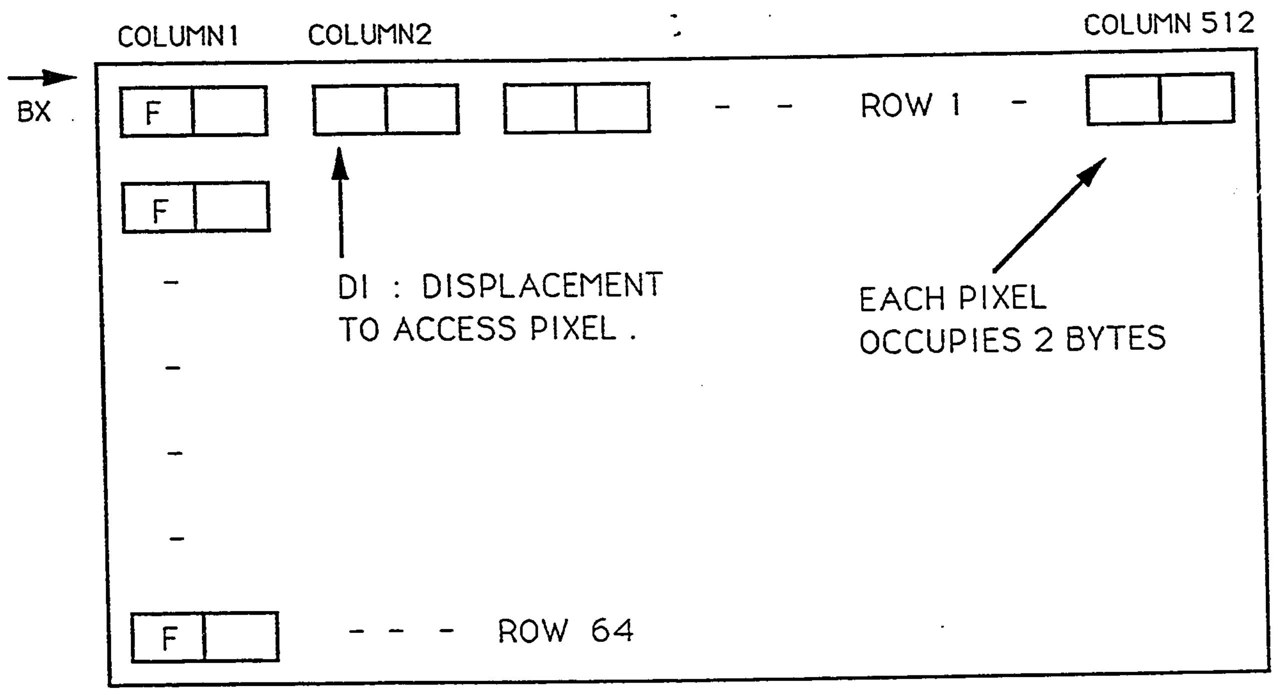
The 2 basic operations are :

* accessing next pixel is done by incrementing DI by 2, DI cannot be greater than 65,535 since one strip is only 64 Kbytes.
* accessing next strip is done by incrementing BX by 0040h, since only 8 strips are available, BX must be less than 0200h.

Henceforth, discussion of the operations will not show the programming for accessing individual strip or pixel since they all follow the same template; for the sake of brevity and ease, only the operations on each pixel will be explained assuming that the pixel value has been obtained by the algorithm of Fig 4.6 .

MEMORY BASE ADDRESS : A0000H

DISPLACEMENT TO ACCESS STRIP : BX



COLUMN1
COLUMN2
COLUMN 512
BX
F
ROW 1
F
DI : DISPLACEMENT TO ACCESS PIXEL .
EACH PIXEL OCCUPIES 2 BYTES
F
ROW 64

AN INDIVIDUAL PIXEL ADDRESS IS GIVEN BY :

= MEMORY BASE ADDRESS A0000H

+ CONTENTS OF REGISTER BX

+ CONTENTS OF REGISTER DI

Fig 4.5 : Accessing Pixel within Strip

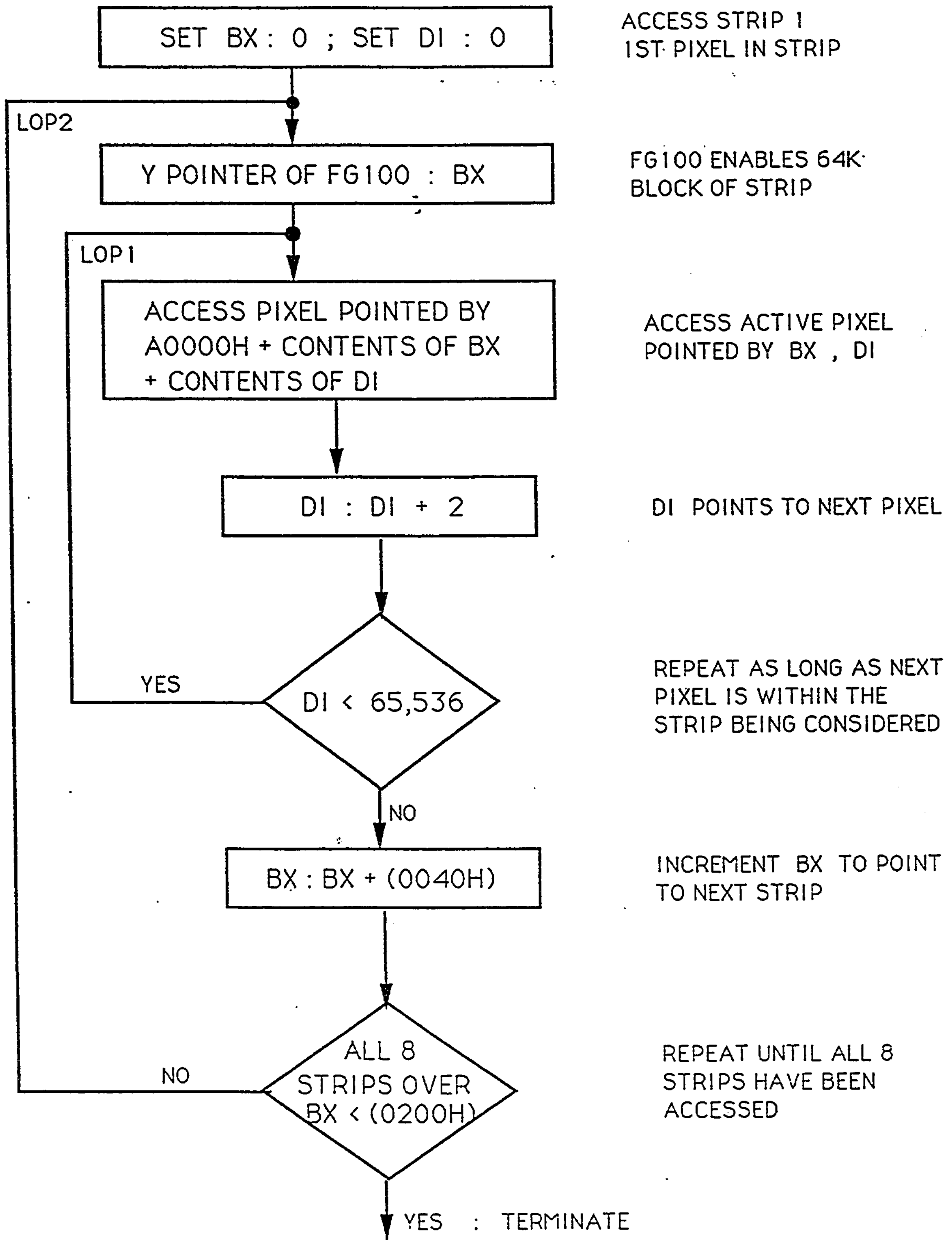

SET BX : 0 ; SET DI : 0
ACCESS STRIP 1
1ST PIXEL IN STRIP
LOP2
Y POINTER OF FG100 : BX
FG100 ENABLES 64K BLOCK OF STRIP
LOP1
ACCESS PIXEL POINTED BY A0000H + CONTENTS OF BX + CONTENTS OF DI
ACCESS ACTIVE PIXEL POINTED BY BX , DI
DI : DI + 2
DI POINTS TO NEXT PIXEL
YES
DI < 65,536
REPEAT AS LONG AS NEXT PIXEL IS WITHIN THE STRIP BEING CONSIDERED
NO
BX : BX + (0040H)
INCREMENT BX TO POINT TO NEXT STRIP
NO
ALL 8 STRIPS OVER BX < (0200H)
REPEAT UNTIL ALL 8 STRIPS HAVE BEEN ACCESSED
YES : TERMINATE

Fig 4.6 : Pixel-by-pixel Access Template

4.2.4 Negative / Binary Operations

NEGATIVE IMAGE

This procedure reverses each pixel gray level value, i.e. a black pixel is set to white and vice versa. Each pixel is examined once, complemented and written back in frame memory (Appendix A3)

```
"READ  PIXEL TO AL"      : AS IN FIG 4.6
NOT  AL
INC  AL                  : REVERSE PIXEL VALUE
ADD  AL,0FFH
"WRITE  PIXEL TO AL"     : AS IN FIG 4.6
```

BINARY IMAGE

This compares each pixel gray level value to a threshold value (cutoff) and sets the pixel to 1 if its grey level is higher than the threshold and 0 otherwise. The algorithm (Appendix A4) is :

```
"READ PIXEL TO AL"  : AS IN FIG 4.6
COMPARE AL TO CUTOFF VALUE IN CL
IF AL  >  CL , SET PIXEL TO FFh ( WHITE )
      ELSE   SET PIXEL TO 00H ( BLACK )
"WRITE PIXEL BACK TO FRAME"
```

Note that the operator is given the choice of selecting any cutoff value (stored in register CL). This is done by interactively asking the operator to enter a cutoff value (in hexadecimal) which is echoed back to the monitor. The procedure to read a hexadecimal byte (2 digits between 0 - 9 , A - F) is given in Appendix A5.

4.3 Memory Areas Used (Frame, SECTOR, FMCOUT)

BINARY IMAGE SAVED IN HOST COMPUTER

Accessing of the FG-100 frame by the host PC is very tedious and slow as it involves changing the strip address 8 times. Furthermore each pixel occupies 1 byte of memory in the frame. To make the contour extraction, the saving and coding of images more efficient, it is needed to store the image as a 512 rows by 512 columns by 1 bit (0 or 1) to a memory area of size 32 Kbytes (this area is denoted as 'SECTOR').

'SECTOR' area is illustrated in Fig 4.7, we note that each 8 consecutive pixels are stored (packed) into 1 byte. One row of pixels therefore occupies 512/8 or 64 bytes. The packing procedure is very similar to the binary conversion procedure of Appendix A4 except that after comparing each gray level pixel to cutoff and setting it to 0 or 1, the resulting binary pixel has to be 'packed' , 8 pixels at a time into 1 byte and once the byte is packed, it has to be written into SECTOR area of host PC's memory. The packing is done by shifting each successive pixel right into the packed byte.

'FMCOUT' memory area is reserved to store the FMC contour of the object, the first 2 bytes contain the number of pixels on the contour, the next 4 bytes contain the coordinates of the start pixel (row, column) and the rest of the area contains the FMC of all border pixels.

PACKING procedure given in Appendix A6 does just that. The flowchart is given in Fig 4.8 (the pixel access is not given; it is as the one in Fig 4.8). One important step before accessing image in SECTOR for contour extraction is the BLANKING OUT OF COLUMN 1 of the image; FG-100 digitizer sets all pixels in column 1 to a value of FFh indicate beginning of a new row.

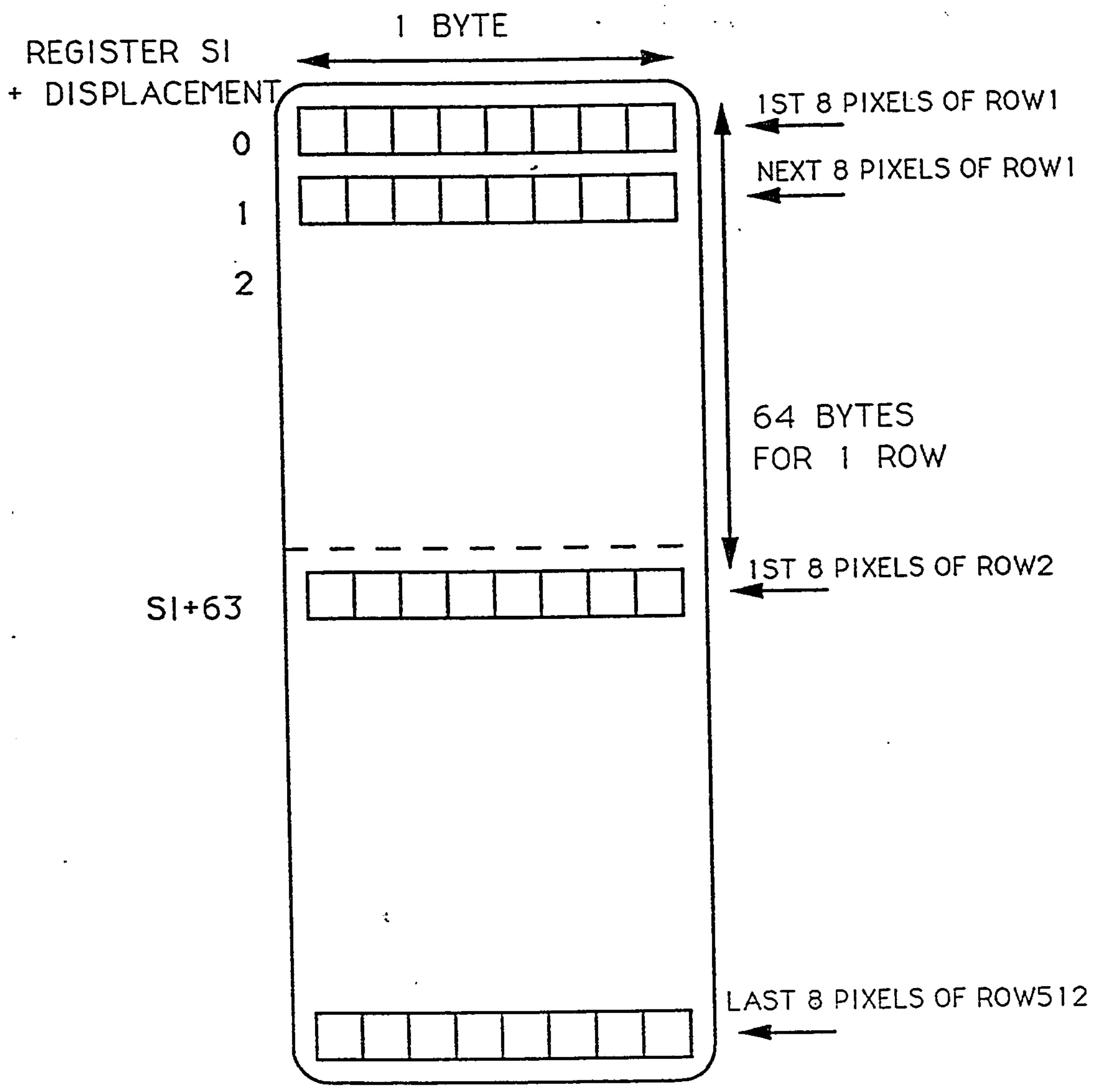


Fig 4.7 : 'SECTOR' host area for binary image

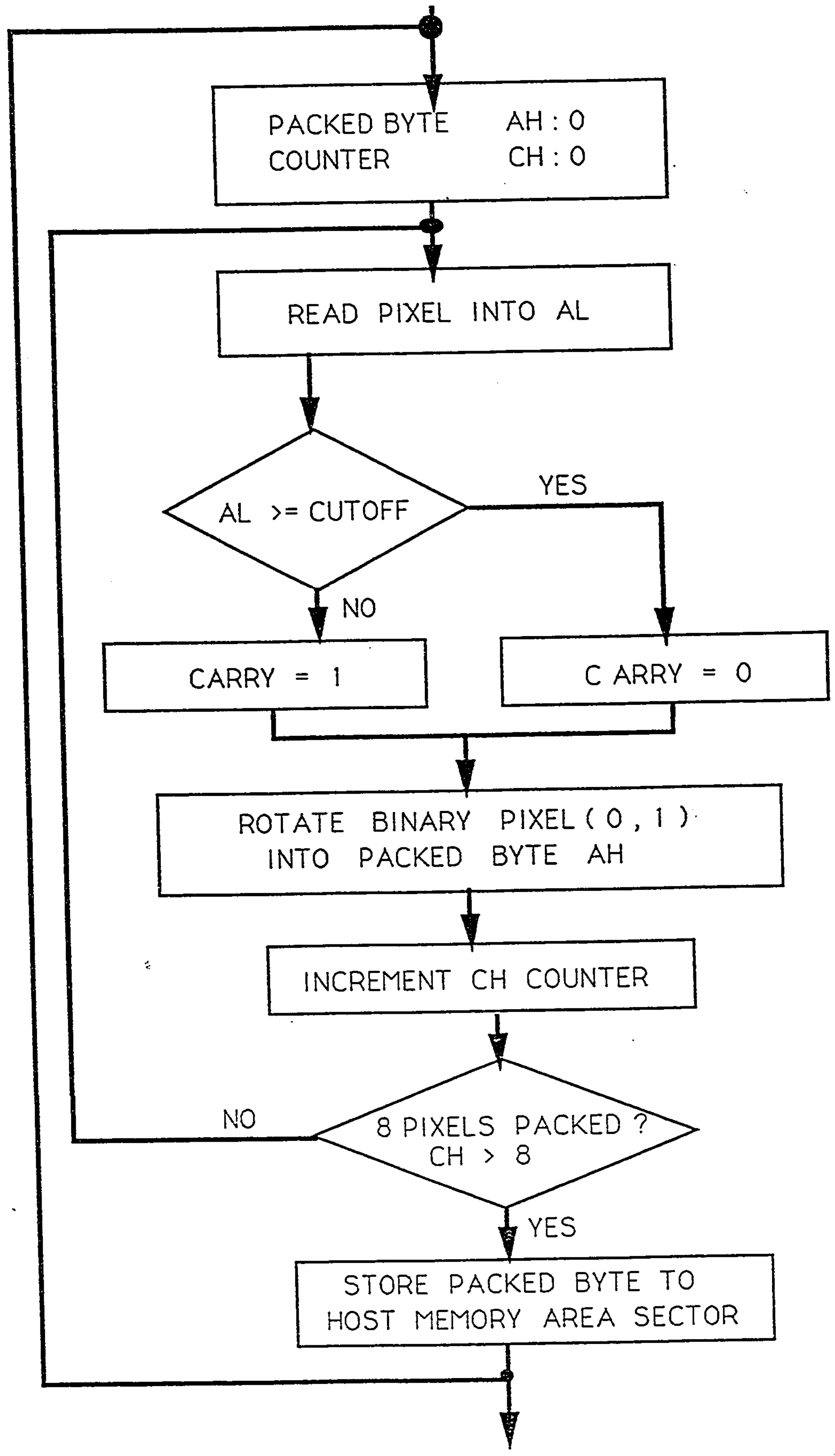


Fig 4.8 : Algorithm to pack the gray-level image to binary one in host memory.

since column 1 will always be on the background of the object, it has to be set to 00h and this is dine in procedure BLANKCOL in Appendix A7. Only the binary bit of column 1 in SECTOR is cleared, not the byte in the FG-100 frame.

UNPACKING procedure, given in Appendix A8, does exactly the opposite of the packing procedure; it unpacks a packed byte of 8 binary pixel values and puts the values of each pixel back to 1 byte board frame at the corresponding row and column. This is used to view images, previously saved binary images (on floppies) on the television monitor.

In short the Assembly language is accessing 3 main memory areas :

i) frame area of FG-100 - 512 K of gray level pixels.

ii) SECTOR area in local memory of PC - 32 K of binary pixels. This area contains 0 for background and 1 for object and will be used to obtain the FMC chain. One row takes 64 bytes.

iii) FMCOUT local memory area which is reserved for -

a) number of points on border .

b) X, Y coordinates of start pixel - X_0 , Y_0

c) the FMC of the object's contour

Since the contour length in pixels of any object is limited by the 512×512 image size, the area of FMCOUT was set to a maximum possible value of 8 Kbytes.

4.4 Diskette Saving / Loading

Referring back to the image acquisition menu proposed in section 3.4.1, the

CLEAR, SNAP, GRAB, BINARY and NEGATE have been explained. The next related batch of procedures to be expanded on in this section are those concerned with utilities for loading / saving the binary image or the FMC.

The utilities for loading / saving from / to a diskette in drive A involves 2 groups :

i) saving / loading binary image

ii) saving / loading FMC of the object's contour

The binary image is saved from the SECTOR area to a file on diskette e.g F1.bin and loading a binary image transfers a file to SECTOR area; the FMC operations involve swapping between FMCOUT area and files on diskette e.g F1.fmc

A double-sided density diskette has the following capacity :

<u>Sector</u>		
0	:	Boot Area for DOS
1-4	:	File Allocation Table (FAT)
5-11	:	Directory Area
12-719	:	File Area

Sectors 12 - 719 (each sector of 512 bytes) are available for data storage or a total of 362,496 bytes. This allows for the storage of 8 'binary' files and 8 'fmc' files i.e. F1.bin, f2.bin, . . ., f8.bin and f1.fmc, f2.fmc, . . ., f8.fmc . Each .bin file occupies 32,768 bytes and each .fmc file takes up 8192 bytes corresponding naturally to the 'SECTOR' area and 'FMCOUT' area used in the Assembly program. Operations for loading from or saving to a file in Assembly require the exact starting sector of each file. These values are given in Table 4.1.

FILE	HEXADEC	DECIMAL
F1.bin	000C	12
F2.bin	004C	76
F3.bin	008C	140
F4.bin	00CC	204
F5.bin	010C	268
F6.bin	014C	332
F7.bin	018C	396
F8.bin	01CC	460
F1.fmc	020C	524
F2.fmc	021C	540
F3.fmc	022C	556
F4.fmc	023C	572
F5.fmc	024C	588
F6.fmc	025C	604
F7.fmc	026C	620
F8.fmc	027C	636

Table 4.1 : Diskette files & addresses

4.4.1 Initializing Diskette

The INIT procedure prepares all .bin and .fmc files on a newly formatted empty diskette in drive A. The procedure reads the diskette, checks if it is empty and creates the files sequentially.

A typical creation of a file will be :

```
var
        j    : integer :
        fb1  : file :
        buf  : array [  1 . . 1024  ]  of boolean :
        writ : word :
begin
        for j:=  1 to 1024         [ buffer area contains all 0's ]
            buf  [ j ]  :=  false :
        assign(fb1,'A:f1.fmc') :  [ create a file on diskette ]
        rewrite(fb1) :
        reset(fb1,1024) :
        for j:=  1 to 8            [ put all 0's in file ]
            blockwrite(fb1,buf,1,writ) :
        close(fb1) :
```

This file creation procedure is repeated for all 16 files, the 8 .bin files are created first sequentially followed by the 8 .fmc files such that they occupy contiguous sectors specified in Table 4.1. Each file contains all 0's.

4.4.2 Diskette Files

All operations involving diskette allows the operator to interactively select which file to operate on. The 16 files are listed on the menu and the user chooses a number between 1 and 8 to select the desired file. An Assembly procedure converts the user's choice to the starting sector of that file.

The procedure <u>READSTART</u> (Appendix A9-a)reads a .bin file number and calculates its starting sector in hexadecimal (Table 4.1) between 000C and 01CC as :

start sector = (filenumber - 1) * 64 + 12

e.g if F3.bin is required and the operator chooses '3', then

SECSTART = (3 - 1) * 64 + 12 = 140 (decimal)

The procedure <u>READSTART</u> (Appendix A9-b)reads a .fmc file number and calculates its starting sector as :

start sector = (filenumber - 1) * 16 + 524

e.g if F4.bin is required and the operator chooses '4', then

SECSTART = (4 - 1) * 16 + 524 = 572 (decimal)

4.4.3 Saving / Loading Diskette files

Once the operator selects the file to work upon and the start sector calculated, the binary image (SECTOR) or Freeman Contour (FMCOUT) is transferred to / from the diskette.

The saving operation is carried out by the following set of Assembly statements :

```
" save all registers "
    MOV   AL,00H        : SELECT DRIVE A
    MOV   CX,0040H       : BLOCK SIZE OF 64
MOV   DX,SECSTART     : STARTING SECTOR OF BLOCK
LEA   BX,SECTOR       : ADDRESS OF BLOCK TO MOVE
INT   26H             : MOVE BLOCK TO DISKETTE
POPF
" restore all registers "
```

The above group of instructions moves a block of memory 'SECTOR' whose address is loaded in register BX to a block of sectors of diskette A (in register AL) whose first sector is in register DX; the size of the block to be transferred (in sectors) is in register CX and the write operation is carried out by the DOS Interrupt of 26H. The above procedure (see Appendix A10-a) moves the binary image from SECTOR to a specific file on diskette.

The routine given in Appendix A11-a is basically the same except that the area moved from memory is FMCOUT and the block size is only 16; the block moved contains the Freeman code of the object.

The loading procedures are similar to the saving procedure given above with the difference that Interrupt 25H is used which moves a number of sectors from diskette to a block in memory.

Appendix A10-b reads the binary image from floppy to memory area SECTOR and Appendix A11-b moves a previously saved FMC contour from floppy to memory area FMCOUT.

4.5 Obtaining FMC

Once the binary image is obtained from the previous algorithm, (Fig 2.7-a), it is necessary to obtain the FMC of the contour (Fig 2.7-b). Different methods exist to follow the contour of a binary object; all of them assume that the convolution mask (Fig 4.10-a) is initially centered on a start pixel of the contour and the next pixel on the contour is detected and the frame centered on that new pixel with the FMC as the direction from the original frame center to the new frame center; this step is repeated until the start pixel is met again : to know if the contour is totally extracted, the row and column numbers of the start pixel as well as those for the current mask center has to be known, at least temporarily - they are not needed once the tracking algorithm is completed.

4.5.1 DUDA's Method

DUDA's method is the simplest of contour tracking algorithm and the algorithm is given below :

1. start at a pixel on contour.
2. if in a "1" pixel, turn left and take a step.
3. if in a "0" pixel, turn right and take a step.
4. repeat steps 2,3 and terminate when start pixel reached.

Fig 4.9 illustrates the operation of the DUDA's method. Some problems exist [DUD73]:

i) note that the top left pixel (which is 8-connected) is never visited and completely ignored.

ii) holes in the figure might cause the algorithm to loop forever.

iii) the contour tracked depends on the start pixel
e.g. the top left pixel would have been visited if pixel(4,5) had been entered from the bottom instead of from the side.

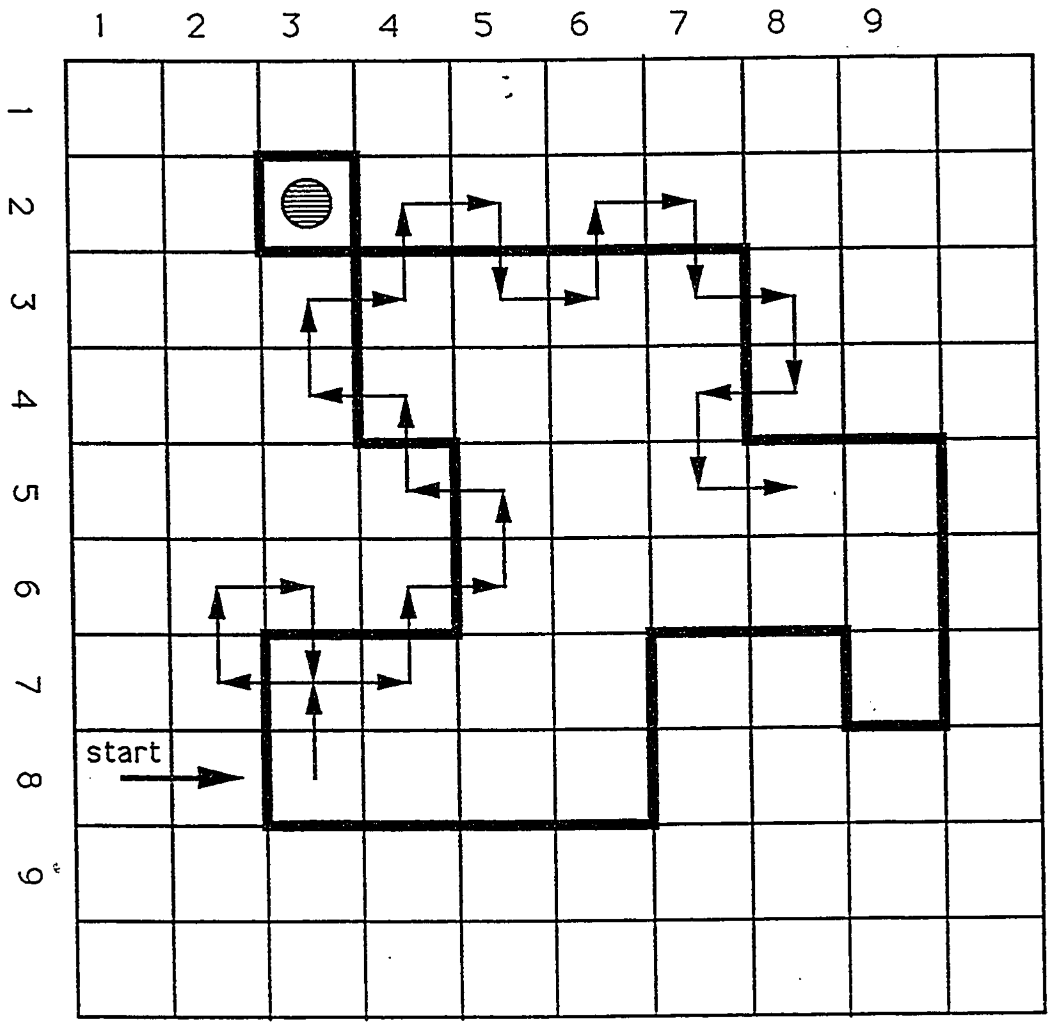


Fig 4.9 : DUDA's Method

The problems are the result of the extreme simplicity of the method; only 3 bits are used at each point to decide which way to turn : 2 bits to indicate the direction the pixel was entered and 1 bit to determine if the pixel is 0 or 1. Other contour following algorithms avoid those problems but examine more pixels surrounding the current pixel and hence need more computations.

4.5.2 Background-to-Object Transition

A more robust algorithm is based on detecting a background to object or 0-to-1 transition. Referring to Fig 4.10-b, assuming that the mask is centered on pixel(3,3), the task is to find the next pixel on the contour. The algorithm is :

```
    check pixels[b,a] = [0,1] -> next pixel = a
    check pixels[a,d] = [0,1] -> next pixel = d
    check pixels[d,g] = [0,1] -> next pixel = g
    etc
```

This algorithm predictably checks pixel pairs [b,a], [a,d], [d,g], [g,h], [h,i], (i,f], [f,e] and [c,b] in this fixed anti clockwise order.

The main drawback is the slowness of the algorithm, i.e. if the 3×3 convolution mask is centered on pixel(5,4), it is needed to check 8 pairs of pixels (worst case) to determine that the next pixel is pixel(4,4).

An algorithm which accelerates the detection of the next pixel is more effective and the one implemented in this thesis is faster because it takes into account the current direction of the contour to minimize the pixel pairs to be checked.

A	B	C
D	E	F
G	H	I

A) 3 by 3 convolution mask

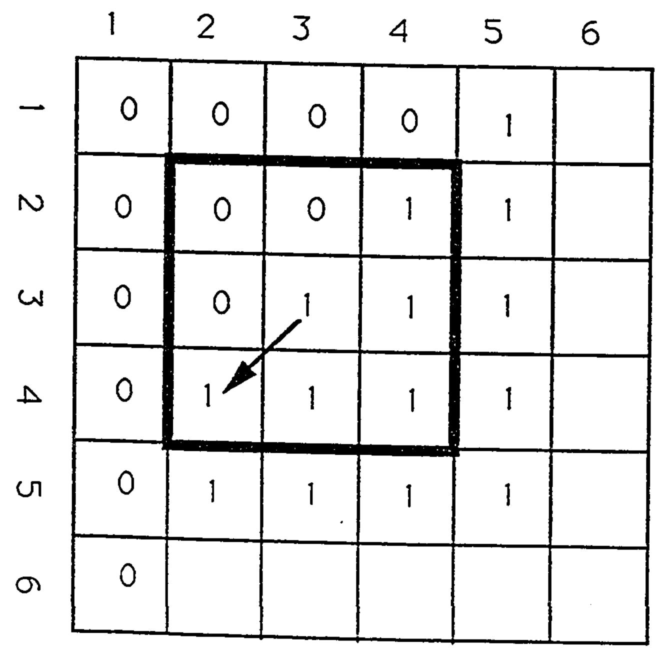


Fig 4.10 Findind next contour pixel

4.5.3 Directional 0-to-1 Transition

The algorithm implemented takes into account the previous direction of the contour or the FMC direction of the last 2 pixels on the contour and looks for the next border pixel in the same direction; the argument is that in the binary image the contour is usually digitized as numerous long straight segments with a certain regularity along each segment - the probability of finding the next pixel along the same direction as a current segment is much higher.

Referring to Fig 4.11 and let the 3×3 mask be centered on pixel(2,3) and let the direction be 5 (this is direction of current pixel(2,3) with respect to previous pixel(1,4)); the algorithm will first check pixel pairs[d,g] before any other pixel pair because the likelihood that pixel g is the next border pixel is very high. In this case pixel pair[d,g] = [0,1] and the next pixel g is found in only 1 pair match versus the 3 pair matches required in the previous algorithm.

A flowchart for the routine (DIR5) to detect a next pixel along direction 5 is given in Fig 4.12 where the 3×3 mask is centered on pixel e. First pixel d is checked , if it is equal to 0, pixel g is checked for 1 and if so the next pixel is g and the FMC is still 5; most of the time the only pixel pair to be checked will be d and g and the small loop (left of flowchart) will be repeated continuously as long as the direction 5 is maintained. Only if the next pixel in direction 5 is not found will the other pixel pairs be checked. If the next pixel is in another direction, i.e direction 6, then DIR5 routine outputs and FMC value of 6 and transfers control to another routine DIR6 where the job is continued. DIR6 routine will check pixel pair[g,h] as first candidate because pixel h lies along direction 6.

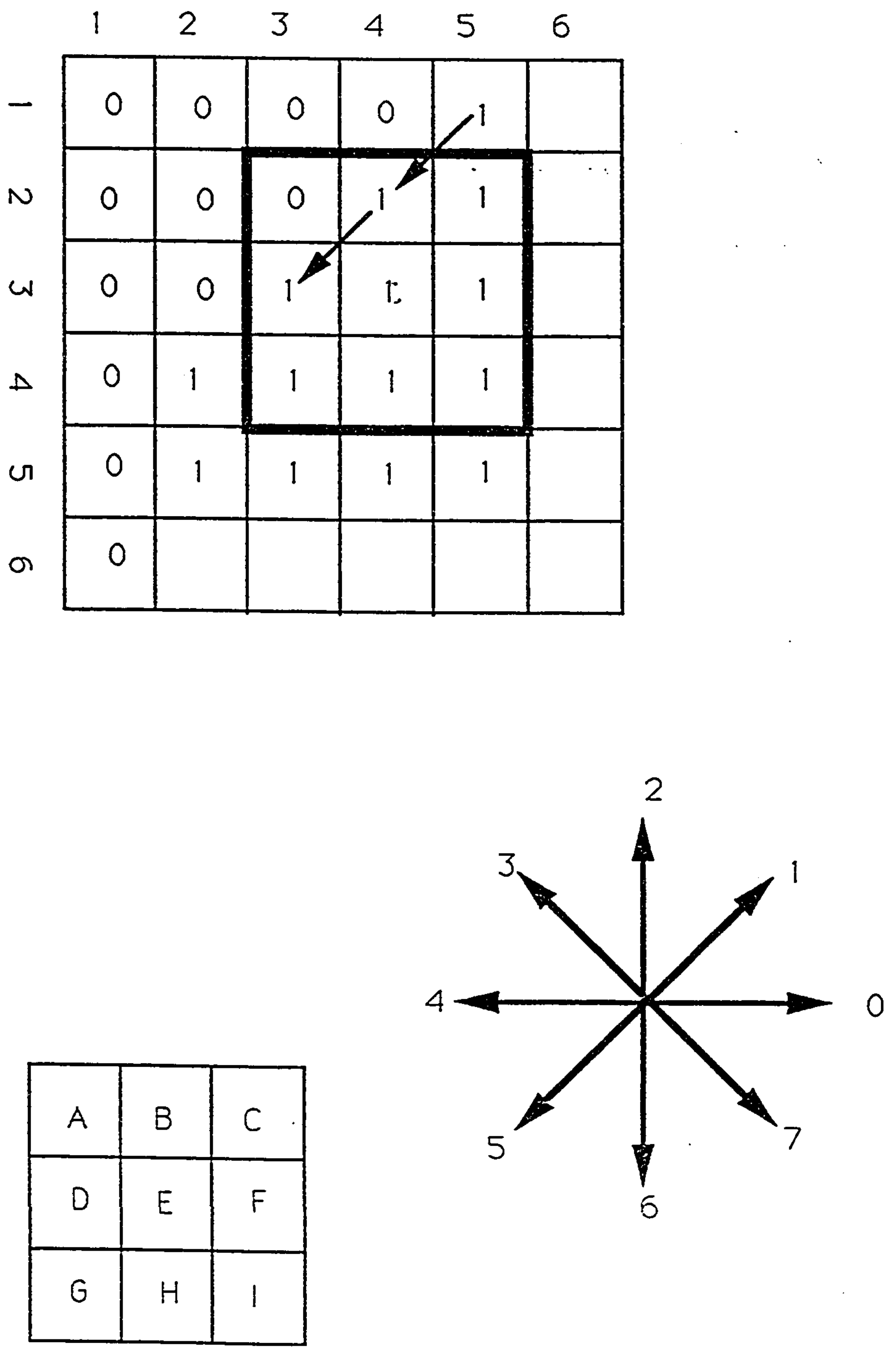


Fig 4.11 : Directional 0-to-1 transition

Overall 8 such routines, DIR0, DIR1, DIR2, DIR3, DIR4, DIR5 ,DIR6, DIR7 exist. It might be argued that this is too long but the point here is that only the length of the Assembly code is long but that the speed of execution of the overall tracking algorithm is faster since most of the time only a small loop in each routine will be running over and over; the other steps of the routines, i.e. checking the other pixel pairs not along the current direction have to be checked anyway no matter what algorithm is employed.

The Assembly code for one such routine DIR5 is given in Appendix A12; the other routines follow exactly the same pattern of the flow chart of Fig 4.12 except that each routine checks the pixels along its direction. The code in the Appendix follows Fig 4.12; the only addition is that every pixel must be examined to determine if it is the start pixel (all contour followed).

The value of the next pixel direction or the Freeman Code is kept in register AH and stored in memory area FMCOUT.

The two instructions

```
        CALL  GO1
        CALL  GETBIT
```

will check the pixel value along direction 1 and keeps its binary value 0 or 1 in the "carry" flag of the status register for the quickest comparison. More about these 2 calls will be given later.

Before explaining the full tracking algorithm, it will be useful to clarify the operations involved. Referring to Fig 4.7 or SECTOR area which contains the whole binary image (8 pixel bits is kept in 1 byte), let

register BX : points to 1 byte in sector which has the 8 bits (0 or 1) for 8 consecutive pixels on 1 row; pixels at columns 1,2,..8 , pixels for columns 9,10,..16 are kept in single bytes.

register AL : points to 1 individual pixel within those 8 pixels.

Each row occupies 64 bytes e.g. row 1 starts at location 0000h of sector, row2 at displacement 0040h of sector, row3 begins at 0080h . . etc. The pixel at row 2, column 5 will be kept at location 0040h of sector in the 5th bit at that byte.

The following will help the reader to better understand how the values of registers BX and AL change as the mask is moved from pixel to pixel in any of the 8 directions (up , down, left, right, north east, . . etc).

pixel(2,5)	=>	BX : 0040h	&	AL : 5
pixel(3,5)	=>	BX : 0080h	&	AL : 5
pixel(1,5)	=>	BX : 0000h	&	AL : 5
pixel(2,8)	=>	BX : 0040h	&	AL : 8
pixel(2,9)	=>	BX : 0041h	&	AL : 1
pixel(1,7)	=>	BX : 0000h	&	AL : 7
pixel(1,9)	=>	BX : 0001h	&	AL : 1
pixel(3,7)	=>	BX : 0080h	&	AL : 7
pixel(4,9)	=>	BX : 0081h	&	AL : 1

Referring to Appendices A14 and A15 and assuming that a 'pointer' of register BX and AL refers to a specific pixel in SECTOR, then

<u>GO2</u> (up) routine moves the pointer up one row; the only operation is to decrement BX by 0040h since each row requires 64 bytes.

i.e. if current pixel(2,5) : BX = 0040h & AL = 5 and going up results in pixel(1,5) : BX = 0040h & AL = 5 .

<u>GO6</u> (down) routine moves the pointer down one row; only BX has to be

decremented by 0040h .

i.e. if current pixel(2,5) : BX = 0040h & AL = 5 and going down results in pixel(3,5) : BX = 0080h & AL = 5 .

<u>GO4</u> (left) routine is not as simple because going one pixel left might cross the byte boundary; so first AL is decremented to point to pixel left of current one and if byte boundary is crossed (AL = 0), then the preceding byte with its last bit (AL = 8) contains the desired pixel.

i.e. if current pixel(2,9) : BX = 0041h & AL = 1 and going left results in pixel(2,8) : BX = 0040h & AL = 8 .

AL is decremented to 0 , boundary crossed , AL set to 8

boundary crossed , so DX decremented to 0040h

<u>GO0</u> (right) routine must also check if the byte boundary is crossed so first AL is incremented to point to a pixel to right of current one and if byte boundary is crossed (AL = 9), then the succeeding byte with its first bit (AL = 1) contains the desired pixel.

i.e. if current pixel(2,8) : BX = 0040h & AL = 8 and going right results in pixel(2,9) : BX = 0041h & AL = 1 .

AL is incremented to 9 , boundary crossed , AL set to 1

boundary crossed , so DX incremented to 0041h

The next four routines given in Appendix A14 are slightly more involved because both the row and column have to be changed as well as the byte checked if it is crossed. Only one of them will be explained as the logic is similar.

<u>GO1</u> (north-east) follows the logic :

decrement BX by 0040h to point to row up.

increment AL to point to pixel on right.

if AL = 9 , byte crossed , set AL to 1.

if byte crossed , increment BX.

i.e if current pixel(2,8) : BX = 0040h & AL = 8 and going to

north-east pixel(1,9) : BX = 0010h & AL = 1 .

The next 3 routines are used in the tracking algorithm they are listed in Appendix A15 and they are :

STARTPIX merely saves the pointer values (row & column) of the starting pixel into 4 bytes.

ENDPIX is important to stop the tracking algorithm; as the contour is traversed from start pixel in an anti clockwise direction, the algorithm should terminate when current pixel coincides with the start pixel. Basically it compares the row and column numbers of the current pixel with the row and column of start pixel which were saved in STARTPIX routine.

GETBIT routine is needed to check if the pointed pixel is 0 or 1 and the fastest way to check this is to put the pixel value in the CARRY flag of the status register where it can be checked for very efficiently in Assembly. The byte of SECTOR containing the pixel bit is merely shifted into the CARRY bit; how many times the shift is done depends on the AL value.

The TRACKING ALGORITHM follows the flowchart given in Fig 4.13 and the Assembly listing is given in 4 parts in Appendices A16, A17, A18 & A19.

The first step of the algorithm initializes the starting address of the two areas needed SECTOR and FMCOUT, then initializes the pointer to current pixel to first row and column of image.

The second part tries to locate the topmost black pixel (1st pixel lying on object's contour); this is done by checking the pixel value (call GETBIT routine) to be 1 and if it is not, the succeeding pixel is checked (the next pixel will be on the right or if the end of a row is reached, the next row is scanned). The end of this step results in a start pixel or will fail if the whole 512 × 512 array of pixels are all 0's.

The 3rd step of the tracking algorithm saves the coordinates of the start pixel which is needed to stop the tracking. Steps 3 and 4 are listed in Appendix A17.

Step 4 finds the second pixel with respect to start pixel

this is needed to get the initial current direction of the contour segment. Since the start pixel was obtained in a top to bottom and left to right fashion, the second pixel can be only in the south east, south, south west or west direction. Once the FMC of the second pixel is obtained and current direction obtained, the control is passed to step 5 of the algorithm listed in Appendix A18. The current direction is kept in register AH.

Step 5 will transfer control to one of the 8 routines DIR0, DIR1, . . DIR7 (explained earlier) given that the pointer BX and AL refers to current pixel and the direction of scan is in AH. If AL = 5 , then control goes to DIR5 which tracks the contour and as long as the next pixel is in direction 5 , execution stays in DIR5 in a small and efficient loop. Once a pixel is found which is not along direction 5, the new direction is loaded in AH and control passes back to the beginning of step 5 where the contents of AH is used to direct control to any of the 8 routines.

Note that every time the DIR routines are executed, it outputs the FMC code of the pixel to FMCOUT and also checks if the end of the contour is reached; if the end is reached it loads a special number 09 in register AH.

The last step (Appendix A19) will only write the coordinates of the start pixel and the contour perimeter at the beginning of area FMCOUT.

4.5.4 Displaying the Contour only

The algorithm to display the contour of an image given only the FMC code of the image, the number of pixels on the boundary and the start pixel coordinates. The flowchart is given in Fig 4.14 and the listing in Appendix A19 & A20. This algorithm is used either to display the contour of a digitized image interactively in the learning stage or can be used to view the contour of a previously stored contour from floppy. At first, the SECTOR area which contains the binary image of what will be displayed on the TV monitor has to be cleared to 0's; then the perimeter of the contour, the row and column of start pixel are loaded from FMCOUT ; a pointer is set up to the 1st FMC of the contour and a counter to keep track of perimeter.

Next, the FMC of the pixel is read and depending on its value, registers BX & AL are changed such that they point to that particular pixel and its value is set to 1 (dark). This part is repeated for all the pixels on the border.

Once all the border pixels have a 1 in their corresponding bits in SECTOR, the last thing is just to unpack SECTOR to the frame memory area of the FG-100 (discussed previously).

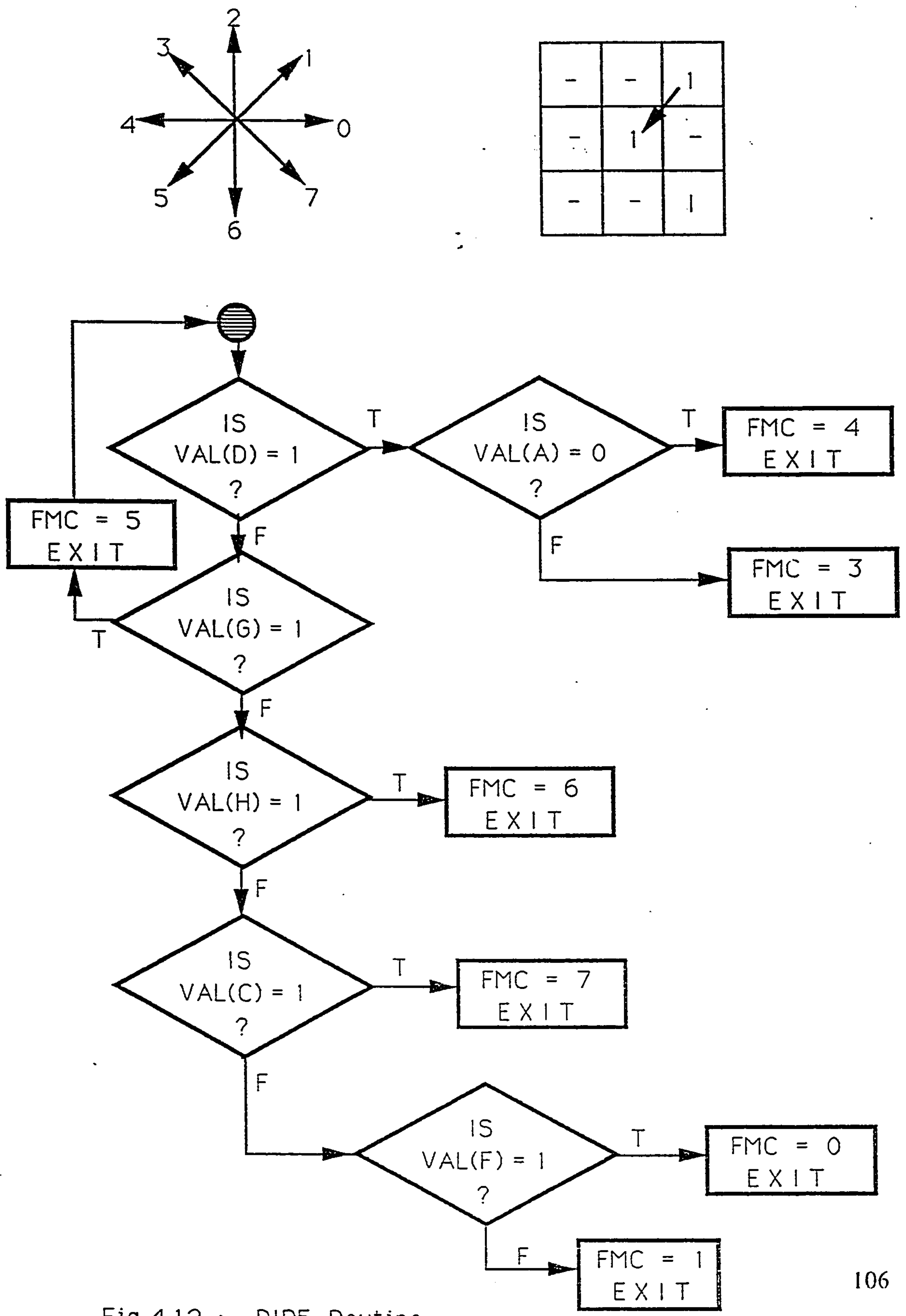
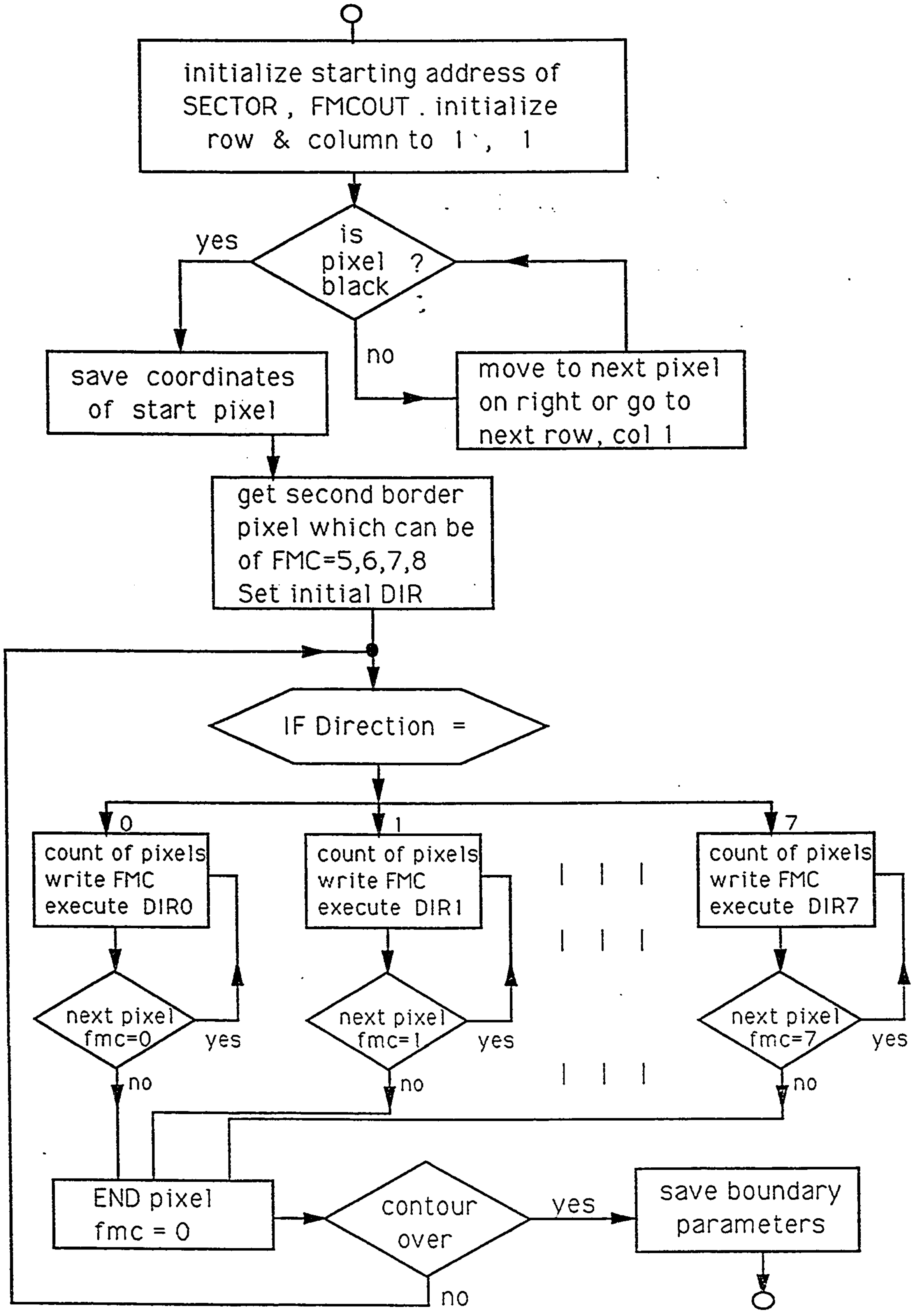


Fig 4.12 : DIR5 Routine



initialize starting address of SECTOR , FMCOUT . initialize row & column to 1 , 1
is pixel black ?
yes
no
save coordinates of start pixel
move to next pixel on right or go to next row, col 1
get second border pixel which can be of FMC=5,6,7,8 Set initial DIR
IF Direction =
0
1
7
count of pixels write FMC execute DIR0
count of pixels write FMC execute DIR1
count of pixels write FMC execute DIR7
next pixel fmc=0
next pixel fmc=1
next pixel fmc=7
yes
no
END pixel fmc = 0
contour over
save boundary parameters

Fig 4.13 : Tracking Algorithm

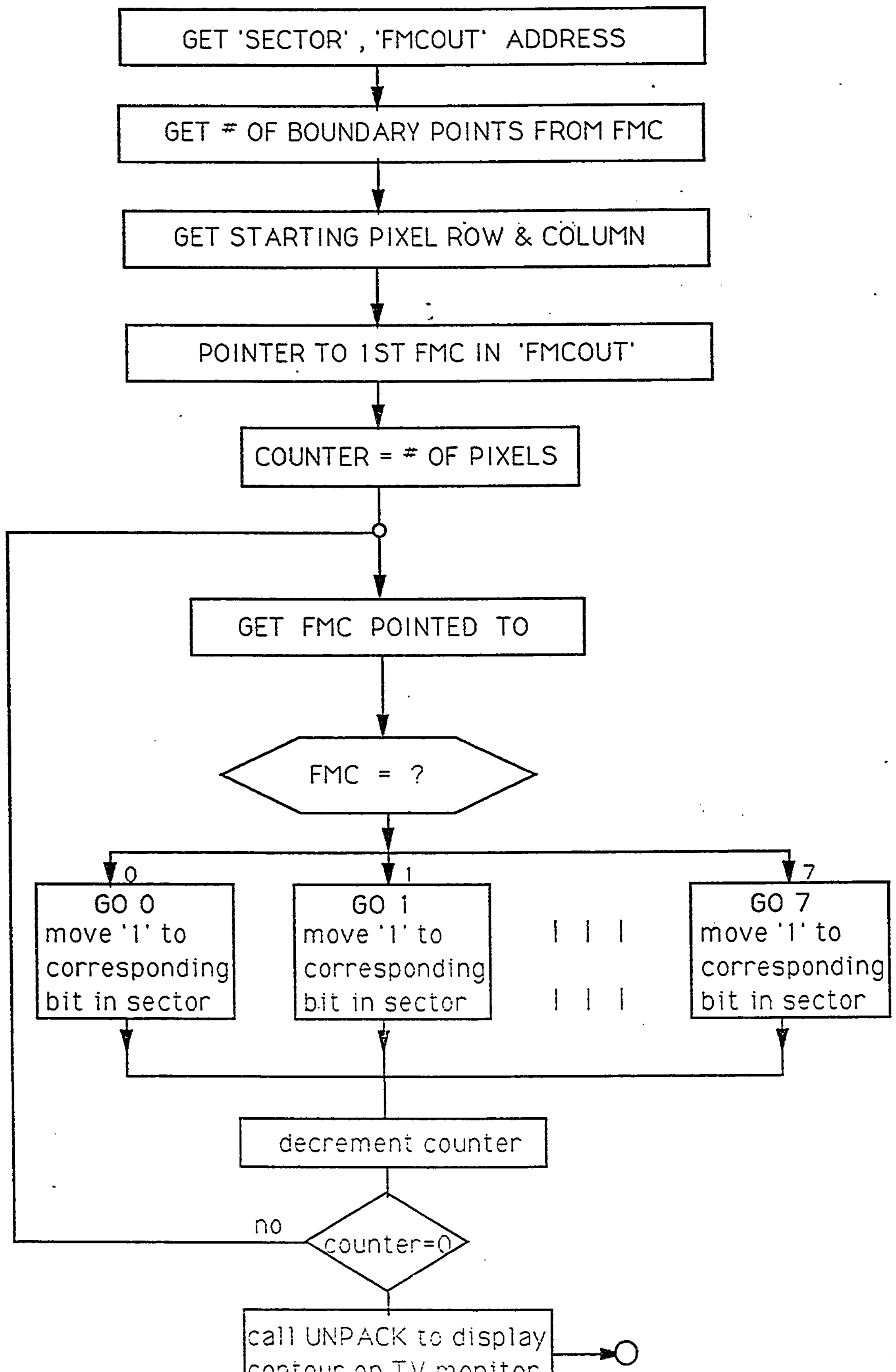

GET 'SECTOR' , 'FMCOUT' ADDRESS
GET # OF BOUNDARY POINTS FROM FMC
GET STARTING PIXEL ROW & COLUMN
POINTER TO 1ST FMC IN 'FMCOUT'
COUNTER = # OF PIXELS
GET FMC POINTED TO
FMC = ?
0
1
7
GO 0 move '1' to corresponding bit in sector
GO 1 move '1' to corresponding bit in sector
GO 7 move '1' to corresponding bit in sector
decrement counter
no
counter=0
call UNPACK to display contour on TV monitor

Fig 4.14 : Display Contour

CHAPTER 5

INVARIANT REPRESENTATION & DESCRIPTOR ARRAY

5.1 Relative Freeman Code

All the previous algorithms of Chapter 4 were carried out in Assembly. The rest of the programs were implemented in Pascal. Once the Freeman Chain Code of the contour is obtained as $FMC(C_k) = 0, 1, 2, 3, 4, 5, 6, 7$, it will be converted to a relative Freeman coding as already explained in Section 3.1.2 as

$RFMC(C_k) = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12$

The RFMC is given in Fig 3.1. The conversion is not as straight forward because it is not always required to translate a 0 to 8, a 1 to 9 . . etc but rather the translation is needed only in the context of the direction of the segment being examined. The algorithm checks each FMC(k) with its predecessor FMC(k-1) and only if the difference indicates a marked difference, is it assumed that the FMC has to be modified. The algorithm is given in Fig 5.1 and the listing is in Appendix B1.

Let the FMC for a segment corner be :

. . .'6'7'7'7'7 **'0** '0'0'0'1'1'1'1'0'0'0'0'1 . . .

The RFMC outputted by the algorithm will be :

. . .'6'7'7'7'7 **'8'8'8'8** '9'9'9'9'8'8'8'8'9

and definitely not

. . .'6'7'7'7'7 **'8** '0'0'0'1'1'1'1'0'0'0'0'1'

This explains why even if the difference between successive FMC's are small the RFMC is not necessarily equal to the FMC. i.e. the difference between 0 and 7 demands a translation but what about the change from 0 to 0 and 0 to 1 ?

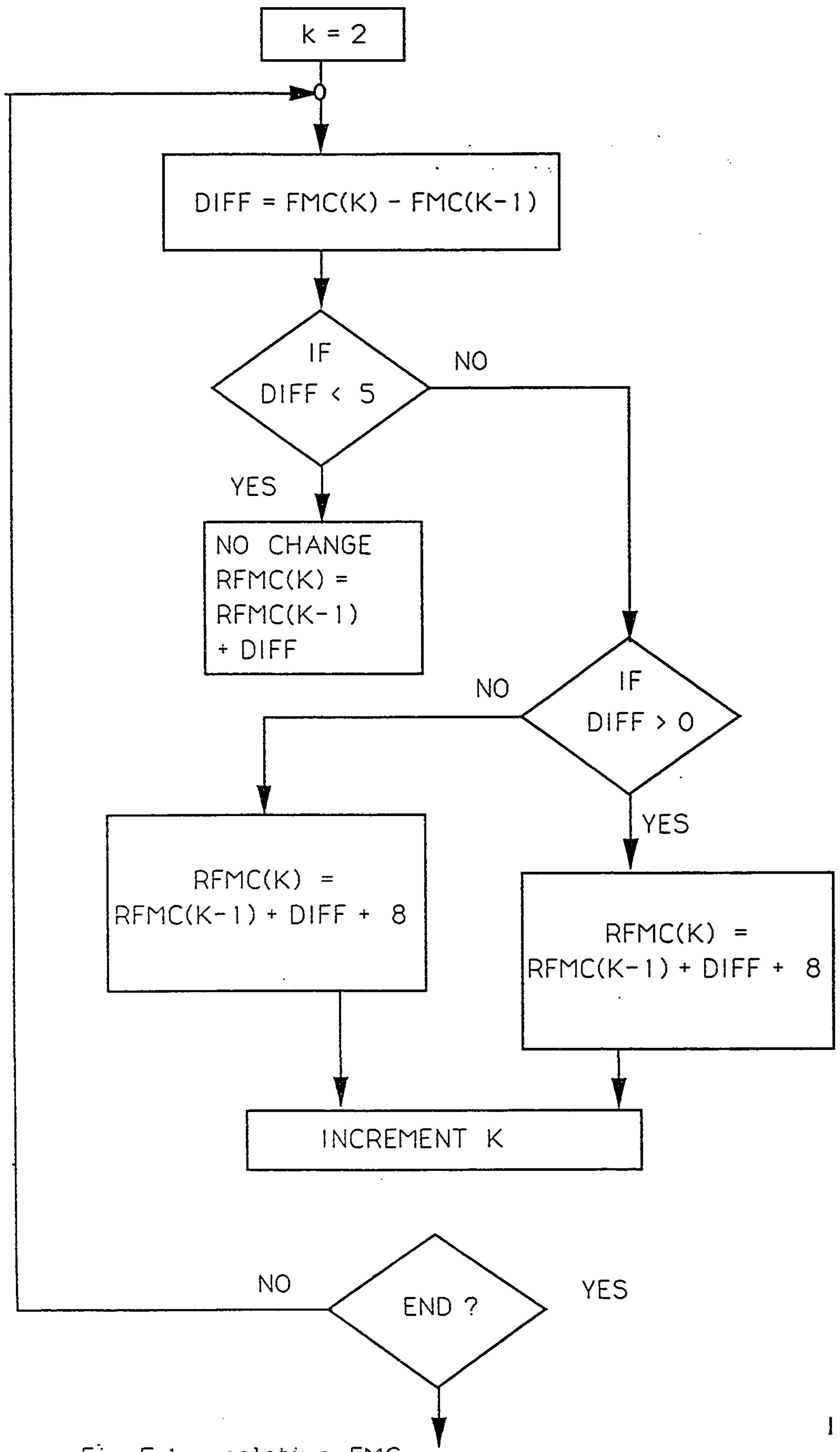

k = 2
DIFF = FMC(K) - FMC(K-1)
IF DIFF < 5
NO
YES
NO CHANGE RFMC(K) = RFMC(K-1) + DIFF
IF DIFF > 0
NO
YES
RFMC(K) = RFMC(K-1) + DIFF + 8
RFMC(K) = RFMC(K-1) + DIFF + 8
INCREMENT K
NO
END ?
YES

Fig 5.1 : relative FMC

Since the last 0 was changed, it implies the new 0's and 1's encountered must also be changed. This is why the step

RFMC(k) := RFMC(k-1) + diff , is used instead of the presumed simpler

RFMC(k) := FMC(k) , because the former takes into account any previous translation in effect.

If the difference is more than 5 or that the West boundary has been crossed, a translation is warranted. There are 2 possible ways that this boundary can be crossed : first a chain code of '7'7'7'7 **'0** '0'0'0' , 7'7'7 **'1** '1' , '6'6'6'6 **'2** '2'2' etc or in an anti clockwise direction and second chain codes like '1'1'1'1 **'7** '7'7'7', '9'9'9'9 **'6** '6'6'6' , '10'10'10'10 **'7** '7'7'7' etc or in a clockwise direction.

For the first case, a value of 8 is added to the previous RFMC and the difference while for the latter 8 is decremented. Note that again the RFMC of previous pixel is used and not the FMC to maintain consistency in direction.

Appendix B1 also shows how the file containing the FMC of the object is read from Pascal; the user is interactively asked the name of the file and appends the directory where the file is found (here in a subdirectory C:subdir) and appends the extension of .fmc as the file type. The file is then read into a buffer which is an array of integer values or the FMC values. Note that the first byte of the file contains the number of pixels on the contour; this value is read into variable fmccnt ; the step (to be explained in Section 5.2.2) is set to 4 and the number of 4-pixel groups set to 1/4 of fmccnt.

5.2 Obtaining Corner Points of Contour

This section explains the algorithm used to get the coordinates of the corner points. As discussed in Section 3.1.3 the method of gradient change over 4-pixel groups is used and two levels of the Polar model is found : one which detects the very sharp points of inflexion and the other detects even slight deviations in straight lines (shown in Fig 3.3).

5.2.1 Gradient Change Method

The algorithm detects a break point by examining successive line segments (comprising of 4 pixels) and checking if the gradient of that line segment fits within a lane of the current segment i.e. if the small segment lies in the current direction of the segment; if it does, the small segment belongs to the segment and the next small segment is compared. When a small segment differs markedly from the current segment gradient, a break point is detected : its X and Y coordinates are recorded and a new segment started.

The conventions used are (Section 3.1.3) :

S_k = summation of RFMC of 4 consecutive pixels.

$\overline{S_k} = \frac{S_k}{4}$ = the gradient or direction of small segment.

$S_{k(ref)}$ = gradient of the contour segment examined.

ε = the difference tolerated between small and large segment.

In the flowchart of Fig 5.2 and the listing of Appendix B2, the following variables are used :

j : pointer to the beginning of a 4-pixel group

sumcomp : $S_{k(ref)}$ for Coarse model.

sumcompl : $S_{k(ref)}$ for Detailed model.

sum : summation of FMC of 4 pixels : S_k .

$\overline{S_k}$ is not used in the algorithm because the constant division by 4 would slow down the algorithm; in fact the summation or the average of the summation work just as well.

dircnt : number of pixels on contour.

dirptr : pointer to number of 4-pixel groups looked at.

xx : x ordinates of current pixel group.

yy : y ordinates of current pixel group.

vertx : array to save X ordinates of corners (Coarse)

verty : array to save Y ordinates of corners (Coarse)

vertx1 : array to save X ordinates of corners (Detailed)

verty1 : array to save Y ordinates of corners (Detailed)

ptrdetail : pointer linking corner of Detailed to Coarse.

ptrdetail1 : pointer linking corner of Coarse to Detailed.

The algorithm for obtaining the Coarse model is given in Fig 5.2 but the program was implemented with the algorithm of Fig 5.3 which finds the Coarse and Detailed models in one combined iteration of the contour foı reasons of efficiency. The listing of Appendix B2 corresponds to Fig 5.3 but the simplified algorithm of Fig 5.2 will be used for explanation for the sake of understanding.

Initialization is done by setting j = 1st pixel point of contour, setting a pointer to the first X and Y coordinates to be saved, setting the current x and y coordinates of pixel under test as 0 , 0 (assume the 1st topmost pixel on contour is the origin);

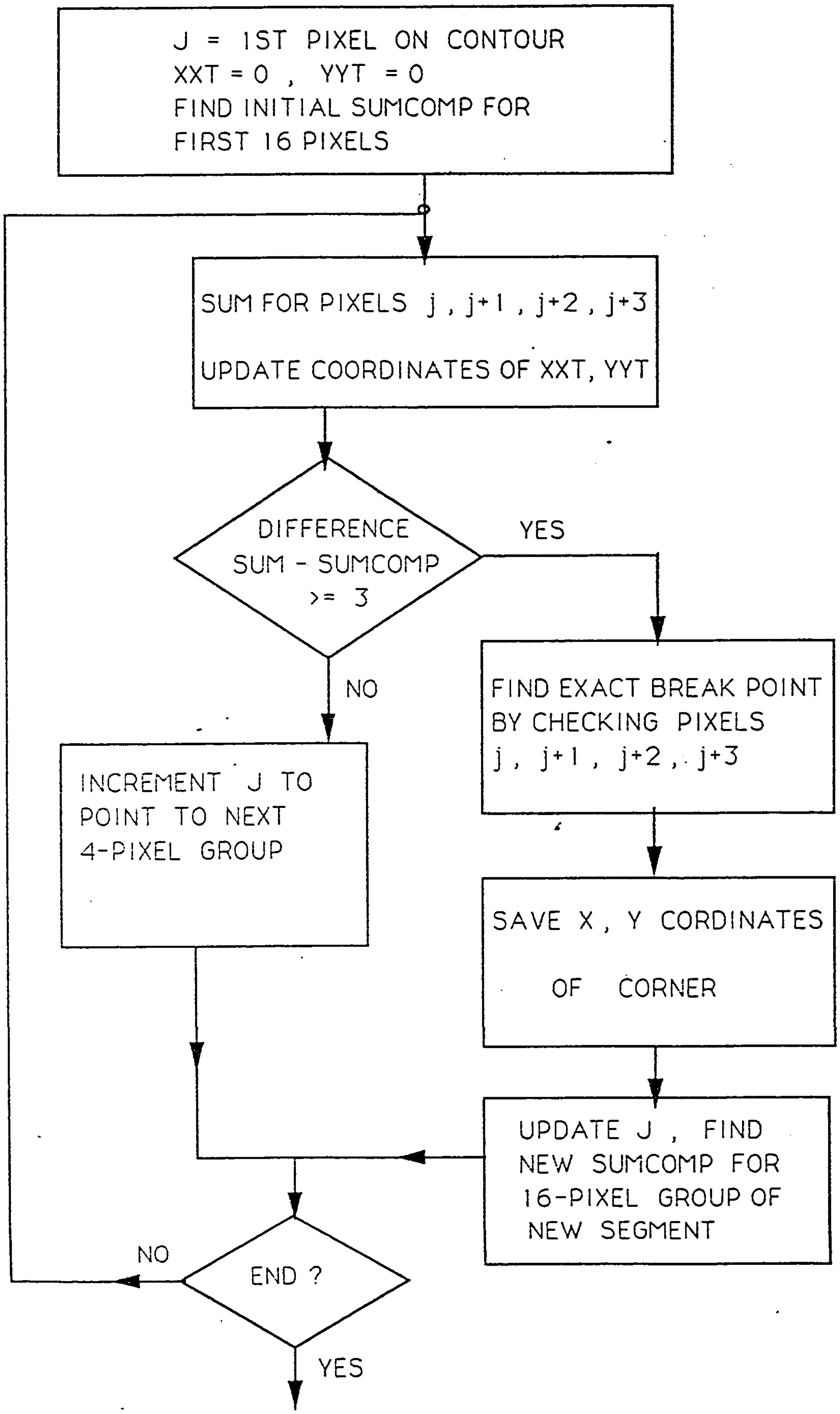


Fig 5.2 : Finding Corner Points of Coarse Model

A value of sumcomp (current direction of large segment) is needed and this was calculated as the average of the FMC of 16 pixels (explained later).
Now successive 4-pixel groups are considered and their sum computed and the coordinates of the current pixel is updated to point to the beginning of the 4-pixel group; Note that variables xx and yy always contain the x and y ordinates of the 1st pixel in the 4-pixel group; xx and yy are updated as the algorithm moves from pixel to next pixel by simply examining the direction of the next pixel and changing xx and yy accordingly. e.g referring to Appendix B2 if the next pixel direction is 7 , xx is incremented and yy decremented , keeping in mind that the origin of the contour is the topmost pixel. Two small arrays xxt and yyt , each of size 4 temporarily stores the x and y ordinates of each pixel in the 4-pixel group while xx and yy keeps only those of the 1st pixel of group. xxt and yyt are needed to save the exact coordinates of the corner.
Next the difference between sum and sumcomp is found. If the difference is less than 3, it is assumed that the small 4-pixel segment lies on the same line of the large segment and hence the tracking algorithm goes on by setting j to point to the next 4-pixels and the previous steps are repeated as long as the end of the contour is not reached.
If the difference is greater than 3, then a sharp break has been found and some processing is required. The first thing to do is to find the exact break point within the 4-pixels.

5.2.2 Exact Break Point

Let the RFMC of the contour around a corner point be :

..44454445444544777877787778.. for a segment of direction East changing to one of direction South East. Let the 4-pixel groups be 4445 4445 4445 4477 7877 . etc and their corresponding sums be 17, 17, 17, 22, 29. Let the tracking algorithm be on the first segment with a general direction of sumcomp = 17. Now the j pointer is pointing to the group 4477 and the current sum = 22. The difference of (22-17)=5 indicates a sharp change in direction but the problem is how to know if the corner occurs at pixel 1 or pixel 2 or pixel 3 or pixel 4 of that group. This exact break is found by multiplying each RFMC of the pixel by 4 and comparing it to sumcomp; i.e.

4-pixel group : 4477 and sumcomp : 17

pixel 1 : 4 , 4*4 = 16 , diff(16,17) : 1 => not corner point.

pixel 2 : 4 , 4*4 = 16 , diff(16,17) : 1 => not corner point.

pixel 3 : 7 , 7*4 = 26 , diff(26,17) : 10 => corner point.

Once the exact corner point is found, the corresponding X and Y coordinates are stored in arrays vertx and verty (this is where xxt and yyt comes in handy). The pointer to next entry in vertx and verty must also be updated and j is to point to the 1st 4-pixel group on new large segment. In addition a new value of sumcomp is calculated for the new large segment of the contour (group of 16 pixels used).

The above steps are repeated until the end of the contour is reached. The combined algorithm implemented (Coarse & Detailed) is slightly more complicated although the tracking logic is the same; it will be explained in a while.

5.2.3 4-Pixel Groups

The program implemented in the final version uses gradients of small segment of 4 pixel groups. Why was this value chosen instead of 3-pixel, 5- or 6-pixel groups ? In fact the algorithm was tested for 3- , 4- , 5- and 6- pixel groups, the results of which will be given in Chapter 7.

The optimal value was found to be 4. The grouping itself will determine the number of iterations of the algorithm although the summations will be done a constant number of times. If the grouping is of 'step' pixels, then the number of iterations is equal to perimeter divided by step; hence the larger the step the smaller the number of iterations.

A step of 3 was not satisfactory because it does not overlook the digitization noises over local pixels; Steps of 5 or 6 also was not satisfactory because every time a break is found between the group and large segment, the accurate break point has to be found. This is not satisfactory especially for the detailed model where a lot of corner points are expected.

5.2.4 Sumcomp (over 16 pixel)

Every time a new segment along the contour is detected, the average gradient of that segment is required. An accurate value for that gradient is required because all the small segments of 4-pixels will be compared to it to determine the next break. A value which extends over the first 16 pixels at the start of the segment was used instead of the usual 4 because it was noticed that at most corners in the digitized images, local digitization noise exists because the FMC directions are changing suddenly. This noise is most often found at the first 4 or 5 pixels of the new segment and if the sumcomp was found over only those beginning pixels, the value of sumcomp is not accurate and does not represent the overall direction of that segment. To cancel the effect of the stray starting pixels of a new segment, the first 16 pixels were considered to get the value of a better fitted sumcomp.

5.2.5 Epsilon (gradient tolerance)

The difference between successive 'sum' of 4-pixel groups and 'sumcomp' of the large contour segment is used to detect corner points. Different values were experimented with, the results of which will be listed in Chapter 7.

A difference of 3 was found to be the best for the Coarse model as it accurately detects segments differing by 30° or more . A value of 4 was not good because segments differing by angles less than $45^\circ - 50^\circ$ were not detected.

Obviously for the Detailed model a choice of epsilon equal to 1 or 2 existed since it should detect variations which are overlooked by the Coarse model. The only choice was 2 since epsilon of 1 was so detailed that it broke the contour into hundreds of small segments with differing angles of around 5 degrees. Breaking the contour into hundreds of pieces imposes 3 problems :

- the storage needed to store the x and y ordinates of all the corner points is too large to be efficient.
- a large number of segments does not necessarily represent the image more accurately than required for the application.
- more segments in the Detailed models imply a longer time in the calculation to compute the area between the graphs of two images; this will slow down the recognition steps.

5.3 Coarse & Detailed Models

5.3.1 Combined Algorithm

The combined algorithm in Fig 5.3 and in Appendix B2. The addition needed is if the difference in sum and sumcomp is < 3 (no corner for Coarse), another check is made to see if the difference might mean a vertex for the detailed model.

If a corner for the Detailed model is found, the coordinates are loaded in the vertx1, verty1 and its pointer updated but nothing is written into vertx, verty. Sumcomp is not changed, only sumcomp1 is set to reflect a new 'detailed' segment. Initially sumcomp1 is set to sumcomp.

If a corner for the Coarse model is found, the exact corner vertex is found and this vertex will be both in the Coarse and Detailed models and so a link is set to indicate the matching on vertices from both models. So once the X and Y ordinates are saved in vertx, verty, vertx1, verty1 and the pointing link set, both pointers vertptr, vertptr1 are updated; the new value of sumcomp is computed and sumcomp1 is set to that value.

Typically sizes of vertx1, verty1 will be slightly or greatly larger than the sizes of vertx, verty depending on the shape of the object being straight lined object or curved contour object. The indices in the arrays of the Coarse and Detailed models where the same vertices are stored are used in the pointer links. From the 2 models of Fig 5.4 , vertex 2 of Coarse and vertex 2 of Detailed match - the starting vertex. Vertex 6 of Coarse corresponds to vertex 8 of Detailed. The pointers reflect this - i.e pointer for vertex 6 of Coarse = 8 and pointer for vertex 8 of Detailed = 6.

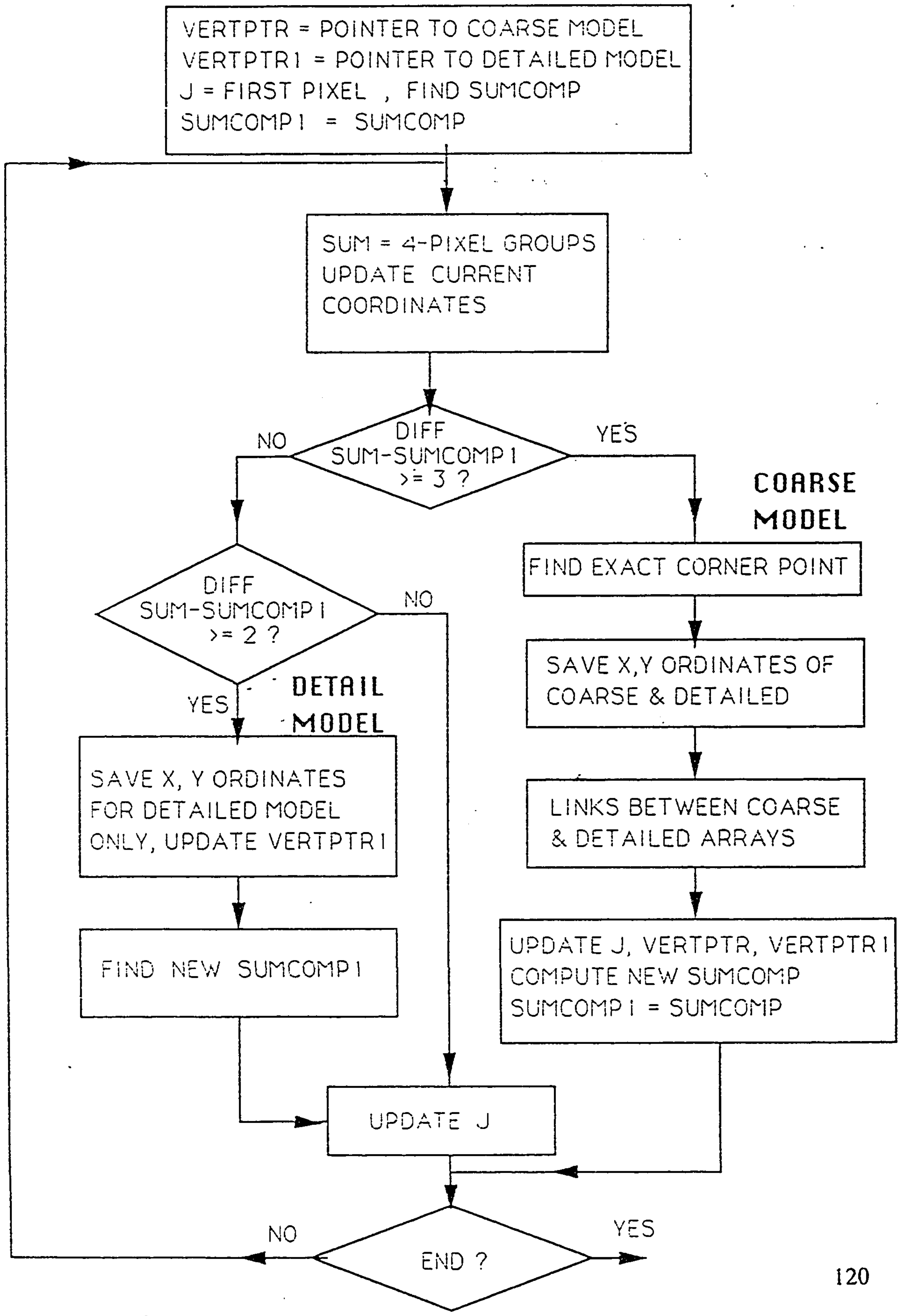

VERTPTR = POINTER TO COARSE MODEL
VERTPTR1 = POINTER TO DETAILED MODEL
J = FIRST PIXEL , FIND SUMCOMP
SUMCOMP1 = SUMCOMP
SUM = 4-PIXEL GROUPS
UPDATE CURRENT COORDINATES
DIFF SUM-SUMCOMP1 >= 3 ?
NO
YES
COARSE MODEL
FIND EXACT CORNER POINT
DIFF SUM-SUMCOMP1 >= 2 ?
NO
YES
DETAIL MODEL
SAVE X,Y ORDINATES OF COARSE & DETAILED
LINKS BETWEEN COARSE & DETAILED ARRAYS
SAVE X, Y ORDINATES FOR DETAILED MODEL ONLY, UPDATE VERTPTR1
FIND NEW SUMCOMP1
UPDATE J, VERTPTR, VERTPTR1
COMPUTE NEW SUMCOMP
SUMCOMP1 = SUMCOMP
UPDATE J
NO
YES
END ?

Fig 5.3 : Combined Coarse & Detailed Models Generation

5.3.2 Data Structures

Besides the arrays presented, the segment length and the angle change (ρ , θ) are saved in two 1-dimensional arrays LENG, THEDEG and LENG1, THEDEG1 . Fig 5.4-a shows the data structures which are needed in the later stages; an actual data from an image is given. Note how the corners which replicate in both models are linked by the indices of array PTRDETAIL and PTRDETAIL1. The 2 graphs for the same object, obtained from their polar representations are given in Fig 5.4-b; note that the matching corner points also match in the graphs and that a long segment of the coarse model matches the smaller segments of the detailed model.

5.4 Non-Square Pixel Correction

It was noticed that once an object's polar model was obtained at different orientations that the angles and lengths of the segments vary markedly but that they do not vary if the object is only translated. This led to some confusion until it was realized that the pixels of the CCD camera does not consist of uniform square pixels but rather rectangular ones. This fact was verified in 2 steps namely :

- A straight line was digitized along the horizontal and vertical directions and it was noticed that the same line was longer (in number of pixels) along the vertical direction.
- A circle was digitized and the horizontal and vertical diameters measured and the same factor was found.

COARSE MODEL

	LENGTH	ANGLE	POINTER
2	189	109'	2
3	87	68.4'	3
4	60	123.6'	4
5	48	-24'	6
6	85	-18.2'	8
7	129	101'	10
	LENG	THEDEG	PTRDETAIL

DETAIL MODEL

	LENGTH	ANGLE	POINTER
2	182	110'	2
3	87	70'	3
4	37.3	127'	4
5	23.9	-10.5'	0
6	34.9	-15.2'	5
7	13.0	-10.4'	0
8	62.2	-7.9'	6
9	22.8	-10.4'	0
10	129	103'	7
	LENG1	THEDEG1	PTRDETAIL1

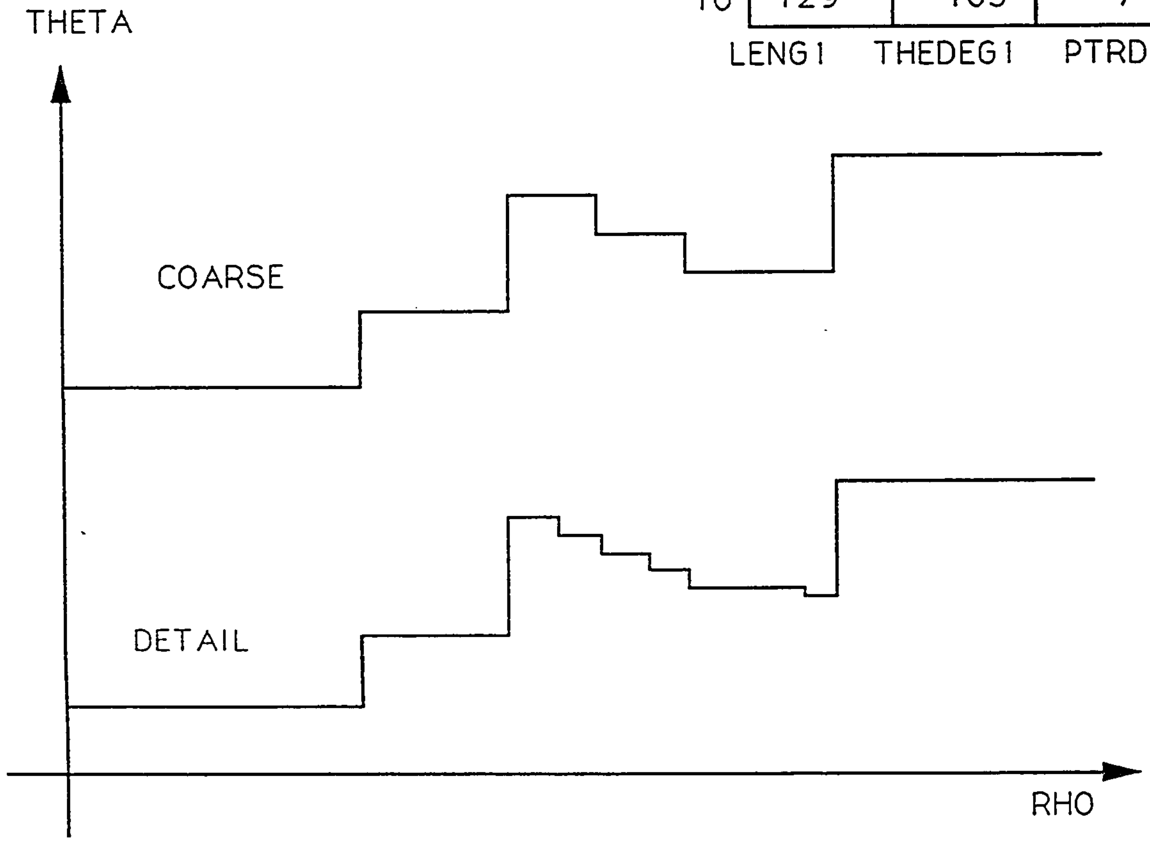


Fig 5.4 : Relation between Coarse & Detail Models

The only way to correct the problem was through software as the camera cannot be changed. It was noticed that the same line oriented vertically would be aroud 3/4 th of the vertical line. The initial solution was to somehow insert 1 extra row of the image for every 3 original rows. different methods were tested on the line in Fig 5.6. Only part of the line is shown. The actual values for :

$\alpha = 74.3^\circ$ L1 = 16 , L2 = 6 , L = 17.08 & L/L1 = 1.038

5.4.1 Inserting new rows

The **first** method was to replicate the new so that the new pixel is equal to that of the 3rd row and it's replicated on the same column as shown in Fig 5.7-a. The results obtained were :

$\alpha = 62.1^\circ$ L = 19.23 & ratio L/L1 = 1.13

The **second** way was to replicate the pixel along the same direction as the pixel of the 3rd row w.r.t. the 2nd row. The results were :

$\alpha = 65.8^\circ$ L = 21.93 & ratio L/L1 = 1.09

The **third** method copies the pixel from the succeeding row and on the same column. The results were :

$\alpha = 57.7^\circ$ L = 22.47 & ratio L/L1 = 1.18

The **fourth** method copies the pixel along the direction of pixel of next row i.e pixel of row5 w.r.t. row 4 . The results :

$\alpha = 57.7^\circ$ L = 22.47 & ratio L/L1 = 1.18

The results were rather poor, so these methods were dropped; they would also increase processing time as the new row must be inserted for every 3 rows.

CCD PIXEL PICKUP OF CAMERA

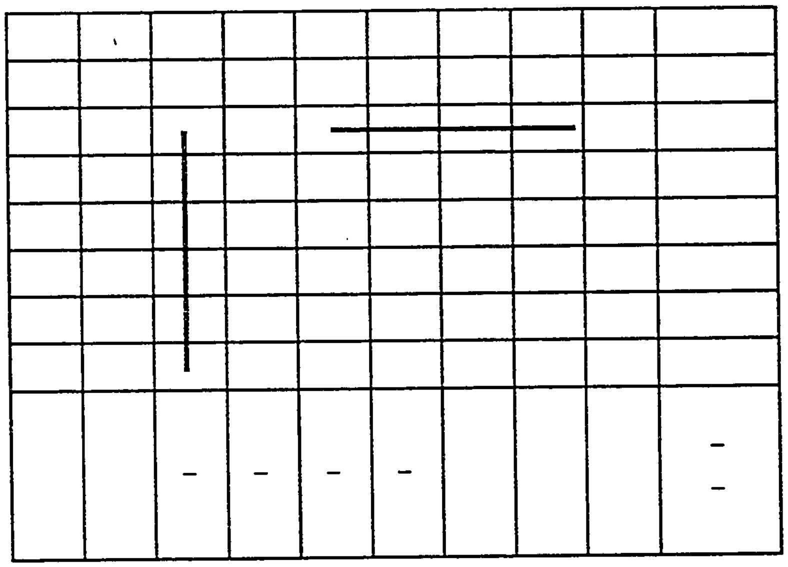
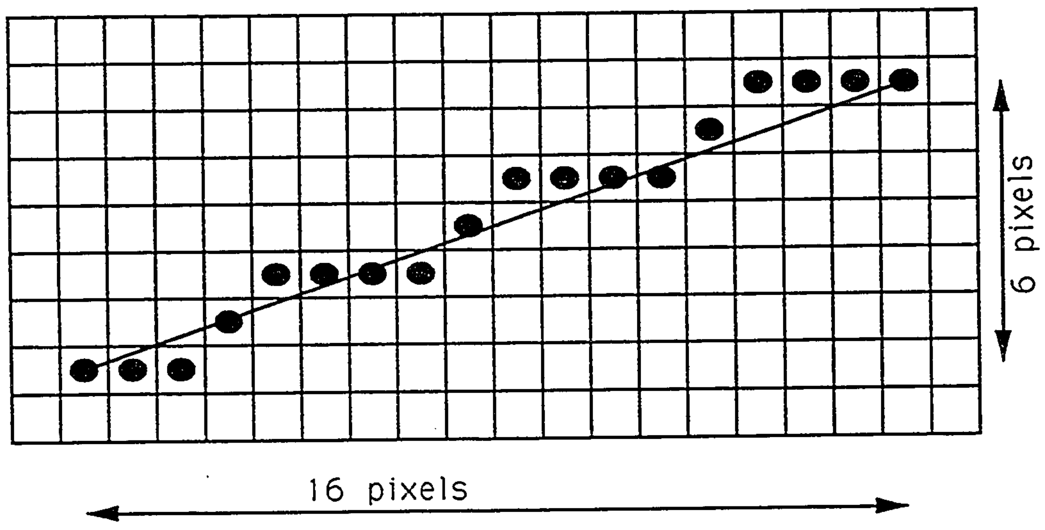


Fig 5.5 : Same line digitized horizontally and vertically with pixel lengths of 3 and 5

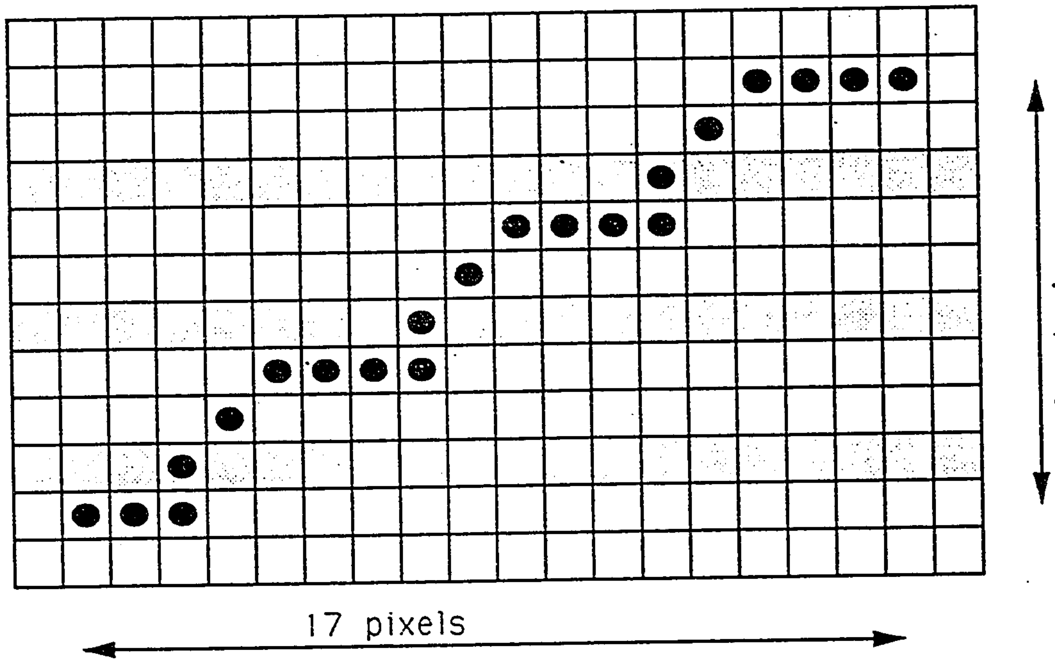


Length of diagonal line : 17.08 pixels

Angle theta (diagonal with x-axis) : 69.44 deg.

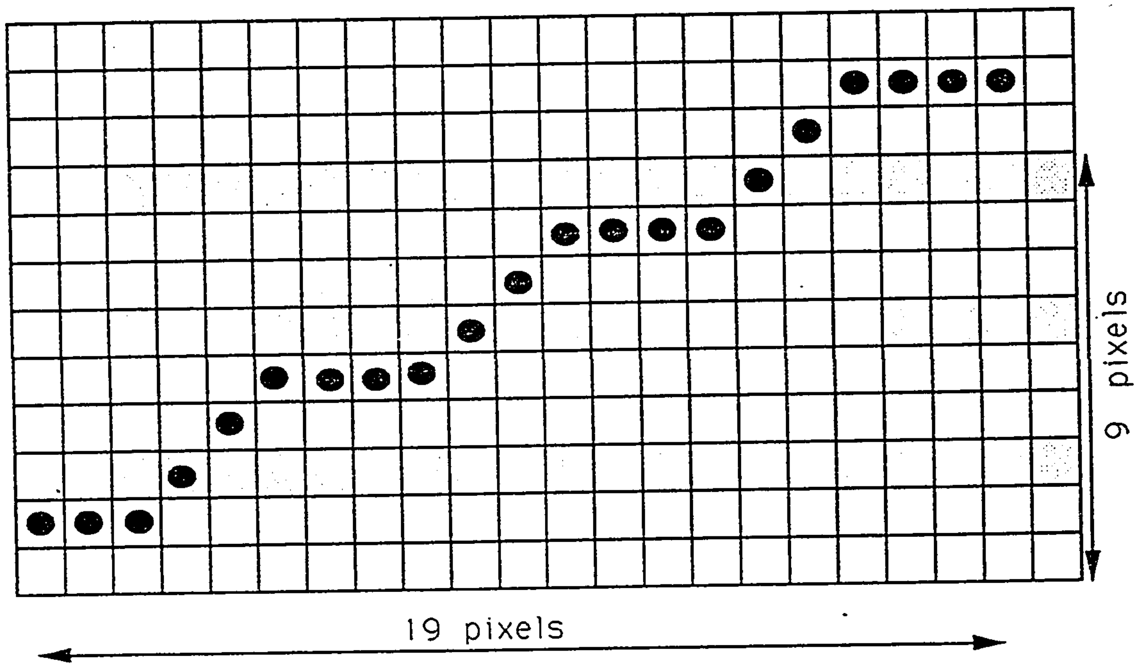
Fig 5.6 : Test line for correction

a) method 1 (new row shaded)



Length of diagonal line : 19.23 pixels
Angle theta (diagonal with x-axis) : 62.10 deg.

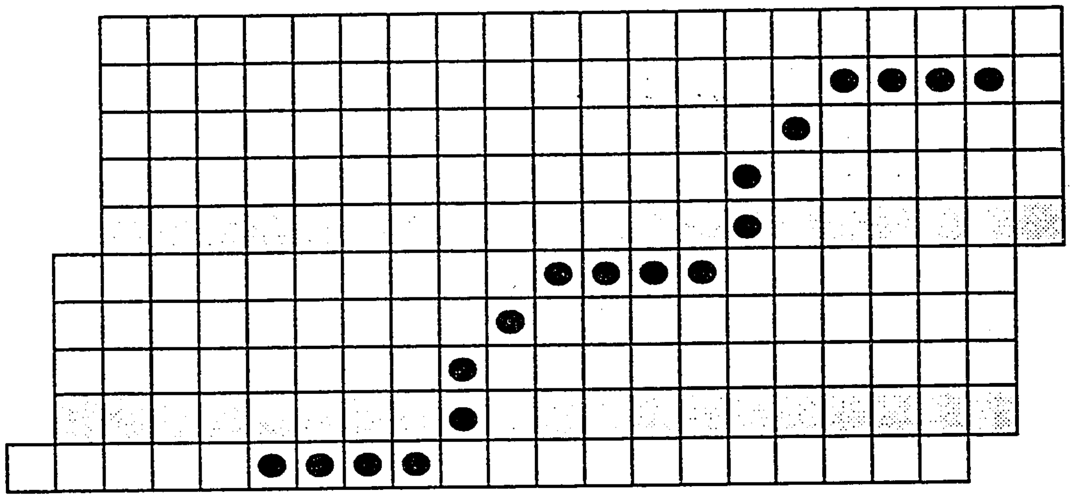
b) method 2 (new row shaded)



Length of diagonal line : 21.93 pixels
Angle theta (diagonal with x-axis) : 65.77 deg.

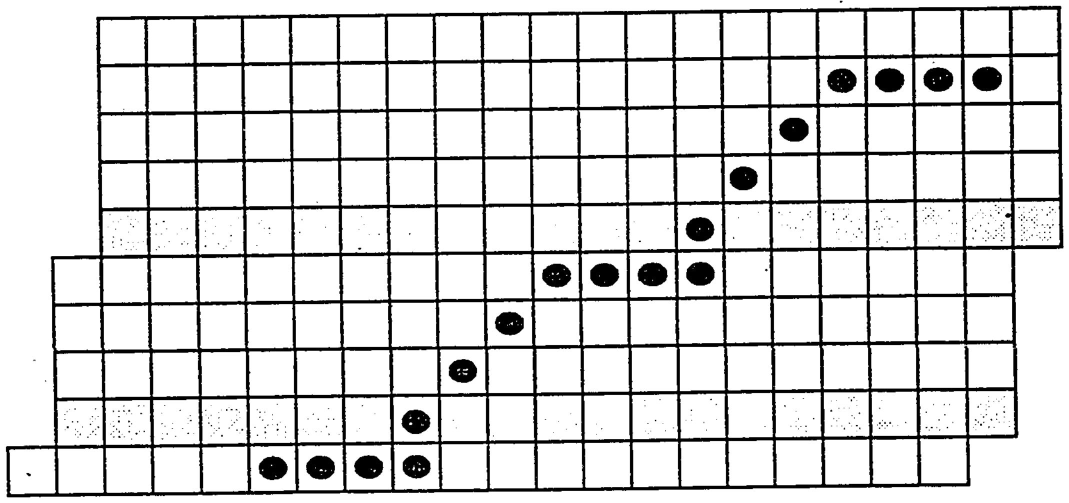
Fig 5.7 : Methods 1,2 for non Square Pixel Correction

a) method 3 (new rows shaded)



Length of diagonal line : 22.47 pixels
Angle theta (diagonal with x-axis) : 57.72 deg.

b) method 4 (new rows shaded)



Length of diagonal line : 22.47 pixels
Angle theta (diagonal with x-axis) : 57.72 deg.

Fig 5.8 : Methods 3, 4 for non square pixel correction

5.4.2 Correction by Scaling

The solution which was implemented was the simplest one and gave the best results. It does not operate on row-by-row level but rather on the vertices only, which cuts down a lot of processing. The method was tested on the 2 segments of Fig 5.9-a with actual values of

$\alpha = 63.0°$

L1 = 2.0 , L2 = 1.4 & ratio L/L1 = 1.43

The digitized image of the segments produced the values (in pixels as shown in Fig 5.9-b. There existed a choice of scaling either all dimensions along the X- or Y- axis; the Y-axis was chosen arbitrarily. For each segment of the image, only the length along the Y-axis has to be multiplied by a factor, the scaling factor; different values for this factor were tried experimentally and the results will be shown in Chapter 7. The best value found for the camera-object distance considered was found to be around 0.67 . The results obtained are shown in Fig 5.9-c and they are the best possible results which could be obtained under the situation.

The Pascal listing for the correction is given in Appendix B3, note that only Y coordinates of each corner point is multiplied directly by the scaling factor. There is no need to need to calculate the displacement along the Y-axis for each segment as the Origin of the X- and Y-axes is centered on the first corner point of the image.

A similar scaling is done for the verty1 of the detailed model but is not listed in the appendices.

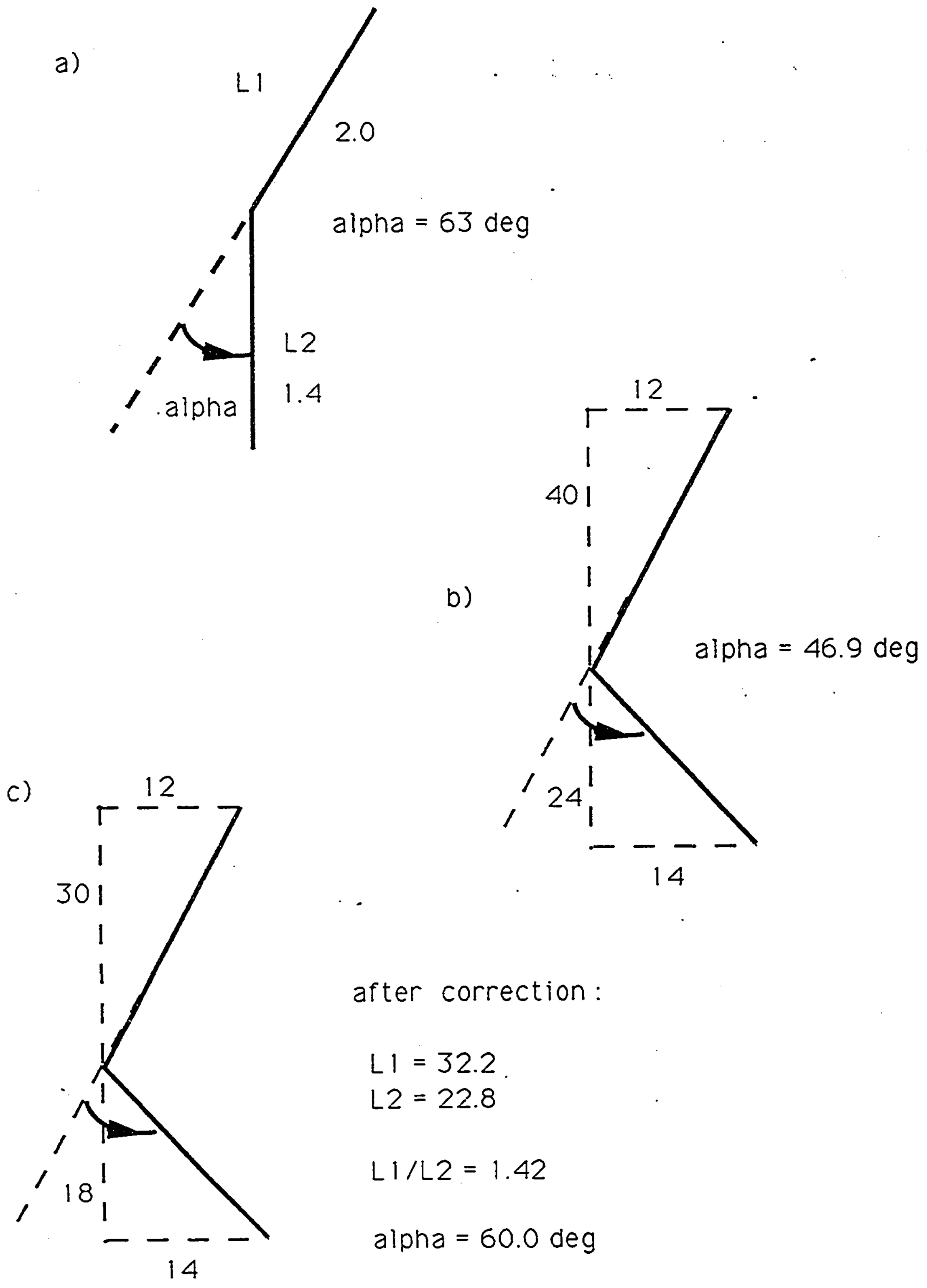


Fig 5.9 : Scaling along Y-axis

5.5 Obtaining Polar Model

Obtaining the polar model involves calculating the length and angle change for each segment from the x- and y- coordinates of the vertices.

5.5.1 Obtaining Segment Lengths

This step is straight forward, given the segment linking the points P1 (x1,y1) and P2 (x2,y2) , the length is obtained from the equation :

$$\text{length} = \sqrt{(x2 - x1)^2 + (y2 - y1)^2}$$

The Pascal listing is given in Appendix B3. A similar loop is implemented for the detailed model (not shown). Note that the difference between x- and y- ordinates of each vertex is saved as they will be used in the next step of finding the angle change between consecutive segments.

5.5.2 Obtaining Angular Change

Obtaining the change in angles of segments proved to be a more difficult step as it involves :

- finding the exterior angle between segments.
- finding the direction of the exterior angle.

Referring to Fig 5.10 which shows 2 segments A-B and C-D, the problem is to find the angle. The angle needed is α or γ but not the interior angle β . Furthermore even if the absolute value of angle α will not change for the 2 figures, their direction will change if segments A-B-C or segments C-B-A is considered; i.e. the angle formed by segment B-C w.r.t. segment A-B is clockwise or $+\alpha$ but segment B-A w.r.t. segment C-B forms an anti clockwise or $-\alpha$.

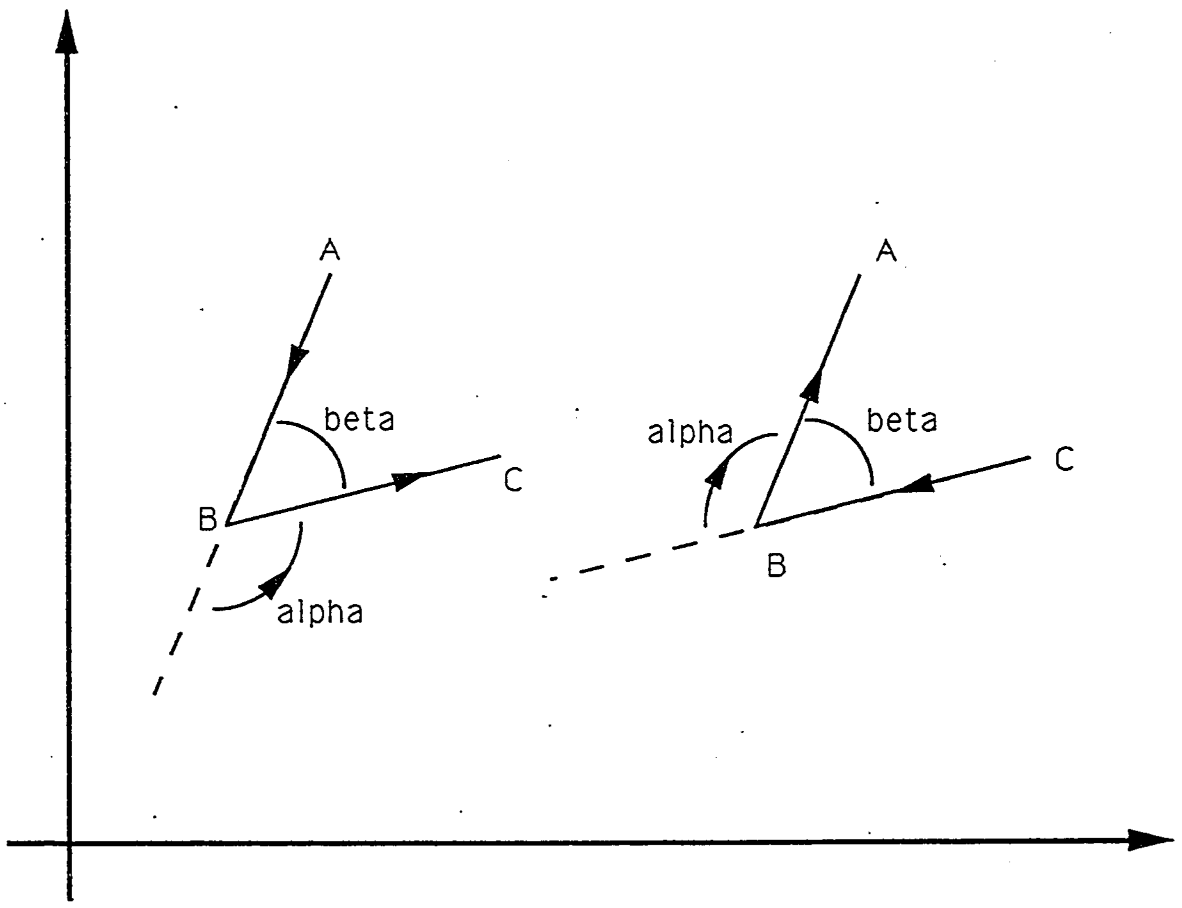


Fig 5.10 : Correct angular change

Several methods were tried and they are :

The ***COSINE*** method implied finding the angles supported by

the 2 segments on the Y-axis. Referring to Fig 5.11 , where the angle $+\theta$ is to be found between segments A-B and B-C, the first step of obtaining the angle α (formed by A-B and Y-axis) or angle β (by B-C and Y-axis) proved problematic because the ARCCOS function returns the same value for segments A-B or D-E.

The 2nd problem is, even if α , β can be obtained unambiguously, the value of angle θ must be obtained and not the interior angle γ . Sometimes $\theta = (\beta - \alpha)$ or at other times $\theta = 360° - (\beta - \alpha)$ depending on the which of the 2 angles are bigger. Furthermore ARCCOS function has to be evaluated

The ***TANGENT*** method also proved problematic for nearly the same reasons :

- segments in the 2nd and 4th quadrant support the same angles with the Y-axis; also segments of 1st and 3rd quadrants.
- ARCTAN function must be evaluated twice.
- ARCTAN cannot be evaluated for vertical lines.
- choosing between the interior or exterior angles add one selection step in the procedure.

The ***SLOPE*** method which uses the equation :

$\theta = (m1 + m2) / (1 - m1*m2)$,

where m1, m2 refers to the slope of the 2 segments also failed because it does not recognize the direction of the segment e.g segment A-B is the same as segment B-A .

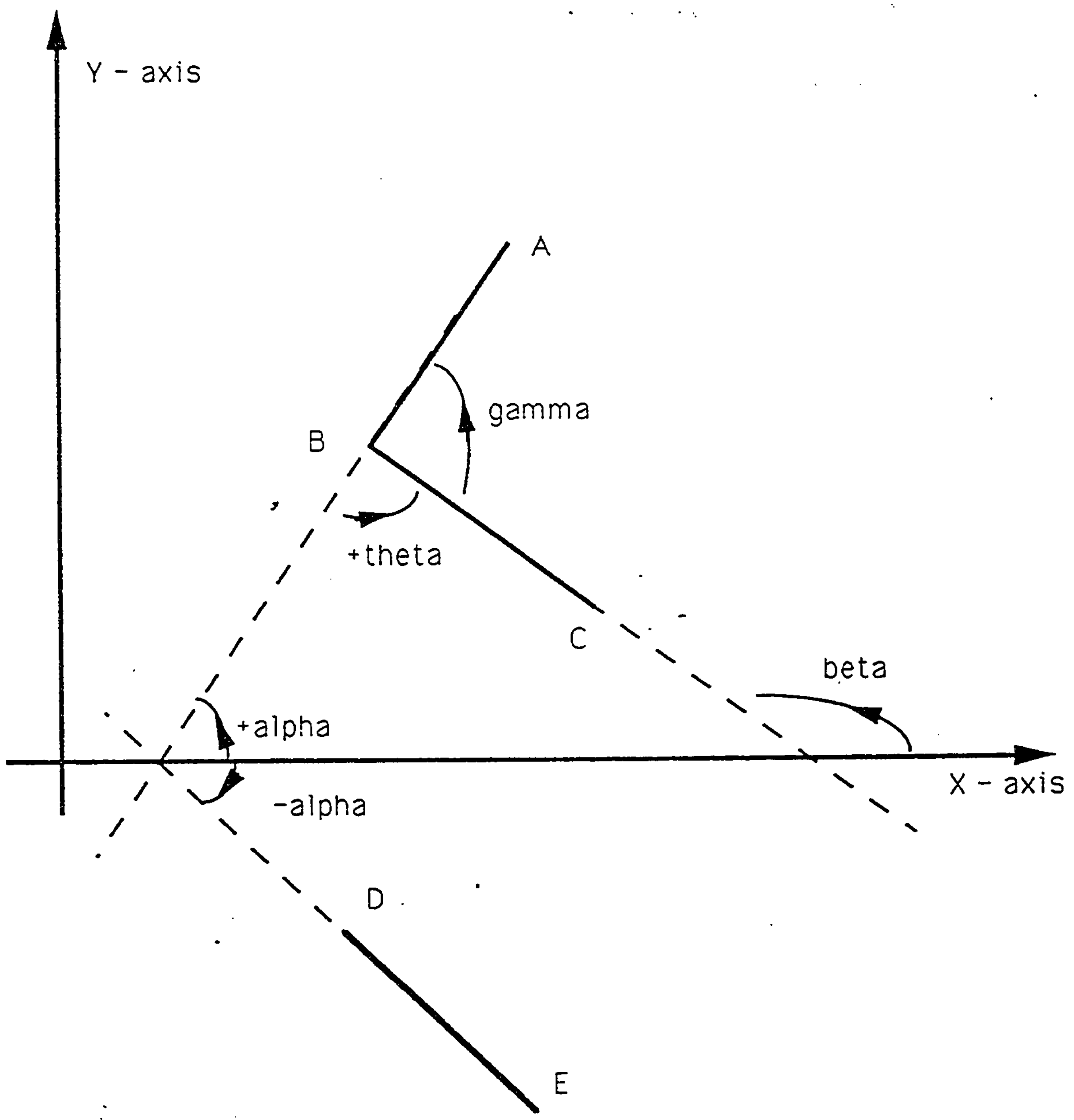


Fig 5.11 : Using the Arccos function

The ***VECTOR*** method (Fig 5.12) below has the benefits :

$$\text{COS}\ \theta = \frac{a1\ b1\ +\ a2\ b2}{|\,a\,|\ +\ |\,b\,|}$$

- ARCCOS function is evaluated only once.
- the exterior angle is obtained directly.
- the length and vectors are already available.

The calculation of angle θ is listed in Appendix B4. Note that the function ARCCOS could not be implemented directly in Turbo Pascal and was calculated using the available ARCTAN function :

$$\text{ARCCOS(x)} = \frac{\text{ARCTAN}\ (\sqrt{1 - x^2}\ )}{x}$$

Part 1 of Appendix B4 shows the evaluation of the angle. It takes care of vertical segments, two segments which lie on the same line and the addition of π if required. The vector method still does not identify the direction of angle change (clock or anti clock wise) but this problem is encountered in any method. Finding the ***DIRECTION OF ANGLE CHANGE*** between successive segments has to take care of :

- the direction of the segment i.e. for segment A-B of Fig 5.12 going from vertex A to B is different than going from B to A.
- the succeeding segment lies on the left or right of previous segment; i.e. whether vertex C lies on the left of segment A-B.

The procedure for getting the sign of the angle between segments is listed in Appendix B4 (2nd part).

It does the following :

a. find the slope of the first segment A-B : if A-B is vertical, the slope is set to a very large value, if not it is calculated from the ordinates of vertices A and B.

b. The X-ordinate of the succeeding vertex, C, is fitted to the equation of the straight line A-B .

c. The Y-ordinate of C is compared to the fitted Y value to determine if vertex C lies on the left or the right of line A-B extended.

d. lastly, the direction of segment A-B is considered. i.e. if vertex C lies on the left of segment A-B, it will lie on the right of segment B-A.

5.6 Normalizing Perimeter

One of the requirements is that the system identify an object even if its size, orientation or camera distance is changed and must match to the same polar models in the database.

The polar model obtained so far consists of only the segment lengths and incremental angles; the lengths are given in actual pixel lengths.

The model is of : Polar(obj) $= \{ P_i \}$ where $P_i = (\rho_i , \theta_i)$,

ρ_i = length of segment i .

θ_i = angular change between segments ρ_i , ρ_{i-1}

Two such models exist, $\{ P_{i(c)} \}$ and $\{ P_{i(d)} \}$. Also, the perimeter, fmcent, is given in actual pixel lengths. What is needed is to map the object to an invariant model; the most efficient (computation wise) mapping was to normalize the actual perimeter to a fixed value so that no matter what size the object is actually, its model always refers to one with perimeter of 2000 pixels.

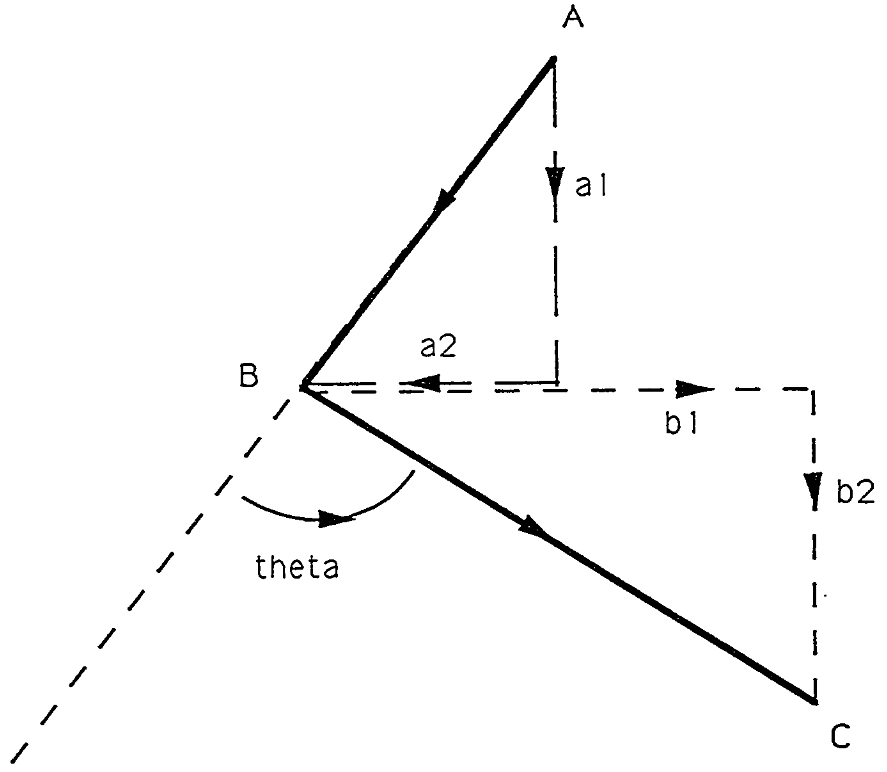


Fig 5.12 : The Vector Method

Only the ρ_i must be normalized because the θ_i is invariant under magnification.

The procedure for normalizing is listed in Appendix B5. It simply finds the ratio of the normalized perimeter (2000) to the actual perimeter (fmccnt) and multiplies each ρ_i of $\{ P_{i(c)} \}$ and $\{ P_{i(d)} \}$ with that factor.

A variable 'normdiff' which is a fixed fraction of the normalized perimeter is calculated to be used later.

5.7 Descriptor Array

Once the polar models are normalized, the next step is obtaining the important features or the Descriptor array (explained in Chapter 3). The features needed with the associated variables used in the program are listed below.

The features are only extracted from the coarse model $\{ P_{i(c)} \}$ because it overlooks noise and represents the object or class more generally.

Some of the features are already available like :

actual perimeter in pixels, fmccnt.

number of segments in Coarse model, count1.

number of segments in detailed model, count2.

The rest of the features are calculated in the procedure 'findfields' listed in Appendix B6. The following points are to be noted :

- even though 'count' is already available from the tracking algorithm, very small segments (less than 30 over a 2000 perimeter) might still exist due to noise and they are not counted; hence the value of count is decremented if such very small sides are encountered. This occurs very rarely in practice.

- any segment larger than 400 pixels (over perimeter of 2000) is counted as a long segment.

- the maximum positive or negative angle is found by a simple 'IF' checking in the iteration.
- large positive angles are those that exceed $+89°$.
- large negative angles were taken to be greater than $-49°$. It was noticed that if the image's contour is traversed in an anti clockwise direction, very few of the θ_i's were negative; they proved to be a good representative feature.

The variable used were as follows :

FEATURE	VARIABLE
largest segment length	larger1
number of large segments	alargeside
number of large positive angles	alargeanglep
number of large negative angles	alargeanglen
largest positive angle change	amaxanglep
largest negative angle change	amaxanglen
numb of segments in coarse model	count1
numb of segments in detailed model	count2

CHAPTER 6

LEARNING, DATABASE & RECOGNITION

This chapter will cover :

- the learning stage, where the models of a class as well as their features are saved. New classes can be added or new objects within existing classes can be inserted.
- the database which is responsible for saving the classes, their objects, their descriptors as well as its detailed polar segments. The files are organized in the DB in flat or indexed files for accelerating the recognition steps. The DB can also be viewed by the operator.
- the recognition phase where an unknown object will be matched to a cluster of classes. The sorted union of the clusters can be viewed. The distance test can then be applied to the unknown object and its matched class.

An overview of the system will be useful. Fig 6.1 shows the main menu available to the operator where he has the choice of first **Initialize FG-100** Board; here basically he sets up the digitizer to correctly operate with the host IBM-AT and initializes the board and checks visually that the snap, grap functions operate correctly; he can also adjust the lighting, while level . etc for optimal conditions. The second choice is the **Image Acquisition** where the user basically snaps an image, convert it to binary, extract the FMC contour and can save the binary image or the FMC contour to a file on diskette; these options were presented in Chapter 4. The third option is the **Learning & Recognition** submenu; the detailed modules are given in Fig 6.2 . The options under this are :

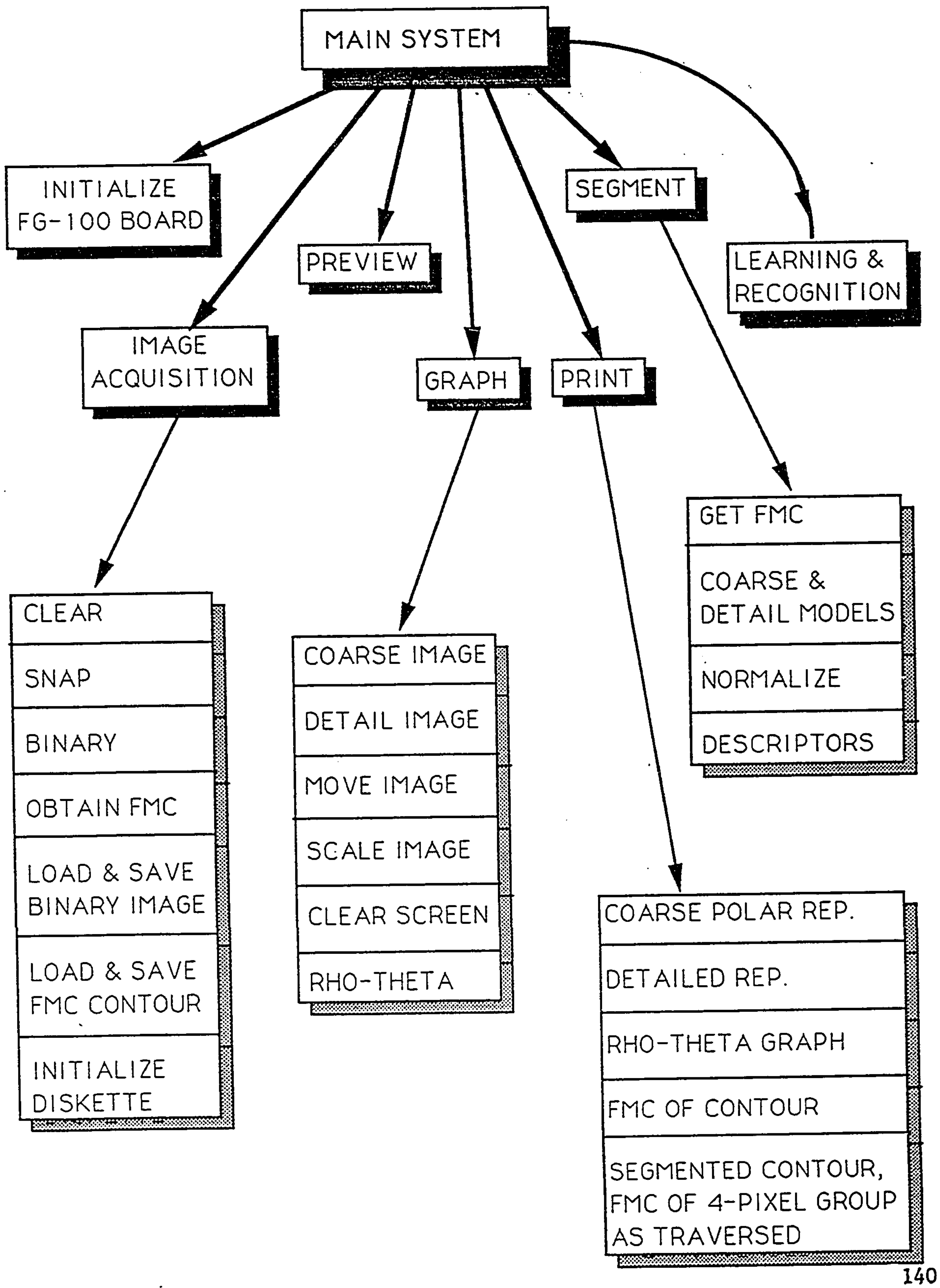


Fig 6.1 : System Modules Implemented

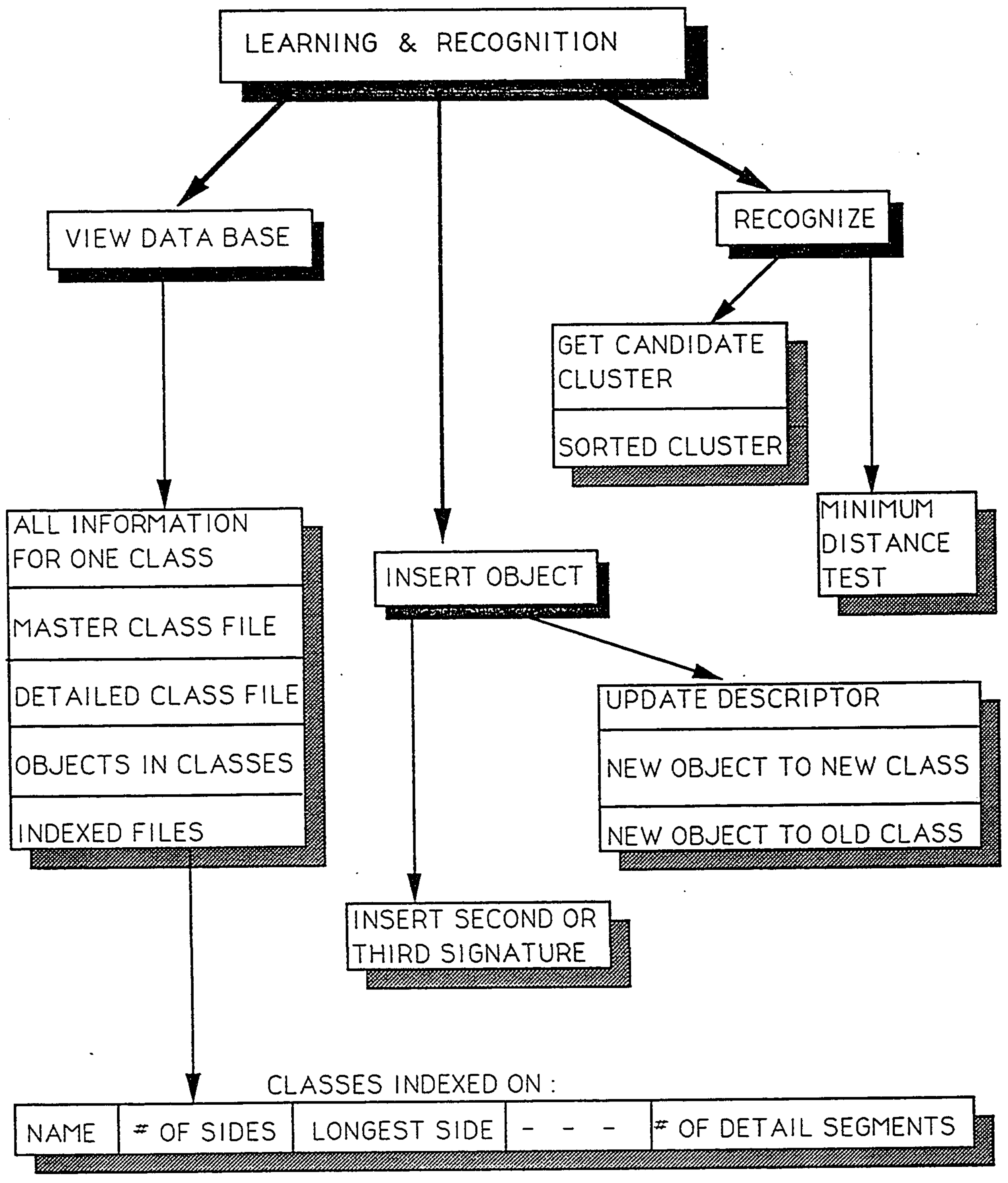


Fig 6.2 : Learning & Recognition Modules

- Obtain the FMC contour of an object, track the contour and obtain the segments at the coarse and detailed level, calculate the polar models, normalize the contour (or ρ_i to 2000 and finally obtain the Descriptor array from the Coarse model; all these steps were presented in Chapter 5.
- Graphical option allows the user to visually check an object's contour on the monitor of the IBM-AT, view either the Coarse image or the Detailed one or both as well as their 'rho-theta' graphs. This is a useful step before inserting a new object to the database since the operator can decide which class to include a new object. The image can also be moved across the monitor and displayed at different scalings.
- Print utilities allow the operator to obtain a hard copy of the two models, the coordinates of the vertices, the lengths and angular change of segments; also the rho-theta graphs can be printed as well as all the FMC of the contour. A useful tool made available is allowing the printing of the FMC of the 4-pixel groups, the detecting of corner points of both models as the contour is being dynamically tracked in the segmentation algorithm; this is necessary as a debugging tool.
- Viewing the database permits us to look at all the files in the DB, all objects within classes, detailed models of a class as well as the indexed files; the indexed files are sorted on each feature of the Descriptor array.
- Recognition matches an unkown object to its nearest class applying each decision results in different cluster which can be viewed as well as the final sorted union of the clusters where the closest classes with the frequency of match is listed. Finally the results of the minimum distance test is seen.

6.1 Database

The Turbo Pascal Database Toolbox is a small code database which provides the programmer with all the minimum tools of a DB upon which he can access through Turbo Pascal programs; the overhead is minimum and hence the efficiency is very good. The basic functions of a DB are implemented and they are :

- creation of flat or sequential files with separate attribute definition; each attribute can be of maximum length of 256 bytes and the record length can be upto 64 Kbytes.
- creation of index files
  an indexed file sorts a flat file with one or more attributes as key. The sorting utilizes the QuickSort algorithm, the most efficient sort available.
- insertion, deletion or updating of records of the sequential relations; the indexed files are immediately updated.
- sequential or indexed search for records is implemented through B+ trees, the fastest for finding and retrieving database information.
- a maximum of 15 files can be opened simultaneously.
- maximum number of records per relation is 2 million.
- TABUILD is a utility available to programmers if they desire to increase the efficiency of the DB by resetting the page size, the maximum # of pages in memory; the user can experiment and check the comparisons needed per key search, the searches satisfied in RAM memory and the disk searches needed. This is useful for very large DB's where the files are too large to fit in RAM simultaneously and have to be paged in on demand.

Databases revolve around the use of files; their functions are explained w.r.t.

Database Toolbox so that the procedures implemented in this thesis will be clear.

6.1.1 Flat & Indexed Files

Flat or sequential files store the overall and detailed information as records inserted sequentially. Each record is divided into attributes of character type (fields of numeric format must be converted to strings). An example of a flat file is given in Table 6.1 where 5 records are kept each with 3 attributes. The records are kept in the relation in the order of insertion.

Searching for records in such flat files is slow as the access is sequeential; i.e. to find all records with longest side between 200 and 400 can be done only by examining each record of the file one at a time. The process can be accelerated by indexed files. Let the flat relation be indexed on attribute 'longest' , i.e. the records are sorted according to the 3rd attribute. The resulting index file is given in Table 6.2-a. Note that only the record number and the attribute indexed on is duplicated and not the whole record as it will be wasteful to save whole records especially if they consist of many fields. Table 6.2-b shows a file indexed on field 'sides'.

Now searching for querries is very efficient as all the records sharing similar attributes are grouped together. Database toolbox , unlike some DB's allow index files with duplicate keys.

RELATION CLASS.DBF

	name	sides	longest
record_1	class-a	12	300
record_2	class-b	18	200
record_3	class-c	10	400
record_4	class-d	18	600
record_5	class-e	16	500

Table 6.1

RELATION LONGEST.NDX

	record_#_of_CLASS.DBF	attribute_'longest'
record_1	2	200
record_2	1	300
record_3	3	400
record_4	5	500
record_5	4	600

Table 6.2

RELATION SIDES.NDX

	record_#_of_CLASS.DBF	attribute_'sides'
record_1	3	10
record_2	1	12
record_3	5	16
record_4	2	18
record_5	4	18

Table 6.3

6.1.2 Interface & Creation of Relations

Relations are created in a separate file of extension .typ . A file is defined as a record of attributes each of type string, i.e. for the relation CLASS.DBF of Table 6.1, the definition will be :

```
classrec = record
            stat      : longint     :  used by system
            name       : string(20)   :  1st attribute
            sides     : string(2)     :  2nd attribute
            longest    : string(4)     :  3rd attribute
end :
maxdatatype = classrec :
maxkeytype  = string(20) :
```

This interface file (DB.TYP) is compiled by the TABUILD utility of Database Toolbox from the following command : **TABUILD DB.TYP**

The next step is creating the relations and index files:

```
uses DOS, TACCESS :
[ $I DB.TYP ]
var
      gen : datafile :
      gen1,gen2 : indexfile :

makefile ( gen , 'class.dbf' , sizeof(classrec) ) :
makeindex( gen1, 'sides.ndx' , 2 , 1 ) :
makeindex( gen2, 'longest.ndx' , 4 , 0 ) :
```

The MAKEFILE proceudre physically creates the file and names it class.dbf,

the record length in the 3rd parameter while the first parameter is the file name used by the system (gen) unlike the actual name (class.dbf) used by DOS.
The MAKEINDEX procedure is similar except that the 3rd parameter specifies the length of the key and the fourth one indicates if duplicates in the key are allowed or not.

Opening the files for insetion, deletion or querries is done by the following :

```
        openfile ( gen , 'class.dbf' , sizeof(classrec)  ) :
        openindex( gen1, 'sides.ndx' , 2 , 1 ) :
        openindex( gen2, 'longest.ndx' , 4 , 0 ) :
```

6.1.3 Insertion, Deletion & Searching

Adding a record to a normal relation or an indexed one with duplicates allowed is straight forward; first the contents of the record are assigned or read, next the record is added to the normal file and the inserted record number is returned in the second parameter of the Addrec function. This record number is needed if a related indexed file has to be updated. Note that since each attribute is stored as a string, any numeric input must be converted to a string; this is done by the function INTTOSTR listed in Appendix C3.

```
           recordnum : longint :
                readln(classrec.class) :
                readln(side):  classrec.sides    := inttostr(side) :
                readln(long):  classrec.longest := inttostr(long) :
                Addrec( gen , recordnum , classrec ) :
                Addkey( gen2, recordnum , classrec.longest ) :
```

Updating records for index files which do not allow duplicates is slightly more involved since we cannot insert a record if the new record's key is already present in the indexd relation; so first the index file is searched for the key, if found the new record with dupplicate key cannot be inserted. The steps are given below :

```
recordnum : longint :
          readln(classrec.class) :
          readln(side): classrec.sides   := inttostr(side) :
          readln(long): classrec.longest := inttostr(long) :
          tempcode := classrec.sides :
          Findkey( gen1 , recordnum , tempcode );
          If not OK then
              begin
                      Addrec( gen , recordnum , classrec ) :
                      Addkey( gen1, recordnum , classrec.sides ) :
              end :
          Else
                  write(' Duplicate attribute "sides" exist ') :
```

Searching for records is achieved by the four functions : Findkey, Searchkey, Nextkey and Prevkey. Findkey searches the file for a string given in the third parameter; the latter's value is destroyed; if the search is successful, variable OK is set to 1 and the recordnum is given :

```
write(' enter class to look for : ) :
read (lookclass) :
tempclass := lookclass :
Findkey( gen , recordnum , tempclass ) :
```

```
If OK then
        write(' class found ', . . . ) :
Else
        write(' class not found '):
```

Once the record is found, the neighbor records can be accesses Nextkey gives the succeeding record while Prevkey the preceding one.

FINDKEY function will search the index for a key that exactly matches the search key but the function SEarchkey allows DB search using only a partial key. SEARCHKEY takes the same parameters as Findkey but looks for any record with a key greater than or equal to the key being sought. An example : a querry to locate all classes in the DB with 'longest' side between 350 and 550 (although no classes exist with exact value of 350. The following procedure locates the first record with its 'longest' field greater or equal to 350 and continuously reads the succeeding records as long as they satisfy the querry that the 'longest' is less or equal to 550.

```
Searchkey( gen2 , recordnumber , inttostr(350) );
If OK then
    begin
        Getrec( gen , recordnumber , classrec ) :
        repeat
            write(' class found : ', classrec.longest ) :
            Nextkey( gen2 , recordnum , searchcode ) :
            Getrec( gen , recordum , classrec ) :
        until classrec.longest < = 500 :
```

```
        end :
    Else
        write(' all records less than 350 ') :
```

Deletion of records is pretty simple. Once the desired record is located with Findkey, Searchkey . etc , the record is deleted but first that record in all indexed files must be deleted before the master file record is deleted, i.e. to delete the record for class_a , assuming an index file for the 'name' attribute exists as gen3 .

```
    Findkey( gen3 , recordnum , 'class_a' ) :
    If OK then begin
            Deletekey( gen3 , recordnum , temp ) :
            Deleterec( gen  , recordnum ) :
    end :
```

6.1.4 DB Relations Implemented

Four master files are created, some of them were briefly explained in Chapter 3 and they are :

- GENOBJ.DBF is the main master file which contains the classes and their descriptors. This file is opened and used with the name GEN . Fig 6.3-a shows the attributes as :

* class	:	name of class
* sides	:	number of sides in Coarse model
* maxside	:	longest segment of contour
* largeside	:	number of sides > 250
* maxanglep	:	largest positive angular change

* maxanglen	: largest negative angle
* largeanglep	: # of angles > +80 degrees
* largeanglen	: # of angles < -5 degrees
* sidesdet	: # of sides in Detailed model
* numobjects	: number of objects in this class
* subclass	: # of reference signatures for class

The function of each attribute is self explanatory execpt for 'subclass' which denotes if the class has more than one detailed representation stored; this is needed for objects with not one single large starting segment but maybe 2 or 3 equally large segments. The graph of the object varies markedly depending from which of these large sides the model starts. More will be explained later.

- DETOBJ.DBF (used as DET) keeps the Detailed model(s) of a class; one class may have 1, 2 or 3 signatures. Fig 6.3-b shows the attributes as :

* class	: name of class
* sideord	: which segment of the contour
* length	: length of each segment
* angle	: angular change of segment

Note that if for example class_a has 5 segments, they will be entered in DET file in sequence; which segment will become segment number 1 will depend on the longest side of the contour : this is motivated by the recognition algorithm which has to find the distance between the graphs of 2 objects (Section 3.1.8) and to do so, the graphs must coincide at some common segment.

- OBJECT.DBF (used as OBJ) is given in Fig 6.3-c, the attributes being :

* class	: name of class
* objname	: name of object in class
* perimeter	: actual physical perimeter of object
* distance	: actual object-camera distance

One class may have several objects and the only way to differentiate them is some actual physical parameter; one is the actual perimeter (not the normalized one) of the object , another is the distance from the object to the camera lens. The first was available and used but no attempts was made to recognize an object within a class if the class was recognized.

- CLUST.DBF (or CLS), given inn Fig 6.3-d was used in the recognition algorithm when finding the classes with highest number of descriptors matching to those of unknown object. The attributes are :

* class	: name of class
* count	: frequency class matched

The interface file DB.TYP is listed in Appendix C2, its use was explained in Section 6.1.2. Once the interface to DB is run from TABUILD, the files can be created as listed in Appendix C3. The two functions INTTOSTR which converts all numeric values to strings (needed for relation fields) and STRTOINT which is the converse are also given in the same Appendix.

In addition to the sequential files, 11 index files are created to allow direct and efficient retrieve the data stored in the .dbf files; GEN1, GEN2, . . GEN8 are index files which have as key each attribute of the GENOBJ.DBF (or each

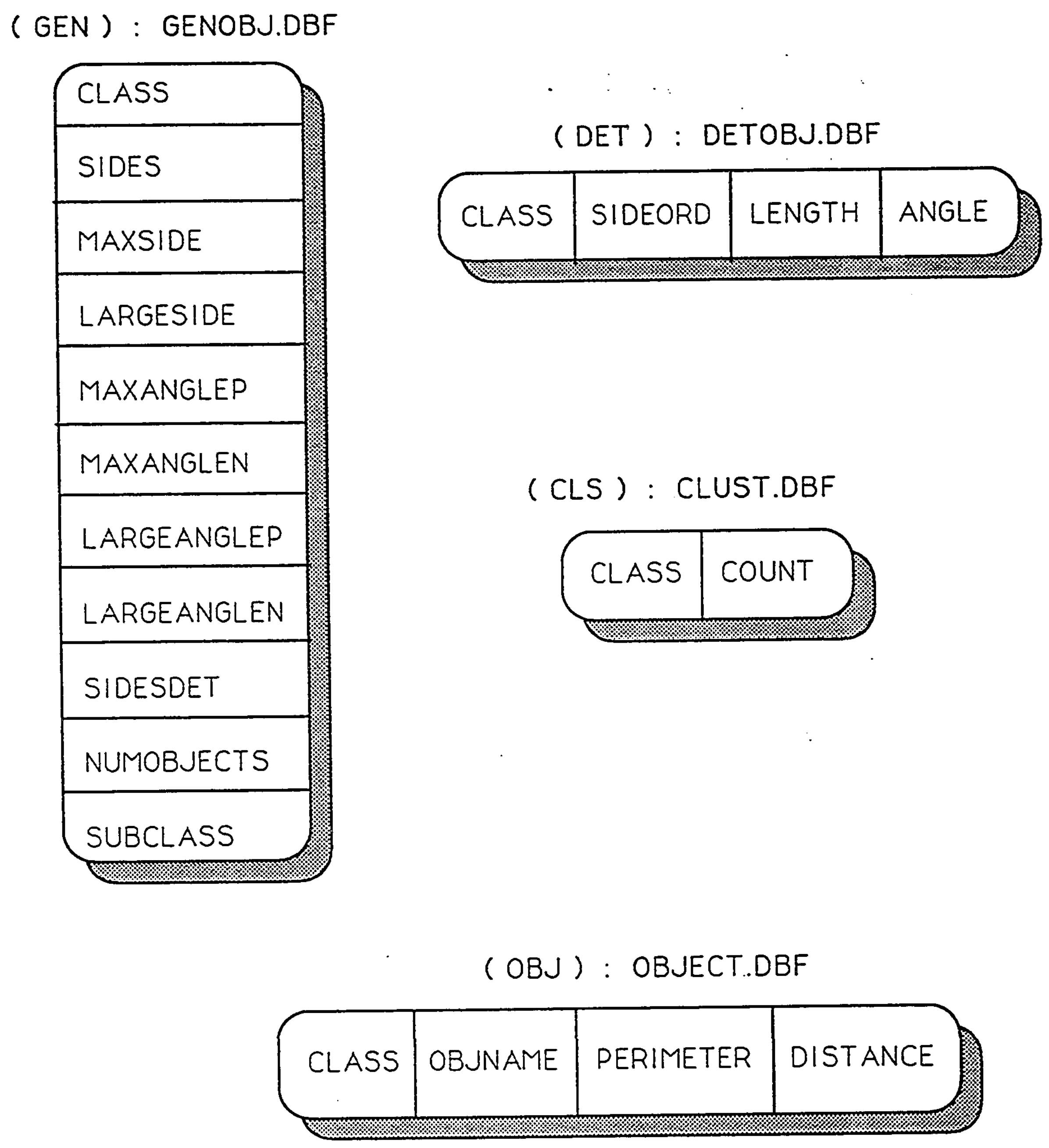


FIG 6.3 : Data Base File Organization

feaure of the descriptor) while DET1 is indexed on the 2 attributes class and sideord of DETOBJ.DBF . The index file OBJ1 have as keys attributes class and object of OBJECT.DBF . CLS1 is indexed on the combined attributes class and count of CLUST.DBF .

The files are opened by the statements given in Appendix C4, the format of which were explained. Similar statements close the file at the end of the session.

6.2 Learning Stages

The learning menu provides many menu options allowing the operator to segment an image contour, view or print it, preview the descriptor array before adding it to the DB in an old or new class; he may further view the DB.

6.2.1 Reference Signatures

It is clear how the descriptors of an object class are found and used but how exactly is the detailed polar model kept and in what sequence of segments.

Referring to Fig 6.4, the object given can be aquired at different positions or orientations (a, b) and their detailed polar models, although exactly the same in value of ρ_i and θ_i , will vary in the sequence; i.e. the Det_{obj} for (a) will have ρ_1 as L1 while that of (b) will have ρ_1 as L3. Naturally their rho-theta graphs will differ and if a match of distance between their two graphs is calculated, it would fail in recognizing the match even though both graphs refer to the same object. Two ways implemented in literature are :

- repeat the rho-theta graph for the perimeter (signature) end to end such that an extended signature which covers the perimeter two times is kept (Fig 6.5-a) .

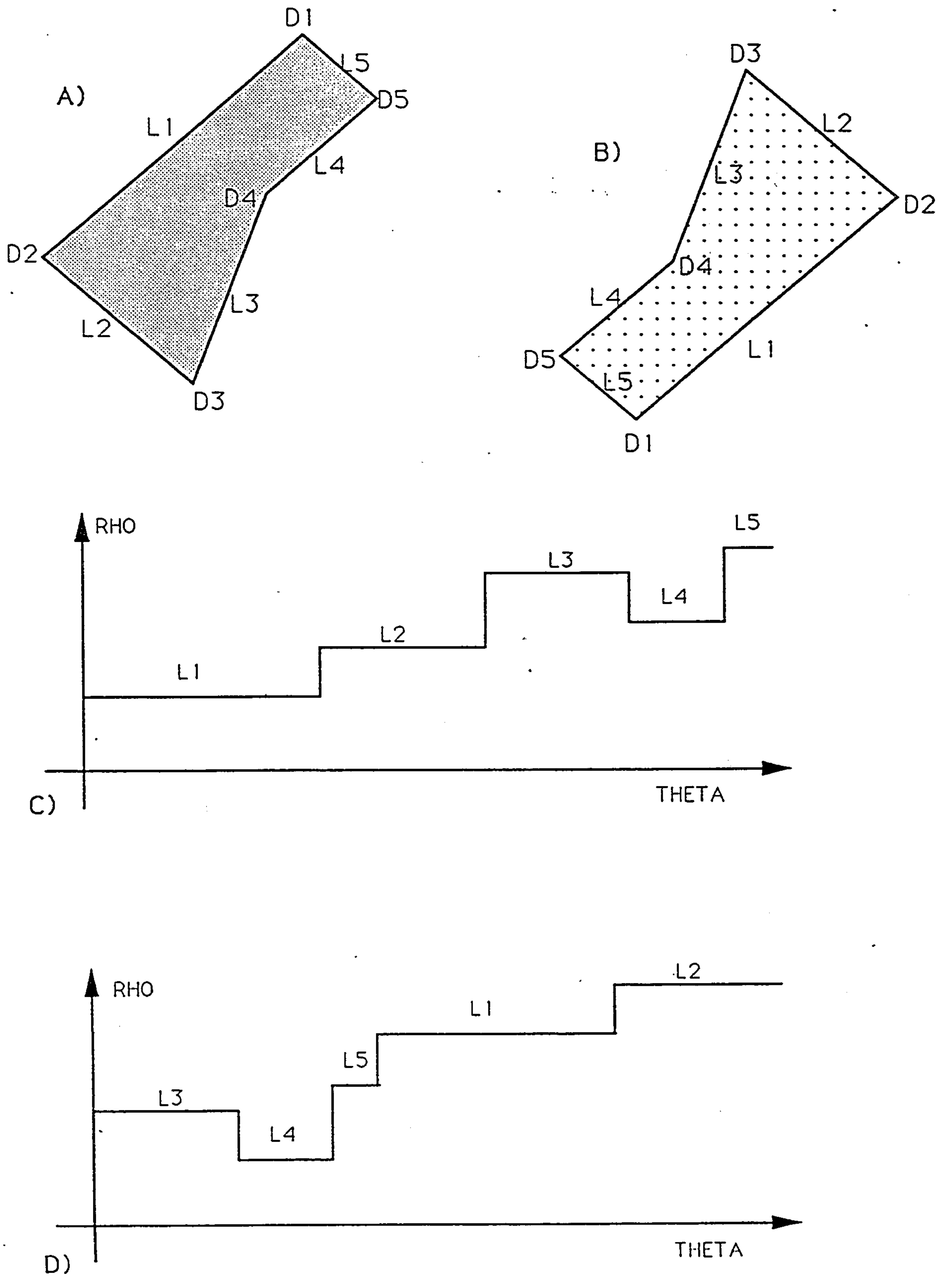


Fig 6.4 : Reference Signature

- the signature is not extended but when two signatures need be matched, one signature is kept fixed and the other is repeated shifted and matched (Fig 6.6-b); this method is obviously slow especially for objects of many segments. i.e. if unknown object is Fig 6.4-b while the object known to the DB is Fig 6.4-a, let Fig 6.5-b represent the signature stored in DB while Fig 6.5-b1 is the unkown signature, the first match fails. The signature b1 is shifted to get b2 which is mapped to b and again the match fails; this shifting or circular rotating of the signature continues until b4 matches b successfully.

None of the methods were implemented because they are costly instead the signature of an unknown object was shifted prior to its insertion to the DB. The shifted signature is obtained such that the first segment is always the longest; we call this the Reference signature $SIGNAT_1$. Given an unprocessed signature e.g. Fig 6.5-b1, it will not be stored as such but shifted such that the $SIGNAT_1$. satisfies the rule $\rho_1 > \rho_j, j = 2, 3, .. n$ The reference signature for Fig6.4 -a and -b will always be the same, as seen in Fig 6.5-b and -b4. The distance test is very much simplified.

One problem with reference signatures is the confusion resulting from having 2 or 3 nearly equally longest segments. If the object is regular like a square or regular pentagon, there is no problem since the signature is symmetrical in their ρ_i , $\dot{\theta}_i$ but what if the object is not regular like that of Fig 6.6-a ? The reference signature requires first finding the longest segment but for this object, there exists 2 longest sides namely L1 and L6 such that 2 reference signatures must exist in the DB to take care of the object appearing in any position.

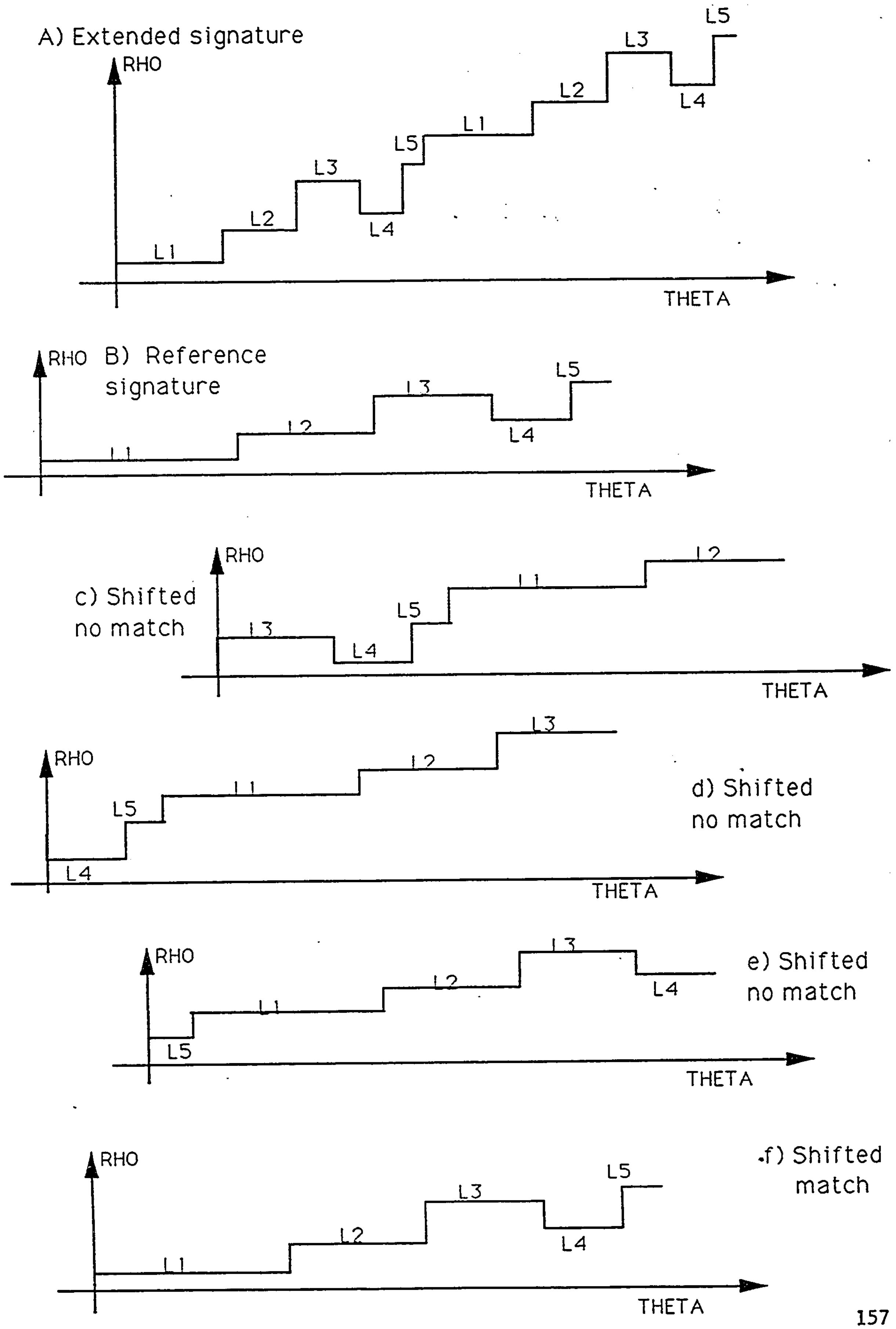


Fig 6.5 : Shifting Reference signature

Therefore the DB will save both SIGNAT_1 and SIGNAT_2 .

Procedure **sortmaxes** from Appendix C5 finds out the longest segments of the contour of an object. Note that the **contunit** segmentation algorithm has made available the polar models in arrays leng, leng1, angle, angle1 and the links between the two models in ptrdetail, ptrdetail1 and the size of each model in vertptr (count1) and vertptr1 (count2).

The first loop in **sortmaxes** locates the longest segment of the contour and saves it in variable **larger1.** A pointer **largepos1** keeps the value of which segment of original signature is longest; this is needed for shifting.

The second loop goes back and checks each segment to see if it is equal or nearly equal to larger1. If a new segment of length differing by a value (30 in this case), it is considered as a 2nd largest segment and variable **larger2** keeps the length, **largepos2** points to the segment correspondind to 2nd largest segment and a flag **largestat2** is set to 1 to indicate that 2 reference signatures exist.

The loop also checks for a possible third longest side, **larger3** , keeps a pointer **largepos3** and set a flag **largestat3** to 1 to indicate 3 reference signatures exist.

Once the longest side(s) are located, the signature has to be circularly rotated such that ρ_1 corresponds to the longest segment; this is done in procedure **dataint** listed in Appendix C6 which does 2 functions : one to reorder the ρ_i , θ_i to obtain the reference signature and two convert all the values to integer ('round' function in Pascal).

Basically the procedure transfers all the ρ_i , θ_i from and after the longest segment to the first elements of new arrays called length1, angle1 ; then the

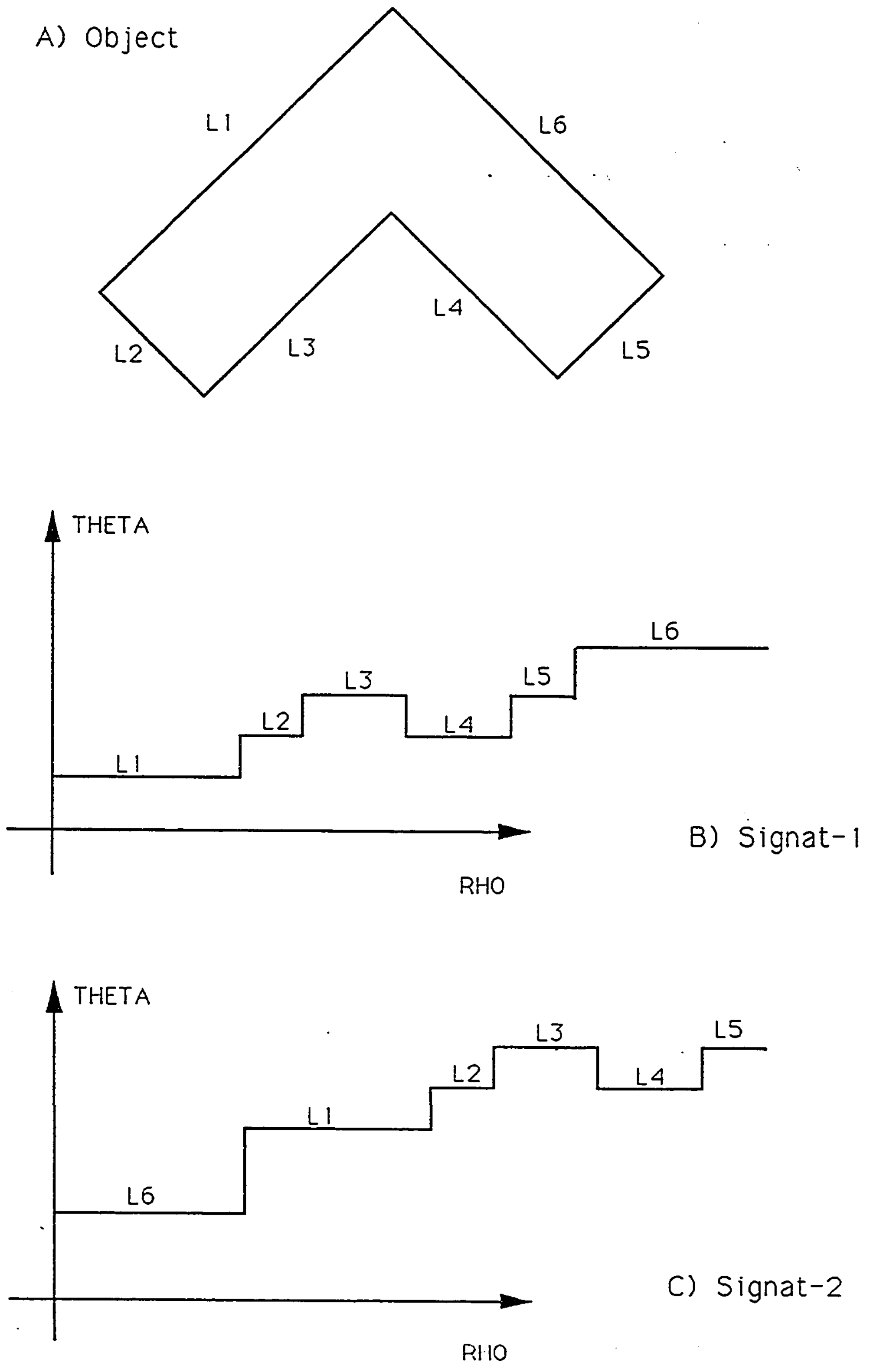


Fig 6.6 : More than one Reference Signature

ρ_i , θ_i from segment 1 upto the segment preceding the longest one (of old signature) are transferred in sequence to the last part of arrays length1, angle1. Procedure dataint will be called once, twice or thrice depending on how many reference signatures exist.

6.2.2 Learning Stage Utilities

Referring to Appendix C1 (main menu of Learning & Recognition), the operator once he segments an image, normalizes it, obtains the reference signature is free to insert the new object to the DB but before he does this, he can view the normalized object (graphically), print the models and signatures, view the DB classes.

The **preview utility** allows the display of the descriptor array for the object, its detailed model etc. The procedure is given in Appendix C7 and just prints out to the CRT all these values and waits for the user to view them and then returns to the control menu.

The **graphics utility** is a useful visual tool, the functions of which were given in Fig 6.1 . Procedure **graphutil** of Appendix C8 shows the submenus; they can be controlled in the graphics mode so the operator can choose any command from within the graphics mode without switching to text mode; the menu is key driven and simple to use. The graphics mode is switched to by the 'detect' and 'setgraphmode' functions to CGA mode and some default values for the center, the scaling factor are set and the object drawn by calling procedure **contdraw** from Appendix C9. Contdraw links up the vertices of the objects by straight lines (the X- and Y- coordinates are known) starting with the 1st vertex as

center (offset by variables XCEN, YCEN); the object's coordinates are scaled up or down (by variables XSC, YSC).
Both the Coarse and Detailed models can be viewed separately or overlapped and even the refernce signature can be displayed from procedure **rhotheta** of Appendix C10. Once the axes are drawn, the first segment of the signature is drawn horizontally; the second segment is also drawn as a horizontal but is shifted up or down by a displacement proportional to the angular change between 2nd and 1st segment; these steps are repeated for all steps.
The whole CGA screen can also be dumped to the printer by the Print Screen key through DOS.
On exiting from the graphic utilities, the text mode is restored.

The **printshow utility** prints both models, their graphs, the FMC and more importantly the Segmentation algorithm as it goes through the contour 4-pixel group at a time. The 'write(lst, . . .)' function which instead of writing to the monitor, dumps the characters to the connected printer. Appendix C11 shows the loop which prints out the Coarse model parameters.

Viewing the DB utility allows the interactive examination of the current DB by calling procedure **displaydb** (Appendix C1) which in turn displays the view DB submenu of Fig 6.2 . Retrieving the records from the .dbf files (to view genobj, detobj or object relations) is done by a simple loop whose index runs from 1 to number of records for the file and using 'Getrec ..' procedure, and then displaying the record.

Viewing the index files is as simple and is listed as procedure **printfields** in

Appendix C11 (not the whole procedure is given, only for 3 index files); the procedure's parameter of 1 to 8 specifies which index file is viewed. The first step is to set the pointer to the 1st record of the index file by the call ' Clearkey (file name) ' and then to step through each record in that file by ' Nextkey (filename , record number , record area) ' followed by a ' Getrec (file name , record number , record area) '. The record number increases from 1 to the last record (known as genreccount). Once the record is read, it is written out, string attributes being converted to numeric first.

Viewing all the information for a certain class is shown in Appendix C13; the 1st step is to show the master record for the class which the user asks for. A ' findkey ' to GEN1 with the required class locates the record number needed for the class, and if found, the record is read with ' getrec ' and all the attributes for that class are shown.

All the objects within that class is displayed next, the related index file OBJ1 is searched for the class and when found accesses for as many times as there are objects in that class; number of objects is obtained from 'numobjects' attribute of master record just read.

The detailed polar model is displayed next by searching the DET1 index file with the search key as the class + 1 (1st side); a loop stepping through all the succeeding records reads the records and displays them (how many times depends on the number of segments in the detailed model - from 'sidesdet ' attribute of master record).

Timimg utility is useful in testing the system and generating response times of various steps; more will be given in Chapter 7. If it is required to find how long a process takes, the Turbo Pascal function ' gettime (hh, mm, ss, hh) ' will save the hour, minute, second, 1/100 th second into variables hh, mm, ss, hh respectively. To time a process, gettime is called twice (as shown in Appendix C14) and the difference in those 2 calls is calculated. A simple substraction of the variables will not do the job because care must be taken when 1/100 th seconds cross from 99 to 0 or when second cross from 59 to 0; the procedure timediff (Appendix C14-b) takes care of it.

6.2.3 Inserting Objects to DB

Once an object is segmented and its descriptor array obtained and previewed, it can be added to the DB in the 'update' option of Appendix C1. The procedure **addclassobj** of Appendix C15 is called and it essentially follows the simplified flowchart given in Fig 3.7 (Supervised Learning module). Note at this point, all the attributes needed to add to the DB files, namely the Descriptor array, the detailed model are already available and that a $SIGNAT_1$ is already aranged. Adding the object (Appendix C14) first prompts for the class name to insert the object to. A search is made in the GEN1.NDX file to quickly locate the presence of the class by 'Serachkey' command.

If the class already exists, adding a new object involves only the updating of 'numobjects' attribute of the GEN file and appending a new record for the OBJ file. It is not required to add a new record for the class in the GEN file, nor add a record to the DET file since all objects within a class share the same features. The user is prompted to ensure his wish to ensure the object and then procedure **addobjtoclass** of Appendix C16 is called. The name of the new object is needed, the perimeter is already known and a new record is added to OBJ file. The OBJ1 file indexed on OBJ is also updated by using the combined key of class name and object name. Finally, the 'numofobjects' attribute is incremented by 1 and the record rewritten back to GEN file (by 'Putrec').

If the class to insert is a new one, procedure **addclasstodb** directs the flow to other procedure to create a record for the class with the descriptor, insert the detailed model and insert the first object for the class.

The first call is to **inputgenrec** in Appendix C17 which simply inserts the record of GEN file with the Descriptor array, the name of the class, sets 'numobjects' attribute to 1 as this is the 1st object of this class and finally 'subclass' attribute is 0 to indicate the presence of only 1 reference signature used up to that point.

The **addkeytogen** procedure (Appendix C18) is called next to actually insert the created record to GEN file of the DB as well as updating all the related index files.

Next, the object name and perimeter is added to OBJ and index OBJ1 is also updated, essentially the same steps in procedure **addobjtoclass.**

The last update is inserting the detailed model or $SIGNAT_1$ inside the DET file; this is done by procedure **inputdetrec** of Appendix C19. Note that this procedure accepts an argument of 0, 1 or 2 (corresponding to $SIGNAT_1$, $SIGNAT_2$, $SIGNAT_3$.) If the detailed model refers to $SIGNAT_1$, then the name of the class is used as such, but if $SIGNAT_2$, or $SIGNAT_3$ is inserted, the classname is affixed with a '1' or a '2'. The ρ_i and θ_i of SIGNAT is then inserted in the file DET, record by record. Each record corresponds to one segment and the key attribute is the classname and segment number. The

DET1 index relation is also updated.

The **addclasstodb** described above, called from Appendix C1. After the insertion of a new object in a new class, it is also required to insert $SIGNAT_2$, or $SIGNAT_3$ if they exist; they exist if variables largestat2 and largestat3 are 1 (set in procedure sortmaxes). So if $SIGNAT_2$ exists, the operator is prompted automatically.
The next important step is to reorder the signature to obtain $SIGNAT_2$ by calling **dataint.**
Then procedure **addsubclass** is called and the new signature added to DB.

6.2.4 Last Record

The operator goes through the Learning stage and adds all the objects within classes and the system is now ready to accept the image of any newly acquired object and recognize it as belonging to any of the classes in the DB.
One problem remained : it was noticed that while searching the DB relations, specially the master file GEN that sometimes the record pointer loops past the end of the last recod. This is specially true when searching for attribute values. The problem was resolved by inserting a last record at the very end of the file GEN or GENOBJ.DBF such that all its attributes have a value much greater than those for any class. Appendix C20 lists procedure **lastrec** which creates a record with all 'numeric' attributes of 2000 (an arbitrary maximum number). Once all the GEN files and all related index relations are opened, the last record is added by 'Addrec' and all index files also updated through 'Addkey'.

6.3 Recognition Stages

The recognition is broken in two major stages, namely find all classes satisfying an unknown object's descriptors resulting in several classes each being possibly the candidate matching class with different probabilities; the next stage is applying the minimum distance test between each of the candidate classes's signature(s) to that of the unknown object, the order of the test depending on the probability of the candidate classes. When the recognition stages are run, it is implied that the unknown object has been segmented and its descriptor array is available. The recognition stage follows the chart given in the Recognition module of Fig 3.7 .

6.3.1 Cluster Classes

The first stage is obtaining all classes which have similar descriptors to that of the unknown object; since the descriptor array consists of many features, many classes will be found, each satisfying some features. The trick is to find the classes which have the highest number of features common; therefore each time a candidate class is found which matches in a certain feature, a frequency count is incremented for that class. At the end of the search of all features of the descriptor array, the class with the highest frequency count is the class the unknown object belongs to with the highest probability. The class with 2nd highest frequency count is the next most probable belonging class if the object does not belong to the first class.

The design of the Clustering algorithm proposed in Chapter 3 asks for finding one cluster of classes for each feature and then finding the union of these separate clusters. The algorithm implemented however simplifies the clustering process so that the 'cluster union' step is not necessary.

Procedure **testsearch** from Appendix C21 first determines the bounds of search for each feature of the Descriptor array; i.e. if the unknown object's maximum positive angle = 130°, it is not practical to find only the classes which satisfy the feature of exactly 130°. A more practical approach will be to find all classes of DB with their maximum positive angle between 125° & 135°. That is why when the procedure **searchclasses** is called, an upper and lower bounds are specified as arguments.
The call ' searchclasses (n , upper , lower) ' will find all the classes such that feature **n** of their Descriptor array lies between the **upper** and **lower** bounds (± value of feature n of unknown object).
Searchclasses is called as many times as their are features in the Descriptor array, in this case 7. Naturally the tolerances for upper and lower bound will vary on the meaning of the feature; i.e. the number of segments cannot have a wide range of search while angles can cover a tolerance range of ± degrees.

Procedure **searchclasses** will find all the classes satisfying feature n within upper and lower bounds and for each class update its frequency of occurrence. The procedure (Appendix C22) first converts upper and lower to strings as the attributes of GEN are such. Then the index files are searched to find the first class satisfying lower bound; this is found through 'Searchkey' command with serach key equal to lower bound. Note that if feature = number of sides, index file GEN2 will be used or if feature = maximum positive angle, index GEN5 is accessed. Also to be noted that the classes in each class are already sorted on their feature value; therefore once the 1st class satisfying lower bound is found,

all the candidate classes which will satisfy the range lower - upper are located immediately after in contiguous records. Index files provide the most efficient manner of accessing all the candidate classes.
Once the first class is read (by 'Getrec'), obtaining the other classes is possible by simply reading each succeeding record of the index file (loop 'locs1' in Appendix C22) and accepting each class with feature below the upper bound. If the class's feature exceeds the range of upper, all the candidate classes have been found and the loop terminates.

Each time a candidate class is found, its total frequency of occurrence must be updated. This is done by calling **insertclust** listed in Appendix C23. Before the recognition starts, the two 1-dimensional arrays **clscls** (cluster class) and **clscnt** (class frequency count) are all set to 0 's and the variable **numcls** (number of candidate classes) is also 0 - no candidate classes found yet.
Each time a candidate class is found in procedure searchclasses routine insertclust is called : either the class found already exists among the candidate classes of array **clscls** in which case, the only thing to do is to increment the frequency count for that class; this is exactly what Appendix C23 shows.
However if the class found is not already a candidate class in **clscls** it must first be inserted in the array, its corresponding frequency count in array **clscnt** set and the variable **clscnt** incremented to show the increase of total candidate classes.

One important improvement implemented in the insertclust procedure is the association of a weight to the frequency count; i.e. some features of the Descriptor are more powerful than others, for example negative angles much greatly define the shape of an object than the normalized maximum segment length; so if a candidate class satisfies the former feature, its frequency count should be increased by a higher value than if it satisfied the latter feature. So each ′ Inserclust (candidate class , weight) ′ when called for different features will vary the value of this weight.

Once all the classes satisfying the Descriptor search with their frequency count is obtained, they are sorted such that the most probable class (class with highest hit) is first in the list; the procedure **sortclust** given in Appendix C24 uses the Bubble Sort technique to arrange the arrays clscls and clscnt in descending order of clscnt (Fig 6.7) .

The operator can look at the resulting sorted cluster by calling procedure **showclust** of Appendix C25.

6.3.2 Minimum Distance Test

The last step in the recognition is to match the class with highest 'match' probability through a minimum distance test as shown in Fig 3.7 (Recognition Module); the distance test is finding the area between the 'graphs' of the unknown object and the identified class. If the test fails, the next class in the list is taken and tested until one class is found such that the area between its graph and that of the unknown object is below a minimum theshhold. The procedure **testsearch2class** in Appendix C26 will first transfer the normalized ρ_i , θ_i of the unknown object (from arrays length2, angle2) to two 1-dimensional arrays **la** and **ta** which will be used by the procedure areadiff. Variable **maxa** is set to count2, or the number of detailed segments for the unknown object.

Candidate class	Frequency of occurence
class f	5
class b	4
class s	8
class e	3
–	–
–	–
class p	2

Fig 6.7 : Class clustering & their frequency

Next the ρ_i , θ_i of the class to identify with is retrieved from the DB and transferred to temporary arrays **lb** and **tb** which are also passed to areadiff. Variable **maxb** is set to the number of detailed segments for that class, or from the attribute genr.sidesdet .

The DB is accessed by first checking if the class exists and getting its record number in the master file (genrecnum) and then accessing the Detailed model for one subclass of that class (note that one class can have 1, 2 or 3 reference signatures). The function 'Searchkey' is used to access the first segment of the Detailed model, stored in DET1 index file. Once this first record is read, all the following segments of the Detailed model are read sequentially since they are sorted by segment number and since the number of segments is known. Each time a segment (its length and angle) is retrieved in the record, it is transferred to arrays lb and tb.

The last step is calling procedure **areadiff** listed in Appendix C27, which takes as input arguments : **la** , **ta** and **maxa** for one Detailed Model and **lb** , **tb** and **maxb** for the second Detailed Model and calculates the area between their two graphs and returns that value in variable **ar** .

The algorithm followed by procedure **areadiff** will be explained by referring to Fig 6.8 which shows 2 graphs to be matched in (a) and (b). The following points are to be noted :

- the ρ and θ for each individual segment is absolute, i.e. the length of a segment is only its own length and not its length relative to the origin of the graph. The same applies to the angles.

- the 1st segments of the 2 graphs do not necessarily match in their θ.

- the number of segments for the 2 graphs are not always equal.

- lastly, as noted in Section 3.1.8, the area between the 2 graphs will not be found iteratively over a fixed length δ_p as given equation

$$\text{dis}(\{P_{i(d)}\},\{P_{i(d)}\}) = \sum_{rho=0}^{perimeter} |\theta_i(\rho) - \theta_j(\rho)|\ \delta_p$$

Rather the area between corresponding segments will be calculated over the whole segment length; this will obviously accelerate the process.

The first thing done by areadiff is to align the first segments of the two graphs, ta(1) and tb(1) ; this can be done by equating one to the other or setting both to 0 . Graph A and B are now aligned in Fig 6.3-c; the next step is to calculate the ta(i) and tb(i) for the 2nd and onwards segments so they are relative to the graph's origin.

Variables **i** and **j** are used to point to which 2 segments of graph A and B are current. They are initialized to 1. Referring to Fig 6.3-c, the area between segment 1 of A and segment 1 of B is to be found, not by discrete intervals δ_p but over a long interval; this interval is the minimum of la(1) and lb(1). This minimum of 2 corresponding segments is found by procedure **min** given in Appendix C28. It takes two segment lengths **x** and **y** as input and calculates the difference between the lengths of x and y and returns this difference in y. Further, it returns a 1 or 2 in variable k to indicate if length x or y is the smaller.

Procedure areadiff calculates the area between the 2 segments by multiplying this minimum length by the difference in their angles. This partial area is summed to variable ar .

Next i and j must be updated to move on to the next segments. If both segments were equal ($y = 0$), then both i and j are incremented. If $k = 1$, segment i is smaller than segment j, so only i is incremented. If $k = 2$, the reverse is done. These steps are repeated over all the segments of A and B until the whole perimeter is traversed.

This difference in area between the graph of an unknown object and that of a class from the DB is returned to the control module, and if it is within a minimum 'matching' distance, the unknown object has been recognized and is accepted as belonging to that class.

CHAPTER 7

RESULTS & ANALYSIS

All the algorithms presented in the previous chapters were tested for their behaviour. Some of the parameters used in the algorithms like the scaling factor to correct non square pixels were obtained by experimentations. The results of some of the tests can be shown in this paper while others like the display of the contour on the TV monitor, the grapical utilities, the printing utilities cannot. The following sections give some brief results and timimg statistics.

7.1 Image Acquisition & Diskette Utilities

The image acquisition utilities performed without problems. The GRAB, SNAP, CLEAR operations worked instantaneously.

7.1.1 Binary Image

Converting the grey level image G(512,512) to the binary image B(512,512) worked smoothly; the speed of the process is determined by :

- the speed of access of the FG-100 frame, access had to be done through 8 consecutive 'read' of the board.
- the clock speed of the INTEL 80286 processor on board the IBM-AT, equal to 6 MHz.
- the resolution of the image, 512 rows by 512 columns.

All of these factors were fixed and if changed, could tremendously increase the speed of the process, namely, the image digitized and stored on the FG-100 board could be accessed as one block in one 'read' if the PC could run a different OS which does not restrict access of memory to a maximum block of 64 Kbytes; second, if an INTEL 80386 or even 80486 were used with speeds of

over 20 MHz, the efficiency will increase greatly and lastly if instead of using a 512 by 512 image, a resolution of 256 by 256 could be used, it would accelerate the process without losing any details of the image.
The threshold value is operator selectable and it was found that in normal room lighting, a hexadecimal value of around 40h to 50h proved ideal.

7.1.2 Obtaining FMC of Contour

The algorithm implemented to extract the border pixels of the image was successful in all tested objects. The following should be noted :

- the Directional 0-to-1 transition algorithm implemented fares much better than Duda's method (Section 4.5.1) in that it never misses part of the image nor is the contour unpredictable.
- the implemented algorithm is better than the usual 0-to-1 transition technique; although no detailed comparisons were made, it is estimated that it is at least 50 % faster for typical straight segmented objects. The reason is that it does not predictably check for the neighboring pixel in a fixed manner (Section 4.5.2) but rather uses the current segment direction to find the most probable candidate neighbor; for straight line segments it will successfully find it at the first trial.
- the algorithm simultaneously returns a linked chain of the contour and converts the image to a 1-pixel thin border. In some other research, 2 or 3 steps are needed namely the segmentation process, the edge thinning process and the segment linking step.
- the coordinates of the start pixel is obtained directly, the perimeter in pixels is also calculated without any extra cost.

- the interior pixels within the object are not included in the contour; this is good so that the algorithm does not stray into the interior pixels and loop forever.

The results of the algorithm cannot be shown here since the isolated contour is displayed on the TV monitor.

7.1.3 Diskette Utilities

The utilities for initializing a diskette in drive A , the saving and loading of both the binary image of the FMC contour, selecting specific files on diskette to operate on all perform very well. Images or contours saved can be redisplayed on the TV monitor without a hitch.

A possible improvement would be to allow the saved files to reside on hard disk rather than on a floppy as the seek and access time of hard disks are more favorable; the problem would be to find the exact sector numbers of these files. Unlike a floppy, the sector numbers of files on hard disk will vary depending on in which directory they reside, if they have been erases, the size of the files, the overwriting of new files . etc and taking care of all this from the Assembly level is tedious.

7.2 Segmentation Algorithm

The contour tracking algorithm which breaks up the border to corner points and obtain the 2 hierarchies of Polar models is not simple; it worked fine for all straight lined objects but some problems were encountered for complex objects with very many curves; the contour is broken up into too many segments.

The advantages of the algorithm are :

- it makes use of the FMC directly.

- it saves the coordinates of all corner points.

- it calculates the ρ_i , θ_i directly.

- it finds 2 hierarchies of Polar models within one loop.

- it gives a representation which is invariant under translation, rotation. Size of the object is also invariant when the model is normalized as well as the object to camera distance.

- it corrects the problem of non square pixels through software.

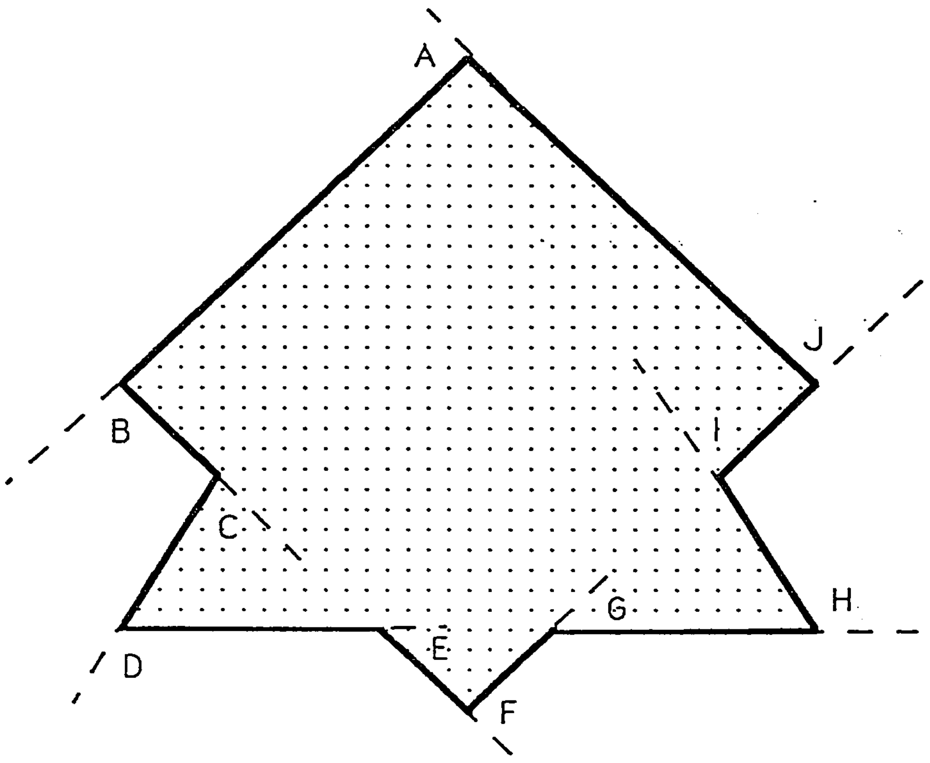
The time taken is given further and was very good but it can be further improved if executed on a faster machine.

7.2.1 Coarse & Detailed Model

The two models were generated with accuracy and their correspondence was exact. The pointer linking both models also were accurate. Fig 5.4 shows the 2 models and their correlation.

7.2.2 Effect of Orientation

The algorithm fares very well for the same object, no matter how it is translated or rotated in space. Many objects were tested from simple triangles to the more complex one given in Fig 7.1 . The actual angles and lengths of the segments are also given. (all the angles are given in degrees). The object was tested for at least 10 different positions (translation, rotation) but only 3 will be shown for brevity. Tables 7.1, 7.2, 7.3 shows the Coarse Model representations after the algorithm is run. The position of the part is shown at the top of each table. The lengths and angular chage are shown as well as the errors or deviation (absolute) from the actual values of the object. Only the deviations for the angle



ANGLE	IN DEGREES
A	+90
B	+90
C	-75
D	+120
E	-45
F	+90
G	-45
H	+120
I	-75
J	+90

Fig 7.1 : tested object

were shown because the lengths were very accurate and showed litte deviation in their ratio with the actual segment lengths..

The results are very good. A maximum error of only 5 degrees occurred; this is favorable especially if the fact that the digitizing pixels not being square is considered. It is proposed that if a better suited CCD camera was used, the results would be even better.

Therefore it was verified that the Polar representation successfully modelled the object in an invariant way.

7.2.3 Non Square Pixel Correction

The fact that the pixels of the CCD camera are not square brings out the serious problem of obtaining different angles and lengths of the same object when positioned differently. This effect is random and cannot be predicted and had to be corrected through software or else the whole recognition stages would fail. It was shown in Section 5.4.2 how the correction was achieved; the scale factor was obtained after many trials some of which only can be listed, in Table 7.4 . Comparing the angles with the actual ones (Fig 7.1), a scale of 0.6700 was found to be the best possible choice. It did not completely correct the effect but it minimized it.

7.2.4 Object - Camera Distance

It was noticed that if the object is positioned too far from the camera, it would appear so small that its distinctive shape features are overlooked and if positioned too close, the shape is distorted due to a problem common to all photographic lenses termed aberration. Trials were run to find a range of optimal distances; this assumption is valid considering that most image recognition systems fix the object to camera distance.

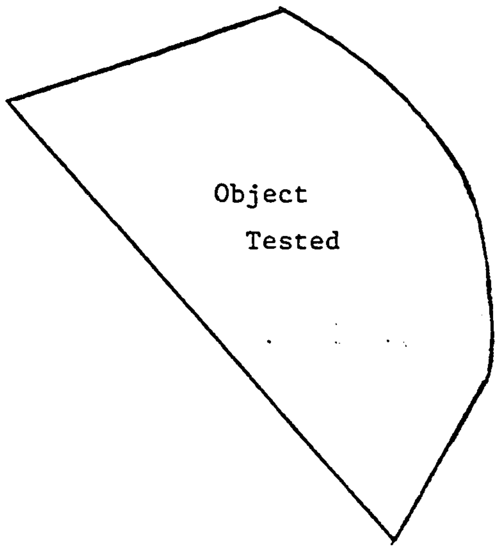
Besides varying the focusind distance, the shape of the object itself was changed; the object of Fig 7.1 was replicated in 3 sizes : large, medium and small. Tables 7.5, 7.6 and 7.7 shows some of the results of the 3 objects at different focuses. The actual corresponding distances were:

distance 1	:	46 cm .	distance 2	:	40 cm .
distance 3	:	35 cm .	distance 4	:	32 cm .
distance 5	:	27 cm .	distance 6	:	23 cm .
distance 7	:	17 cm .	distance 8	:	14 cm .

The results confirm that very small objects at far distances give the worst results; the best results are for large objects at almost any distance or for medium objects at medium distances. In general a distance range of 25 cm to 40 cm was suitable. Naturally these values will change for different lenses, i.e. if very small components are to be recognized, a different kind of lens like a close-up or macro lens will yield better results.

7.2.5 Gradient Tolerance (Epsilon)

The value of ε used to detect the corner points of the Coarse and Detailed Models were chosen as 3 and 2 (Section 5.2.5); the value was reached after different trials. Some of the results are shown in Fig 7.2; the resulting models are shown as graphs for better grasping. A value greater than 3 is not suitable as it is not accurate enough while a value of 1 results in too many redundant segments.



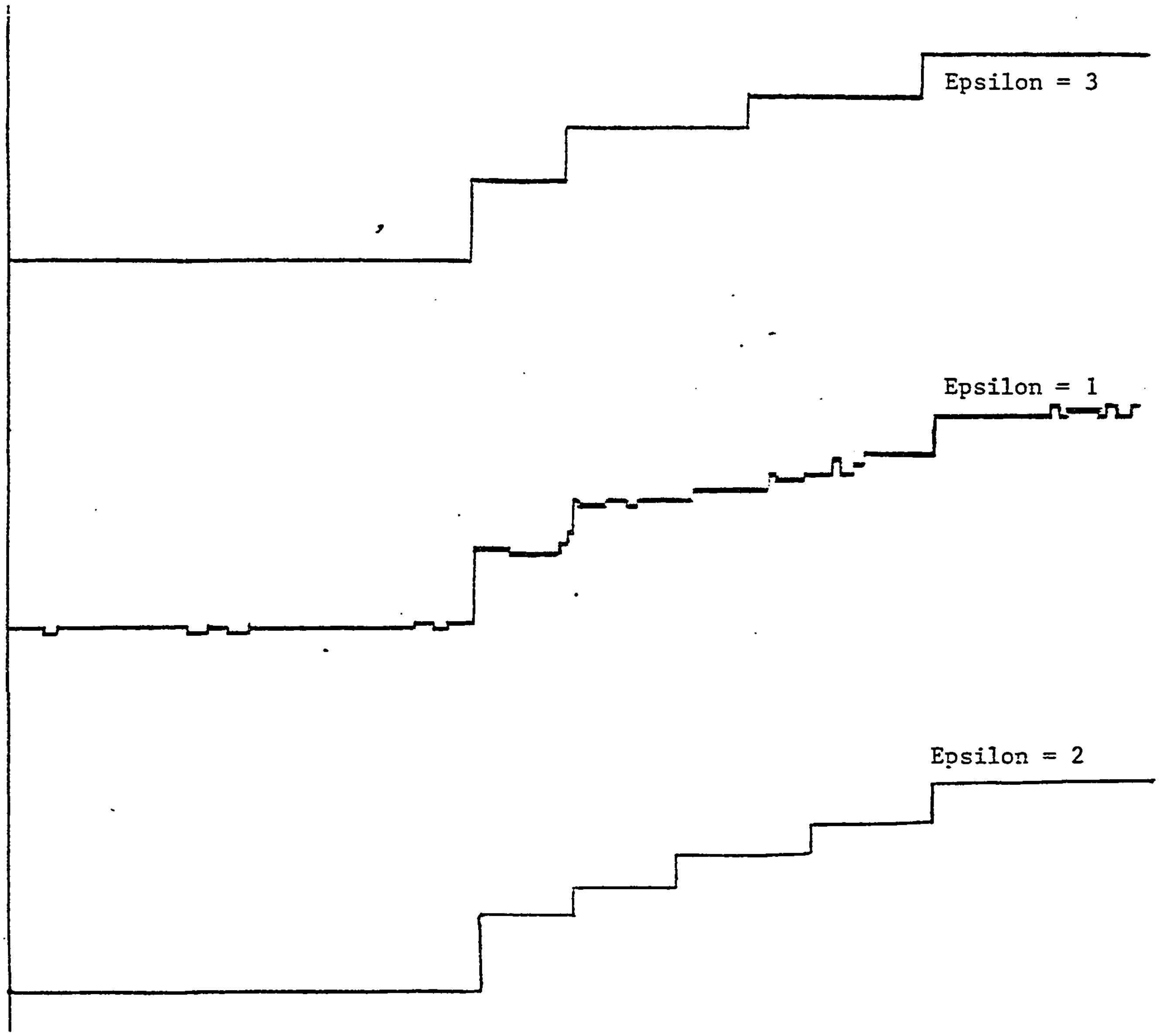


Fig 7.2 : Different Gradient Tolerances

7.3 Learning Stage Results

The Learning and Recognition algorithms were tested for their performance; the objects tested for are illustrated in Fig 7.3 . Twenty objects were selected; the number was considered enough in comparison to other researches done in literature where the test was carried out on as few as 4 objects or in Juvin's case on 10 objects.

The Descriptor arrays for all the 20 objects are listed in Table 7.8 ; it can be noticed that a wide variety of objects with varying number of segments, angles and complexity. The objects are not of industrial type in nature since the system is just a pilot one.

The classes, when entered in the DB, were named as Classa, Classb, etc. but for this paper A, B, etc. will be used.

The Learning stage provides :

- segmenting the contour of an object in a quick manner.
- normalizing to get totally invariant representation.
- previewing Descriptor of object before its insertion.
- graphical view of both models and their graphs.
- viewing the current DB.
- adding object to existing class.
- adding object to a new class
- adding upto 3 reference signatures in DB.

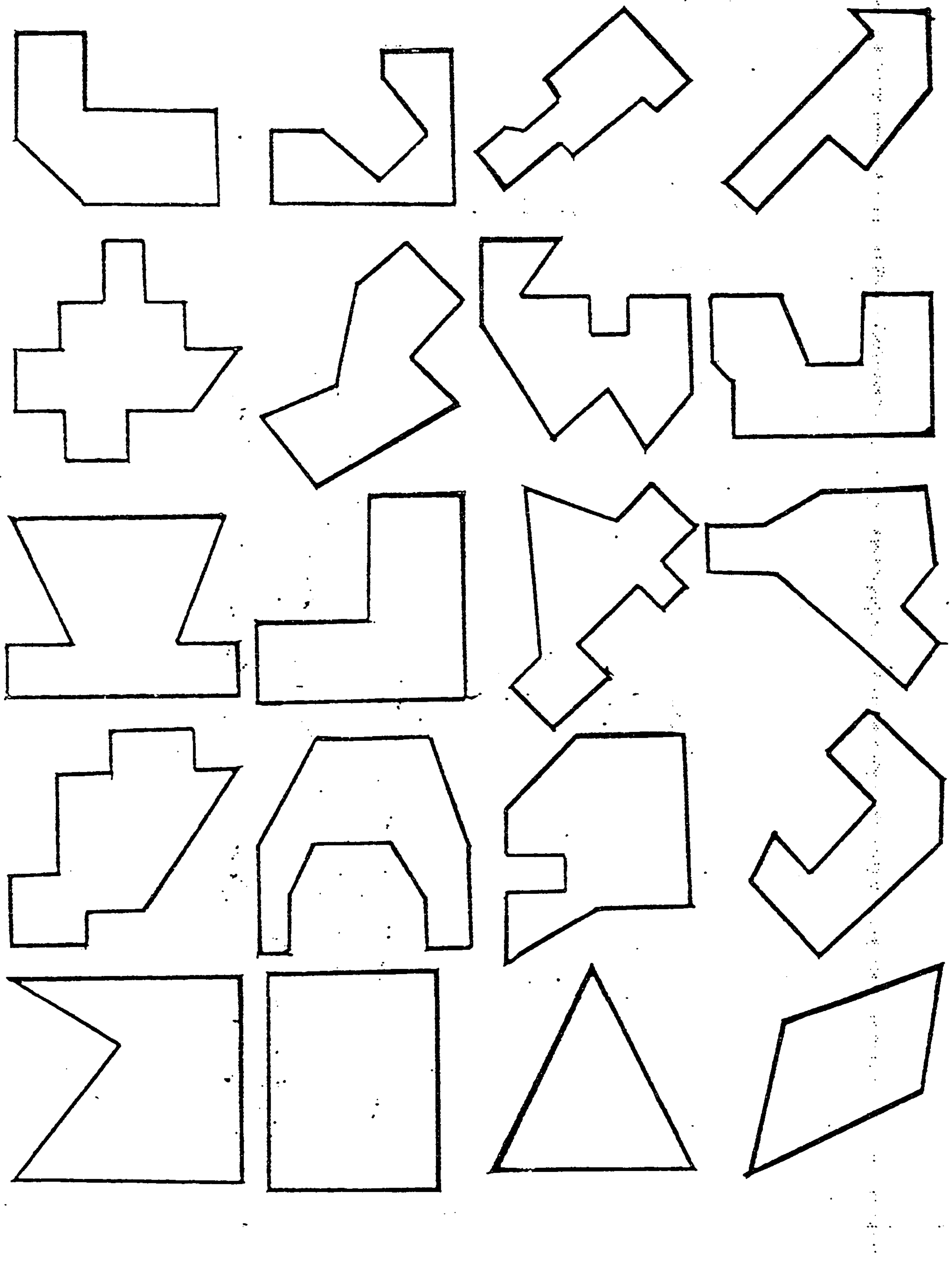


Fig 7.3 : Objects tested on.

7.3.1 Time to Obtain Polar Models

The Learning stage first reads the FMC of the contour and obtains the normalized models and their reference signatures. The time taken by each step is listed in Table 7.9 ; it shows the length of each process as actually carried out on each of the 20 objects.

The first column, **Segment** , lists the time taken by the segmentation algorithm; it totals the time to obtain the 2 Polar models (ρ_i , θ_i) and the time for non square pixel correction. The time listed is in 1/100 th of a second. An average time of 68/100 th second was a good result which can be easily tripled by a faster machine.

The second column, **Longest Sides** , refers to the time taken to obtain the longest side of the contour; at times 2 or 3 nearly longest sides are found and the time listed includes the finding of all 3. The average time was a negligible 0.5/100 th second.

The third column, **Reference Signature** , tabulates the moment taken to reorder the Polar models, starting from the longest segment, to obtain the reference signature which is needed for the last step of Recognition. Again the average time was indeed small, at 0.8/100 th second.

The last column, **Normalize** , indicates the total time to normalize the models to a constant perimeter of 2000. The average time is around 0.8/100 th of a second.

7.3.2 Time to Update to DB

Once the normalized reference signatures of a new object is obtained, getting its Descriptor array and inserting it in the DB is done. The timing statistics for the involved steps are listed in Table 7.10 .

It should be noted that all the times related to the DataBase operations are influenced heavily by the size of the RAM of the PC. The DB obviously cannot keep all the master and index files in Direct Access Memory if the latter is limited. The IBM-AT , on which these timimg statistics were obtained, had the following limitations :

* the clock speed of INTEL 80286 is 6 MHz .

* the RAM available was 512 Kbytes.

The clock speed of 6 MHz irrespectively slows down all the operations be it the program execution or DB search even if the latter is residing in RAM.

The 512 Kbyte RAM available slows down the Learning and Recognition in a different way. The RAM will be occupied by the DOS, the compiled programs executing and the remaining space will be filled by the DB files; not all of them can be loaded so only a limited pages are read from the mass storage (hard disk). If a search or update to a DB file is executing and the related data is in RAM, so much the better since the access time is a mere 150 nanoseconds; the problem arises if the record needed is not in the RAM but on the hard disk which has a relatively longer access time of at least 12 milliseconds. The PAGE containing the record is located, paged in while another page of the RAM not being used is paged out back to the hard disk. All this contribute to a larger delay.

The timing statistics in the following tables show that at times, the same opertions take a 1/100 th second while at other times they take 10/100 th second; this is not random but due to the fact that sometimes the record is already in RAM and at other times has to be paged in.

An obvious solution is to use an INTEL 80386 or higher which are available commonly with at least 2 Mbyte of RAM. The efficiency of the Recognition will greatly improve as the DB operations will only take place from the Direct Access Memory and the clock speed of 20 MHz or more accelerates the execution of the program itself.

The first column, **Descriptor** , shows the time to extract the features of the Descriptor array of each class; the average time is 1.7/100 th second.

Column 2, **Assign to GENREC** , counts the time taken to update all the features of an object to the record GENREC, which will be inserted in the master file of the DB. The average time was less than 1/100 th of a second.

Column 3, **Add Record** , times how long it takes to really insert the GENREC record in the master file and the time taken to update the 8 related index files. The average time is 6.8/100 th second;

The fourth column, **Add object** , times how long it takes to add an object to a class and update the related index file and it averages 2.5/100 th second; note that most of the times in that column are in fact less than 1/100 th second but for some cases they are 6/100 or 22/100; the reason is paging in of DB records.

The last column, **Add Detailed** , refers to the insertion of the reference signature (up tp 3) of the Detailed model; this takes more time than the others because a lot of records have to be inserted and indexed. Specially here, a faster PC with more RAM, will greatly help. Note the fluctuation of the times between 11/100 to 61/100 th second.

7.4 Recognition

The Recognition algorithm consists of finding all the neighboring classes and applying the minimun distance test. The advantages of the Recognition are :

- the representation stored in the DB is invariant under rotation, translation or scaling.

- first hierarchy model kept, the Coarse or the overall features of the class used in accelearating the matching process by clustering all classes with feature near those of the unknown object.

- second hierarchy is the detailed model, kept for exact match between unknown object and class.

- a DataBase is used to accelerate all searches and make the storage of the models more efficient. Indexed files are used.

- the first level of search obtains the cluster of classes and their frequency of match. Indexed files efficiently allows direct identification of classes matching a feature.

- a combined algorithm is used which locates all classes satisfying a feature and obtains their union.

- a weight is assigned to each feature depending on how important a shape identifier it is.

- for each feature searched for, an upper and lower bound of error is considered to make the recognition more robust.

- the weights assigned to each feature and the bounds of search can be easily modified by the programmer to tailor the system to an application with particular dictionary of objects.

- if the programmer wishes to use different set of Descriptor array, i.e. some other features not included which might be more important in shape demarkation, all he has to do is calculate them in the procedure Getfields and use them instead; the variable names of the attributes are not important, they are just dummy names and can represent any new feature.

- a minimum distance test can be applied with any reference signature.
- the minimum distance test does not calculate the difference in area with a fixed iteration but calculates the area difference over whole segment lengths.

7.4.1 Recognition Algorithm

The Recognition algorithm was tested for all the objects of Fig 7.1; the testing was carried for all possible combinations of matches.

The weights assigned for each feature of the neighborhood search were found out by trial and error (Appendix C22); some features were judged more important than others on a rule of thumb basis.

Procedure **Testsearch** was tested by taking one object of each class and finding the cluster satisfied for each feature; the resulting cluster after the seasrch for each feature was examined and checked if the required classes were located; the results were very good as shown in Table 7.11 .

The first column lists the time taken to get the resulting cluster of classes satisfying all the decisions and obtaining their frequency of occurence; the average was 0/36 th second.

The second column times the sorting of the total cluster such that the candidate class with highest frequency is first; the time was about 0/00 th second.

The last column shows the resulting sorted cluster after the testsearch procedure is executed; the results proved excellent since

- for 19 of the 20 cases, the class with highest probability of match was actually the correct class of the unknown object.
- the difference in frequency count of the first and second class of the sorted

cluster is always at least 2, i.e. for 19 cases the algorithm never returned two classes with equal probability of being the class looked for, hence no ambiguity in the recognition.

- only 1 of the 20 cases tested resulted in a class with highest frequency not actually belonging to the class of the unknown object, i.e. ClassH ; even then the required class was indeed found but not on the first but third hit and its frequency was only 2 lower than the first hit class.

7.4.2 Minimum Distance Test

Once the sorted cluster of all candidate classes are found, one of them is the identified class, usually the first one. To be 100 per cent certain that it is actually the correct class, the minimum distance test is applied to find the correlation (difference) in areas of the graphs of unknown object and class.
Testing was carried out by finding this difference in area for each unknown object to each of the 61 reference signatures in the DB. Part of these results are shown in Table 7.12 and they are very encouraging. The first column shows the area difference between an object of ClassA to reference signatures for ClassA, 1ClassA, ClassB . etc . (note 1 class may have more than 1 signature).

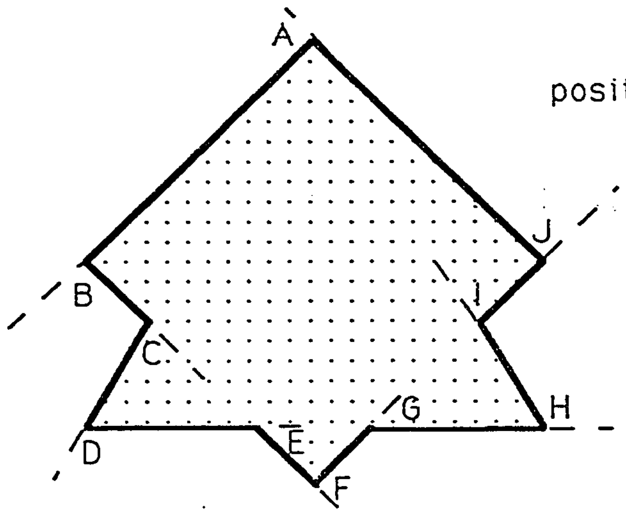
A total of 252 comparisons were carried out with the conclusion that if an object and its correct class were correlated by their graphs, the distance is always < 20,000 , 50 % of these resulted in a distance of < 10,000.
However if an unknown object was matched to a different class than its own, the distance is always > 40,000.
Therefore the minimum distance test is simple to formulate :

If dis ($\{P_{d\text{-unknown}}\}$, $\{P_{d\text{-DBclass}}\}$) $\leq$ 20,000 $\rightarrow$ unknown object belongs DBclass .

The timings for the minimum distance test were also measured and given in Table 7.13 .

Column 1 shows the time required to obtain the Detailed model representation from the DB; this step which takes an average of 0/13 th second, can be decreases as pointed out by increasing the RAM size.

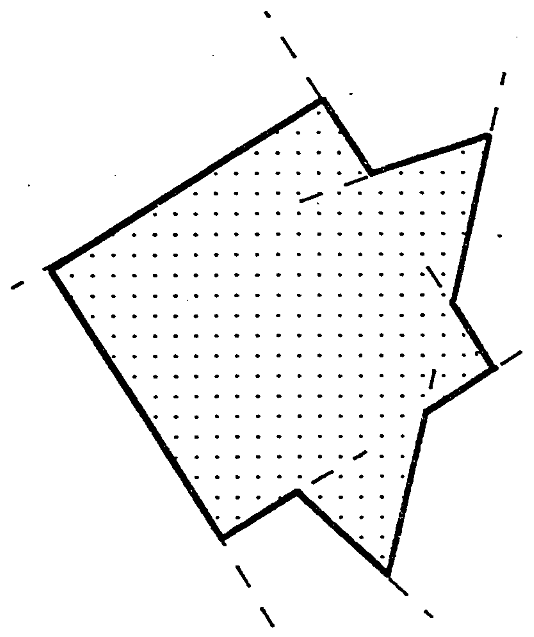
Column 2 lists the time to find the distance between 2 graphs and it averages to nearly 0/0 th second.



Orientation 1

Length	Angle	Deviation
147.2	91.2	1.2
38.8	86.5	3.5
39.6	-72.4	2.6
58.1	117.7	2.3
54.4	-43.0	2.0
58.4	88.2	1.8
54.0	-41.9	3.1
39.7	117.9	2.1
39.3	-70.0	5.0
147.2	85.8	4.2

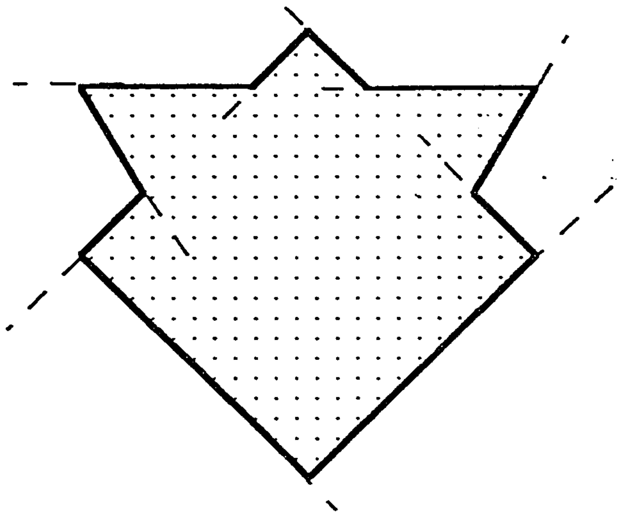
Table 7.1 : Varying the orientation of part only



Orientation 2

Length	Angle	Deviation
146.1	87.2	2.8
147.4	90.4	0.4
37.9	87.0	3.0
39.7	-71.4	3.6
59.1	115.4	0.4
52.6	-41.3	3.7
59.6	86.9	3 1
53.1	-42.3	2.7
41.1	119.6	0.4
40.1	-71.6	3.4

Table 7.2 : Varying the orientation of part only



Orientation 3

Length	Angle	Deviation
56.0	88.1	1.9
56.0	-44.3	0.7
42.1	118.2	1.8
38.5	-71.4	3.6
146.8	88.0	2.0
146.3	90.7	0.7
38.7	86.2	3.8
40.2	-70.2	4.8
56.0	117.8	2.2
56.7	-43.1	1.9

Table 7.3 : Varying the orientation of part only

ALL ANGLES IN DEGREES

Actual	**90**	**90**	**72**	**120**	**45**	**90**	**42**	**120**	**78**	**90**
S=0.6400	89	91	73	116	43	82	37	115	71	89
S=0.6500	89	89	71	116	43	82	37	115	69	87
S=0.6600	91	88	70	115	43	84	40	116	69	86
S=0.6700	92	88	70	115	42	85	40	115	68	86
S=0.6800	92	87	68	114	42	85	40	115	68	85
S=0.6900	93	86	68	113	41	87	43	116	67	85
S=0.6470	90	90	72	116	44	82	38	116	70	88
S=0.6480	90	90	72	115	43	84	40	116	70	88
S=0.6490	90	90	72	115	43	84	40	116	69	86
S=0.6475	90	90	72	116	44	*5	40	116	70	88

Table 7.4 : Varying the Scale Factor

OPBJECT SIZE : LARGE
camera-object distance : d2, d3, d4
distance2 distance3
distance4

Leng	Angle	Dev	Leng	Angle	Dev	Leng	Angle	Dev
116.6	89.5	1.5	126.3	89.5	1.5	141.0	89.5	1.5
29.1	89.6	1.4	32.7	88.8	1.2	35.3	87.4	2.6
33.2	-77.1	2.1	35.4	-73.3	1.7	39.2	-73.5	1.5
48.0	119.3	0.7	52.0	117.3	2.7	56.0	119.4	0.6
43.3	-40.8	4.2	44.6	-42.7	2.3	54.1	-42.6	2.4
43.4	86.8	3.2	50.3	85.2	4.8	54.4	86.6	3.4
47.1	-41.3	3.7	48.0	-41.0	4.0	56.0	-42.7	2.3
32.7	116.2	3.8	35.4	120.6	0.6	40.9	117.7	2.3
30.2	-70.7	4.3	34.2	-72.9	2.1	36.7	-72.8	2.2
117.2	88.6	1.4	128.5	88.7	1.3	143.4	91.0	1.0

Table 7.5 : Varying object size and camera distance.

OPBJECT SIZE : MEDIUM
camera-object distance : d4, d5, d6

distance4			distance5			distance6		
Leng	Angle	Dev	Leng	Angle	Dev	Leng	Angle	Dev
111.1	88.4	1.6	121.2	88.0	2.0	132.5	88.4	1.6
30.7	86.7	3.3	33.5	88.3	1.7	36.8	87.2	2.8
33.2	-71.5	3.5	39.7	-73.0	2.0	40.3	-70.5	4.5
51.0	118.6	1.4	56.0	120.6	0.6	60.0	117.9	2.1
36.1	-39.4	5.6	40.9	-45.4	0.4	42.7	-43.6	1.4
35.8	85.0	5.0	41.7	88.7	1.3	47.1	85.6	4.4
48.0	-43.4	1.6	54.0	-41.9	3.1	56.0	-40.9	4.1
35.4	118.7	1.3	37.7	116.8	3.2	42.8	115.5	4.5
30.0	-73.2	1.8	32.5	-69.4	5.6	32.7	-69.5	5.5
110.8	90.0	0.0	122.2	87.3	2.2	134.0	89.8	1.2

Table 7.6 : Varying object size and camera distance.

OPBJECT SIZE : SMALL
camera-object distance : d5, d6, d7

distance5			distance6			distance7		
Leng	Angle	Dev	Leng	Angle	Dev	Leng	Angle	Dev
98,9	89.9	1.1	115.5	89.9	0.1	129.2	90.0	0.0
23.5	84.6	5.4	28.3	90.8	0.8	30.9	88.2	1.8
23.9	-69.7	5.3	28.0	-70.3	4.7	31.6	-71.1	3.9
36.1	117.0	3.0	36.0	113.4	6.6	44.1	114.9	5.1
39.0	-41.1	3.9	49.6	-39.7	5.3	53.0	-40.8	4.2
45.0	84.6	5.4	48.7	82.4	7.6	57.8	83.0	7.0
32.0	-40.4	4.6	37.0	-38.6	6.4	42.0	-37.1	7.9
25.2	115.9	4.1	28.1	111.0	9.0	31.7	114.4	5.6
24.9	-69.0	6.0	26.8	-62.4	12.5	31.4	-68.2	6.8
98.9	88.2	1.8	111.9	83.5	6.5	128.2	86.6	3.4

Table 7.7 : Varying object size and camera distance.

Class	#of sid	max sid	big sid	max pos ang	max neg ang	#of lrg ang	#of neg ang	#of sid dtl	num obj	sub cls
A	10	390	2	120	-74	6	4	11	1	1
B	8	461	2	131	-89	4	2	8	1	0
C	13	307	1	95	-89	6	4	16	1	1
D	9	370	2	95	-91	5	2	12	1	1
E	7	300	1	93	-92	4	1	8	1	2
F	10	424	2	91	-87	5	3	12	1	0
G	13	264	0	102	-94	7	4	18	1	1
H	8	425	1	107	-86	5	2	9	1	0
I	16	196	0	128	-91	8	6	18	1	2
J	10	356	1	98	-82	4	3	14	1	0
K	13	293	0	117	-90	7	5	14	1	0
L	6	447	2	91	-87	5	1	8	1	1
M	8	475	2	114	-14	6	2	11	1	0
N	9	383	1	110	-88	4	2	12	1	0
O	10	345	2	122	-89	5	3	12	1	1
P	9	369	2	105	-72	3	3	13	1	1
Q	12	357	1	132	-87	7	4	14	1	0
S	3	677	3	137	0	3	0	4	1	1
T	4	591	4	91	0	4	0	4	1	1
U	5	464	3	136	-86	4	1	6	1	1

Table 7.8 : Objects tested

Class	Segment	Longest	Reference	Normalize
A	0/71	0/0	0/0	0/0
B	0/60	0/0	0/0	0/0
C	0/93	0/0	0/0	0/5
D	0/72	0/0	0/5	0/0
E	0/55	0/0	0/0	0/0
F	0/77	0/0	0/0	0/0
G	0/99	0/6	0/0	0/5
H	0/60	0/5	0/0	0/0
I	1/04	0/0	0/6	0/0
J	0/82	0/0	0/0	0/0
K	0/94	0/0	0/0	0/6
L	0/50	0/0	0/5	0/0
M	0/71	0/0	0/0	0/0
N	0/77	0/0	0/0	0/6
O	0/71	0/0	0/0	0/0
P	0/83	0/0	0/0	0/5
Q	0/82	0/0	0/0	0/0
S	0/83	0/0	0/0	0/0
T	0/33	0/0	0/0	0/6
U	0/44	0/0	0/0	0/5

Table 7.9 : Statistics for Obtaining Polar Models

Class Descrip	Get	Assign Genrec	Add Class	Add Object	Add Detailed
A	0/0	0/0	0/5	0/0	0/27
B	0/0	0/0	0/5	0/0	0/11
C	0/6	0/0	0/5	0/0	0/33
D	0/6	0/0	0/5	0/0	0/28
E	0/0	0/0	0/6	0/6	0/11
F	0/6	0/0	0/6	0/0	0/22
G	0/6	0/0	0/6	0/0	0/38
H	0/0	0/0	0/15	0/0	0/11
I	0/0	0/0	0/11	0/0	0/61
J	0/0	0/0	0/5	0/6	0/33
K	0/0	0/0	0/11	0/6	0/27
L	0/0	0/0	0/6	0/5	0/22
M	0/0	0/0	0/6	0/0	0/22
N	0/0	0/0	0/5	0/0	0/33
O	0/6	0/0	0/5	0/0	0/28
P	0/0	0/0	0/11	0/22	0/44
Q	0/5	0/0	0/6	0/0	0/60
S	0/0	0/0	0/11	0/0	0/11
T	0/0	0/0	0/5	0/6	0/11
U	0/0	0/0	0/6	0/0	0/16

Table 7.10 : DataBase Operations

Object from class	Time_to_get all cluster	Sorting cluster	Resulting merged cluster
A	0/27	0/0	(A,9),(O,6),..
B	0/49	0/0	(B,11),(M,9),..
C	0/33	0/0	(C,11),(G,8),..
D	0/55	0/0	(D,11),(N,8),..
E	0/33	0/0	(E,10),(L,7),..
F	0/55	0/6	(F,11),(J,10),(O,8),..
G	0/33	0/0	(G,11),(C,10),(K,6),..
H	0/44	0/0	(N,11),(M,10),(H,9),..
I	0/16	0/0	(I,11),(O,2),..
J	0/39	0/0	(J,11),(O,8),..
K	0/22	0/0	(K,11),(G,6),..
L	0/38	0/0	(L,11),(E,9),..
M	0/38	0/0	(M,11),(E,8),..
N	0/44	0/0	(N,11),(H,9),..
O	0/39	0/0	(O,10),(J,9),(F,7),..
P	0/38	0/6	(P,11),(J,8),..
Q	0/33	0/0	(Q,11),(C,8),..
S	0/28	0/0	(S,11),(T,7),..
T	0/27	0/0	(T,11),(S,5),..
U	0/37	0/0	(U,11),(L,8),..

Table 7.11 : Results of Clustering Algorithm

Reference Signature	Object_of ClassA	Object_of ClassB	Object_of ClassC	Object_of ClassD
ClassA	5613			
1ClassA	72469			
ClassB	77343	11064		
ClassC	58685	98852	9782	
1ClassC	74696	125274	84479	
1ClassD	80691	104720	90747	14225
ClassD	65917	123572	70576	109546
ClassE	49753	83380	66884	50920
1ClassE	51877	105622	50430	85054
2ClassE	59741	100422	46436	105664
2ClassF	54356	101308	74173	81427
-	-	-	-	-
-	-	-	-	-

Table 7.12 : Minimum Distance Test

Object_from_class	Get_D_ Detailed_Rep.	Get_area
A	0/11	0/0
B	0/16	0/0
C	0/17	0/0
D	0/22	0/0
E	0/16	0/0
F	0/11	0/0
G	0/16	0/0
H	0/22	0/0
I	0/16	0/5
J	0/22	0/0
K	0/5	0/0
L	0/16	0/0
M	0/0	0/0
N	0/11	0/0
S	0/5	0/0
T	0/5	0/0
U	0/6	0/0

Table 7.13 : DataBase Operations

CHAPTER 8

CONCLUSION AND FUTURE WORK

The problem of recognizing an object's digital image from a dictionary of prestored images is slow. In this work, it was shown how a multi - level hierarchy accelerates the search. The Freeman Chain Code of an object's contour was extracted and translated to a normalized Polar model which is invariant under translation, rotation or scaling. Two Polar models were extracted, namely the Coarse model from which the Descriptors needed to minimize the search were obtained, and the Detailed model which stores the precise silhouette of the object used in an exact minimum - distance match. The two models were stored in a Database; In the Learning stage, the user is provided with numerous facilities to view the image, its silhouette, its segmented contour(s), its Descriptors, its Polar graph and is allowed saving or loading utilities. The Recognition Stage which also interfaces with the Database is an interactive system allowing the operator to match an unknown object to a class of objects. A summary of the average time to complete crucial operations is listed in Table 8.1 . The total average Recognition time is around 120 hundredth of a second or 1.2 second.

As future work, it would be recommended that the system's performance be improved. The Recognition time can be easily decreased by :

- using a PC with faster CPU clock (instead of 6 MHz)
- using lower resolution images, i.e. 256 by 256 pixels.
- using a PC with more RAM, instead of 512 Kb, this allows the whole Database to be memory resident (instead of being paged in from secondary storage); all the DB operations will be fastened tremendously.
- parallelizing or pipelining some of the operations, many of which are particularly suited and are usually implemented in parallel in the industry :
  * Image Thresholding and Contour Extraction can be easily modified to operate in a pipeline fashion.
  * Extracting the Detailed and Coarse Models can be done in parallel without any extra communication cost.
  * Recognition stage is particularly suited for parallelism where the Clustering and Area - match can be carried out on separate classes in parallel.

Time to	**Sec**	**100th_sec**
Generate overall and detailed Polar Models from FMC	0	68
Find the longest side(s) for Reference Signature	0	0.5
Reorder Polar Model to obtain Reference Signature	0	0.8
Normalize Polar Models	0	0.9
Get Descriptor Array	0	1.7
Assign Descriptors to DB	0	0
Update GEN file & all Indexes	0	6.8
Add object to Class/Create new Class	0	2.5
Add Detailed Polar Model to DB	0	27.5
Find Cluster of Classes	0	36
Sort Cluster on Frequency	0	0
Calculate Distance between two Detailed Polar Models	0	0
Obtain Detailed Model DB (DET Relation)	0	12.7
Average Total Recognition Time	**1**	**20**

Table 8.1 : Timing Statistics Summary

REFERENCES

AND86 Andrew R. and Mahdi S., "On the rotational invariance of Fourrier descriptors of discrete images", Identification and Pattern Recognition, Tome II, 18-20 June 1986.

BHA82 Bhannu B. and Faugeras O., "Shape matching of 2-D objects using a hierarchical stochastic labelling technique", Conference on Pattern Recognition and Image Processing, IEEE 1982.

CHE82 Cheng J. and Huang T., "Recognition of curvilinear objects by matching relational structures", Conference on Pattern Recognition and Image Processing, IEEE 1982.

COR83 Corby N., "Machine vision for robots", IEEE Transactions on Industrial Electronics, Vol 1E-30, no 3, August 1983.

DAT87 "Database Toolbox, Turbo Pascal ver 4.0", Borland International, 1987.

DUD73 Duda R. and Hart P., "Pattern Classification and Scene Analysis", John Wiley & Sons, New York, 1973.

FG187 "FG-100-AT : user's manual", Imaging Technology Incorporated, 1987.

GUE89 Guerra C. and Levialdi S., "Computer Vision : Algorithms & Architectures", Advances in Machine Vision, Springer-Verlag, New York Inc, 1989.

GON83 Gonzalez R. and Safabakhsh R., "Computer Vision Techniques for industrial inspection and robot control : a tutorial overview", IEEE, 1983.

GON87 Gonzalez R. and Wintz P., "Digital Image Processing", Addison-Wesley, Reading, Massachusetts, 1987.

HAR83 Harrington S., "Computer Graphics : a programming approach", McGraw-Hill Inc, Tokyo, 1983.

HOL79 Holland S., Rossol L. and Ward M.,"Consight-I : a vision controlled robot system for transferring parts from belt conveyors", Computer Vision and Sensor based robots, Plenum, 1979.

HEN89 Hendengren K., "Methodology for Automatic Image-based inspection of industrial objects", Advances in Machine Vision, Springer-Verlag, New York Inc, 1989.

HOL88 Holzner S., "Advanced Assembly Language on the IBM-PC", Prentice Hall Press, 1988.

HUN84 Hung S. and Kasavand T., "On the chord property and its equivalences", Pattern Recognition, 7th Conference, vol 1, Aug 1984.

JUV82 Juvin D. and Dupeyrat B., "ANIMA (ANalyses of IMAges) : a quasi real time system", Conference on Pattern Recognition and Image Processing, IEEE 1982.

JUV86 Juvin D. and Dupeyrat B., "An heuristic method of classification and automatic inspection of parts for ANIMA", Identification & Pattern Recognition, Tome II, 18-20 June 1986.

KAS84 Kasvand T. and Otsu N., "Recognition of line shapes based on thinning, segmentation with good connectivity algorithms and regularization", Pattern Recognition, 7th Conference, vol 1, Aug 1984.

KLA89 Klafter R., Chmielewski T. and Negin M., "Robot Engineering", Prentice Hall Inc., Englewood, 1989.

MAR82 Martin H., Goesta G. and Hans K., "Consistency operation for line and curve enhancement", Pattern Recognition and Image Processing, IEEE 1982.

PAS89 "Turbo Pascal ver 5.0 : User's Guide and Programmers' Reference Guide", Borland International,1989.

PAV77 Pavlidis T., "Structural Pattern Recognition", Springer Publishing, New York, 1977.

PER86 Person E., "Hierarchical Correlation for fast industrial object location", Pattern Recognition in Practice II, 1986.

ROS84 Rosenfeld A., "Image Analysis : Problems, Progress and Prospects", Pattern Recognition, vol 17, no 1, 1984.

RUM84 Rummel P. and Beutel W., "Workpiece Recognition and Inspection by a model-based scene analysis system", Pattern Recognition, vol 17, no 1, 1984.

SAF89 Safari B., "Putting DSPs to work" Byte Magazine, December 1989.

SCO82 Scott B., Parag P. and George L., "Computer Recognition of overlapping parts using a single camera", Pattern Recognition and Image Processing, IEEE 1982.

STO84 Stockman G.and Morawski P., "2-D object aquisition using circular scanning", Pattern Recognition, vol 17, no 1, 1984.

SUE86 Suetens P. and Oosterlinck A., "Industrial Pattern Recognition", Pattern Recognition in Practice II, 1986.

TIO82 Tio J., McPherson A. and Hall E., "Curved Surface measurement for robot vision", Pattern Recognition and Image Processing, IEEE 1982.

TOU74 Tou T. and Gonzalez R., "Pattern Recognition Principles", Addison-Wesley, Reading, Massachusetts, 1974.

TSU82 Tsung-Wei S. and Yee-Hong Y., "Goal Directed Segmentation", Pattern Recognition and Image Processing, IEEE 1982.

WAL81 Wallace T., Mitchell O. and Keinosuke F., "3-Dimensional shape analysis using local shape descriptors", IEEE on PAMI-3, no 3, May 1981.

WAT84 Watson L., Arvind K., Erich R. and Haralick R., "Extraction of lines and regions from grey-tone line drawing images", Pattern Recognition, vol 17, no 5, 1984.

WIE86 Wiecek B., "Binary pattern recognition and comparison", Identification and Pattern Recognition, Tome II, 18-20 June 1986.

WUR84 Wu R. and Stark H., "Rotation-invariant pattern recognition using optimum feature extraction", Pattern Recognition in Practice II, 1986.

```
:----------------------------------set fg100 regs FOR access-----
SET_REG         PROC    near
         :
         mov dx,0300h
         mov ax,4040h                 :memory control
         out dx,ax
         mov dx,0302h
         mov ax,0000h                 :host mask
         out dx,ax
         mov dx,0304h
         mov ax,07ffh                 :video mask
         out dx,ax
         mov dx,0308h
         mov ax,0000h                 :x pointer
         out dx,ax
         mov dx,030ah
         mov ax,0000h                 :y pointer
         out dx,ax
         mov dx,030ch
         mov ax,0044h                 :pointer control
         out dx,ax
         mov dx,030eh
         mov ax,0407h                 :cpu address control
         out dx,ax
         mov dx,0310h
         mov ax,0010h                 :x spin constant
         out dx,ax
         mov dx,0312h
         mov ax,0000h                 :y spin constant
         out dx,ax
         mov dx,0316h
         mov ax,3000h                 :lut control
         out dx,ax
         mov dx,031ch
         mov ax,0008h                 :zoom control
         out dx,ax
         ret
SET_REG         endp
```

Appendix A1 : Setting up fg-100 registers.

```
:--------------------------------CLEAR image display-----------
CLEAR            PROC    far
         mov dx,0300h
         mov ax,5050h              :
         out dx,ax
         mov dx,0306h
         mov ax,0000h              :
         out dx,ax
         mov dx,0300h
         mov ax,1010h              :
         out dx,ax
         mov dx,031ah
         mov ax,0d940h              :
         out dx,ax
    lopl:
         mov cx,0ffffh
    watt: xor bx,bx
          loop watt
         mov dx,031ah               :
         in  ax,dx                  :
         and ah,0f0h                :
         cmp ah,0c0h                :
         jne lopl                   :
         ret
CLEAR            endp
:--------------------------------SNAP new image----------------
SNAP             PROC       far
         mov dx,031ah
         mov ax,0f940h              :status/control SNAP
         out dx,ax
           mov cx,0ffffh
   delay:  add ax,bx
           loop delay
         mov dx,031ah
         mov ax,0e940h              :status/control SNAP
         out dx,ax
         ret
SNAP             endp
:--------------------------------GRAB continuously-------------
GRAB             PROC  far
         mov dx,031ah
         mov ax,0f940h              :status/control SNAP
         out dx,ax
         ret
GRAB.            endp
```

Appendix A2 : a-Clear , b-Snap , c-Grab operations

```
:---------------------------------convert TO NEGATive image------
NEGAT          PROC  far
         call SET_REG
         xor  bx,bx
nlop3:
         mov   ax,bx                 :
         mov   dx,030ah              : set frame ypointer
         out   dx,ax
         xor   di,di                 : di - offset within strip

 nlop2:                              :
         push  ds                    :
         mov   dx,0a000h             :
         mov   ds,dx                 :
         mov   al,{di}              :read pixel value TO al
         pop ds
                                     :
            not al
            inc al
            add al,0ffh

         push  ds                    :
         mov   dx,0a000h             :
         mov   ds,dx                 :
         mov   dl,al
         mov   {di},dl               :
         pop ds
 np2:                                : get 0 or 1 in packed ah
         inc   di                    : go TO next next byte of
         inc   di                    : frame strip
                                     :
         cmp   di,0000h              :end of frame strip?
         jnz   nlop2                 :IF no REPEAT packing
         add   bx,0040h
                                     : IF yes  go TO next frame
         cmp   bx,02c0h              : strip only IF remaining
         jb    nlop3                 : frame strip left
         call reSET_REG
         call res_cut
         ret
NEGAT          endp
```

Appendix A3 : Converting the image TO its NEGATive

```
bin      PROC  far              : convert TO binary display
         call set_cut
         call SET_REG
         xor  bx,bx
         call READ_HEX
         mov cl,dl
 lop3:   mov   ax,bx              :
         mov   dx,030ah           : set frame ypointer
         out   dx,ax
         xor   di,di              : di - offset within strip
  lop2:  push  ds                 :
         mov   dx,0a000h          :
         mov   ds,dx              :
         mov   al,{di}           :read pixel value TO al
         pop ds
                                  :
         cmp   al,cl              :
         jb    pl
                                  : TO 0 IF white
         push  ds                 :
         mov   dx,0a000h          :
         mov   ds,dx              :
         mov   dl,0ffh
         mov   {di},dl            :
         pop ds
         jmp p2

  pl:    push  ds                 :
         mov   dx,0a000h          :
         mov   ds,dx              :
         mov   dl,00h
         mov   {di},dl           :read pixel value TO al
         pop ds
                                  :
  p2:    inc   di                 : go TO next next byte of
         inc   di                 : frame strip
                                  :
         cmp   di,0000h           :end of frame strip?
         jnz   lop2               :IF no REPEAT packing
         add   bx,0040h
                                  : IF yes  go TO next frame
         cmp   bx,02c0h           : strip only IF remaining
         jb    lop3               : frame strip left
         call reSET_REG
         call res_cut
         ret
bin            endp
```

Appendix A4 : Converting a gray level image TO binary

```
:---------------------------------read hexadecimal byte---------
READ_HEX  PROC near
         :
         push bx
         push ax
         call ONE_HEX
         mov dl,al
         mov cl,04h
         shl dl,cl
         call ONE_HEX
         add dl,al
         mov ah,02h
         int 21h
         pop   ax
         pop   bx
         ret
READ_HEX endp
:---------------------------------used by READ_HEX--------------
ONE_HEX  PROC near                :reads one hex digit
         :
         push dx
         mov ah,08h
     y1: int 21h
         cmp al,30h
         jb y1
         cmp al,46h
         ja y1
         cmp al,39h
         ja  y2
         mov ah,02h
         mov dl,al
         int 21h
         sub al,30h
         pop dx
         ret
     y2: cmp al,41h
         jb  y1
         mov ah,02h
         mov dl,al
         int 21h
         sub al,37h
         pop dx
         ret
 ONE_HEX endp
```

Appendix A5 : Reading a hexamdecimal digit from keyboard

```
pack      PROC  near       : pack frame image TO binary image in sector
          lea   si,sector
          xor   bx,bx              : bx - offset within sector
                                   :
 loop3:   mov   ax,bx
          mov   cl,06h             :divide by 64 TO get ypointer
          shr   ax,cl              :for frame strip
          mov   dx,030ah           : set frame ypointer
          out   dx,ax
                                   :
          xor   di,di              : di - offset within strip
                                   :
  loop2:  mov   ch,00h             :counter - pack 8 pixels
          xor   ah,ah              : ah - keep packed pixels
          mov   cl,01h             :shift count is 1
                                   :
  loop1:  push  ds                 :
          mov   dx,0a000h          :
          mov   ds,dx              :
          mov   al,{di}           :read pixel value TO al
          pop ds
                                   :
          cmp   al,cutoff          :cut-off value ?
          jb    pp1
          clc                      : TO 0 IF white
          jmp   short pp2
  pp1:     stc                     : TO 1 IF black
                                   :
  pp2:     rcl   ah,cl             : get 0 or 1 in packed ah
          inc   di                 : go TO next next byte of
          inc   di                 : frame strip
                                   :
          inc   ch                 :count 8 times ?
                                   :
          cmp   ch,08h             :IF less than 8
          jl    loop1              : go back TO pack more
                                   :
          mov   {si+bx},ah      :store packed ah in store+offset
          inc   bx
                                   :next location in sector
                                   :
          cmp   di,0000h           :end of frame strip?
          jnz   loop2              :IF no REPEAT packing
                                   : IF yes  go TO next frame
          cmp   bx,8000h           : strip only IF remaining
          jb    loop3              : frame strip left
                                   :
          call blankcol
          ret
 pack           endp
```

Appendix A6 : Converting a gray image TO binary and transfer TO 'sector' area: each 8 pixels TO 1 byte.

```
:-----------------------------blank out column 1 and 2 of sector
:                              from far - e.g. pascal program
PBLANKCOL        PROC   far
          push ax
          push bx
          push cx
          push dx
          push bp
          push si
          push di
          push ds
          push es
     :
                                      : blank out columnn 1 since fg100
          lea si,sector               : sets them TO 1 FOR some reason
          xor bx,bx
          mov dl,00h
 blankl : mov   {si+bx},dl
          inc   bx
          mov   {si+bx},dl
          add   bx,003fh
          cmp   bx,8000h
          jb    blankl
                                   :
    :
          pop es
          pop ds
          pop di
          pop si
          pop bp
          pop dx
          pop cx
          pop bx
          pop ax
         ret
 PBLANKCOL        endp
```

Appendix A7 : Blank out column 1 of frame memory

```
UNPACK   PROC  near    :     UNPACK binary image in sector TO frame
         lea   si,sector
         xor   bx,bx                 : bx - offset within sector
 r3:     mov   ax,bx
         mov   cl,06h                :divide by 64 TO get ypointer
         shr   ax,cl                 :for frame strip
         mov   dx,030ah              : set frame ypointer
         out   dx,ax
                                     :
         xor   di,di                 : di - offset within strip
 r2:                                  :
         mov   al,{si+bx}
         xor   ch,ch                 : ah - keep packed pixels
         mov   cl,01h                :shift count is 1

  r1:    shl   al,cl
         jb    s1
               push  ds                     :
               mov   dx,0a000h              :
               mov   ds,dx                  :
               mov   dl,0ffh
               mov   {di},dl                 :read pixel value TO al
               pop ds
               jmp short s2
                                      :
  s1:                                  : TO 1 IF black
               push  ds                     :
               mov   dx,0a000h              :
               mov   ds,dx                  :
               mov   dl,00h
               mov   {di},dl                :read pixel value TO al
               pop ds
                                      :
  s2:                                  : get 0 or 1 in packed ah
          inc   di                    : go TO next next byte of
          inc   di                    : frame strip
                                      :
          inc   ch                    :count 8 times ?
                                      :
          cmp   ch,08h                :IF less than 8
          jl    r1                     : go back TO pack more
                                      :
          inc   bx
          cmp   di,0000h              :end of frame strip?
          jnz   r2                     :IF no REPEAT packing
                                      : IF yes  go TO next frame
          cmp   bx,8000h              : strip only IF remaining
          jb    r3                     : frame strip left
          ret
 UNPACK         endp
```

Appendix A8 : Unpacks 'sector' area TO frame memory

```
;---------------------------reads binary file start and get address
READSTART  PROC far
         :
         push dx
         push cx
         call READ_HEX
         mov  file,dl
         dec  dl
         xor  dh,dh
         mov  cl,06h
         shl  dx,cl
         xor  bx,bx
         mov  bx,000ch
         add  dx,bx
         mov  secstart,dx
         pop cx
         pop dx
         ret
READSTART endp
;---------------------------reads fmc -  file start and get address
readfmc  PROC far
         :
         push dx
         push cx
         push bx
         call READ_HEX
         mov  file,dl
          dec  dl
          xor  dh,dh
          mov  cl,04h
          shl  dx,cl
          xor  bx,bx
          mov  bx,020ch
          add  dx,bx
          mov  secstart,dx
         pop bx
         pop cx
         pop dx
         ret
readfmc endp
```

Appendix A9 : Read file number and find starting sector

```
:---------------------------load binary image from sector TO diskette
:  input : secstart should define file address TO write TO
SEC2DISK        PROC  far
          push ax
          push bx
          push cx
          push dx
          push bp
          push si
          push di
          push ds
          push es
            mov   al,00h              :drive a
            mov   cx,0040h            :count of 64 sectors
            mov   dx,secstart         :start read from secstart
            lea   bx,sector           :load from sector table
            int   26h                 :write  TO diskette
            popf
                        :" pop all registers which were pushed"
          pop. es
           .  .
          pop ax
          ret
SEC2DISK        endp
:---------------------------load binary file from diskette TO sector
:  input : secstart should define file address TO read from
DISK2SEC        PROC  far
          push ax
          push bx
          push cx
          push dx
          push bp
          push si
          push di
          push ds
          push es
            mov   al,00h              :drive a
            mov   cx,0040h            :count of 64 sectors
            mov   dx,secstart         :start read from secstart
            lea   bx,sector           :load from sector table
            int   25h                 :read  TO diskette
            popf
                        " pop all registers which were pushed"
          pop es
           .  .
          pop ax
          ret
DISK2SEC        endp
```

Appendix A10 : Saving / loading binary image

```
:----------------------------save fmc image from fmcsec TO diskette
:  input : secstart should define file address TO write TO
FMC2DISK        PROC  far
          push ax
          push bx
          push cx
          push dx
          push bp
          push si
          push di
          push ds
          push es
            mov   al,00h              :drive a
            mov   cx,0010h            :count of 16 sectors
            mov   dx,secstart         :start read from secstart
            lea   bx,fmcout           :load from sector table
            int   26h                 :write  TO diskette
            popf
                       " pop all registers which were pushed"
          pop es
            .  .
          pop ax
         ret
FMC2DISK        endp
:----------------------------load fmc file from diskette TO fmcsec
:  input : secstart should define file address TO read from
DISK2FMC        PROC  far
          push ax
          push bx
          push cx
          push dx
          push bp
          push si
          push di
          push ds
          push es
            mov   al,00h              :drive a
            mov   cx,0010h            :count of 16 sectors
            mov   dx,secstart         :start read from secstart
            lea   bx,fmcout           :load from sector table
            int   25h                 :read  TO diskette
            popf
                       " pop all registers which were pushed"
          pop es
            .  .
          pop ax
         ret
DISK2FMC        endp
```

Appendix A11 : Saving / loading of fmcout (contour)

```
:-----------------------direction 4- keep outputing fmc=1  unless
dir5     PROC  near
dir5rep:
               inc cx
               push si
               push bx
                mov si,di
                mov bx,cx
                mov {si+bx},ah
               pop bx
               pop si
         call  endpix        :IF current=start quit process (ah=00)
         jb    dir5end
         :
         call go4
         call getbit
         jnb  dir512
             call go8
             call getbit
             jb   dir5111
                   call go6
                   mov  ah,04h
                   jmp  dir5end
             dir5111:   mov  ah,03h
                        jmp  dir5end
 dir512 : call go6
          call getbit
          jnb  dir5121
               mov ah,05h
               jmp dir5rep
  dir5121: call go0
           call getbit
           jnb  dir5122
                mov ah,06h
                jmp dir5end
   dir5122: call go0
            call getbit
            jnb  dir5123
                 mov ah,07h
                 jmp dir5end
     dir5123: call go2
              call getbit
              jnb  dir5124
                   mov ah,00h
                   jmp dir5end
       dir5124: call go2
                mov ah,01h
 dir5END: ret
 dir5     endp
```

Appendix A12 : DIR5 routine

```
;-----------------------go up routine -
go2       PROC  near
          sub   bx,0040h
          ret
go2       endp
;-----------------------go down routine -
go6       PROC  near
          add   bx,0040h
          ret
go6       endp
;-----------------------go left routine -
go4       PROC  near
          dec   al
          cmp   al,00h
          ja    go4r
           mov  al,08h
           dec  bx
  go4r:   ret
go4       endp
;-----------------------go right routine -
go0       PROC  near
          inc   al
          cmp   al,08h
          jbe   go0r
           mov  al,01h
           inc  bx
  go0r:   ret
go0       endp
```

Appendix A13 : Accessing individual pixels in 'sector' in horizontal and vertical directions

```
;----------------------go north-east routine -
go1      PROC  near
         sub   bx,0040h
         inc   al
         cmp   al,08h
         jbe   go1r
          mov  al,01h
          inc  bx
  go1r:  ret
go1      endp
;----------------------go south-east routine -
go7      PROC  near
         add   bx,0040h
         inc   al
         cmp   al,08h
         jbe   go7r
          mov  al,01h
          inc  bx
  go7r:  ret
go7      endp
;----------------------go north-west routine -
go3      PROC  near
         sub   bx,0040h
         dec   al
         cmp   al,00h
         ja    go3r
          mov  al,08h
          dec  bx
  go3r:  ret
go3      endp
;----------------------go south-west routine -
go5      PROC  near
         add   bx,0040h
         dec   al
         cmp   al,00h
         ja    go5r
          mov  al,08h
          dec  bx
  go5r:  ret
go5      endp
```

Appendix A14 : Accessing individual pixels in the 4 diagonal directions.

```
:----------------------save row & bit# of starting pixel on contout
startpix PROC  near
        :mov   row,bx
         mov   dx,bx
         mov   bit,al
         ret
startpix endp
:----------------------check IF current pixel is equal TO start
:                                     carry =  0     IF not equal
:                                           =  1     IF equal
endpix   PROC  near
        :cmp   row,bx
         cmp   dx,bx
         jne   endp1
         cmp   bit,al
         jne   endp1
         mov   ah,00h
         stc
         jmp  endp2
endp1:   clc
endp2:                                :  testing
         ret
endpix   endp
:----------------------get bit corresponding TO current pixel in carry
getbit   PROC  near
         push  ax
         push  bx
         push  cx
         push  si
         :
         mov   cl,al
         mov   ah,{si+bx}
         shl   ah,cl
                 :
         pop   si
         pop   cx
         pop   bx
         pop   ax
         ret
getbit   endp
```

Appendix A15 : Startpix, Endpix & Getbit routines

```
:----------------------convert contour of screen image TO fmc & save
scrfmc   PROC  far
    :
         push ax
         push bx
         push cx
         push dx
         push si
         push di
         push ds
                                  :get first black bit of image
          lea si,sector           :  si: start address of sector
          lea di,fmcout           :  di: start address of fmcout
          xor bx,bx               : pointer within sector ( row )
          mov cx,00004h           : pointer TO output area in fmcout
          mov al,00h              : bit #
          :
          call getbit
          jnb  sc1
                   :print out some error message - wrong file loaded
                   jmp scrfmcend
    sc1:
          call getbit             :get first black pixel
          jb   sc2                :found
          call go0                :not found , move TO next pixel
          cmp  bx,7fffh           :stop IF past end-of-image file
          jb   sc1
          :
                     :print some message 'image file is blank ..sorry
                     jmp scrfmcend     :return
```

Appendix A16 : Locating start pixel

```
sc2:
        call startpix           :save co-ordinates of start pixel
                                :and locate next pixel
                        :test
                        push bx
                        push dx
                         mov dl,bh
                         call write_hex
                         mov dl,bl
                         call write_hex
                         call READ_HEX
                         call line
                        pop dx
                        pop bx
                        :
        call go5
        call getbit
        jnb  sc3
          :---------------output fmc=5  TO area fmcout
          mov  ah,05h
          jmp scrloop
   sc3: call go0
        call getbit
        jnb  sc4
          :-----------------output fmc=6
          mov ah,06h
          jmp scrloop
   sc4: call go0
        call getbit
        jnb  sc5
          :------------------output fmc=7
          mov ah,07h
          jmp scrloop
   sc5: call go2
        call getbit
        jnb  sc6
          :------------------output fmc=0
          mov ah,00h
          jmp scrloop
        :
   sc6: :isoalted pixel , ignore and move TO next one
           jmp sc1
              :
```

Appendix A17 : Obtaining 2nd pixel w.r.t. 1st pixel & getting initial direction

```
 scrloop :

            cmp ah,09h          :  end of contour
            je  scrfinish
             cmp  ah,00h
                 jne  scr2
                  call dir0
                  jmp  scrloop
    scr2:    cmp  ah,01h
                 jne  scr3
                  call dir1
                  jmp  scrloop
    scr3:    cmp  ah,02h
                 jne  scr4
                  call dir2
                  jmp  scrloop
    scr4:    cmp  ah,03h
                 jne  scr5
                  call dir3

                  jmp  scrloop
    scr5:    cmp  ah,04h
                 jne  scr6
                  call dir4
                  jmp  scrloop
    scr6:    cmp  ah,05h
                 jne  scr7
                  call dir5
                  jmp  scrloop
    scr7:    cmp  ah,06h
                 jne  scr8
                  call dir6
                  jmp  scrloop
    scr8:    cmp  ah,07h
                 jne  scr9
                  call dir7
                  jmp  scrloop
    scr9:    jmp  scrfinish
:
scrfinish:
```

Appendix A18 : Contour tracking algorithm.

```
scrfinish:
                                     :
            sub cx,0004h
            push si
              mov si,di
              xor bx,bx
              mov {si+bx},cx              :length of fmc points saved
              :
              inc bx
              inc bx
              mov {si+bx},dx              :save row of start pixel
              :
              inc bx
              inc bx
              mov dl,bit
              mov {si+bx},dl              :save bit # of start pixel
            pop si
            :
scrfmcEND:
          pop ds
          pop di
          pop si
          pop dx
          pop cx
          pop bx
          pop ax
           ret
scrfmc   endp
```

Appendix A19 : Saving perimeter, coordinates of 1st pixel.

```
:----------------------------show contour of object using fmc
contour         PROC  far
          push ax               : push all the registers
           .   .
          push es
                                : blank out sector
          lea si,sector
          xor bx,bx
          mov dl,0ffh
   contl: mov  {si+bx},dl
          inc  bx
          cmp  bx,8000h
          jb   contl
                                 :
              lea si,fmcout
              xor bx,bx
              mov cx,{si+bx}              :get # of fmc points
              add cx,0004h
              :
              inc bx
              inc bx
              mov dx,{si+bx}              :get  row of start pixel
              :
              inc bx
              inc bx
              mov al,{si+bx}              :get  bit # of start pixel
            :
          lea  si,sector
          lea  di,fmcout
          mov  bx,dx

 cont2:
        push       si
        push       bx
          mov si,di
          mov bx,cx
          mov ah,{si+bx}
        pop bx
        pop si
```

Appendix A20 - Displaying the contour on tv

```
            cmp  ah,01h
                 jne  scc2
                  call go5
                  jmp  sccloop
    scc2:   cmp  ah,02h
                 jne  scc3
                  call go6
                  jmp  sccloop
    scc3:   cmp  ah,03h
                 jne  scc4
                  call go7
                  jmp  sccloop
    scc4:   cmp  ah,04h
                 jne  scc5
                  call go8
                  jmp  sccloop
    scc5:   cmp  ah,05h
                 jne  scc6
                  call go1
                  jmp  sccloop
    scc6:   cmp  ah,06h
                 jne  scc7
                  call go2
                  jmp  sccloop
    scc7:   cmp  ah,07h
                 jne  scc8
                  call go3
                  jmp  sccloop
    scc8:   cmp  ah,08h
                 jne  scc9
                  call go4
                  jmp  sccloop
    scc9:   jmp  countend
 sccloop :                              : set pixel in sector TO '1'
               push cx
                  mov cl,al
                  mov ah,{si+bx}
                  rcl ah,cl
                  clc
                  rcr ah,cl
                  mov {si+bx},ah
               pop cx
          dec cx
          cmp cx,0004h
          ja cont2
               call UNPACK
countEND:      pop es              : pop all registers
                .  .
               pop ax
          ret
contour        endp
```

Appendix A20 (continued)

```
{-------------------------------------get file TO extract contour---}
 filnam := '              ' :  chl:=' ': contstatus := 0:
 gotoxy(5,3):write('diskette file drive a : proceed {y/n} : '):
 textcolor(5):gotoxy(65,3):readln(chl):gotoxy(65,3):write(chl):
 IF chl <> 'y' THEN exit:textcolor(3):
 gotoxy(5,5):write('enter file name + 1 blank : '):textcolor(5):
 gotoxy(33,5):readln(filnam):gotoxy(33,5):write(filnam):
 filnam := 'c:\limafmc\'+copy(filnam,1,pos(' ',filnam)-1 )+'.fmc' :
 assign(fbl,filnam): reset(fbl,4): blockread(fbl,bufinf,1,writ):
 close(fbl):
 assign(fbl,filnam): reset(fbl,1000): blockread(fbl,fmc,1,writ):
 close(fbl):

step   := 4:
fmccnt := bufinf{1}:                    { # of pixels on contour }
dircnt := fmccnt div step :

{---------------------------------------relative fmc----------------}
   fmcn{6} := fmc{6} :
   k:= 7 :
   REPEAT
      diff  := fmc{k} - fmc{k-1} :
      IF ( diff < 5 ) and ( diff > -5 ) THEN    {boundary not crossed}
         fmcn{k} := fmcn{k-1} + diff
      ELSE
         BEGIN
           IF ( diff >= 5 ) THEN                {boundary change +   }
              fmcn{k} := fmcn{k-1} + diff - 8
           ELSE                                 {boundary change -   }
              fmcn{k} := fmcn{k-1} + diff + 8 :
         END:
      inc(k) :
   until k > (fmccnt+5 ) :
```

Appendix B1 : Read fmc file and convert TO rfmc

```
{-------------------------------sum of 4-pixel groups & breakpoints  }
   posln := 3 :     ptrdetail{2} := 2 :     ptrdetail1{2} := 2 :
   vertx{2} := 0 : verty{2} := 0 : vertptr := 3 :
   vertx1{2} := 0 : verty1{2} := 0 : vertptr1 := 3 :
   j  :=  6      :   dirptr := 1 :   xx := 0 : yy:= 0 :
   sumcomp :=   fmcn{j+1}+fmcn{j+2}+fmcn{j+3}+fmcn{j+4}  +
                fmcn{j+5}+fmcn{j+6}+fmcn{j+7}+fmcn{j+8}  +
                fmcn{j+9}+fmcn{j+10}+fmcn{j+11}+fmcn{j+12}  +
                fmcn{j+13}+fmcn{j+14}+fmcn{j+15}+fmcn{j+16} :
   sumcomp := round( sumcomp / 4.0  ):
   sumcomp1 := sumcomp :
REPEAT
   sum := 0:  xxt{0} := xx :     yyt{0} := yy :
   FOR  k:= 0 TO step-1 do
        BEGIN
           sum := sum + fmcn{j+k} :
                                              { update xx , yy coordinates }
                    CASE fmcn{j+k} of
                        0,1,7,8,9,-1                  :   inc(xx) :
                        2,6,10,14,-2                  :   xx := xx  :
                        3,4,5,11,12,13,-3,-4,-5       :   dec(xx) :
                    END:
                    CASE fmcn{j+k} of
                        0,4,-4,8,12                   :   yy := yy  :
                        1,2,3,-5,-6,9,10,11           :   inc(yy) :
                        5,6,7,-1,-2,-3,13,14          :   dec(yy) :
                    END:
              xxt{k+1} :=  xx : yyt{k+1} := yy :
      END:

                                          { overlook sharp break in corner }

         IF  abs(sum-sumcomp) >= 3 THEN
         BEGIN

           foundbreak := 0 :
           FOR k:= 0 TO (step-1) do
               BEGIN
                  IF ( abs( (fmcn{j+k}*4) - sumcomp )  > 4  )
                      and ( foundbreak = 0 )     THEN BEGIN

                      j := j + k   :
                      vertx{vertptr} := xxt{k} :
                      verty{vertptr} := yyt{k} :
                      vertx1{vertptr1} := xxt{k} :
                      verty1{vertptr1} := yyt{k} :
                      xx := xxt{k} : yy := yyt{k} :
                      foundbreak := 1 :
                   END:
               END:
```

Appendix B2 : Find corner points FOR 2 levels

```
        IF foundbreak = 0 THEN BEGIN
                j := j + 2    :
                vertx{vertptr} := xxt{2} :
                verty{vertptr} := yyt{2} :
                vertx1{vertptr1} := xxt{2} :
                verty1{vertptr1} := yyt{2} :
                xx := xxt{2} : yy := yyt{2} :
         END:

                ptrdetail{vertptr} := vertptr1 :
                ptrdetail1{vertptr1} := vertptr :

       sumcomp :=fmcn{j+0}+fmcn{j+1}+fmcn{j+2}+fmcn{j+3} +
                  fmcn{j+4}+fmcn{j+5}+fmcn{j+6}+fmcn{j+7}+
                  fmcn{j+8}+fmcn{j+9}+fmcn{j+10}+fmcn{j+11}+
                  fmcn{j+12}+fmcn{j+13}+fmcn{j+14}+fmcn{j+15} :

       sumcomp := round(sumcomp / 4.0)   :
       sumcomp1 := sumcomp :

       inc(vertptr) : inc(vertptr1) :

   end { IF diff >= 3 }
   ELSE
   IF  abs(sum-sumcomp1) >= 2 THEN
   BEGIN

                j1 := j + 2 :
                vertx1{vertptr1} := xxt{2} :
                verty1{vertptr1} := yyt{2} :
                ptrdetail1{vertptr1} := 0 :
                inc(vertptr1):

      sumcomp1 :=fmcn{j1+0}+fmcn{j1+1}+fmcn{j1+2}+fmcn{j1+3} +
               fmcn{j1+4}+fmcn{j1+5}+fmcn{j1+6}+fmcn{j1+7}+
               fmcn{j1+8}+fmcn{j1+9}+fmcn{j1+10}+fmcn{j1+11}+
               fmcn{j1+12}+fmcn{j1+13}+fmcn{j1+14}+fmcn{j1+15}:

       sumcomp1 := round(sumcomp1 / 4.0)   :

   END:    {diff >= 1 }

   IF  abs(sum-sumcomp) < 3  THEN inc(j,4) :


  inc(dirptr):
until  (  (j-3) >= fmccnt   ) :
```

Appendix B2 (continued)

```
{---------------------------------------------------show vertices---}
 yscale := 0.67  :
 FOR k:=3 TO (vertptr-1) do
  BEGIN
   verty{k} :=  verty{k}*yscale
  END:

 vertx{1} := vertx{vertptr-1} : verty{1} := verty{vertptr-1} :
 vertx{vertptr} := 0 : verty{vertptr} := 0 :

 FOR k:=2 TO (vertptr-1) do

  BEGIN                                     { length of segment     }
     delx{k} := vertx{k+1} - vertx{k} :
     dely{k} := verty{k+1} - verty{k} :
     leng{k} := ( sqrt( sqr(delx{k}) + sqr(dely{k}) ) ):
  END:
 delx{1}:=delx{vertptr-1}:dely{1}:=dely{vertptr-1}:
 leng{1}:=leng{vertptr-1}:
 delx{vertptr}:=delx{1}:dely{vertptr}:=dely{1}:
 leng{vertptr}:=leng{1}:
```

Appendix B3 : Correction FOR non square pixels

```
 posln := 7 :

for k:= 2 TO (vertptr-1) do
BEGIN
   x:= ( delx{k-1}*delx{k}+dely{k-1}*dely{k})/(leng{k-1}*leng{k}) :
{-----------------------------------------------get arccos x----}
    IF x > 1.00000 THEN
      therad := 0.00000
    ELSE
       IF x < -1.00000 THEN
          therad := 3.141592654
       ELSE
         IF x = 0.0 THEN
            therad := 1.570796327
         ELSE
            IF x > 0.0 THEN
                therad := arctan( sqrt(1.0-sqr(x)) / x )
            ELSE
                therad :=  pi +  arctan( sqrt(1.0-sqr(x)) / x )   :

   thedeg{k} := (therad*180/pi) :
{-------------------------------------------get direction of angle--}
   IF delx{k-1}  = 0 THEN
       mm := 9999
   ELSE
       mm := dely{k-1}/delx{k-1} :

   cc := verty{k} - (mm*vertx{k}) :
   fn := mm*vertx{k+1} + cc :
   IF verty{k+1} >  fn THEN
      sign := +1
   ELSE
      sign := -1 :


   IF ( vertx{k} < vertx{k-1} )
      or ( ( delx{k-1} = 0 ) and ( verty{k-1} > verty{k} ) ) THEN
       sign := sign * -1 :

   thedeg{k} := sign*thedeg{k} :

  END:
```

Appendix B4 : Find lengths & angles of segments

```
{---------------------------------normalize perimeter TO 2000---}
PROCEDURE NORMPERIM:
var      sm : integer :
BEGIN
   {************************************} GETTIME(hh,m7,s7,h7) :
   normscale := 2000.0 / fmccnt :   { normalize lengths 2000/perimeter
   normdiff := 20 * normscale :
   FOR sm := 2 TO vertptr - 1 do
          leng{sm} := leng{sm}*normscale :
   FOR sm := 2 TO vertptr1 - 1 do
          lengl{sm} := lengl{sm}*normscale :
  {************************************} GETTIME(hh,m8,s8,h8) :
END:
```

Appendix B5 : Mormalize perimeter

```
{-----------------------------determine fields of database----------}
PROCEDURE FINDFIELDS:
var f : byte :
BEGIN
  {************************************} GETTIME(hh,m3,s3,h3) :
  alargeside := 0 : asides := count1 :
  amaxanglep := 0: amaxanglen := 0: alargeanglep :=0: alargeanglen :=0
  FOR f := 1 TO count1 do BEGIN
   IF length1{f} < 30 THEN             { neglect very small lengths }
           dec(asides):
   IF length1{f} > 294 THEN            { count all sides > 150 }
           inc(alargeside):
   IF angle1{f} > amaxanglep THEN      { find maximum positive angle}
           amaxanglep := angle1{f}:
   IF angle1{f} < amaxanglen THEN      { find maximum NEGATive angle}
           amaxanglen := angle1{f}:
   IF angle1{f} > 80  THEN             { count angles more +80  deg }
           inc(alargeanglep):
   IF angle1{f} < -5   THEN            { count angles less -50  deg }
           inc(alargeanglen):
     {************************************} GETTIME(hh,m4,s4,h4) :
  END:
 END:
```

Appendix B6 : Ontaining Descriptor array

```
{-------------------------------------main---------------}
BEGIN
    clrscr:textcolor(5): textbackground(0):
    gotoxy(30,10): write(' opening database files ...'):
    openGENfiles:openobjfiles:opendetfiles:
    GENRECcount := filelen(GEN) - 1 :
    menuhl{1}:= 'a:file':     menuhl{2}:= 'normal':
    menuhl{3}:= 'graph ':     menuhl{4}:= 'print ':
    menuhl{5}:= 'prevue':     menuhl{6}:= 'update':
    menuhl{7}:= 'recogn':     menuhl{8}:= 'match ':
    menuhl{9}:= 'dbase ':
  REPEAT
    textbackground(0):textcolor(3):clrscr:
    horbar(9,1,1,80,1,menuhl,selbarhorl):
    GENRECcount := filelen(GEN) - 1 : GENdetcount:=filelen(det) - 1 :
    GENobjcount := filelen(obj) - 1 :
CASE selbarhorl of
 1 : BEGIN    FINDCONT :                  { obtain contour }
                        count1:=vertptr-2: count2:=vertptrl-2:
                        normdiff := 30 : SORTMAXES:
                        lgpos1 := largepos1:  lgpos2 := largeptr1:
                        DATAINT(lgpos1,lgpos2,
                                corx1,cory1,point1,angle1,length1,
                                corx2,cory2,point2,angle2,length2):
     END: { CASE 1 }
 2 : BEGIN            NORMPERIM:
                      SORTMAXES:
                      lgpos1 := largepos1:  lgpos2 := largeptr1:
                      DATAINT(lgpos1,lgpos2,
                              corx1,cory1,point1,angle1,length1,
                              corx2,cory2,point2,angle2,length2):
                      FINDFIELDS:
     END: { CASE 2 }
        3 : GRAPHUTIL1 :
        4 : PRINTSHOW:
        5 : BEGIN    PREVIEWFIELDS:   END:
 6 : BEGIN                                  { add main class TO db }
      gotoxy(1,5):write(' is data normalized ? {y/n} :          '):
      gotoxy(32,5):readln(opt): IF opt = 'n' THEN goto ml :
      window(1,2,80,25):clrscr:ADDCLASSTODB:
      window(1,2,80,25):clrscr:     add subclass 1  IF user desires }
      IF largestat2 = 1 THEN BEGIN
        gotoxy(1,5):write('add subclass,for 2nd largest side {y/n}:'):
        gotoxy(57,5):readln(opt): IF opt = 'n' THEN goto ml :
        lgpos1 := largepos2:  lgpos2 := largeptr2:
        DATAINT(lgpos1,lgpos2,corx1,cory1,point1,
                angle1,length1,corx2,cory2,point2,angle2,length2):
        addsubclass(1):
      END:
```

Appendix C1 : Main Learning & Rrecognition Module

```
      window(1,2,80,25):clrscr:     { add subclass 2  IF user desires}
      IF largestat3 = 1 THEN BEGIN
        gotoxy(1,5):write('add subclass,for 3rd largest side {y/n}:'):
        gotoxy(57,5):readln(opt): IF opt = 'n' THEN goto m1 :
        lgpos1 := largepos3:  lgpos2 := largeptr3:
        DATAINT(lgpos1,lgpos2,corx1,cory1,point1,
                angle1,length1,corx2,cory2,point2,angle2,length2):
        addsubclass(2):
      END:

END:   { sel = 6 }


       7 : BEGIN      gotoxy(1,3):write(' normalizing model ... '):
                      NORMPERIM:   SORTMAXES:
                      lgpos1 := largepos1:  lgpos2 := largeptr1:
                      DATAINT(lgpos1,lgpos2,
                              corx1,cory1,point1,angle1,length1,
                              corx2,cory2,point2,angle2,length2):
                      FINDFIELDS:
                      TESTSEARCH:
            END:


8 :BEGIN      gotoxy(1,3):write(' normalizing model ... '):
        NORMPERIM:   SORTMAXES:
        lgpos1 := largepos1:  lgpos2 := largeptr1:
        DATAINT(lgpos1,lgpos2,corx1,cory1,point1,angle1,length1,
                              corx2,cory2,point2,angle2,length2):
        FINDFIELDS:
        TESTSEARCH2class :

   END:
       9 : BEGIN   DISPLAYDB :   END:
   END: { CASE }
    window(1,2,80,25):textbackground(0):clrscr:textcolor(3):
m1: until selbarhor1 = 0 :
    gotoxy(20,10):write(' closing all database files ...'):
    closeGENfiles:closeobjfiles:closedetfiles:
    textcolor(3):textbackground(0):window(1,1,80,25):clrscr:
```

Appendix C1 (continued)

```
type
 codestr=string{10}:
 string2=string{2}:
{ FOR GENobj.DBF : info about main features of object              }
 GENREC = record
  GENstat      : longint :  {  used by system FOR deleting         }
  class        :  codestr:  {  name of object user defined         }
  sides        :  string2:  {  number of sides                     }
  maxside      :  string2:  {  longest side                        }
  largeside    :  string2:  {  number of sides > 400               }
  maxanglep    :  string2:  {  largest positive angle              }
  maxanglen    :  string2:  {  largest NEGATive angle              }
  largeanglep  :  string2:  {  number of angle > 80                }
  largeanglen  :  string2:  {  number of angle < -50               }
  sidesdet     :  string2:  {  number of sides FOR detailed object }
  numobjects   :  string2:  {  number of objects in the class      }
  subclass     :  string2:  {  same class , longest sides shifted  }
 END:
{ FOR detobj.DBF : detailed rho-theta info of object sides         }
 DETREC = record
  detstat      :  longint :
  class        :  codestr:  {  class  name   together with         }
  sideord      :  string2:  {  side number starting from longest   }
  length       :  string2:  {  length of side                      }
  angle        :  string2:  {  relative angle TO previous side     }
 END:

{ FOR clust.DBF  : heuristic search keeps count of near matches    }
 CLUSTREC = record
  cluststat    :  longint :
  class        :  codestr:  {  class  name                         }
  count        :  string2:  {  how many times object was found     }
 END:
{ FOR object.DBF : heuristic search keeps count of near matches    }
 OBJREC = record
  objstat      :  longint :
  class        :  codestr:  {  class name  together with           }
  objname      :  codestr:  {  object name  are indexed            }
  perimeter    :  string2:  {  absolute perimeter                  }
  distance     :  string2:  {  dist between object and camere      }
 END:

 maxdatatype=GENREC:
 maxkeytype=string{20}:
```

Appendix C2

```
uses dos,crt,taccess:
{$i db.typ}
var
 GEN , det , cls ,obj : datafile :
 clsl , detl , objl ,
 GEN1,GEN2,GEN3,GEN4,GEN5,GEN6,GEN7,GEN8 : indexfile:
 GENr : GENREC: detr : DETREC: clsr : CLUSTREC: objr : OBJREC :
 tempstr12 : string{12} :
 CLUSTRECnum , GENRECnum : longint :

function inttostr ( n : integer ) : string2 :
  BEGIN
    n := n + $8000:
    inttostr := chr(hi(n)) + chr(lo(n)) :
  END:

function strtoint ( s : string2 ) : integer :
  BEGIN
    strtoint := swap( ord(s{1}) ) + ord(s{2}) + $8000 :
  END:

PROCEDURE createGENfiles:
 BEGIN
   makefile ( GEN   ,'GENobj.DBF'      , sizeof(GENr) ):
   makeindex( GEN1  ,'classnam.NDX'    , 10         , 0 ):
   makeindex( GEN2  ,'sides.NDX'       , 2          , 1 ):
   makeindex( GEN3  ,'maxsid.NDX'      , 2          , 1 ):
   makeindex( GEN4  ,'largesid.NDX'    , 2          , 1 ):
   makeindex( GEN5  ,'maxangp.NDX'     , 2          , 1 ):
   makeindex( GEN6  ,'maxangn.NDX'     , 2          , 1 ):
   makeindex( GEN7  ,'largangp.NDX'    , 2          , 1 ):
   makeindex( GEN8  ,'largangn.NDX'    , 2          , 1 ):
 END:
PROCEDURE createdetfiles:
 BEGIN
   makefile ( det   ,'detobj.DBF'    , sizeof(detr) ):
   makeindex( detl  ,'classide.NDX'  , 12           , 0 ):
 END:
PROCEDURE createclustfiles:
 BEGIN
   makefile ( cls  ,'clust.DBF'     , sizeof(clsr) ):
   makeindex( clsl ,'clusnam.NDX'   , 12           , 0 ):
 END:
PROCEDURE createobjfiles:
 BEGIN
   makefile ( obj  ,'object.DBF'     , sizeof(objr) ):
   makeindex( objl ,'classobj.NDX'   , 20           , 0 ):
 END:
```

Appendix C3

```
PROCEDURE openGENfiles:  BEGIN
   openfile ( GEN   ,'GENobj.DBF'     , sizeof(GENr) ):
   openindex( GEN1  ,'classnam.NDX'   , 10         , 0 ):
   openindex( GEN2  ,'sides.NDX'      , 2          , 1 ):
   openindex( GEN3  ,'maxsid.NDX'     , 2          , 1 ):
   openindex( GEN4  ,'largesid.NDX'   , 2          , 1 ):
   openindex( GEN5  ,'maxangp.NDX'    , 2          , 1 ):
   openindex( GEN6  ,'maxangn.NDX'    , 2          , 1 ):
   openindex( GEN7  ,'largangp.NDX'   , 2          , 1 ):
   openindex( GEN8  ,'largangn.NDX'   , 2          , 1 ):
 END:
     PROCEDURE opendetfiles: BEGIN
        openfile ( det   ,'detobj.DBF'    , sizeof(detr) ):
        openindex( det1  ,'classide.NDX'  , 12          , 0 ):
      END:
    PROCEDURE openclustfiles: BEGIN
       openfile ( cls  ,'clust.DBF'     , sizeof(clsr) ):
       openindex( cls1 ,'clusnam.NDX'   , 12          , 0 ):
    END:
     PROCEDURE openobjfiles: BEGIN
        openfile ( obj  ,'object.DBF'     , sizeof(objr) ):
        openindex( obj1 ,'classobj.NDX'   , 20          , 0 ):
      END:
```

Appendix C4

```
PROCEDURE SORTMAXES:  BEGIN
   larger1 := leng{2} : largepos1 := 2 : largeptr1 := ptrdetail{2}:
   FOR sm :=3 TO vertptr - 1 do BEGIN { find longest side }
    IF leng{sm} > larger1 THEN BEGIN
       larger1 := leng{sm} :
       largeptr1 := ptrdetail{sm} :
       largepos1 := sm :
    END:  END:
   largestat2 := 0 : largestat3 := 0 :  {find 2nd 3rd largest sides }
   FOR sm :=2 TO vertptr - 1 do BEGIN
   largediff :=  larger1 - leng{sm}:
     IF ( sm <> largepos1 ) and ( largediff <= normdiff ) THEN BEGIN
       IF largestat2 = 0 THEN BEGIN larger2 := leng{sm} :
                                    largeptr2 := ptrdetail{sm}:
                                    largepos2 := sm :
                                    largestat2 := 1 :  end
       ELSE IF largestat3 = 0 THEN BEGIN larger3 := leng{sm} :
                                    largeptr3 := ptrdetail{sm}:
                                    largepos3 := sm :
                                    largestat3 := 1 :  end :
     END: end :
 END:
```

Appendix C5

```
{------------------------------convert all data TO integer---------}
                              { reorder sides starting from longest }
PROCEDURE  DATAINT(lgpos1,lgpos2:integer:var corx1,cory1,point1,angle1
                  length1,corx2,cory2,point2,angle2,length2:vert):
var m1,ind1:byte:
BEGIN
     IF lgpos1 < vertptr THEN BEGIN
      FOR m1 := lgpos1 TO vertptr - 1  do BEGIN
         ind1 := m1-lgpos1+1:
         corx1{ind1}    := round( vertx{m1} ) :
         cory1{ind1}    := round( verty{m1} ) :
         point1{ind1}   := ptrdetail{m1} :
         angle1{ind1}   := round(thedeg{m1}):
         length1{ind1}   := round(leng{m1}):
      END:
     END:
     IF lgpos1 > 2 THEN BEGIN
      FOR m1 := 2 TO  lgpos1 - 1  do BEGIN
         ind1:=vertptr-lgpos1+m1-1:
         corx1{ind1}    := round( vertx{m1} ) :
         cory1{ind1}    := round( verty{m1} ) :
         point1{ind1}   := ptrdetail{m1} :
         angle1{ind1}   := round(thedeg{m1}):
         length1{ind1}   := round(leng{m1}):
      END:
     END:
     {**************************************} GETTIME(hh,m5,s5,h5) :
     IF lgpos1 < vertptr THEN BEGIN
      FOR m1 := lgpos2 TO vertptr1 - 1  do BEGIN
         ind1 := m1-lgpos2+1:
         corx2{ind1}    := round( vertx1{m1} ) :
         cory2{ind1}    := round( verty1{m1} ) :
         point2{ind1}   := ptrdetail1{m1} :
         angle2{ind1}   := round(thedeg1{m1}):
         length2{ind1}   := round(leng1{m1}):
      END:
     END:
     IF lgpos2 > 2 THEN BEGIN
      FOR m1 := 2 TO  lgpos2 - 1  do BEGIN
         ind1:=vertptr1-lgpos2+m1-1:
         corx2{ind1}    := round( vertx1{m1} ) :
         cory2{ind1}    := round( verty1{m1} ) :
         point2{ind1}   := ptrdetail1{m1} :
         angle2{ind1}   := round(thedeg1{m1}):
         length2{ind1}   := round(leng1{m1}):
      END:
    END:
     {**************************************} GETTIME(hh,m6,s6,h6) :
END:
```

Appendix C6

```
{-------------------display fields which will be updated TO dbase--}
PROCEDURE PREVIEWFIELDS:
BEGIN
            textcolor(3):
            gotoxy(2,3):write('class name            : '):
            gotoxy(2,5):write('number of sides       : '):
            gotoxy(2,7):write('longest side          : '):
            gotoxy(2,9):write('# of sides > 300      : '):
            gotoxy(2,11):write('maxim pos. angle      : '):
            gotoxy(2,13):write('maxim neg. angle      : '):
            gotoxy(2,15):write('# of angles > 90      : '):
            gotoxy(2,17):write('# of NEGATive angles : '):
            gotoxy(2,19):write('detailed - # sides    : '):
            gotoxy(2,21):write('actual perimeter      : '):
            textcolor(2):
            gotoxy(27,3):write('noname'):
            gotoxy(27,5):write(asides:6):
            gotoxy(27,7):write(round(larger1):6):
            gotoxy(27,9):write(alargeside:6):
            gotoxy(27,11):write(amaxanglep:6):
            gotoxy(27,13):write(amaxanglen:6):
            gotoxy(27,15):write(alargeanglep:6):
            gotoxy(27,17):write(alargeanglen:6):
            gotoxy(27,19):write(count2:6):
            gotoxy(27,21):write(fmccnt:6):
 textcolor(1): FOR m1 := 3 TO 24 do BEGIN
                 gotoxy(38,m1):write('£'):END:
 textcolor(5):gotoxy(41,3):
 write(' -detailed-    side   length    angle'):
 textcolor(3): FOR m1 := 1 TO count2 do BEGIN
 gotoxy(50,m1+3):write(m1:9,length2{m1}:9,angle2{m1}:9):END:
 textcolor(4):gotoxy(50,25):write('press enter ..'):readln:
 END:
```

Appendix C7

```
{------------------------------------------graphics  utilities-----}
PROCEDURE GRAPHUTIL1:
var
   gd,gm : integer :   outs1,outs2,outs3,outs4 : string{3} :
   xcen,ycen : shortint :  xsc,ysc : real : menusel : byte :
BEGIN
   gd := detect : initgraph(gd,gm,''):
   IF graphresult <> grok THEN halt(1) :

   setgraphmode(0) :
                           { delete FOR cga mode }
   CLEARview(0,0,639,199) : setbkcolor(4) :
   men{1}:='object ?':
   men{2}:='main image':
   men{3}:='detailed':
   men{4}:='center':
   men{5}:='scale':
   men{6}:='CLEAR':
   men{7}:='rho theta':
   xsc := 1.5 : ysc := 0.7 : xcen := 0 : ycen := 0 :
   REPEAT
     CLEARview(500,0,639,199):
     str(xsc:3:1,outs1):str(ysc:3:1,outs2):
     str(xcen,outs3):str(ycen,outs4):
     outtextxy(500,100,'scale : '+outs1+'  '+outs2):
     outtextxy(500,110,'center: '+outs3+'    '+outs4):
     select(men,7,menusel):
   CASE menusel of
   1 : BEGIN                    end :
   2 : BEGIN CONTDRAW(corx1,cory1,count1,xcen,ycen,xsc,ysc): end :
   3 : BEGIN CONTDRAW(corx2,cory2,count2,xcen,ycen,xsc,ysc): end :
   4 : BEGIN   flashmsg(0,0,'enter new center'):
               read(xcen,ycen):CLEARview(0,0,135,30):
               end :
   5 : BEGIN   flashmsg(0,0,'enter new scale'):
               read(xsc,ysc):CLEARview(0,0,135,30):
               end :
   6 : BEGIN   blank  : end :
   7 : BEGIN   RHOTHETA(angle1,angle2,length1,length2,count1,count2):
               END:
   else:
    END:
   until menusel = 0 :
   setviewport(0,0,639,199,clipon):
   CLEARdevice :  closegraph : restorecrtmode :
   window(1,1,80,25):textbackground(0):textcolor(3):clrscr:
 END:
```

Appendix C8

```
{----------------------------------------draw polygon-----------}
 PROCEDURE CONTDRAW(xx,yy:vert:n,xcen,ycen:integer:xsc,ysc:real):
 BEGIN
     moveto( round(xsc*xx{1}+xcen+250) ,
     round(ysc*abs(yy{1})+ycen)  ):
     FOR kkl := 2 TO n do BEGIN
        lineto( round(xsc*xx{kkl}+xcen+250) ,
        round(ysc*abs(yy{kkl})+ycen) ):
     END:
     lineto( round(xsc*xx{1}+xcen+250) ,
     round(ysc*abs(yy{1})+ycen)  ):
 END:
```

Appendix C9

```
{-------------------------------------------plot graph----------}
PROCEDURE RHOTHETA(angle1,angle2,length1,length2:vert:
                                     count1,count2:integer):
BEGIN
   line(40,1,40,198): xx:=40:yy:=75:moveto(round(xx),round(yy)):
   FOR rt := 1 TO count1 do BEGIN
     yy := yy - angle1{rt}/6     :  lineto(round(xx),round(yy)) :
     xx := xx + length1{rt}/3    :  lineto(round(xx),round(yy)) :
   END:
   xx := 40 : yy := 150 : moveto(round(xx),round(yy)):
   FOR rt := 1 TO count2 do BEGIN
     yy := yy - angle2{rt}/6     :  lineto(round(xx),round(yy)) :
     xx := xx + length2{rt}/3    :  lineto(round(xx),round(yy)) :
   END:
END:
```

Appendix C10

```
PROCEDURE  DISPOBJINCLASS:
BEGIN
    tempclassobj1 := GENr.class :   posln := 8: textcolor(5):
    gotoxy(2,6):write('    class      object     perimeter'):
    gotoxy(2,7):write('-----------------------------------'):
    SEARCHKEY(obj1,OBJRECnum,tempclassobj1):  textcolor(3):
    IF ok THEN BEGIN
     tempval := OBJRECnum + strtoint(GENr.numobjects) - 1 :
     FOR tempind := OBJRECnum TO tempval do BEGIN
      GETREC(obj,tempind,objr): gotoxy(1,posln):
      write(objr.class:10,objr.objname:10,strtoint(objr.perimeter):9):
      inc(posln):  end : { FOR loop }
    END:
END:
```

Appendix C11

```
      chl:=' ':textcolor(3):
      gotoxy(5,5):write('Is printer is connected  {y/n} : '):
      textcolor(5):gotoxy(65,5):readln(chl):
      IF chl <> 'y' THEN goto 201 :
      writeln(lst,'sides: ',count1,'        perimeter: ',fmccnt):
      writeln(lst,' side #  x-ord  y-ord  length  angle  pointer' ):
      writeln(lst,'------------------------------------------------'):
      FOR rt :=1 TO count1 do BEGIN
          writeln(lst,rt:5,corx1{rt}:9,cory1{rt}:9,length1{rt}:9,
          angle1{rt}:9,point1{rt}:8):
      END:
```

Appendix C12

```
{-----------------------------print out the dbase fields indexed---}
PROCEDURE PRINTFIELDS(fn:integer:msg:string50):
BEGIN
 textcolor(3):
    CASE fn of
       1 :CLEARKEY(GEN1):          2 :CLEARKEY(GEN2):
          .    .    .        .        .         .
    END:  posln := 4:
 FOR k := 1 TO GENRECcount do BEGIN
   CASE fn of
     1:NEXTKEY(GEN1,GENRECnum,tempclass):
     2:NEXTKEY(GEN2,GENRECnum,tempstr2):
          .    .    .        .        .         .
   END:   gotoxy(14,posln):
   IF ok THEN BEGIN
     GETREC(GEN,GENRECnum,GENr):
     CASE fn of
       1:write(k:7,GENRECnum:8,'     :  ',GENr.class:10):
       2:write(k:7,GENRECnum:8,'     :  ',GENr.class:10,'     ',
       strtoint(GENr.sides) ):
          .    .    .        .        .         .
     end
   END: inc(posln):
 END:
END:
```

Appendix C13

```
{---------------------------------------all info FOR a class----}
 gotoxy(5,3):write( 'enter search  class name   : '):
 gotoxy(40,3):readln(searchclass):
 tempclass := searchclass :
 FINDKEY(GEN1,GENRECnum,tempclass):
 IF ok THEN  BEGIN                          { class already exists }
    gotoxy(1,1):write('class  .# of sides .. heading '):
    GETREC(GEN,GENRECnum,GENr):textcolor(3):
    gotoxy(1,4):write(GENr.class):gotoxy(8,4):
    write(strtoint(GENr.sides):6,strtoint(GENr.maxside):8,
          strtoint(GENr.largeside):5,strtoint(GENr.maxanglep):8,
          strtoint(GENr.maxanglen):9,strtoint(GENr.largeanglep):7,
          strtoint(GENr.largeanglen):7,strtoint(GENr.sidesdet):8,
          strtoint(GENr.numobjects):5,strtoint(GENr.subclass):5 ):
   tempclassobj := searchclass :   posln := 8: textcolor(5):
   gotoxy(1,7):write('object name    perimeter '):
   SEARCHKEY(obj1,OBJRECnum,tempclassobj): textcolor(3):
   IF ok THEN BEGIN
      tempval := OBJRECnum + strtoint(GENr.numobjects) - 1 :
      FOR tempind := OBJRECnum TO tempval do BEGIN
         GETREC(obj,tempind,objr): gotoxy(1,posln):
         write(objr.objname:10,strtoint(objr.perimeter):11):
         inc(posln):  end : { FOR loop }
   END:
   tempclasside := searchclass + inttostr(1) :posln := 7:
   gotoxy(45,6):write(' side order    length    angle'):
   SEARCHKEY(det1,DETRECnum,tempclasside): textcolor(3):
   IF ok THEN BEGIN
     tempval := DETRECnum + strtoint(GENr.sidesdet) - 1 :
     FOR tempind := DETRECnum TO tempval do BEGIN
      GETREC(det,tempind,detr): gotoxy(48,posln):
      write(strtoint(detr.sideord):6,strtoint(detr.length):12,
            strtoint(detr.angle):8):
      inc(posln):  end : { FOR loop }
         IF posln > 23 THEN  write('press enter TO continue'):
                        readln:clrscr:posln:=7:
   END:
  gotoxy(55,24):textcolor(4):write(' enter TO go on ..'):readln:
  end  { class already exists }
  ELSE  BEGIN textcolor(4):        { class not found }
    gotoxy(5,8):write(' search class name not found  '):readln:
  end :
```

Appendix C14

```
    GETTIME(hh,m5,s5,h5) :   GETTIME(hh,m6,s6,h6) :
PROCEDURE timediff
(var mmm,sss,hhh:integer:mx2,sx2,hx2,mx1,sx1,hx1:word:msg:str30):
BEGIN
  mmm := mx2 - mx1 :
  sss := sx2 - sx1 :
  IF sss = 0 THEN
     hhh := hx2 - hx1
  ELSE  BEGIN
     sss := 0 :
     hhh := (100 - hx1) + hx2 : end :
     writeln(msg:30,'   : ',mmm:5,' m ',sss:5,' s ',hhh:7,'   hun'):
END:
```

Appendix C15

```
PROCEDURE ADDCLASSTODB: BEGIN
  gotoxy(5,3):write( 'enter new or existing class name       : '):
  readln(userclsname): tempclass := userclsname :
  FINDKEY(GEN1,GENRECnum,tempclass):
  IF ok THEN  BEGIN                          { class already exists }
        GETREC(GEN,GENRECnum,GENr):
        gotoxy(5,4):write('class already exists '):
        write(' objects in class :  ',strtoint(GENr.numobjects):3):
        DISPOBJINCLASS:
        gotoxy(40,6): write('new object perimeter : ',fmccnt:6):
        gotoxy(40,10):write('add object TO class ? {y/n}  :        ')
        gotoxy(71,10):readln(opt):
        IF opt <> 'y' THEN exit
        ELSE BEGIN   ADDOBJTOCLASS  end : { add new object TO class }
     end  { class already exists }
  ELSE  BEGIN { new class TO be added with first object }
     gotoxy(5,8):write(' this is the first object in this class    ')
     gotoxy(5,10):write(' create new class and add object ? {y/n}: ')
     gotoxy(55,10):readln(opt):
     IF opt <> 'y' THEN exit
     ELSE BEGIN                 { first object }
       INPUTGENREC :           ADDKEYtoGEN :
       write(' 1st object name ? : '):   readln(objr.objname):
       objr.class := userclsname :  objr.perimeter:=inttostr(fmccnt):
       objr.objstat := 0 :    objr.distance := inttostr(0):
       tempclassobj:= objr.class+objr.objname :
       ADDREC(obj,OBJRECnum,objr):    ADDKEY(obj1,OBJRECnum,tempclass
       inputDETREC(0):  { insert the detailed sides in detobj.DBF }
     END: { opt not y }
  end :  { new  class 1st obj }
  END:
```

Appendix C16

```
PROCEDURE ADDOBJTOCLASS:
BEGIN
  gotoxy(40,10):write('enter new object name  :                  '):
  gotoxy(65,10):readln(objr.objname):           { get object name  }
       objr.class := GENr.class :  objr.perimeter := inttostr(fmccnt):
       objr.objstat := 0 : objr.distance := inttostr(0):
       tempclassobj :=  objr.class + objr.objname :
       ADDREC(obj,OBJRECnum,objr):
       ADDKEY(obj1,OBJRECnum,tempclassobj ):
       GENr.numobjects := inttostr ( strtoint(GENr.numobjects) + 1 ):
       putrec(GEN,GENRECnum,GENr):   { update record TO add object }
END:
```

Appendix C17

```
PROCEDURE  INPUTGENREC:
BEGIN
            GENr.GENstat       :=  0  :
            GENr.class         :=  userclsname                 :
            GENr.sides         :=  inttostr(count1)            :
            GENr.maxside       :=  inttostr(round(larger1))    :
            GENr.largeside     :=  inttostr(alargeside)        :
            GENr.maxanglep     :=  inttostr(amaxanglep)        :
            GENr.maxanglen     :=  inttostr(amaxanglen)        :
            GENr.largeanglep   :=  inttostr(alargeanglep)      :
            GENr.largeanglen   :=  inttostr(alargeanglen)      :
            GENr.sidesdet      :=  inttostr(count2)            :
            GENr.numobjects    :=  inttostr(1)                 :
            GENr.subclass      :=  inttostr(0)                 :
END:
```

Appendix C18

```
PROCEDURE ADDKEYtoGEN:
BEGIN
            ADDREC(GEN,GENRECnum,GENr):
            ADDKEY(GEN1,GENRECnum,GENr.class             ):
            ADDKEY(GEN2,GENRECnum,GENr.sides             ):
            ADDKEY(GEN3,GENRECnum,GENr.maxside           ):
            ADDKEY(GEN4,GENRECnum,GENr.largeside         ):
            ADDKEY(GEN5,GENRECnum,GENr.maxanglep         ):
            ADDKEY(GEN6,GENRECnum,GENr.maxanglen         ):
            ADDKEY(GEN7,GENRECnum,GENr.largeanglep       ):
            ADDKEY(GEN8,GENRECnum,GENr.largeanglen       ):
END:
```

Appendix C19

```
PROCEDURE inputDETREC(n:byte):
BEGIN
  detr.detstat := 0 :
  CASE n of
   0 :  detr.class   := userclsname:
   1 :  detr.class   := '1' + userclsname :
   2 :  detr.class   := '2' + userclsname :
  END:
  FOR i1 := 1 TO count2 do BEGIN
      detr.sideord := inttostr(i1):
      detr.length  := inttostr(length2{i1}):
      detr.angle   := inttostr(angle2{i1}):
      tempclasside := detr.class + inttostr(i1):
      ADDREC(det,DETRECnum,detr):
      ADDKEY(det1,DETRECnum,tempclasside  ):
  END:
END:
```

Appendix C20

```
BEGIN                              {    last  record }
   openfile ( GEN   ,'GENobj.DBF'     , sizeof(GENr) ):
   openindex( GEN1  ,'classnam.NDX'   , 10         , 0 ):
   openindex( GEN2  ,'sides.NDX'      , 2          , 1 ):
   openindex( GEN3  ,'maxsid.NDX'     , 2          , 1 ):
     -   -   -   -   -   -   -   -   -   -   -   -   -
             GENr.GENstat      :=  0  :
             GENr.class        :=  'lastrec'                 :
             GENr.sides        :=  inttostr(2000)             :
             GENr.maxside      :=  inttostr(2000)  :
             GENr.largeside    :=  inttostr(2000)      :
             GENr.maxanglep    :=  inttostr(2000)      :
             GENr.maxanglen    :=  inttostr(2000)      :
             GENr.largeanglep  :=  inttostr(2000)    :
             GENr.largeanglen  :=  inttostr(2000)    :
             GENr.sidesdet     :=  inttostr(2000)          :
             GENr.numobjects   :=  inttostr(2000)           :
             GENr.subclass     :=  inttostr(2000)           :
             ADDREC(GEN,GENRECnum,GENr):
             ADDKEY(GEN1,GENRECnum,GENr.class         ):
             ADDKEY(GEN2,GENRECnum,GENr.sides         ):
             ADDKEY(GEN3,GENRECnum,GENr.maxside       ):
     -   -   -   -   -   -   -   -   -   -   -   -   -
    closefile(GEN):
    closeindex(GEN1):closeindex(GEN2):closeindex(GEN3):closeindex(GEN4
    closeindex(GEN5):closeindex(GEN6):closeindex(GEN7):closeindex(GEN8
 end.
```

Appendix C21

```
PROCEDURE TESTSEARCH:
var begs,ends : integer : subcls : byte :
BEGIN
 numcls := 0 :
 window(1,2,80,25):clrscr:
       b2  :=  count1             :
       b3  :=  round(larger1)     :
       b4  :=  alargeside         :
       b5  :=  amaxanglep         :
       b6  :=  amaxanglen         :
       b7  :=  alargeanglep       :
       b8  :=  alargeanglen       :

 window(1,2,80,25):clrscr:
 {***********************************}  GETTIME(hh,m27,s27,h27):
 SEARCHCLASSES(2,b2    ,b2+1  ):
 SEARCHCLASSES(3,b3-55 ,b3+55 ):
 SEARCHCLASSES(4,b4    ,b4    ):
 SEARCHCLASSES(5,b5-6  ,b5+6  ):
 SEARCHCLASSES(6,b6+6  ,b6-6  ):
 SEARCHCLASSES(7,b7-1  ,b7+1  ):
 SEARCHCLASSES(8,b8    ,b8    ):
 {***********************************}  GETTIME(hh,m28,s28,h28):

 FOR ind1 := 1 TO numcls do BEGIN
     tempclass := clscls{ind1}:
     FINDKEY(GEN1,GENRECnum,tempclass):
     GETREC(GEN,GENRECnum,GENr):
       subcls := strtoint(GENr.subclass):
       if(  (largestat3=1) and (subcls<>2) ) THEN
                          clscnt{ind1} := 0
       ELSE BEGIN
        if((largestat2=1)and(subcls<>1)) THEN
                          clscnt{ind1} := 0
        ELSE BEGIN
          if((largestat2=0)and(subcls<>0)) THEN
                          clscnt{ind1} := 0:
        END:
       END:
 END:

 SORTCLUST:
 SHOWCLUST:

END:
```

Appendix C22

```
PROCEDURE SEARCHCLASSES(fn,begsid,endsid:integer):
BEGIN
    tempstrint1 := inttostr(begsid): tempstrint2 := inttostr(endsid):
    CASE fn of
         2:   SEARCHKEY(GEN2,GENRECnum,tempstrint1):
         3:   SEARCHKEY(GEN3,GENRECnum,tempstrint1):
         4:   SEARCHKEY(GEN4,GENRECnum,tempstrint1):
         5:   SEARCHKEY(GEN5,GENRECnum,tempstrint1):
         6:   SEARCHKEY(GEN6,GENRECnum,tempstrint1):
         7:   SEARCHKEY(GEN7,GENRECnum,tempstrint1):
         8:   SEARCHKEY(GEN8,GENRECnum,tempstrint1):
    END:
IF ok THEN BEGIN
locs1:
          GETREC(GEN,GENRECnum,GENr):
          CASE fn of
            2:  tempstr2 := GENr.sides          :
            3:  tempstr2 := GENr.maxside        :
            4:  tempstr2 := GENr.largeside      :
            5:  tempstr2 := GENr.maxanglep      :
            6:  tempstr2 := GENr.maxanglen      :
            7:  tempstr2 := GENr.largeanglep    :
            8:  tempstr2 := GENr.largeanglen    :
          END:
          IF ( tempstr2 <= tempstrint2 ) THEN BEGIN
                 CASE fn of
                   2: INSERTCLUST(GENr.class   , 2 ):
                   3: INSERTCLUST(GENr.class   , 1 ):
                   4: INSERTCLUST(GENr.class   , 1 ):
                   5: INSERTCLUST(GENr.class   , 2 ):
                   6: INSERTCLUST(GENr.class   , 3 ):
                   7: INSERTCLUST(GENr.class   , 2 ):
                   8: INSERTCLUST(GENr.class   , 3 ):
                 END:
              CASE fn of
                2: NEXTKEY(GEN2,GENRECnum,tempstrint1):
                3: NEXTKEY(GEN3,GENRECnum,tempstrint1):
                4: NEXTKEY(GEN4,GENRECnum,tempstrint1):
                5: NEXTKEY(GEN5,GENRECnum,tempstrint1):
                6: NEXTKEY(GEN6,GENRECnum,tempstrint1):
                7: NEXTKEY(GEN7,GENRECnum,tempstrint1):
                8: NEXTKEY(GEN8,GENRECnum,tempstrint1):
              END:
              goto locs1:
          END:{ IF <= tempstrint2 }
    END:
END:
```

Appendix C23

```
PROCEDURE INSERTCLUST(newcls:codestr:incwt:shortint):
label insl :
BEGIN
    FOR posln := 1 TO numcls do BEGIN
        IF ( clscls{posln} = newcls ) THEN BEGIN
             inc( clscnt{posln} , incwt) : goto insl : end :
    END:
    inc(numcls):
    clscls{numcls} := newcls :
    clscnt{numcls} := incwt :
insl :
END:
```

Appendix C24

```
PROCEDURE SORTCLUST:
var tempcls : codestr: tempcnt,flag,i,j : byte :
BEGIN
     flag := 1 : i := 1 :
     while ( flag = 1 ) do
     BEGIN
        flag := 0 :
        FOR j:= 1 TO (numcls-i) do BEGIN
            IF ( clscnt{j} < clscnt{j+1} ) THEN BEGIN
               tempcls       := clscls{j} :
               clscls{j}     := clscls{j+1} :
               clscls{j+1}   := tempcls :
                  tempcnt       := clscnt{j} :
                  clscnt{j}     := clscnt{j+1} :
                  clscnt{j+1}   := tempcnt :
                  flag := 1 :
            END: { IF }
        END: { FOR }
        inc(i):
     END: { while }
END:
```

Appendix C25

```
PROCEDURE SHOWCLUST:
BEGIN
  textcolor(4):gotoxy(45,3):write(' class cluster   frequency '):
  textcolor(3):k := 4:
  FOR posln := 1 TO numcls do BEGIN
      gotoxy(40,k):writeln( clscls{posln}:15,'   ',clscnt{posln}:5 ):
      inc(k) : end :
END:
```

Appendix C26

```
PROCEDURE TESTSEARCH2class:
var begs,ends : integer : xx,yy : real :
BEGIN
 window(1,2,80,25):clrscr:
   gotoxy(5,3):write('enter class TO match TO : '):
   gotoxy(35,3):readln(searchclass):
   tempclass := searchclass :
   FINDKEY(GEN1,GENRECnum,tempclass):
   IF ok THEN BEGIN
     maxa := count2 :
     FOR ind1 := 1 TO count2 do BEGIN
        la{ind1} := length2{ind1} :
        ta{ind1} := angle2{ind1}  :  end :
     GETREC(GEN,GENRECnum,GENr):
     maxb := strtoint(GENr.sidesdet) :
     gotoxy(5,6):write('enter sub class TO match TO : '):
     gotoxy(35,6):readln(searchclass):
    maxb := strtoint(GENr.sidesdet) :
    tempclassside := searchclass + inttostr(1) :
    SEARCHKEY(det1,DETRECnum,tempclassside): textcolor(3):
      IF ok THEN BEGIN
         tempval := DETRECnum + strtoint(GENr.sidesdet) - 1 :
         ind1 := 1 :
         FOR tempind := DETRECnum TO tempval do BEGIN
            GETREC(det,tempind,detr):
            lb{ind1} := strtoint(detr.length):
            tb{ind1}  := strtoint(detr.angle):
            inc(ind1):  end : { FOR loop }
   window(1,2,80,25):  clrscr :

   textcolor(5): gotoxy(1,2):write('  length   angle '):
   posln := 3 :textcolor(3):
   FOR ind1 := 1 TO maxa do BEGIN
     gotoxy(3,posln):write(la{ind1}:8,'   ',ta{ind1}):inc(posln):
   END:
   textcolor(5): gotoxy(41,2):write('  length   angle '):
   posln := 3 :textcolor(3):
   FOR ind1 := 1 TO maxb do BEGIN
     gotoxy(43,posln):write(lb{ind1}:8,'   ',tb{ind1}):inc(posln):
   END:
   gotoxy(1,posln):

   IF ( abs(la{1} -lb{1}) <= 50 ) THEN la{1} := lb{1} :
   AREADIFF(maxa,maxb,la,ta,lb,tb,ar):
   gotoxy(1,24):write('                                              '):
   gotoxy(1,24):write(' diff  of graph area : ',ar:15):   readln:
   END: { ok of SEARCHKEY(det1.. }
   end    { ok FOR finkey(GEN1.. }
   ELSE
     writeln( ' class not found '):    { ELSE ok FOR finkey(GEN1.. }
 END:
```

Appendix C27

```
PROCEDURE AREADIFF(maxa,maxb:integer:la,ta,lb,tb:vert:var ar:longint )
 var i,j,k :integer:      t,x,y : integer :
BEGIN
    ta{1} := 0 : tb{1} := 0 :
    FOR i:= 2 TO maxa do  ta{i} := ta{i} + ta{i-1} :
    FOR i:= 2 TO maxb do  tb{i} := tb{i} + tb{i-1} :
    ar:=0:    i:=1:   j:=1:  x:=la{i} : y:= lb{j}:
    REPEAT
       MIN(x,y,k):
       t:= abs( ta{i} - tb{j} ):
       ar := ar + t*x:
       IF (y=0.0) THEN BEGIN
          i:=i+1:
          j:=j+1:
          x:= la{i}: y:= lb{j}:
          end
       ELSE
          IF (k=1) THEN
              BEGIN
                i:=i+1:
                x:=la{i}:
              end
          ELSE
              BEGIN
                j:=j+1:
                x:=y:
                y:= lb{j} :
       END:
 until (i > maxa) or (j > maxb) :
 END:
```

Appendix C28

```
PROCEDURE MIN(var x,y: integer: var k : integer):
var temp : integer:
 BEGIN
 IF (x<y) THEN
    BEGIN
       y := y - x:
       k := 1:
    end
  ELSE
    BEGIN
       temp := x:
       x := y:
       y := temp - y:
       k := 2 :
    END:
 END:
```

Appendix C29