Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

A COMPARATIVE STUDY OF EDGE DETECTION TECHNIQUES

by
Jaskaran Singh Dhaliwal

Thesis submitted to the Faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering, 1990.

APPROVAL SHEET

Title of Thesis: A Comparative Study of Edge Detection Techniques.

Name of the Candidate: Jaskaran Singh Dhaliwal

Master of Science in Mechanical Engineering, 1989 90

Thesis and Abstract Approved:

_____ _____

Dr. R. Dave, Date

Assistant Professor Department of

Mechanical Engineering

_____ _____

Date

_____ _____

Date

VITA

Name : Jaskaran Singh Dhaliwal

Degree and date to be conferred : M.S.M.E., Jan. 1990

Secondary education : Govt. High School Birampur, 1980

College/Institution	Dates	Degree	Date of Graduation
G.K.S.M.G.C., Tanda	80-82	Pre Engg.	June, 1982
G.N.E.C., Ludhiana	82-86	B.E.M.E.	Aug., 1986
N.J.I.T., Newark	87-90	M.S.M.E.	Jan., 1990

Major : Mechanical Engineering

Positions held : Graduate Assistant, CAD Department,
N.J.I.T, Sept. 89 - Present
Graduate Assistant, Mechanical Engineering,
N.J.I.T., Sept. 88 - Sept. 89

ABSTRACT

Title of Thesis: A Comparative Study of Edge Detection Techniques.

Jaskaran SIngh Dhaliwal, M.S.M.E., 1990

Thesis directed by: Professor R. N. Dave

The problem of detecting edges in gray level digital images is considered. A literature survey of the existing methods is presented. Based on the survey, two methods that are well accepted by a majority of investigators are identified. The methods selected are: 1) Laplacian of Gaussian (LoG) operator, and 2) An optimal detector based on maxima in gradient magnitude of a Gaussian-smoothed image. The latter has been proposed by Canny[], and will be referred as Canny's method. The purpose of the thesis is to compare the performance of these popular methods. In order to increase the scope of such comparison, two additional methods are considered. First is one of the simplest methods, based on the first order approximation of the first derivative of the image. This method has the advantage of relatively low amount of computations. Second is an attempt to develop anedge fitting method based on eigenvector least-squared error fitting of an intensity profile. This method is developed with an intent to keep the edge localization errors small. All the four methods are coded and applied on several digital images, actual as well as synthesized. Results show that the LoG method and Canny's method perform quite well in general, and that demonstrates popularity of these methods. On the other

hand, even the simplest method of first derivative is found to perform well if applied properly. Based on the results of the comparative study several critical issues related to edge detection are pointed out. Results also indicate feasibility of the proposed method based on eigenvector fit. Improvements and recommendation for further work are made.

ACKNOWLEDGEMENT

I take this opportunity to record my gratitude and thanks to Dr. Rajesh N. Dave, Assistant Professor, Mechanical Engineering Department of New Jersey Institute of Technology for his valuable guidance throughout the course of this thesis work.

I am extremely thankful to Dr. Joshua S. Greenfeld, Assistant Professor, Civil Engineering Department for his valuable help and suggestions during the thesis work.

My many thanks are due to Mr. Somayajulu Bhamidipati for his great help during the thesis work and typing it.

Table of Contents

CHAPTER I

INTRODUCTION

1.1 IMAGE PROCESSING :

Image processing is the most important part of the machine vision. Machine Vision is becoming more and more important these days because it is being used in manufacturing, inspection of parts, medical applications and robot guiding. Machine vision means making the machines to interact with the environment as human beings do in terms of seeing. But there is much work left to be done. Visual data are most complex and most useful sensory input for humans. Machine vision is concerned with the interpretation of similar visual data. Image Processing is the science of modifying and analyzing pictures. For analysis of images we need to find their edges first. Edges information can be used for segmentation of images or for locating their boundary information. Edge detection is important because edges give the compact description of the objects and objects can be reconstructed from the edge information.

Vision begins with transformation of a flux of photons into a set of intensity values at an array of sensors. The first step in visual information processing is to obtain a compact description of the raw intensity values. The primitive elements of the initial description should ideally be complete in the sense of representing the full information contained in the image, and meaningful (that is, capturing significant properties of the three-dimensional surfaces around the viewer). Physical edges are among the most

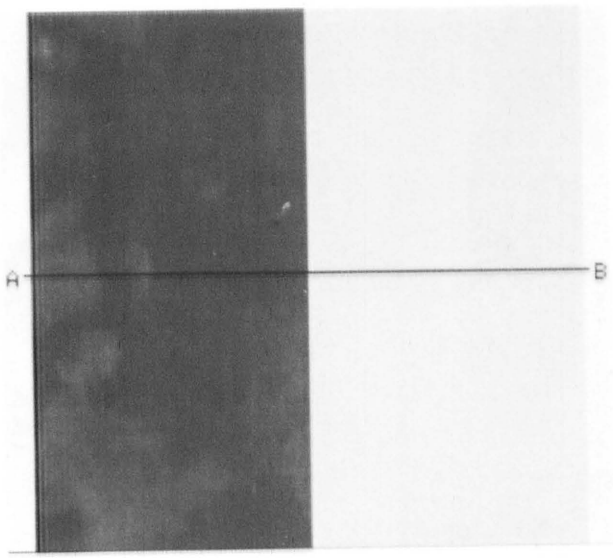
ties of objects since they correspond to object boundaries or to changes in surface orientation or material properties.[1]

1.2 EDGES :

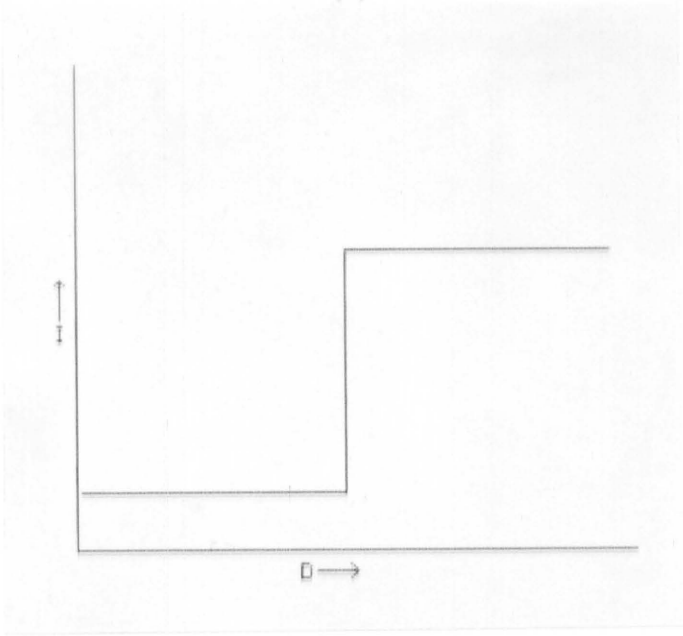
Edges are image attributes which are useful for image analysis and classification in a wide range of application. An edge is defined as a large and sudden change in some image attribute, usually the brightness. The changes in brightness that we are particularly interested in are the ones that mirror significant events on the surface being imaged. These might be the places where surface orientation changes discontinuously, where one object occludes another, where a cast shadow line appears, or where there is a discontinuity in surface reflectance properties. The importance of edges in human visual and perceptual system was first proven by Attneave[1] who showed that a crude outline of an object is an enough information for an object recognition e.g. we can recognize the faces or objects in cartoons easily, which are edges of faces or objects.

The edges are categorized into four types which may occur in an image. These are step edge, ramp edge, roof edge, and spike edge. In the following description Figure (a) represent the image of the edge and Figure (b) represent the edge profile at line AB. These are described as follows:

1. *Step edge* : A step edge can be found in an image if we identify a region of the image which is significantly brighter (or darker) from its surroundings. A step edge exists

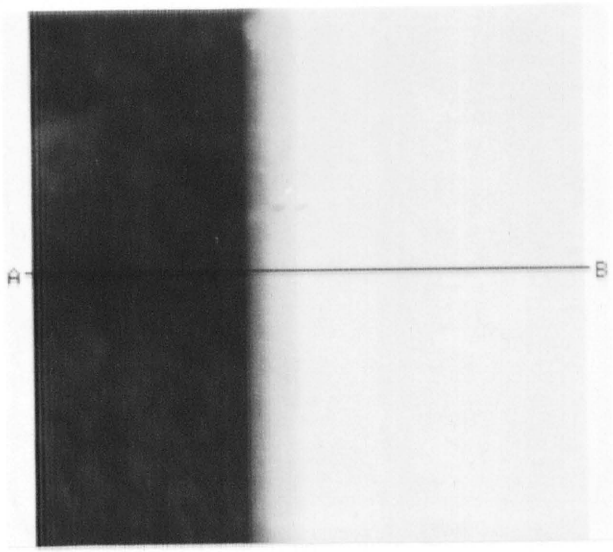


(a)

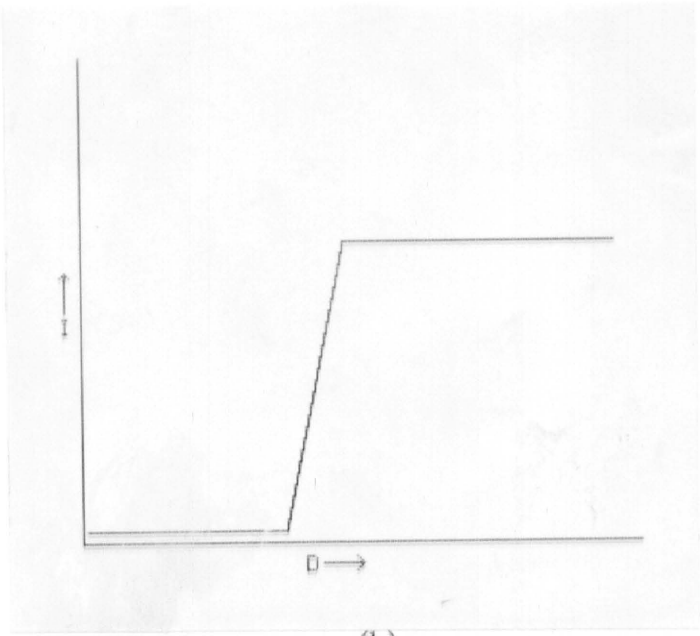


(b)

Figure 1.1 (a) Step edge image (b) Step edge profile at line AB

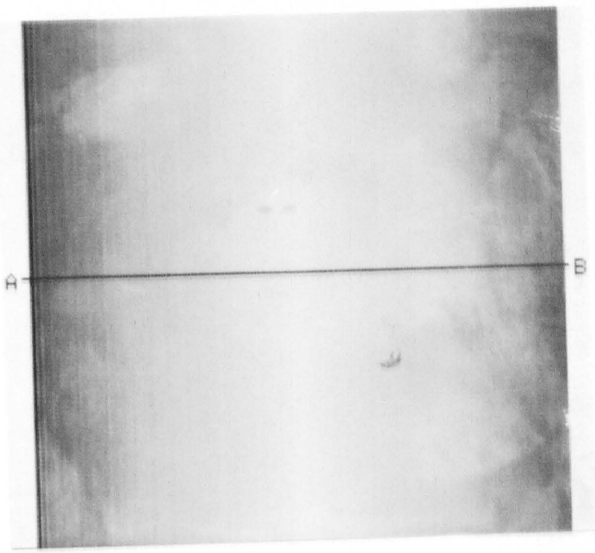


(a)

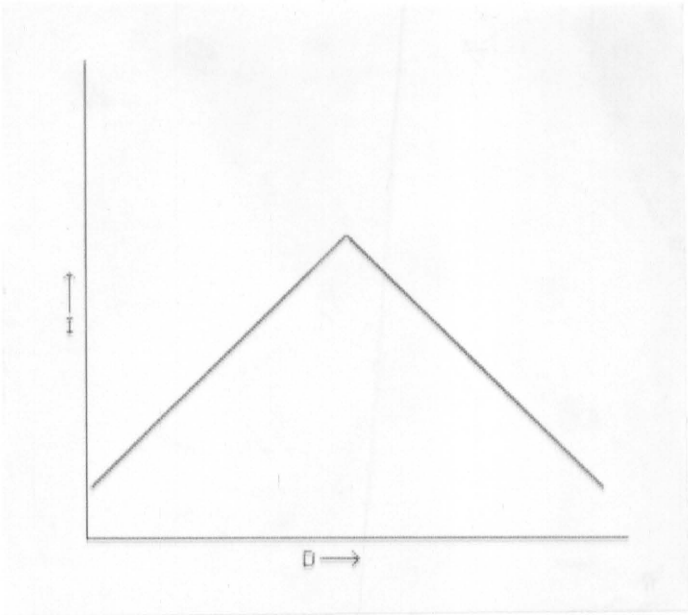


(b)

Figure 1.2 (a) Ramp edge image (b) Ramp edge profile at line AB

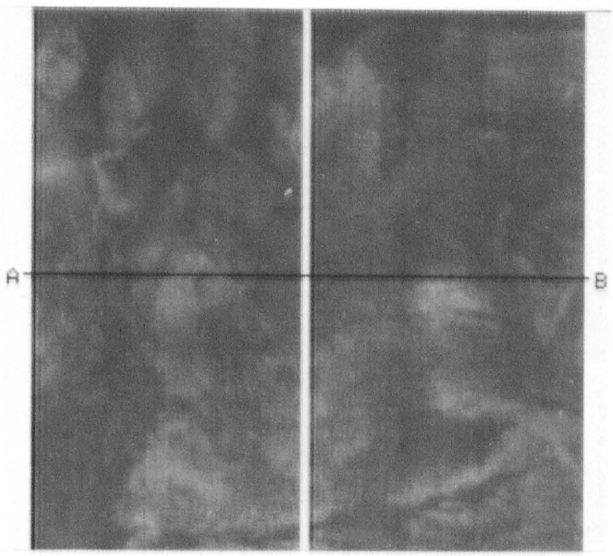


(a)

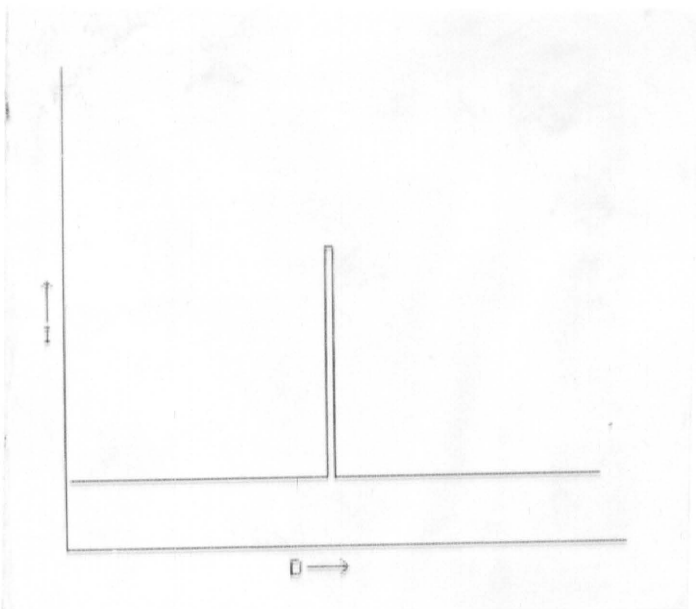


(b)

Figure 1.3 (a) Roof edge image (b) Roof edge profile at line AB

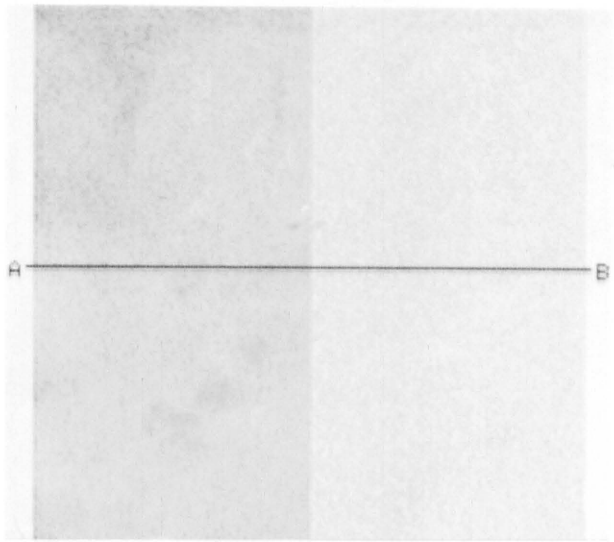


(a)

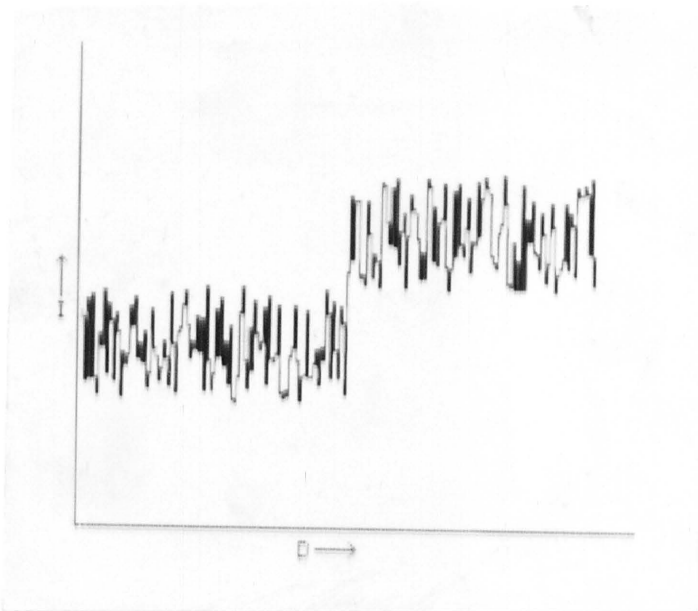


(b)

Figure 1.4 (a) Spike edge image (b) Spike edge profile at line AB



(a)



(b)

Figure 1.5 (a) Noisy edge image (b) Noisy edge profile at line AB

between two neighboring pixels if one belongs to region and other belongs to surroundings. This is an idealized situation and is shown in Figure1.1.

2. *Ramp edge* : This is more common in real images. In a ramp edge the transition from the regions to surroundings over several pixels as shown in Figure1.2.

3. *Roof edge* : Roof edge occurs if the profile of image function steadily increases and after a certain point steadily decreases as shown in Figure1.3. This type of edge is usually not due to object boundaries but rather due to changes in surface orientation with respect to an illumination source.

4. *Spike edge* : Spike edge is composed of two step edges of opposite signs which occur in very short interval and illustrated in the Figure1.4.

The step changes in intensity are important because they correspond to sharp changes in orientation or to object boundaries. Edge detection is a means of generating compact description which preserve most of the structural information in an image.

In actual practice the edge profiles are not smooth as shown previously, but noise present in the image makes the profile to deflect from its actual path as shown in the Figure1.5.

According to Marr[10] there are four main factors responsible for intensity change in an image. Geometry, illumination, reflectance and viewpoint. The purpose of the early vision process is to sort out which changes are due to which factor. It is essential to develop

methods to identify changes of a specific kind. Needless to say that changes due to geometry (edges of objects) are of greater interest while responses due to other factors are of little or no interest and should be suppressed.

1.3 OBJECTIVE :

The objective of this thesis is to compare the most widely used edge detection techniques and to develop a new technique for edge detection. For this purpose literature survey of various edge detection techniques was done and three techniques were selected for implementation i.e., LoG, Canney's edge detector and First derivative method.

1.4 EDGE OPERATORS :

The ultimate goal of edge detection is the characterization of intensity changes in the image in terms of physical process that originated them. The edge operators may include any or all of the following:

1. Magnitude of the edge.
2. Direction of the edge.
3. Reliability of the edge description.
4. Width or blur.

The most simple edge detector is an arbitrary thresholding of gray level values at a certain constant to formulate a binary image. In a binary image all pixels have a value either 1 (white) or 0 (black). Thus, all pixels which have a gray level value larger than the threshold will be assigned the value 1, and the rest will have the value 0. The edges are

formed by the boundaries between the black and white regions. The main problem in devising an edge detector is that edges are a high frequency phenomenon and so is the noise in the image. To avoid detecting noise as edges a low pass filter or a large (in spatial extent) operator is used to average out the noise. But this operation also averages out the edges and is likely to form a single averaged edge from several neighboring edges. Consequently, this degrades the position accuracy of the detected edges.

Edge detectors should be formulated for different contexts. The requirements of many situations are similar and it is possible to design one edge detector for several contexts. The most important step in design of such a detector should be the specification of performance criteria. The edge detector accepts discreet digitized images and processes an "edge map" as its output. The edge map includes information about the position, strength and orientation of edges.

There are different ways to classify the edge operators. One way is to classify them according to the type of mathematical method used for detecting the edges. Mathematical methods can be gradient operators of first, second or higher derivatives to locate discontinuities in the image function. Another mathematical model can be, for example: Model fitting of stored model to the image function. Once we obtain a "good" fit the edges are the boundaries of the recovered model. For the purpose of this work another classification will be used: Directional and Non-Directional edge detectors. The directional edge detector evaluates the intensity changes (usually through a gradient operation) in the direction perpendicular to the edge direction. This class of edge detectors requires that the edge direction should be evaluated prior to analyzing whether to mark

the pixel as being located on an edge. The non-directional or the isotropic edge detector evaluates the intensity changes in the image function in no particular or preferred direction. An example of a mathematical operator which differentiates a function symmetrically in all directions is Laplacian.

Some previous formulation have chosen the first or second derivatives as the appropriate quantity to characterize step edges, and have formed optimal estimates of this derivative over some support, Examples of first derivative operators are the operators of Roberts (14) and Macleod (9), while Modestino and Fries (12) formed an optimal estimate of the two dimensional Laplacian over a large support. Marr and Hildreth (10) suggested the Laplacian of a broad Gaussian since it optimizes the trade-off in localization and bandwidth. According to Canny [3] there are problems with the Laplacian however and the whole concept of derivative estimation seems to have poor foundation

.

There is a second major class of formulations in which image support is approximated by a set of basis functions and the edge parameters are estimated from the modeled image surface. Examples of this technique include the work of Prewitt (1970), Huckell (1971) and Haralick (1982), The methods allow more direct estimates of edge properties such as position and orientation, but since the basis functions are not complete, the properties apply only to a projection of the actual image surface on to the subsurface spanned by the basic functions. According to Canny [3] the basis functions are a major factor in operator performance, especially the ability to localize an edge.

1.5 OVERVIEW OF REMAINING CHAPTERS :

In Chapter 2 a literature review of the edge detectors is given. In Chapter 3 details for implementing two most widely accepted edge detectors LoG and Canny's edge detector, as well as an edge detector based on first derivative are described. LoG is a non-directional edge detector while Canny's edge detector is a directional edge detector. In Chapter 4 details of a new approach based on eigenvector line fitting to the intensity values is described. Chapter 5 contains results and comparison of four edge detectors (LoG, Canny's edge detector, Eigenvector line fitting, and First derivative) as well as conclusions.

CHAPTER II

LITERATURE REVIEW

The computer vision literature is flooded with articles describing edge detectors. Many different techniques for edge detection are discussed in the technical literature and text books on machine vision and scene analysis. This chapter describes some of them. There are two major classifications of edge detectors as listed below:

1. Edge detectors based on first or second derivatives as appropriate quantities to characterize step edges.
2. Edge detectors based on image support approximation by a set of basic functions and edge parameters.

2.1 EDGE DETECTORS BASED ON FIRST OR SECOND DERIVATIVES

2.1.1 L.G. Robert:

L.G. Robert [16] implemented first simple gradient function as an edge operator. His edge detector is based on the assumption that the light intensity is constant or smoothly varying over the image of an object and jumps discontinuously at the intersection with the image of another face. This assumption is valid if the object surfaces are smooth, homogeneous and opaque and lighting is uniform and is arranged to eliminate shadows. In a continuous image plane at which the intensity changes discontinuously are easily identified to be those where the gradient of intensity function is infinite (or larger then the threshold). An approximation to this gradient is given by:

$$R(i,j) \cong g(i,j) = [\{g(i+1,j+1) - g(i,j)\}^2 + \{g(i,j+1) - g(i+1,j)\}e^2]^{1/2} \quad (1)$$

where g(i,j) is the image intensity at pixel (i,j). The direction of gradient is given by angle. where

$$\alpha = -\frac{\Pi}{4} + \tan^{-1}\left[\frac{g(i,j+1) - g(i+1,j)}{g(i+1,j+1) - g(i,j)}\right] \quad (2)$$

This operator is called Robert's cross operator. An edge is present at a pixel (i,j) if R(i,j) > T, where T is a chosen threshold. In a noise free image T can be taken as zero and in a noisy image T is chosen by a trade-off between obtaining all the desired edges and picking too many noise edges. The ideal requirements for Robert's operator are never met in real life images, but it works well in many situations, where controlled lighting is available.

2.1.2 A. Rosenfeld and M. Thurston:

Azriel Rosenfeld and Mark Thurston [18] introduced smoothing step before differentiation. According to them an edge through a point can be detected by comparing the average gray levels in pairs of non-overlapping neighborhoods that meet at a point. Here the relative orientation of the neighborhoods determines the direction of edges that will be detected and the size of the neighborhood determines the width of edges that will be detected. To detect micro-edges small neighborhoods must be used. For digital pictures on a squared grid, square neighborhoods whose signs are powers of 2 are especially convenient. The steps for edge detection are:

1. First picture is shifted to the right by successive amounts 1, 2, 4, 8 ..... 2^{k-1} and these pictures are added point-wise. The result is an array in which the value at each

location (i,j) is the sum of gray levels in the original picture at points (i, j), (i, j-1).....(i, $j-2^k+1$).

2. Picture is shifted downward so that the total number of shifts and adds is 2^k at each point, the sum of the gray levels is a 2^k X 2^k square having the point as lower right hand corner. These sums can be regarded as averages by simply shifting the binary point, because the number of terms in each sum is a power of 2.

3. The difference between the averages for any pair of neighborhoods is computed at every point by a single shift and pointwise subtraction. The edges are detected using neighborhoods of size (mask size) 8, 16, and 32.

Here the problem is the mask size. Large size masks eliminate the small edges and small sizes give noisy edges. All of these operations can be performed very efficiently in parallel using a computer such as Illiac III.

2.1.3 I.D.G. Macleod:

Macleod [10] developed Gaussian mask edge detector in 1970. Under his scheme, edge weights were computed for each point by multiplying the gray level value of corresponding point of a mask and summing. The mask consisted of the difference of two Gaussians displayed perpendicular to the expected edge direction, multiplied by a Gaussian envelop which tapered off parallel to the expected edge direction. This mask is given by:

$$w(x,y) = e^{-(y/t)^2}\left[e^{-(x-p/\rho)^2} - e^{-(x+p/\rho)^2}\right] \quad (3)$$

for edge expected to be in vertical direction.

2.1.4 Robert M. Haralick:

Haralick [5] locates edges at second directional derivative. According to him an edge occurs at a pixel if and only if there is some point in the pixel's area having a negatively sloped zero crossing of the second directional derivative taken in a direction of a non zero gradient at the pixel's center. Thus to determine whether or not a pixel should be marked as a step edge,. its underlying gray tone intensity surface must be estimated on the basis of the pixels in its neighborhood. The directional derivative Edge Finder can be described as:

Let derivative f at the point (r,c) in the direction α be denoted by by $f'_{\alpha}(r,c)$, and is defined as:

$$f'_{\alpha}(r,c) = \lim_{n \to 0} \frac{f(r + h\sin\alpha, c + h\cos\alpha) - f(r, c)}{a} \quad (4)$$

The direction angle is the clockwise angle from the column axis. It follows directly from this definition that

$$f'_{\alpha}(r,c) = \frac{\partial f}{\partial r}(r,c)\sin\alpha + \frac{\partial f}{\partial c}(r,c)\cos\alpha \quad (5)$$

The second directional derivative of f at the point (r,c) in the direction α is given by $f''(r,c)$ and it is obtained by substituting f' for f in (5) as shown below:

$$f''_{\alpha} = \frac{\partial^2 f}{\partial r^2}\sin^2\alpha + \frac{2\partial^2 f}{\partial rc}\sin\alpha\cos\alpha + \frac{\partial^2 f}{\partial c^2}\cos^2\alpha \quad (6)$$

Taking f to be a cubic polynomial in r and c which can be estimated by the discrete orthogonal polynomial fitting procedure as shown:

$$f(r,c) = k_1 + k_2 r + k_3 c + k_4 r^2 + k_5 r c + k_6 c^2 + k_7 r^3 + k_8 r^2 c + k_9 r c^2 + k_{10} c^3 \quad (7)$$

the angle is given by

$$\sin\alpha = k_2/(k_2^2 + k_3^2)^{1/2} \quad (8)$$
$$\cos\alpha = k_3/(k_2^2 + k_3^2)^{1/2} \quad (9)$$

At any point (r,c) the second directional derivative in the direction of α is given by:

$$f''_{\alpha}(r,c) = 6(k_7\sin^2\alpha + 4k_8\sin\alpha\cos\alpha + 2k_9\cos^2\alpha)r + (6k_{10}\cos^2\alpha + 4k_6\sin\alpha\cos\alpha + 2k_8\sin^2\alpha)c + (2k_4\sin^2\alpha + 2k_5\sin\alpha\cos\alpha + 2k_6\cos^2\alpha) \quad (10)$$

For points (r,c) on the line in direction α, $r = \rho\sin\alpha$ and $r = \rho\cos\alpha$ then

$$F''_{\alpha}(\rho) = 6[k_7\sin^3\alpha + k_8\sin^2\alpha\cos\alpha + k_9\sin\alpha\cos^2\alpha + k_{10}\cos^3\alpha]\rho + 2[k_4\sin^2\alpha + k_5\sin\alpha\cos\alpha + k_6\cos^2\alpha] = A\rho + B \quad (11)$$

If for some ρ, $|\rho| < \rho_0$ where ρ_0 is slightly smaller than the length of side pixel, $f''_{\alpha}(\rho) < 0$, $f''_{\alpha}(\rho) = 0$ and $f'_{\alpha}(\rho) = 0$; then there is a negatively sloped zero crossing of the estimated second directional derivative taken in the estimated direction of gradient and the center pixel of the neighborhood is marked as an edge pixel.

2.1.5 Binford:

Binford and co-workers [21] have suggested the use of support -limited filters in filtering step of edge detection. They have used the Harr function

$$F(w) = 1/jw\,(2/D)^{1/2}(1 - \cos wd) \quad (12)$$

in directional fitting or a difference of functions of the type

$$f(x) = \begin{cases} 0 & |x| > D \\ 1/(2d)^{1/2} & 0 < x < D \\ 1/(2d)^{1/2} & -D < x < 0 \end{cases} \qquad (13)$$

for rotational filtering. There are two problems in using this approach:

1. Filtering with support-limited functions does not regularize the image intensity ·ofile, therefore the use of differential operator is unsafe.

2. A strictly support-limited filter, such as a DOB (Difference of Boxes) can not be ›rrectly sampled, and it is difficult to obtain a good digital representation.

2.1.6 Shanmugam, Dickey and Green:

Shanmugam, Dickey and Green [20] consider the problem of optimizing special ɔmain filters for detecting edges in digital pictures. According to them filter is optimal . that it produces maximum energy within a resolution interval of specified width in the cinity of the edge. The filter has the transfer function:

$$H(w) = k_1 w^2 \exp^{-k_2 w^2} \qquad (14)$$

and it maximizes the proportion of (output) signal energy in a resolution interval ›ntered on the edge for a given bandwidth and resolution width. The filter works well ›r detecting blurred edges in noisy images. The filter is implemented via FFT algorithm. or an ideal edge S(x), the Fourier transform of optical filter is:

$$F_{op}(w) = \begin{cases} k_1 w \Psi_1 (w x_0 / \Omega, c) & | \Omega | \leq r \\ 0 & | \Omega | > r \end{cases}$$

If the input to the filter is an exponentially blurred edge, it will be detected if the width of the region of strong blurring is less than the width of the resolution interval. This is controlled by adjusting the parameter c of the filter (c = I/2). According to Torre and Poggio [21] this edge detector performs very poorly on localization and has the intrinsic feature of giving two maxima of energy in the output of the response.

2.1.7 Guner S. Robinson:

G.S. Robinson [19] has developed an edge detection scheme using 3 X 3 compass gradient masks. Edge angles are quantized to eight equally spaced directions. The compass names indicate the slope direction of maximum response e.g., the North gradient mask produces a maximum output for vertical luminance changes i.e. for horizontal

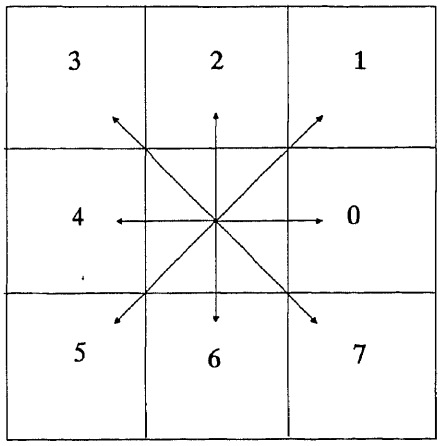


Figure.2.1. The eight principal directions on a 3 X 3 grid

edges. The direction of horizontal edge could be from left to right or from right to left.

Mask no.	Direction of edge	Mask
0	0	1 2 1 0 0 0 -1 -2 -1
1	45	2 1 0 1 0 -1 0 -1 -2
2	90	1 0 -1 2 0 -2 1 0 -1
3	135	0 -1 -2 1 0 -1 2 1 0
4	180	-1 -2 -1 0 0 0 1 2 0
5	225	-2 -1 0 -1 0 1 1 2 0
6	270	-1 0 1 -2 0 2 -1 0 1
7	315	0 1 2 -1 0 1 -2 -1 0

Figure. 2.2. Masks used for Robinson's method

The numbers 0,1,2,.....,7 are used for eight principal directions in a 3 X 3 grid as shown in Figure 2.1.

Edge directions corresponding to the eight compass directions are determined such that the bright side of the edge is always to the left as one moves in the direction of the edge. The application of the first four simple masks to 3 X 3 grid surrounding a picture element gives the gradient magnitude and direction. The gradient picture is obtained by taking the maximum gradient magnitude at each point. The mask which yields maximum gradient value determines the direction of the edge. The masks used are as shown in Figure 2.2.

2.1.8 D. Marr and E. Hildreth:

Marr and Hildreth [11] have proposed an edge detection based on the second derivative of Gaussian. According to them intensity changes are best detected by finding the zero values (or zero-crossings) of $\Delta^2 G(x,y) * I(x,y)$ for image I, where G(x,y) is the two-dimensional Gaussian distribution and Δ^2 is the Laplacian. This scheme is also called LOG (Laplacian of Gaussian). The details of this scheme are given in the chapter 3.

2.1.9 Ramesh Jain and Doug Rheaume:

R. Jain and D. Rheaume [8] have described a two stage method for edge detection. In first stage the potential edge points in the image are obtained using a fast method based on differencing edge detector. The real edge points are determined by applying an edge operation only to the potential edge points. The algorithm can be described as below:

A digital picture P is a two-dimensional array of numbers. An element, P(I,J), of the array represents the gray level of a pixel of the picture. The original picture has NXN pixels and the array P has M X M elements, where M > N. A translation TΔI, ΔJ of P results in translated picture TP such that

$$TP(I, J) = P(I + \Delta I, J + \Delta J),$$

where $\Delta I \leq M - N$ and $\Delta J \leq M - N$.

A difference picture DP for two pictures P1 and P2 is a binary picture such that

If ABS(P1(I, J) - p2(I, J)) > T1 then DP(I, J) = 1

else DP(I, J) = 0.

where T1 is a threshold value.

A binary edge picture BEP is binary picture such that

BEP(I, J) = 1

if DP(I, J) = 1 and at least one of eight neighbors of DP(I,J) = 0.

An edge picture EP obtained from picture such that

EP(I, J) = SOB(I, J)

if BEP(I, J) = 1 and SOB(I, J) T2

else EP(I, J) = 0.

where SOB(I, J) is an edge operator (Sobel operator) applied at point (I,J) of the image.

2.1.10 Alan C. Bovik and David C. Munson:

Bovik and Munson [2] have developed an edge detector based on differencing the median values of local image neighborhoods. According to them most usual approach to the noise problem is to use difference between the average values (or more generally weighted average) of local neighborhoods adjacent to each pixel. It works well if noise is Gaussian, but if the noise is impulsive then the average tends to be inefficient. So they proposed the use of median value (middle value in algebraic rank) of each image neighborhood then the average. The median is well known as effective estimate of location over a broad range of symmetric noise distributions. The edge orientation sensitivity can be described as:

Let the detector be centered over an edge with an arbitrary angular orientation with respect to the detector window. For simplicity let the edge follow a linear path through the central pixel of the window pair. If the image array has a constant value C on one side of the edge and value C + h (h 0) on the other, then for some integer m {0,.....,2n + 1):

A_L contains m elements with value C and (2n + 1 - m) elements with value C + h, and

A_R contains m elements with value C + h and (2n + 1 - m) elements with value C,

where A_L and A_R are left and right sides of the window. Using the absolute difference-of-averages scheme with neighborhood sizes (2n + 1) and predetermined threshold an edge will be deemed present if

$$\left| \frac{mC + (2n + 1 - m)(C + h) - (2n + 1 - m)C - m(C + h)}{2n + 1} \right| \qquad (15)$$

or

$$h > \left| \frac{2m}{2n + 1} \right|^{-1} \qquad (16)$$

However, if the absolute difference-of-medians is used, an edge will be detected whenever

$$h > \tau. \qquad (17)$$

So, the use of the median provides complete invariance to the orientation of the edge, whereas the difference-of-averages scheme provides performance that varies with m.

2.1.11 John Canny:

Canny [3] has investigated the desired properties of an optimal edge detector, based on efficiency of detection and reliability in localization. He has shown through variational methods that optimal odd filter $f_{op}(x)$ in 1-D case is the linear combination of four exponentials. The filter $f_{op}(x)$ is very close to

$$x\, e^{-x^2/\sigma^2}$$

which is the optimal odd filter for minimum uncertainty in detection. Canny's procedure for finding two-dimensional step edges and other type of edges uses directional

operators of varying width, length and orientation. This procedure includes an appropriate thresholding as an essential part, which works well for real life images. More details of this method are given in chapter 3.

2.2 EDGE DETECTORS BASED ON IMAGE SURFACE APPROXIMATION

2.2.1 Manfred H. Huckel:

Huckel [7] has developed an edge detector based on optimally fitting an ideal edge-line to the image intensity values in a small circular neighborhood. The ideal edge-line is determined by a 6-tuple of parameters, three parameters determine the brightness levels b_-, t_-, and t_+ as shown in Figure 2.3 and the other three parameters determine the position, orientation, and width of line. The fitting process consists of determining the values of six parameters for a best fit with the image intensities. Ideally the tuple of parameters is computed such that

$$N = || I - S(\text{tuple}) || \tag{18}$$

is minimum, where I is a vector representing image intensities and S is an ideal edge-line. This minimization process is approximated by expansion of both the input

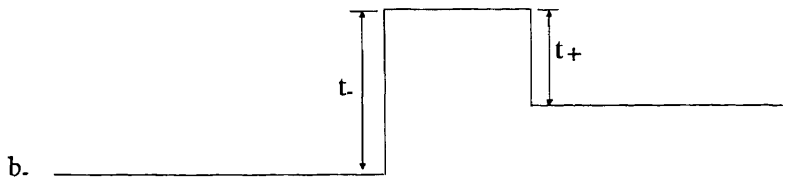


Figure 2.3 Intensity profile of an ideal edge.

image disk and the edge-line in an orthogonal Fourier series. If a is coefficients of expansion for image and s is coefficients of expansion for an ideal edge-line, then

$$N = \sum_{i=0}^{s} (a_i - s_i)^2 \qquad (19)$$

is minimized.

2.2.2 Ramakant Nevatia and K. Ramesh Babu

Nevatia and Babu [15] have described an edge detection scheme based on convolution with small edge like masks. The resulting output is thinned and linked by using edge position and orientation and approximated by piecewise linear segments. Image is convolved with masks corresponding to ideal step edges in a selected number of directions. The magnitude of the convolved output and the direction of the mask giving the highest output at each pixel are recorded as edge data. The edge data consists of magnitude and direction. An edge element is said to be present at a pixel if:

1. The output edge magnitude at the pixel is larger then the edge magnitudes of its two neighbors in a direction normal to the edge.

2. The edge directions of the two neighboring pixels are within one unit (30) of that central pixel.

3. The edge magnitude of the central pixel exceeds a fixed threshold.

Linking of edge pixels is obtained by considering connections of an edge point with 8-neighbors on a 3 X 3 grid. The boundaries are traced using this linking information. Then each boundary segment is approximated by a series of piecewise linear segments. The end point fit algorithm is used for this purpose.

2.2.3 Ralph Hartley:

R. Hartley [6] has developed an edge detector which weighs the neighborhood of a point according to a Gaussian function. The edge are found by fitting ideal edges to the image. A fit is done for each point on a square grid. Each point in the image is weighted so that its influence on the fit computed for a grid point falls off with the distance from a given point to the grid point. An ideal edge is represented by four parameters θ, r, b, and t. θ and r determine the position of the edge as shown in Figure. 1.3.

The other two functions determine the gray levels on each side of the edge. The gray level function of an ideal edge is:

$$\begin{aligned} S(x,y) &= b \qquad x\cos\theta + y\sin\theta <= r \\ &= b+t \qquad x\cos\theta + y\sin\theta > r \end{aligned} \tag{20}$$

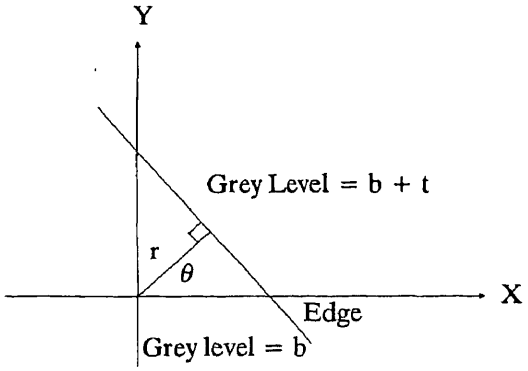


Figure. 1.3. Ideal Edge

for a given image a(x,y), parameters are found for which the difference between the ideal edge and the image is minimized. The weighing factor Q, which determines the relative importance of each point is

$$Q(x,y) = \frac{1}{2\Pi} \exp^{-(x^2 + y^2)/2} \quad (15)$$

If $\{H_i\}$ is the orthogonal bias and $a_i = \langle a, H_i \rangle$ and $S_i = \langle S, H_i. \rangle$ Then the total error is the approximate of a by S is given by

$$||a - S||^2 = \sum_{i=0}^{\infty} (a_i - S_i)^2 \quad (16)$$

For an approximation to find an edge based on a limited range of spatial frequencies, the approximated error is:

$$I^2 = \sum_{i=0}^{5} (a_i - S_i)^2 \quad (17)$$

This can be minimized by setting its partial derivatives with respect to the parameters to zero.

2.2.4 Vishvjit S. Nalwa and Thomas O. Binford

According to Nalwa and Binford [14] differential operators are not adequate for edge detection. Although step-edges contain large first derivatives and zero-crossings of second, the mapping is neither one-one, nor onto. The operators that threshold on the first derivative respond to smooth shading. Surface-fitting approach as given by Prewitt,

Haralick and Huckel also has the problem of choosing adequate basis, i.e. a basis which can accurately represent the feature sought to be detected.

Nalwa and Binford [18] have developed a new approach called a variant of the surface fitting approach to detect step edges. There are significant differences from previous approaches:

1. An oriented one-Dimensional surface i.e. a surface constrained to be constant along some direction is used to reduce noise without severely blurring edge as with circularly symmetric smoothing operators.

2. Pixels are not marked as belonging to an edge, but to detect edgels (edgels are linear edge-elements, each characterized by a direction and location).

3. The Blurring effect of the imaging system, which can be approximately modeled by Gaussian convolution is taken into account.

4. An adequate basis has been found for step-edges, roof edges and line-edges. These are various combinations of tanh function.

5. A step-edgel is detected in a window if the basis constrained to have a step-shape has a better fit (in least square sence) to the data then the relatively unconstrained basis with the same number of parameters.

Outline of The Algorithm for Step-Edge Detection

Following procedure is repeated over the whole image by shifting the window in 1-pixel steps in X and Y directions:

1. Perform least-squares planar-fit to the window using

$$\xi_p = \sum_{x,y=0}^{4} (I[x, y] - (a_0 + a_x x + a_y y))^2 \quad (18)$$

(minimize w.r.t. a_0, a_x, a_y)

Initial Estimate of θ

$$\theta_0 = \tan^{-1}(a_y/a_x) \quad (19)$$

and use the gradient of this fit for initial estimate of direction of variation in the window, assuming that intensity surface is 1-D.

2. Refine the above estimate of direction by fitting 1-D cubic surface with least-square-error criteria using

$$\xi_c = \sum_{x,y=0}^{4} (I[x, y] - (a_0 + a_1 z + a_2 z^2 + a_3 z^3))^2 \quad (20)$$

$$z = x \cos(\theta) + y \sin(\theta) \quad (21)$$

(minimize w.r.t. a_0, a_1, a_2, a_3, and θ)

Calculate the 2, 20 F-Statistic for the planner and cubic fits obtained above. If it is less then the 75% threshold then declare the absence of an edgel.

3. Find the least-square 1-D tanh surface oriented in the direction found in 2 using

$$\xi_r = \sum_{x, y=0}^{4} (I[x, y] - (s \tanh(f[z+p] + k))^2$$

$$z = x \cos(\theta) + y \sin(\theta) \quad (22)$$

(minimize w. r. t. s, p, and k)

The tanh-fit is localized to the nearest 0.1 pixel.

4. Find the least square 1-D quadratic surface oriented in the direction found in 2 using

$$\xi_q = \sum_{x,y=0}^{4} (I[\xi, \psi] - (\alpha_0 + a_1 z + a_2 z^2))^2 \qquad (23)$$

(minimize w.r.t. a_0, a_1, and a_2)

If the least-square-error in this case is less than that for the tanh fit then declare the absence of an edgel.

5. The least-square tanh-fit determines the intensities on two sides of the step and its position in the window. The position of the step-edge is given by the displacement of the tanh term and its intensities are given by the sum and difference of constant term in the basis and the coefficient of the tanh term.

6. Threshold on the step-size determined from above.

CHAPTER III

COMPUTER IMPLEMENTATION DETAILS

The two most widely accepted and used methods, the LoG and Canny's edge detector l the basic method based on first derivative for edge detection are described. Log is a ı directional operator and Canny's edge detector is a directional operator. All the three ɜrators are of gradient type (First type). The description includes reasoning for their ·ivation, advantages and disadvantages.

FIRST DERIVATIVE METHOD:

The method is based on the definition that an edge is a step discontinuity in the image ıction (usually brightness). To find these step discontinuities, take the difference :ween adjacent pixel intensities (first degree approximation to first derivative) along axis and Y-axis.

$$D_x(i, j) = I(i, j) - I(i+1, j) \quad (1)$$
$$D_y(i, j) = I(i, j) - I(i, j+1) \quad (2)$$

D(i, j) is the difference in intensity between the pixels at (i + 1, j) and (i, j) and I(i, s the intensity of pixel at location (i, j) .

The pixel at (i, j) belongs to edge if

$$D(i-1, j) < D(i, j) > D(i+1, j) \quad (3)$$

and

$$D(i, j) > \text{threshold} \quad (4)$$

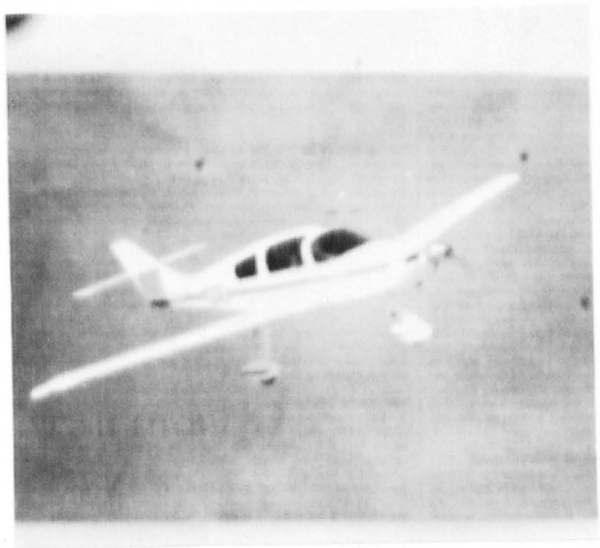
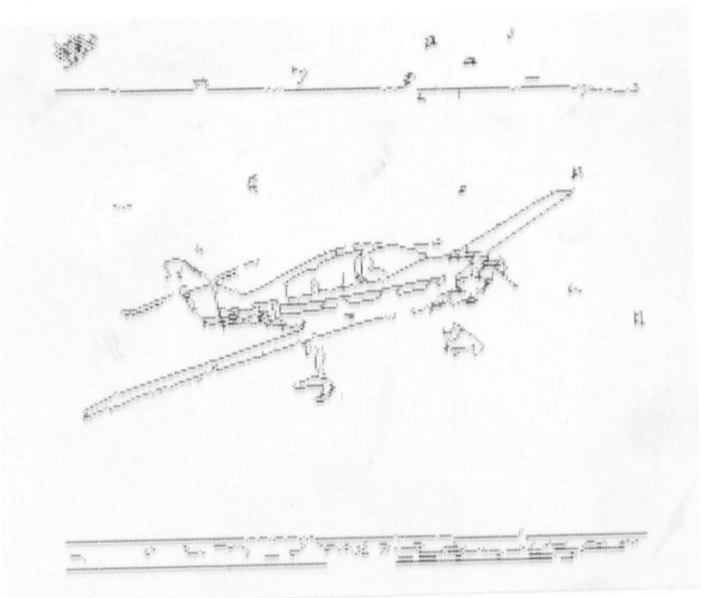
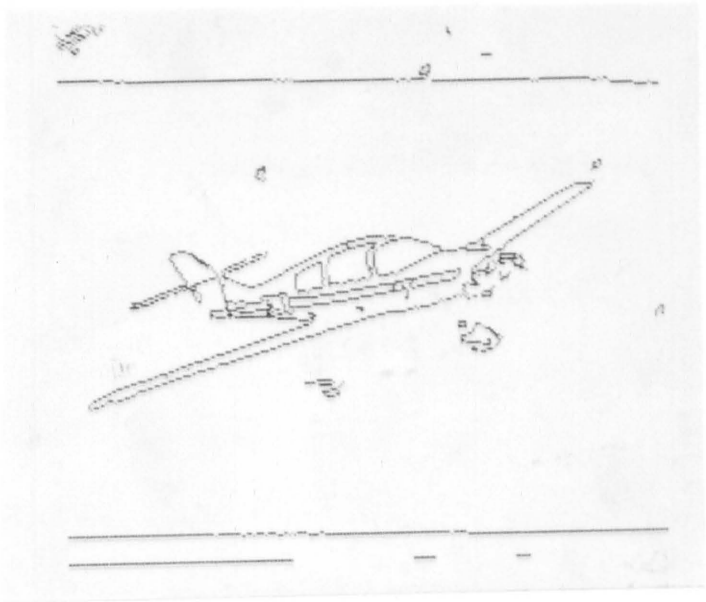


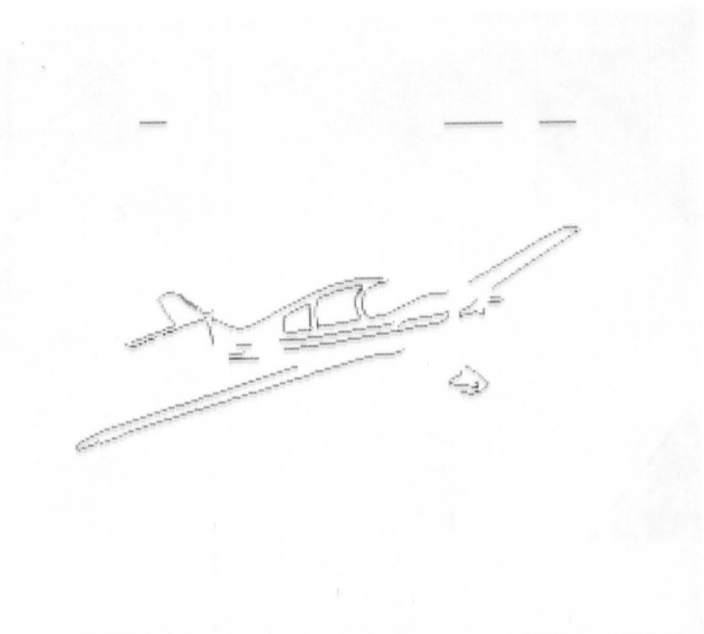
Figure 3.1 Plane image



(a)



(b)



(c)

Figure 3.2 Edges from plane image:
(a) Threshold = 5 and TLEC = 5
(b) Threshold = 15 and TLEC = 5
(c) Threshold = 30 and TlEC = 5
TLEC (Threshold of Length of Edge Contures)

Threshold the results to get the edges. It is found in experiments that the threshold value which gives good results varies from image to image. Lower threshold value gives noisy edges and multiple responses to one edge, but on the other hand the higher threshold eliminates actual edges. This fact is illustrated in Figure 3.1. The results with thresholding the first differences at 10 give multiple responses to the edges as shown in Figure 3.2 (a), if the threshold value is increased to eliminate the multiple responses and noise then some of the true edge are also removed as shown in Figures 3.2 (b) and 3.2 (c). To eliminate the false edges find the length of edge contours using 8-connectivity and threshold the results. The cleaning of output from the above operator is shown in Figure 3.2. This approach works well for simple cases but fails in the presence of large amount of noise. This approach gives output very fast and can be used for rough estimation of edges in many situations. This approach gives very good results if the difference in pixel intensities lies in small interval.

This approach does not sacrifice localization because no averaging is applied to eliminate noise. Detection is not very good as difference in intensity value is taken as fixed which eliminates some true edges and marks some false edges.

3.2 LAPLACIAN OF A GAUSSIAN (LoG) :

In 1980 Marr and Hildreth [11] presented a theory of edge detection, based on the study of human vision system. According to them an optimal filter should fulfill the following requirements:

1. Intensity changes occur at different scales in an image. The filter reduces the range of scales over which intensity changes take place. So the filter should be smooth and band-limited in the frequency domain.

2. The sudden changes in intensity should give rise to peak in the first derivative or zero crossing in the second derivative.

According to Marr and Hildreth human vision system is a non-directional operator, so they chose Laplacian. Gaussian was chosen for blurring the image, because it is smooth and localized both in spatial and frequency domains.

The Gaussian is:

$$G(x, y) = \sigma^2 \exp^{-(x^2+y^2)/2\sigma^2} \tag{5}$$

The Laplacian is:

$$\Delta^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} \tag{6}$$

$$f(x, y, \sigma) = (\Delta^2 g(x, y)) * I(x, y) \tag{7}$$

The convolved image is the result of blurring the image with Gaussian and taking the second derivative (non-directional) using Laplacian.

By law of convolution equation (7) becomes:

$$f(x, y, \sigma) = \Delta^2(G(x, y) * I(x, y)) \tag{8}$$

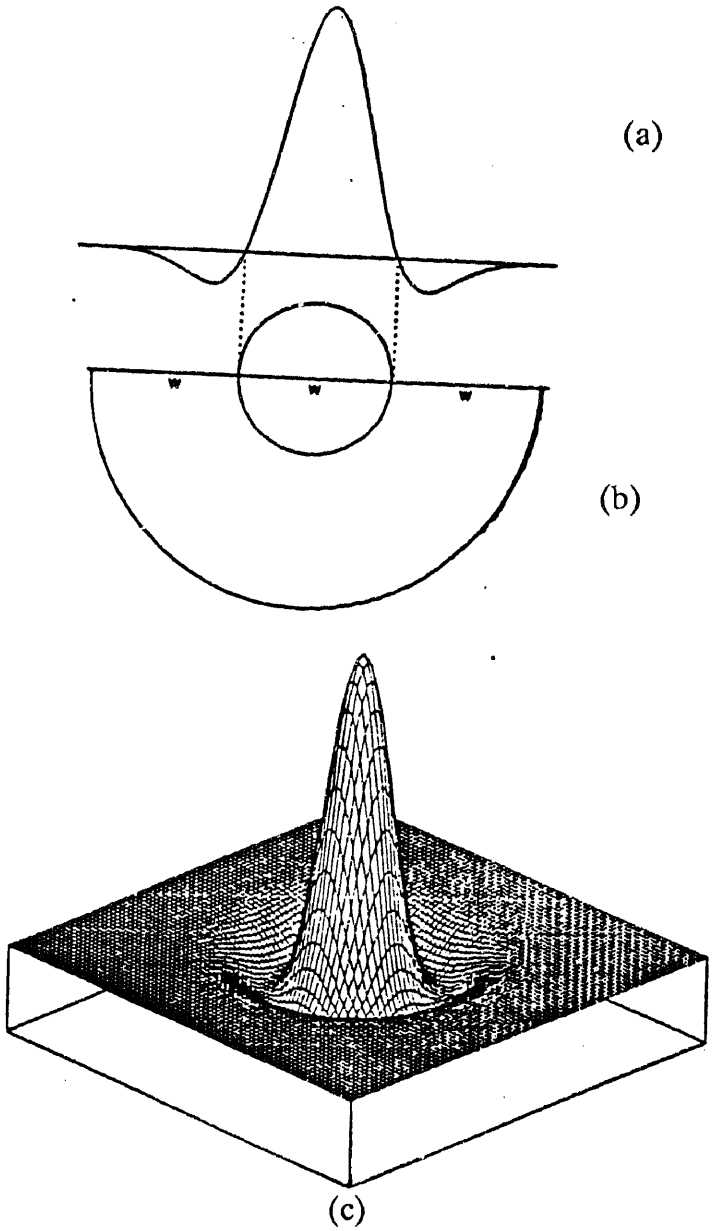


Figure 3.3 The LoG function:
(a) The LoG profile
(b) The LoG as viewed from the top
(c) A 3-D view of LoG

Laplacian of Gaussian can be written as

$$LoG = \Delta^2 G(x, y) = ((x^2+y^2)/\sigma^2 - 2) \exp^{-(x^2 + y^2)/2\sigma^2} \quad (9)$$

This means Gaussian is differentiated first and then applied to the image. This is computationally convenient and simple. The LoG function is shown in Figure(3.3).

The scale of filtering is controlled by the value of σ or alternatively by w which is a function of σ.

$$w = 2\sqrt{2}\sigma \quad (10)$$

As shown in Figure (3.3) "w" is the diameter of inner circle formed by the zero crossing of the LoG.

3.2.1 Finding zero crossings:

The convolution of the image with the LoG gives positive, negative or zero values of each pixel. The pixels which have zero value after convolution are called edge pixels . But the pixels with zero value are rare and the zero crossing occurs between two adjacent pixels having opposite signs. To declare an edge pixel, the pixel having value nearer to zero is chosen, e.g., if two pixels have values -10 and 2 respectively then the pixel with value 2 is chosen as an edge pixel. To find zero crossings the simple scanning of convolved image in horizontal or vertical direction does not work, because if the scan is horizontal then horizontal edge will not be detected. Another problem is that which pixel to mark

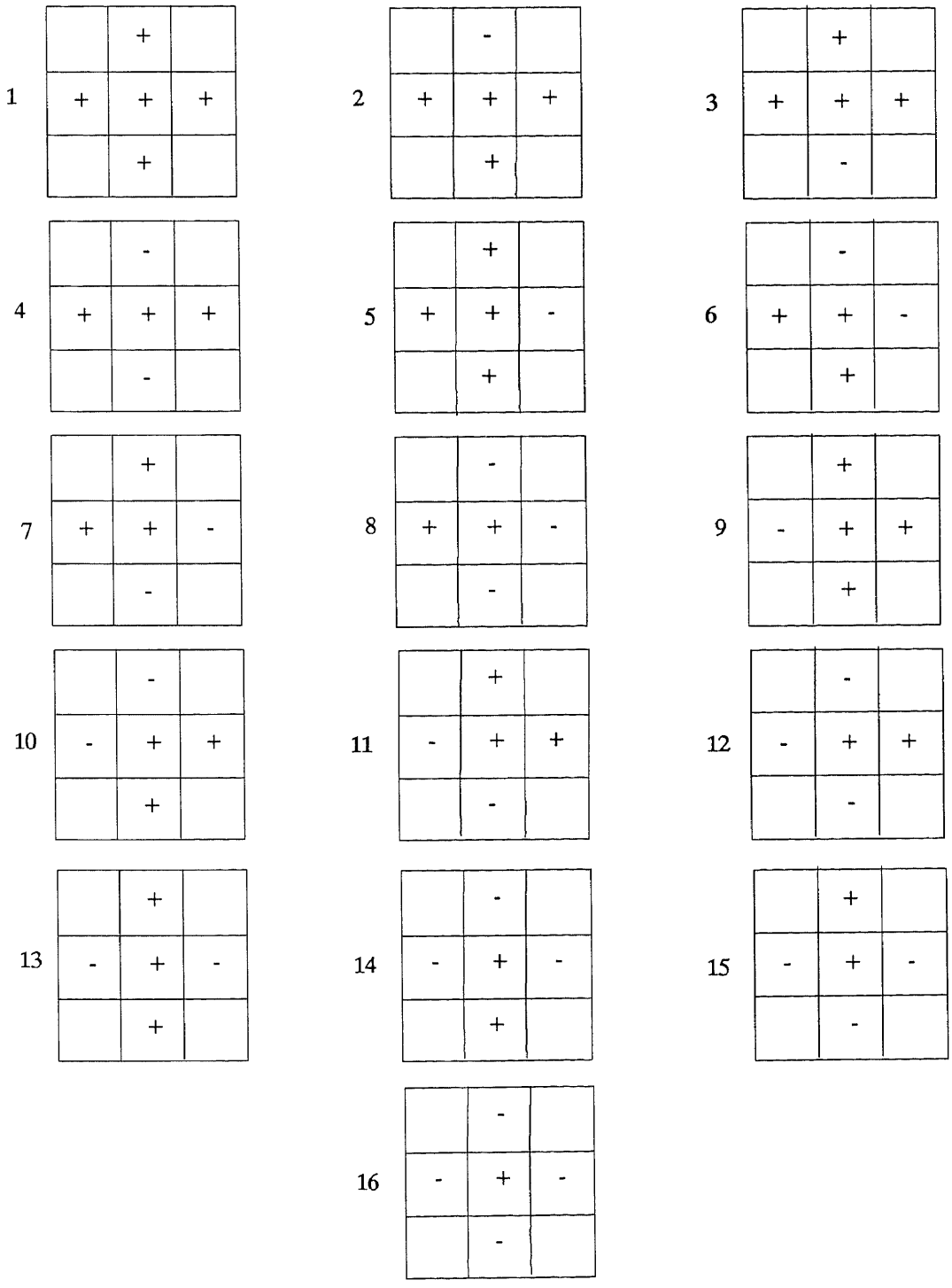


Figure 3.4 Combinations of pixel signs in finding zero crossings

as an edge pixel in case of edges which are neither horizontal nor vertical, in such case information about the neighboring pixels is needed.

To find the most probable pixel as the edge pixel, a method based on 4-connectivity search has been developed [6]. As shown in Figure (3.4), the search is performed in four directions (N, E, S, W). All 16 possible combinations are shown. It is assumed that the center pixel is positive. If the center pixel is zero then this is an edge pixel. In case the center pixel is negative then the signs of all the pixels in the window are reversed. Cases 1 and 16 are trivial. Case 1 shows that there is no edge as there is no change of sign. Case 16 shows an isolated edge, which is ignored due to probability that it is due to noise. The remaining cases can be grouped according to number of candidate pixels present. Cases 2, 3, 5 and 9 have two candidates, cases 4, 6, 7 10, 11 and 13 have three candidates and cases 8, 12, 14 and 15 have four candidates including the center pixel. The pixel with minimum absolute value is marked as an edge pixel.

3.2.2 Properties of the LoG:

The LoG has desirable properties which make it very attractive for machine vision [12]. These can be listed as:

It minimizes the localization in the frequency and spatial domains simultaneously.

The zero crossings from LoG are unique, because no new zero crossings are shown with the increase in σ, but with decrease in σ additional zero crossings may appear due

to the noise present in the image. All zero crossing contours (iso-lines) that appear in the image (excluding those which end at margins of frame) are theoretically closed.

The Gaussian is decreasing about the mean which assigns weights to pixels in filtering process, the weights decrease smoothly with distance from the evaluated pixel.

σ is used as a spatial scalar, a large value of s gives an average signal of image and small d gives almost unsmoothed signal (little filtering).

The LoG is a band pass filter which filters out noise when applied to image. The Log allows obtaining subpixel accuracies by interpolating the zero crossing between two neighboring edge pixels of opposite sign.

3.2.3 Drawbacks of LoG:

The LoG suffers from two major drawbacks. The first is the nonlinear nature of edges. According to Marr and Hildreth [12]:

"Zero crossings of the second directional derivative, in the direction of the gradient, will coincide with the zero crossings of Laplacian of the image if the intensity variation of the image is linear along the line of zero crossings."

This means that, if the edge is not a straight line then zero crossings from the LoG will be displaced. As shown in Figure 3.5 the zero crossing contour at a corner departs from the actual edge.

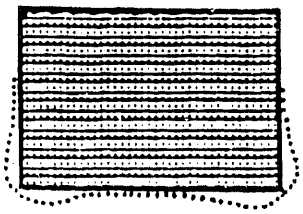
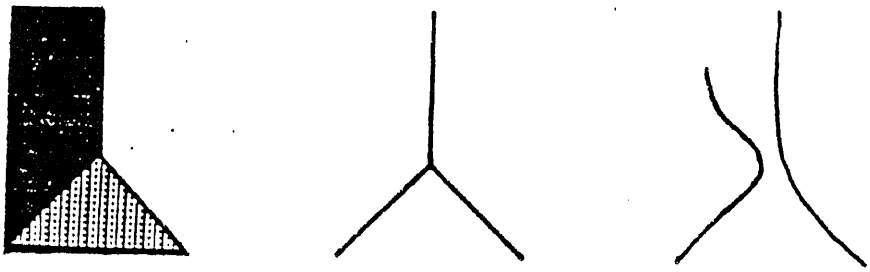


Figure 3.5 Behavior of zero crossings from LoG at a nonlinear edge



(a) (b) (c)

Figure 3.6 Behavior of zero crossing of LoG at a vertex in which odd number of lines converge :

(a) The image
(b) The true edge
(c) Edges as detected using zero crossings of LoG

The second problem is at the vertices at which odd number of edges converge as shown in Figure 3.6. This is because of the property that zero crossings of LoG form a close contours.

These drawbacks are significant only in fine resolution where it is needed to determine the location of the edge accurately.

3.3.4 Computational Considerations:

The amount of calculations involved in convolving the image with LoG is enormous. If the image is an N x N array and window size of LoG is M x M then N x N x M x M multiplications and additions are required. Let image be 256 x 256 array and for $\sigma = 10$, M is 100, then 65.53×10^7 multiplications and additions are needed. Marr and Hildreth [11] suggested the use of difference of two Gaussians (DoG) to approximate Laplacian of Gaussian (LoG).

$$DoG(\sigma_e, \sigma_1) = \frac{1}{2\Pi\sigma_e} \exp^{-x^2/2\sigma_e^2} - \frac{}{2\Pi\sigma_1} \exp^{-x^2/2\sigma_1^2} \qquad (11)$$

It has the Fourier transform

$$DoG(w) = \exp^{-\sigma_e^2 w^2/2} - \exp^{-\sigma_1^2 w^2/2} \qquad (12)$$

The DoG(w) behaves like w^2 for values of w that are small as compared to σ_e and σ_1, so these filters in common with $\Delta^2 G$, approximate second derivative operator.

This equation is for 1-D. For 2-D x is replaced by r where

$$r^2 = x^2 + y^2 \tag{13}$$

The shape pattern of DoG is similar to that of LoG for σ_e and σ_1 to be very close. The advantage of DoG lies in the fact that it can be decomposed into two passes of 1-D Gaussian, one in X direction and other in Y direction. The number of computations reduces from N x N x M x M to N x N x M x 4. So the amount of computation saved is:

$$\frac{N \times N \times M \times M}{N \times N \times M \times 4} = \frac{M}{4} \tag{14}$$

As compared to 65.53×10^7 only 26.21×10^6 multiplications and additions are needed, saving 62.91×10^7 calculations i.e. 95% of calculation load.

Medioni and Nevatia [12] has developed another method of decomposition of LoG itself as shown below:

$$\Delta^2 G(x, y) = h_{12}(x) + h_{21}(x,y) \tag{15}$$

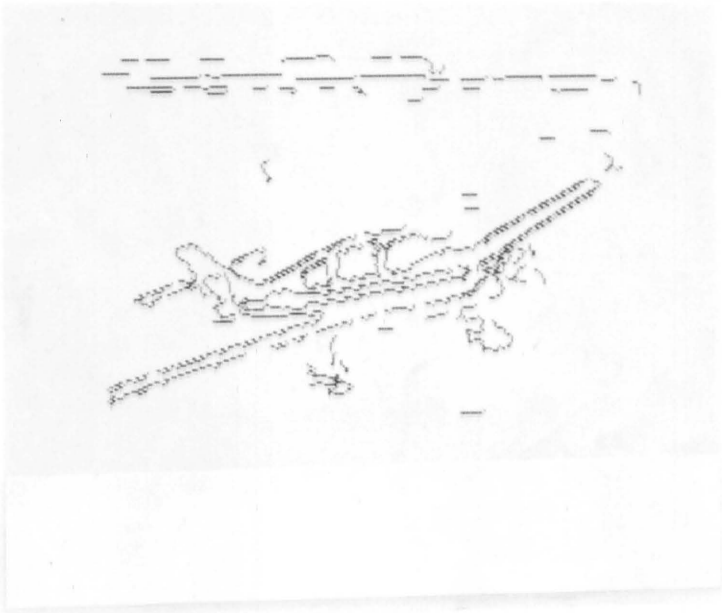
where:

$$h_{12}(x, y) = h_1(x).\, h_2(y) \tag{16}$$
$$h_{21}(x, y) = h_2(x).\, h_1(y) \tag{17}$$

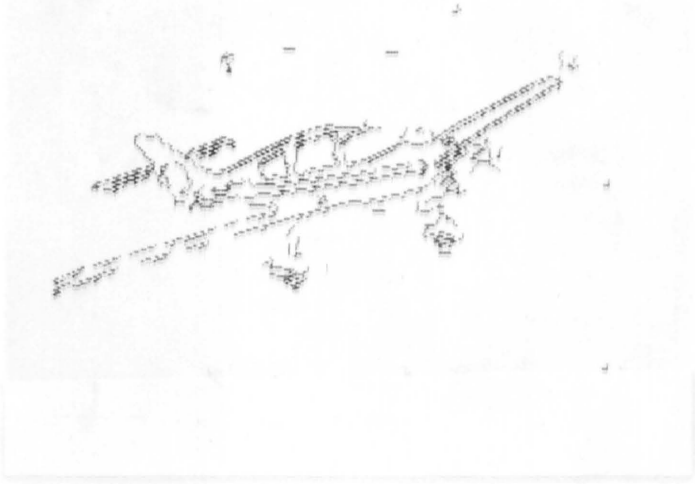
and

$$h_1(z) = (1 - z^2/\sigma^2)\exp^{-z^2/2\sigma^2} \tag{18}$$

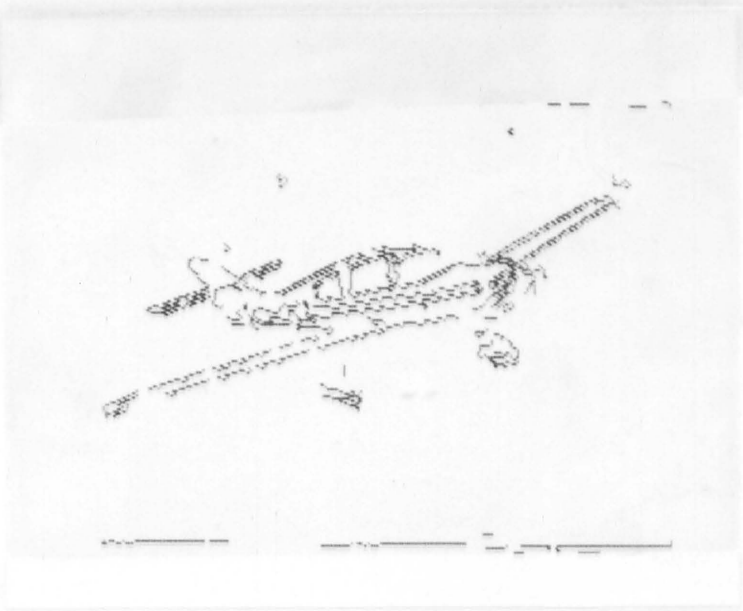
$$h_2(z) = \exp^{-z^2/2\sigma^2} \tag{19}$$



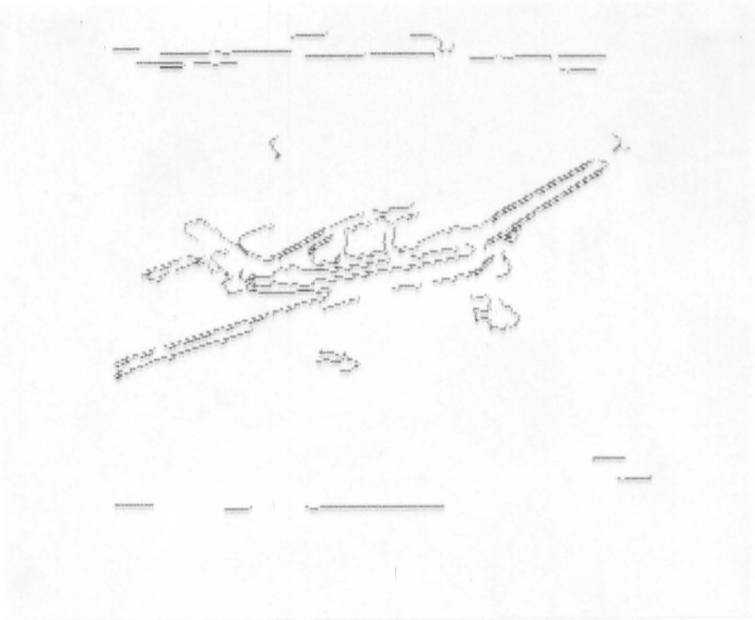
(a)



(b)



(c)



(d)

Figure 3.7 Edges from plane image using LoG :
(a) σ = 1.0 and TlEC = 3
(b) σ = 1.5 and TlEC = 3
(c) σ = 2.0 and TlEC = 3
(d) σ = 3.0 and TlEC = 3

This method also needs 4 passes on a given image. The advantage of this over DoG is that only one standard deviation is variable in the filter.

Experiments are conducted with modified LoG. The results from zero crossings are cleaned by eight connectivity method (described in chapter 4) used to find the length of edge contours from zero crossings and the results are thresholded to eliminate the noisy edges. The experiments show that with the increase in blur factor (σ) the noise is reduced but the true edges are also sacrificed due to more averaging as illustrated in Figure 3.7. As shown in Figure 3.7 (a) the edges found with $\sigma = 1.0$ contain noise, but as the value of σ is increased to eliminate the noise, some of the true edges are also removed as shown in Figures 3.7 (b), 3.6 (c), and 3.6 (d). With the increase in the value of σ, localization is also reduced due to averaging over a large number of pixels.

3.3 CANNY'S EDGE DETECTOR :

Based on pure mathematical considerations, Canny [3] has formulated an edge detector using three performance criteria for an ideal edge detector. He assumes that detection is performed by convolving the noisy edge with a spatial function f(x) and by marking edges at the maxima in the output of this convolution. The three criteria are:

1. Good detection. There should be low probability of failing to mark real edges, and low probability of falsely marking non-edge points. This criterion corresponds to maximizing the signal to noise ratio (SNR).

2. Good localization. The points marked as edge points by the edge operator should be as close as possible to the center of the true edge.

3. Only one response to a single edge.

3.3.1Detection and Localization Criteria

Let the impulse response of the filter be denoted by f(x) and edge by G(x). If the edge is centered at x = 0, then the response of the filter at its center H_G is given by

$$H_G = \int_{-W}^{+W} G(-x)f(x)dx \tag{20}$$

Let the filter have a finite impulse response and be bounded by [-W, W], then the root-mean-square response to noise n(x) is

$$H_n = n_0 \left| \int_{-W}^{+W} f^2(x)dx \right|^{1/2} \tag{21}$$

where n_0^2 is the mean-square noise amplitude per unit length. The first criterion, the output signal-to-noise ratio is given by

$$SNR = \frac{\left| \int_{-W}^{+W} G(-x)\, f(x)\, dx \right|}{n_0 \left| \int_{-W}^{+W} f^2(x)dx \right|^{1/2}} \tag{22}$$

Let $H_n(x)$ be the response of the filter to noise only, and $H_G(x)$ to the edge, and let there be a local maximum in the total response at the point $x = x_0$, then

$$H'n(x_0) + H'G(x_0) = 0 \tag{23}$$

The Taylor expansion of $H'G(x0)$ about the origin gives

$$H'G(x_0) = H'G(0) + H''G(0)x_0 + O(x_0^2) \tag{24}$$

$H_G(x)$ is always symmetric and its derivatives of odd orders are 0 at the origin. The above two equations give

$$H''G(0)x_0 \cong -H'n(x_0) \tag{25}$$

$H'n(x_0)$ is a Gaussian random quantity whose variance is the mean-squared of $H'n(x_0)$, and is given by

$$E[H'n(x0)^2] = n_0^2 \int_{-W}^{+W} f'^2(x)dx \tag{27}$$

Substituting the value of Hn'(x0) and H_G"(0) gives

$$E[x_0^2] = \frac{n_0^2 \int_{-W}^{+W} f'^2(x)dx}{\left[\int_{-W}^{+W} G'(-x)f'(x)dx\right]^2} = \delta x_0^2 \tag{27}$$

where δx_0 is an approximation to the standard deviation of x_0. The localization is defined as the reciprocal of x_0.

$$\text{Localization} = \frac{\left| \int_{-W}^{+W} G'(x)f'(x)dx \right|}{n_0 \left[\int_{-W}^{+W} f'^2(x)dx \right]^{1/2}} \tag{28}$$

Equations (20) and (26) are mathematical forms for the two criteria, and the problem is to maximize both simultaneously. To do so, maximize the product of both i.e. maximize

$$\frac{\left| \int_{-W}^{+W} G(-x)f(x)dx \right| \left| \int_{-W}^{+W} G'(-x)f'(x)dx \right|}{n_0 \left[\int_{-W}^{+W} f^2(x)dx \right]^{1/2} n_0 \left[\int_{-W}^{+W} f'^2(x)dx \right]^{1/2}} \tag{29}$$

3.3.2 Eliminating Multiple Response

The above criterion measures the effectiveness of the filter in discriminating between signal and noise at the center of an edge. It does not consider the behavior of the filter nearby the edge center. The first two criteria can be maximized as follows:

From Schwarz inequality for integrals SNR (18) is bounded above by

$$n_0^{-1} \left[\int_{-W}^{+W} G^2(x)dx \right]^{1/2} \tag{30}$$

and localization (24) by

$$n_0^{-1} \left[\int_{-W}^{+W} G'(x)dx \right]^{1/2} \tag{31}$$

Both bounds are attained and the product SNR and localization is maximized when

$$f(x) = G(-x) \tag{32}$$

in [-W, W]

The mean distance between zero-crossings of f' is

$$x_{zc}(f) = \left[\frac{\int_{-\infty}^{+\infty} f'^2(x)dx}{\int_{-\infty}^{+\infty} f''^2(x)dx} \right]^{1/2} \tag{33}$$

The distance between adjacent maxima in the noise response of f, denoted by xmax, will be twice xzc. This can be set as some fraction of the operator width W.

$$x_{max}(f) = 2x_{zc}(f) = kW \tag{34}$$

Expected number of noise maxima in the region of width 2W is N_n where

$$N_n = \frac{2W}{x_{max}} = \frac{2}{k} \tag{35}$$

Fixing k fixes the number of noise maxima that could lead to a false response.

3.3.3 Optimal Detector

This is done through numerical optimization by finding f which maximizes

$$SNR(f) * Localization(f) - \mu_i P_i(f) \tag{36}$$

where P_i is a function which has a positive value only when a constraint is violated. The larger the value of $_i$ the more nearly the constraints are satisfied. The required function can be approximated by first derivative of Gaussian G'(x), where

$$G(x) = \exp^{-x^2/2\sigma^2} \tag{37}$$

The impulse response of the first derivative filter is

$$f(x) = -(x/^2)\exp^{-x^2/2\sigma^2} \tag{38}$$

For 2-D Gaussian is given by

$$G = \exp^{-(x^2+y^2)/2\sigma^2} \tag{39}$$

3.3.4 Implementation

The general outline of Canny's edge detector is as follows :

1. Convolve the image with a symmetric Gaussian.

2. Take the first directional derivative of the convolved image.

3. Perform a "non-maxima suppress" in the direction of the gradient.

4. Threshold the results from step 3.

5. Perform a fine to coarse comparison.

The first step is to convolve the image with a symmetric Gaussian. The convolution can be performed very efficiently by using the decomposition property of the Gaussian. The decomposition property, is to perform two 1-D passes on the image instead of a full 2-D pass. The decomposition is of their impulse responses into independent linear filters

$$f(x, y) = fx(x, 0) * fy(0, y) \quad (40)$$

The decomposition property of the Gaussian is expressed mathematically by :

$$f * I = [fx * fy] * I = [I * fx] * fy \quad (40)$$

where f(x,y) is the 2-D Gaussian function,

fx, fy are 1-D Gaussian in x and y directions,

I(x,y) is the image function.

The decomposition means that the image is first convolved in the "y" direction only and output of it is then convolved in the "x" direction. For a given convolution mask of MxM, the computation load is reduced from M^2 to 2M.

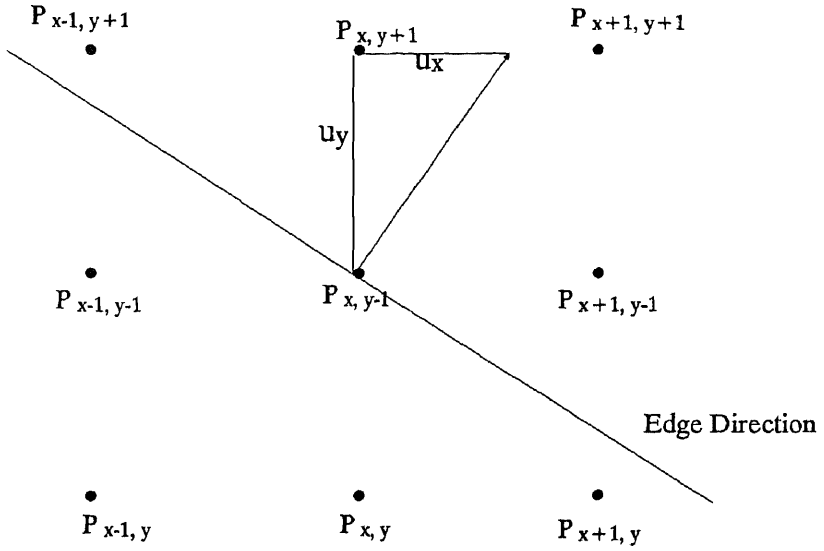


Figure 3.8 The non-maxima suppression operator.

The second step in Canny's edge detector is to take first directional derivative of the filtered image. He has suggested a mask with 30^0 interval in orientation i.e. six masks over the entire image. Sobel operator can also be used for this purpose.

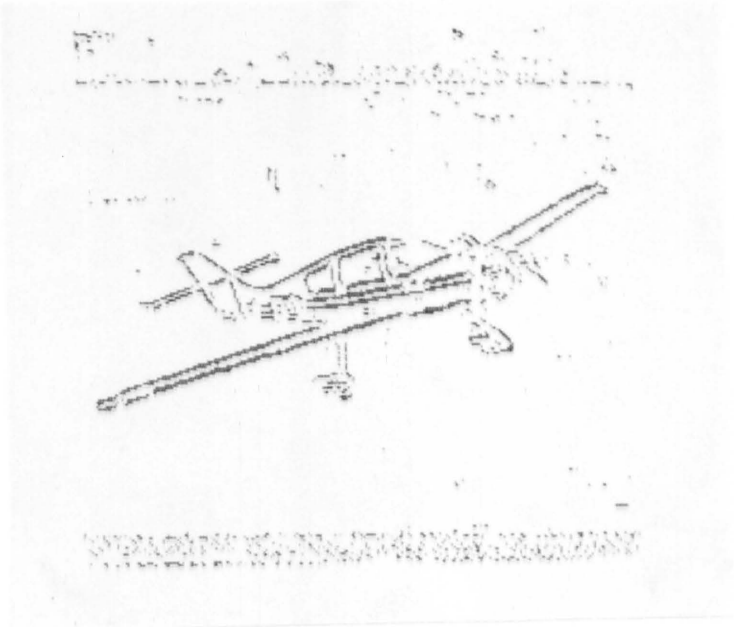
The third step of "non-maxima suppress" is described in Figure 3.8. At each point the value of the gradient perpendicular to the edge direction is interpolated. A pixel is marked as an edge pixel if and only if its gradient magnitude is larger than G1 and G2 where G1 is the interpolated gradient between Px,y + 1 and Px + 1,y + 1 and is given by

$$G_1 = \frac{u_x}{u_y} G(x+1, y+1) + \frac{u_y - u_x}{u_y} G(x, y+1) \qquad (43)$$

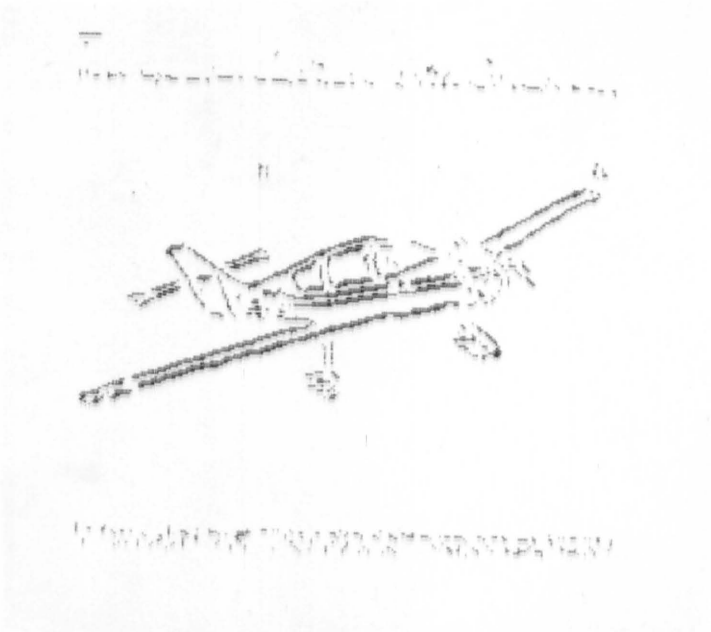
Similarly G2 is interpolated gradient on the opposite side of Px,y and is given by

$$G_2 = \frac{u_x}{u_y} G(x-1, y-1) + \frac{u_y - u_x}{u_y} G(x, y-1) \qquad (43)$$

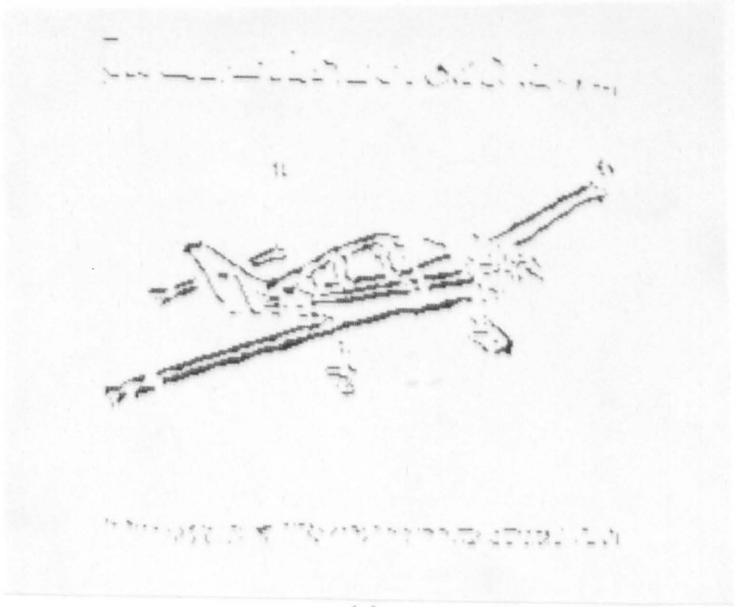
The fourth step is thresholding the edges marked in the previous step to eliminate noisy edges. The thresholding is done with hysteresis as suggested by Canny. If any part of a contour is above a high threshold, that point is immediately output, as is the entire connected segment of the contour which contains the point and which lies above a low threshold. The probability of streaking is greatly reduced because for a contour to be broken it must now fluctuate above the high threshold and below the low threshold. Also the probability of false edges is reduced because the high threshold can be raised without risking streaking. The ratio of the high to low threshold is usually in the range two or three to one.



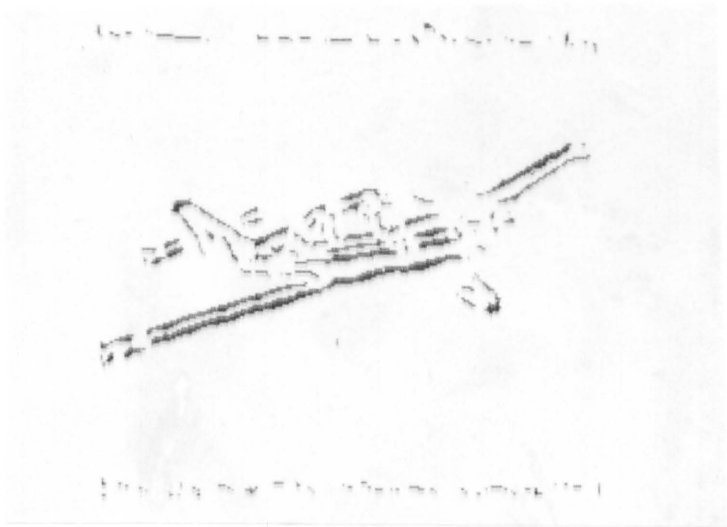
(a)



(b)

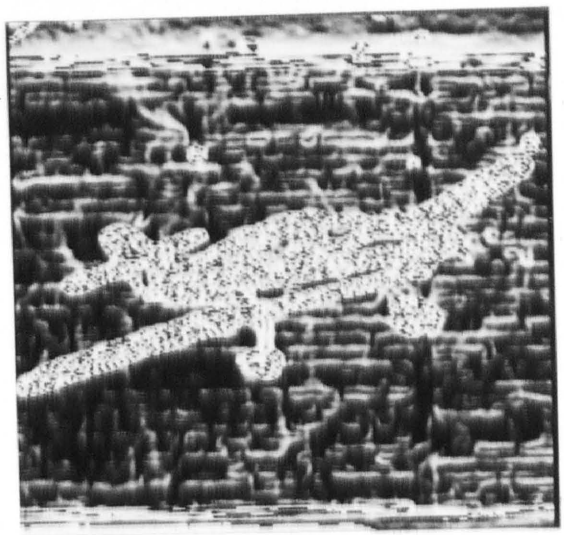


(c)

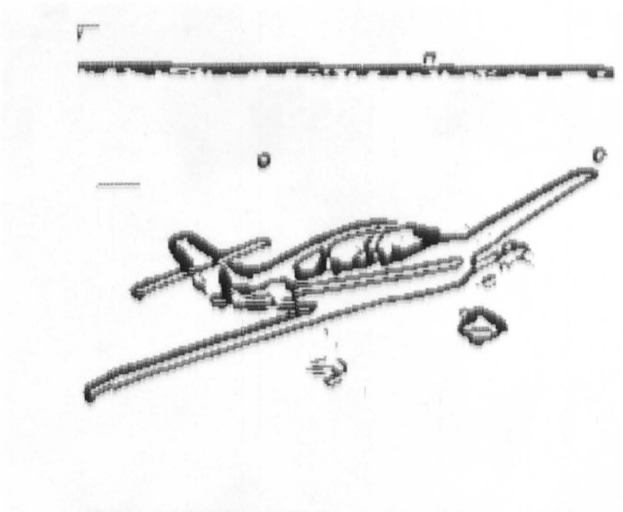


(d)

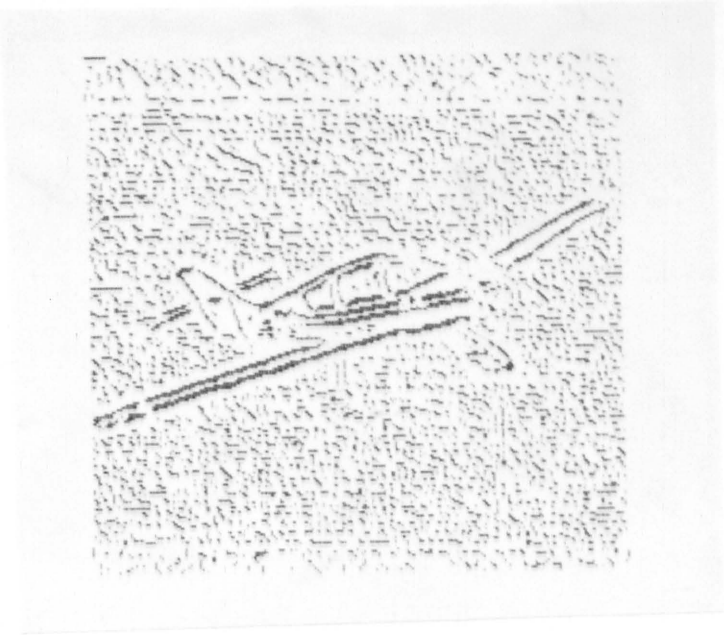
Figure 3.9 Edges from plane image using Canny's method :
(a) σ = 0.5 and TFD = 250
(b) σ = 1.0 and TFD = 250
(c) σ = 1.5 and TFD = 250
(d) σ = 3.0 and TFD = 250
TFD (Threshold of first derivative results)



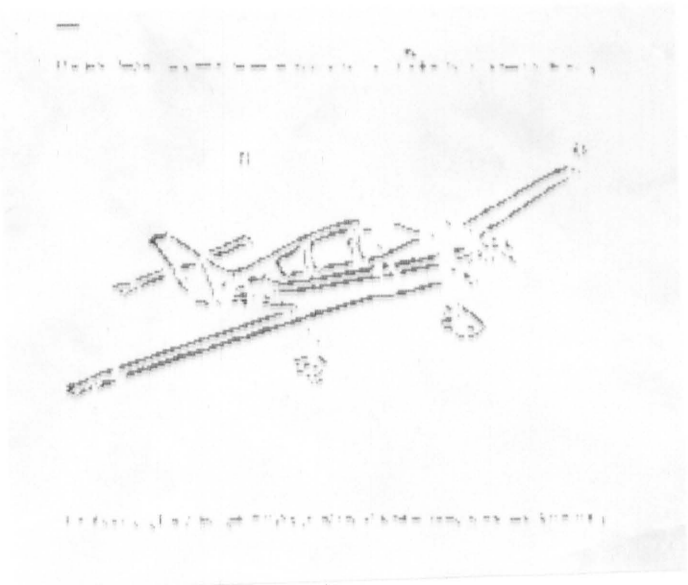
(a)



(b)



(c)



(d)

Figure 3.2 Effect of non-maxima supperssion with $\sigma = 1.0$:
(a) Result after first derivative
(b) Edges from maxima of first derivative
(c) Edges from non-maxima suppression
(d)Edges from non-maxima supperssion
and thresholding

The last step of Canny's edge detector is fine to coarse strategy. The main problem in implementing a coarse to fine strategy for this edge detector is that unlike the LoG, the property of continuity of the edge at different resolutions does not exist. Edges detected at one spatial scale may disappear at a smaller one. To overcome this problem, Canny devised a fine to coarse strategy which is contrary to the coarse to fine strategy suggested by Marr and Poggio in their stereo vision theory. It is also contrary to our intuition that first we look for large pronounced elements in the image and then incrementally examine the fine details. In fine to coarse comparison, edges are first found by convolving the image with a Gaussian of a small spatial scale (small standard deviation). Then, a predicted output for the coarser resolution is synthesized (by convolving the output with a larger Gaussian) and compared to the actual output obtained by convolving the image with the larger Gaussian. Additional edges are marked only if they show significantly greater responses in the actual output, in comparison to that predicted from the synthesis.

Experiments are conducted with different values of blur factor (σ). With the increase in blur factor (σ) noise is reduced but the true edges are also eliminated after a certain value of σ. The value of blur factor depends on the brightness of the image and noise present in the image. The effect of σ is illustrated in Figure 3.8., which shows that as the value of σ is increased from 0.5 to 3.0 the noise is reduced but some of the actual edges are also removed. The increase in the value of σ also reduces localization due to averaging over a large number of pixels. The non-maxima suppression and thresholding cleans the results from the first derivative maxima very efficiently as shown in Figure 3.7. The edges

from the maxima of first derivative contain large amount of noise and multiple responses to edges as shown in Figure 3.8 (b). The non-maxima suppression gives fine edge contours, but lot of false edge (edges due to noise) contours also as shown in Figure 3.8 (c) , these false edges are removed by thresholding the results from non-maxima suppression as shown in Figure 3.8 (d).

CHAPTER IV

EIGENVECTOR EDGE FITTING

In this chapter a new approach for edge detection is described. The approach is based on the definition that an edge is a discontinuity in the image function. In the previous methods some type of filtering (Gaussian or average) is applied to eliminate noise. Two most important performance criteria that an edge detector has to satisfy are, good detection and localization. There is a trade-off between these two criteria i.e. broad operators have good signal to noise ratio but poor localization and vice-versa. Any filter applied to reduce noise decreases localization of the edges to be detected in the next step. In this approach some localization is suffered while detecting edges (discontinuities in image), because eigenvector line is fitted to set of pixels which is like simple averaging. The localization suffers less in this approach because number of pixels taken to fit eigen vector lines is less than the convolution mask size used in other methods to eliminate noise . To eliminate rest of noise (false edges) any kind of averaging filter is not used, but rather noise is eliminated using the fact that noise (false edges) contours are small. For this purpose edge contours are traced using 8-connectivity and results are thresholded based on the length of contours. Thus by doing so we sacrifice very less localization and eliminate most of noise. This approach is described in detail the following section.

4.1 EIGEN VECTOR APPROACH:

This approach is also based on step discontinuity in intensity of image, the fundamental of edge detectors. The pixel location is taken along one axis and intensity value along

the other to find the best eigenvector line fit. The eigenvector line fitting can be described as follows:

The best fit line to a set of points is the one which minimizes the sum of the squares of the perpendicular distances from the points to the line. The best fit line is the unique line through the mean of set points and parallel to the eigen vector of the scatter matrix of those points.

The algorithm for finding eigenvector line fitting is:

1. Find the mean of the set of points.

$$\text{Mean} = \frac{\sum_{i=1}^{n} N_i}{n} \tag{1}$$

where n is number of points and N_i are the points.

2. Standardize the points by subtracting the mean the set from each point.

3. Calculate the scatter matrix. Consider a set of n points with zero mean. Denote ith point (x_i, y_i) as the vector v_i, and the perpendicular distance from v_i to line as d_i, then the task is to find the line minimizing

$$d^2 = \sum_{i=1}^{n} d_i^2 \tag{2}$$

Let the line be characterized by its unit normal vector N through the origin. Then explicitly denoting dependence on N, we get

$$d_i^2(N) = (N \cdot v_i)^2 = (N^t v_i)^2 \tag{3}$$

and

$$\begin{aligned} d^2(N) &= \sum_{i=1}^{n} d_i^2(N) \\ &= \sum_{i=1}^{n} (N^t v_i)^2 \\ &= \sum_{i=1}^{n} (N^t v_i)(v_i^t N) \\ &= \sum_{i=1}^{n} N^t (v_i v_i^t) N \end{aligned} \tag{4}$$

where

$$s = \sum_{i=1}^{n} v_i v_i^t \tag{5}$$

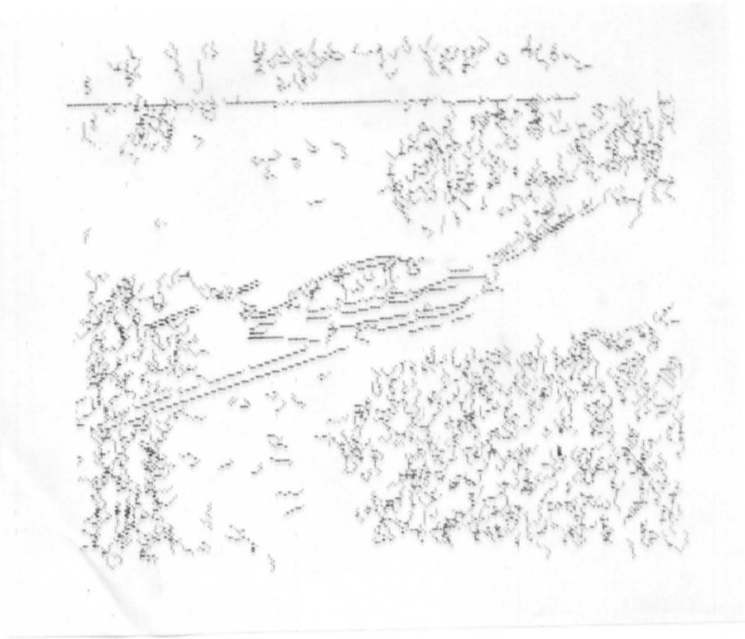
is the scatter matrix of given n points.

4. Find the principal eigenvector of this scatter matrix.

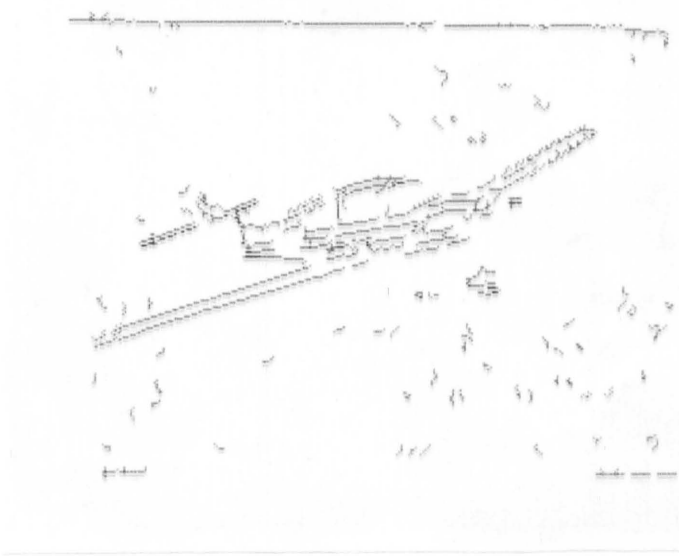
5. Find the best fit line which is the unique line through the mean and parallel to the eigenvector. This best fit line is characterized by the unit normal vector N that minimizes

$$d^2(N) = N^t S N. \tag{6}$$

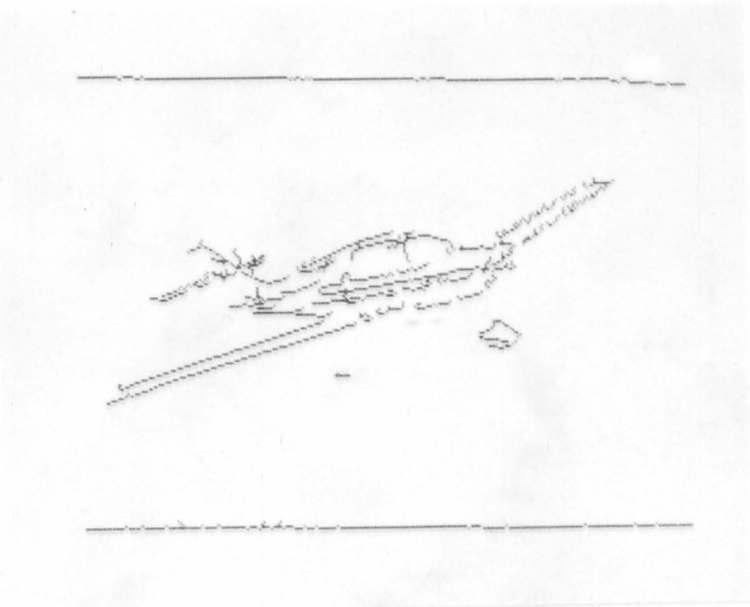
The major advantage of eigenvector line fitting is that it is not dependent on the choice of axis whereas the least squared line fitting is dependent on choice of the axis.



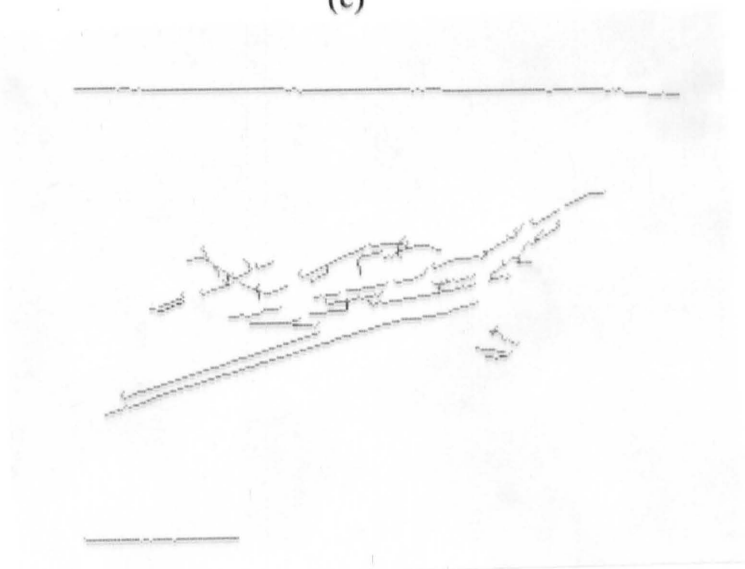
(a)



(b)



(c)



(d)

Figure 4.1 Edges from plane image using Eigen Vector method :
(a) No. of pixels = 3, TD = 30 and TLEC = 5
(b) No. of pixels = 5, TD = 30 and TLEC = 3
(c) No. of pixels = 7, TD = 30 and TLEC = 3
(d) No. of pixels = 10, TD = 20 and TLEC = 3
TD (Threshold of distance)

4.1.1 Implementation:

Implementation of Eigen Vector method can be described as follows:

1.Using above algorithm fit eigenvector lines to intensity and position of pixels along X-axis, taking n pixels at a time.

2. Find distances Dx(i) between two adjacent lines at common point i.

3.To find possible edge points Dx(i) should satisfy:

$$Dx(i-1) < Dx(i) > Dx(i+1) \quad (7)$$

4. Threshold the results to get edges. The pixel (i) is a edge pixel if

$$Dx(i) > TD \quad (8)$$

5. To eliminate noise use 8-connectivity (described in the later part of the chapter) to find length of edge contours and apply threshold to length of edge contours.

The value of threshold in step 4 dependes on the intensity level of the image. Higher threshold gives less noise but also eliminates some of edges as shown in Figure (41).Also larger value of threshold in step 5 results in more the noise eliminatation but poorer localization. Threshold value used for length of edge contours varies from 3 to 15 based on the image . This approach needs further development to get good results.

4.2 DESCRIPTION OF EIGHT CONNECTIVITY ALGORITHM :

The algorithm used to trace contours to find their lengths based on eight connectivity is descrived by Jadhav [9] and is as follows :

1. Mark the edge pixels found from our methods as intensity value 100.

2. Scan the edge data to find the first pixel whose intensity is 100. Change the intensity of starting pixel to 250 and label it as #0.

3. Check if pixel #1 belongs to the contour. If then change its intensity to 250 and label it as #0 and go to step 8. If pixel #1 does not belong to contour then continue.

4. Check if pixel #2 belongs to the contour. If then change its intensity to 250 and label it as #0 and go to step 9. If pixel #1 does not belong to contour then continue.

5. Check if pixel #3 belongs to the contour. If then change its intensity to 250 and label it as #0 and go to step 10. If pixel #1 does not belong to contour then continue.

6. Check if pixel #4 belongs to the contour. If then change its intensity to 250 and label it as #0 and go to step 3. If pixel #1 does not belong to contour then continue.

7. Check if pixel #5 belongs to the contour. If then change its intensity to 250 and label it as #0 and go to step 4. If pixel #1 does not belong to contour then continue.

8. Check if pixel #6 belongs to the contour. If then change its intensity to 250 and label it as #0 and go to step 5. If pixel #1 does not belong to contour then continue.

9. Check if pixel #7 belongs to the contour. If then change its intensity to 250 and label it as #0 and go to step 6. If pixel #1 does not belong to contour then continue.

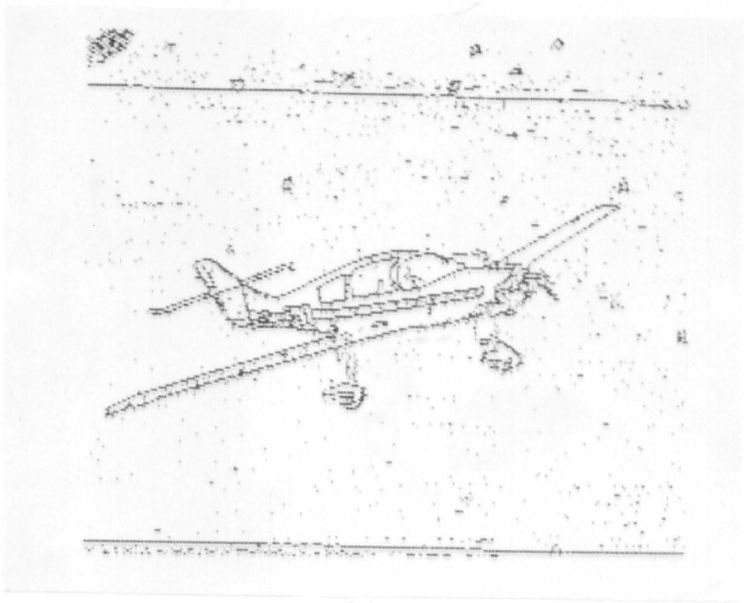
10. Check if pixel #8 belongs to the contour. If then change its intensity to 250 and label it as #0 and go to step 7. If pixel #1 does not belong to contour then continue.

11. Execute steps 3 to 10 till a pixel of intensity 100 is found.

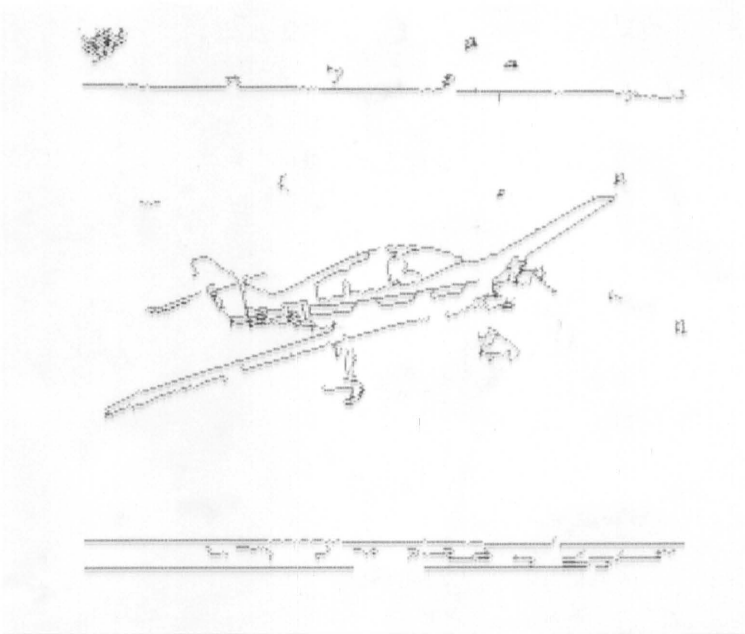
12. A count is kept of the number of edge pixels thus determined along with their coordinates and connectivity according to the chain code as shown in Figure (4.4) .

4.2.1 Description of the above algorithm:

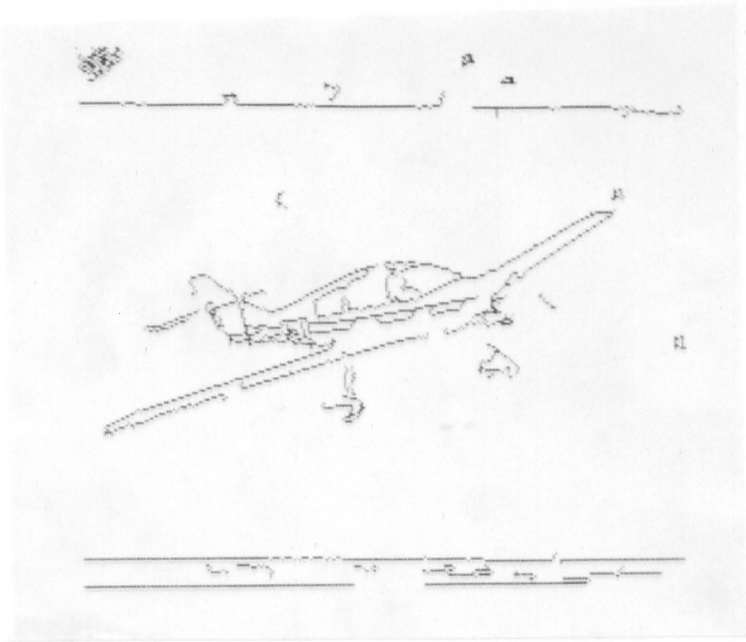
Edge data is scanned and first starting pixel with value 100 is found and its intensity is changed to 250. This pixel is labelled as zero. Since the scanning is from left to right, the pixel immediately to the left of the this starting pixel has already been checked. This checking process is performed in a clockwise manner. The intensity of pixel #1 is read and checked if it is 100. If not the intensity of pixel #2 is checked and then pixel #3 and so on till a pixel with an intensity value of 100 is found. If such a pixel is found then its position is noted and intensity value is changed to 250. Now this pixel which is noted as second point of contour is labelled as #0 and its eight pixel neighborhood is checked as described above. A count is kept for the number of pixels found. In this manner search is continued until all the contours are traced i.e. no pixel with a value of 100 is left. The chain code used for above algorithm is shown in Figure 4.4.



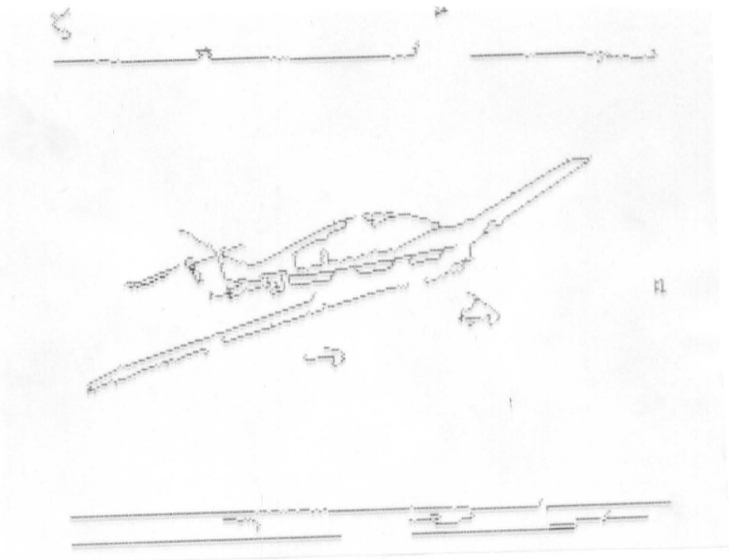
(a)



(b)

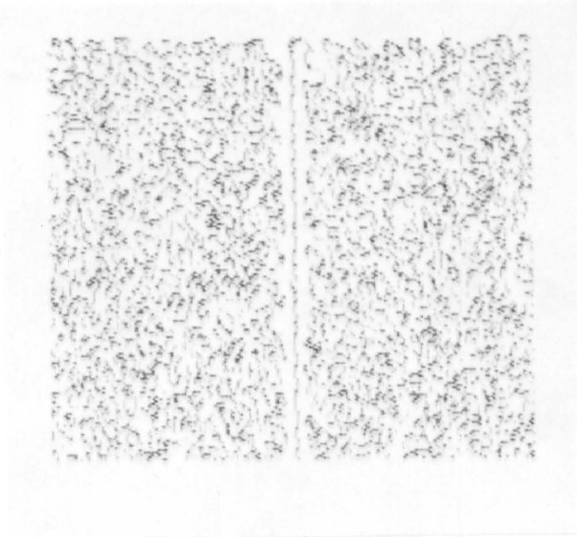


(c)



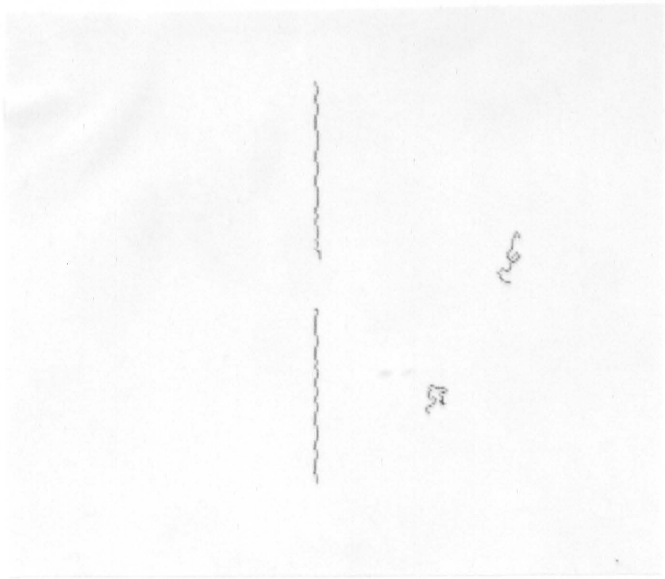
(d)

Figure 4.2 Effect of cleaning using eight connectivity :
(a) Edges from First Derivative method without cleaning
Results after cleaning :
(b) TLEC = 3
(c) TLEC = 5
(d) TLEC = 11

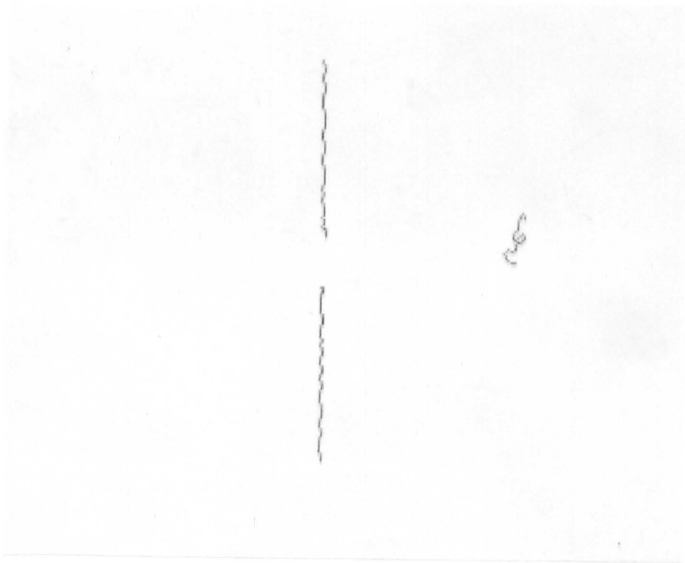


(a)

(b)



(c)



(d)

Figure 4.3 Effect of cleaning using eight connectivity :
(a) Edges from LoG without cleaning
Results after cleaning :
(b) TLEC = 10
(c) TLEC = 20
(d) TLEC = 30

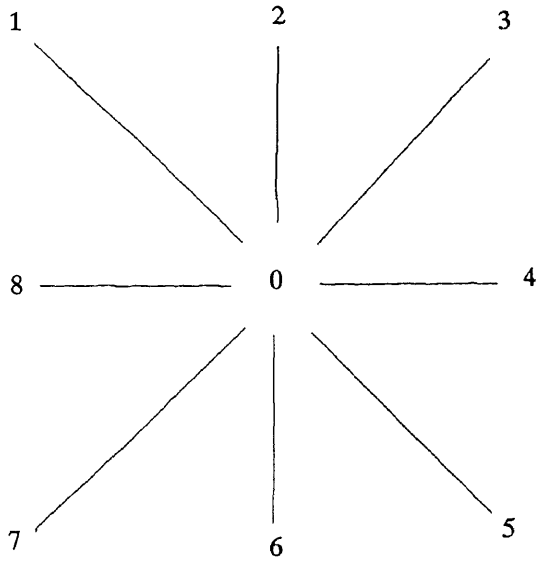


Figure 4.4 Chain code

The cleaning operation of this method is shown in Figure (4.3). The threshold value for length of edge contours is dependent on different situations. Large value of threshold eliminates the noise but also eliminates true edge and on the other hand lower value of threshold gives false edge (edges due to noise) as illustrated in Figure (4.4).

CHAPTER V

RESULTS AND CONCLUSION

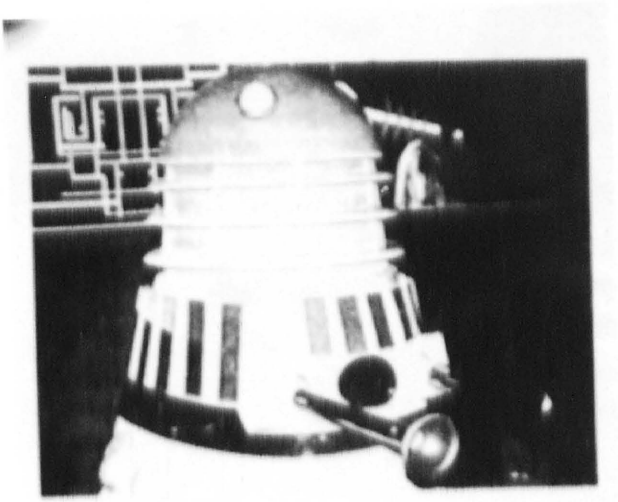
5.1RESULTS:

All the approaches described in the previous chapter, namely, Canny's approach, LoG, First Derivative and Eigenvector line fitting, are applied on several images. The results are shown in Figures 5.1 to 5.5. Generally speaking, conclusions regarding the performance of these methods can be made by observing the results. The conclusions are listed as follows:

The images in Figures 5.1, 5.2 and 5.3 can be considered similar because the intensity variation in all these cases is smooth as the images are of curved or rounded objects. It is observed that the results form Canney's method and First derivative method are better than the other two methods. The following arguments may be made to explain such results:

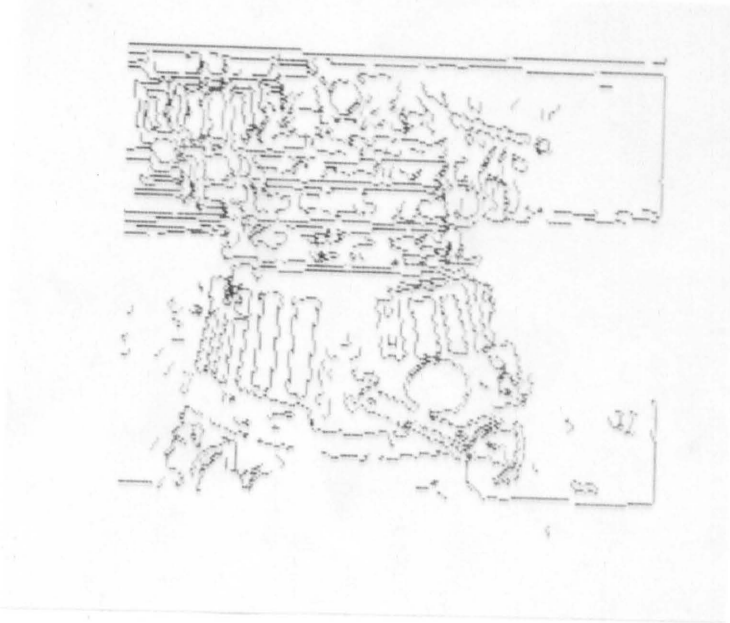
In Canny's method there is a less sacrifice of localization due to averaging (convolution with Gaussian) than for Log because of a smaller convolution mask size for Canney's method.

The results from First Derivative method are even more clear then Canny's method because intensity variation at edges is approximately constant and no localization is suffered because no averaging is applied.

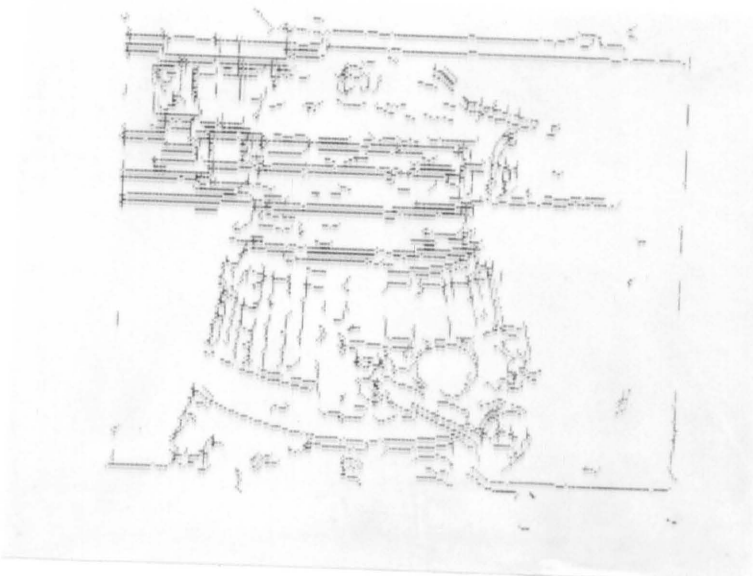


(a)

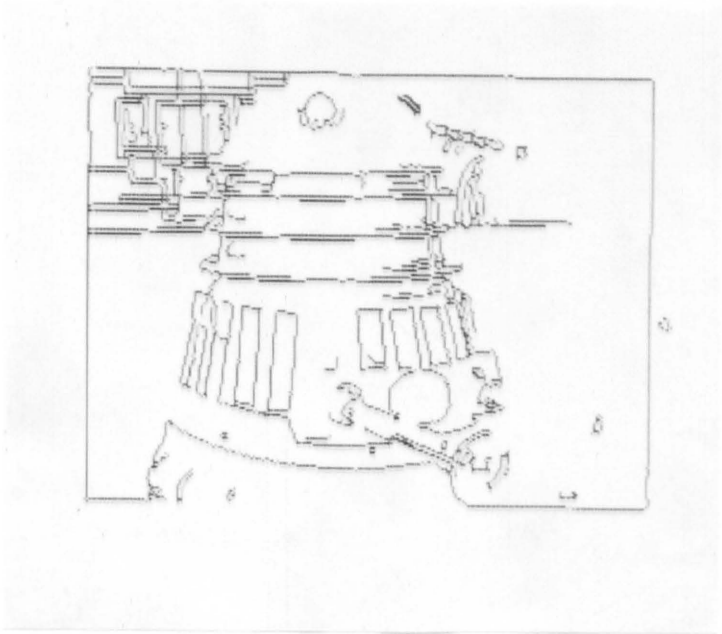
(b)



(c)

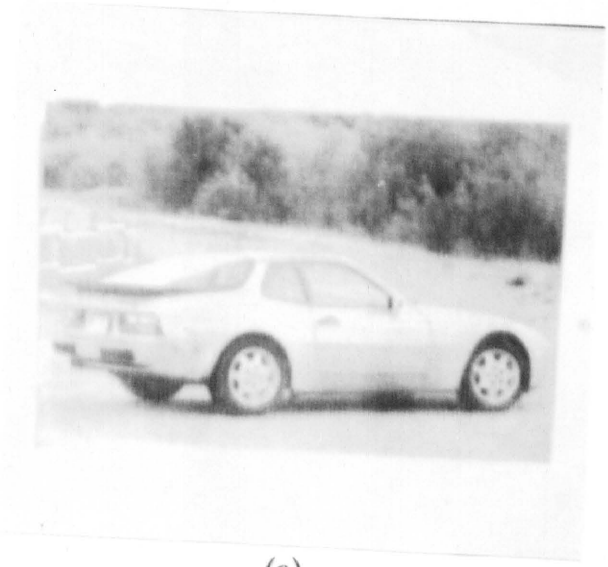


(d)

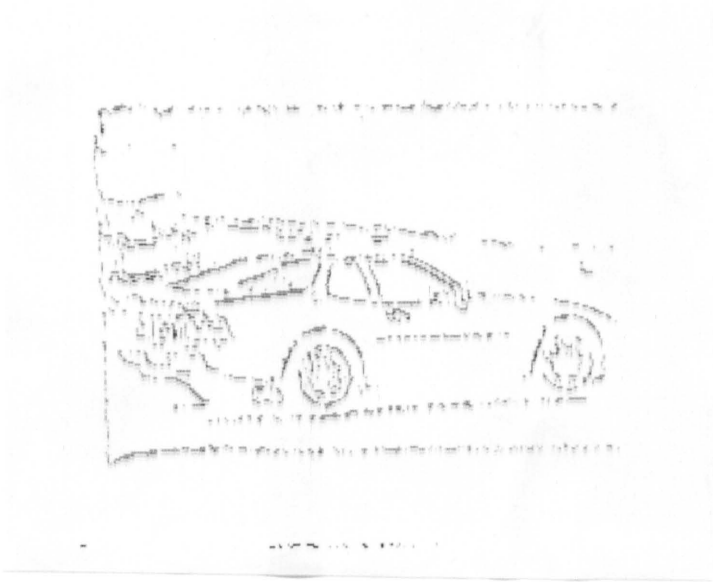


(e)

Figure (5.1) (a) Image
Edge results from :
(b) Canny's Method with $\sigma = 1.5$ and TFD $= 300$
(c) LoG with $\sigma = 3.0$ and TLEC $= 5$
(d) Eigen Vector method with No of pixels $= 7$,
TD $= 30$ and TLEC $= 3$
(e) First Derivative with Threshold $= 20$ and TLEC $= 5$

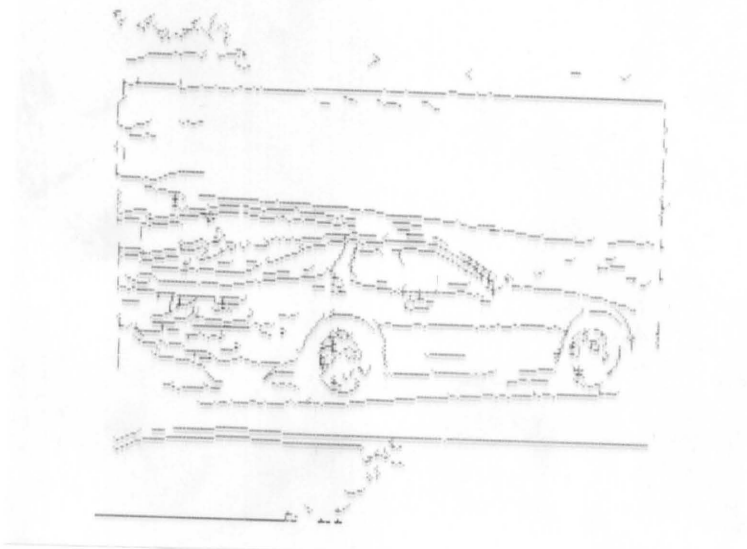


(a)

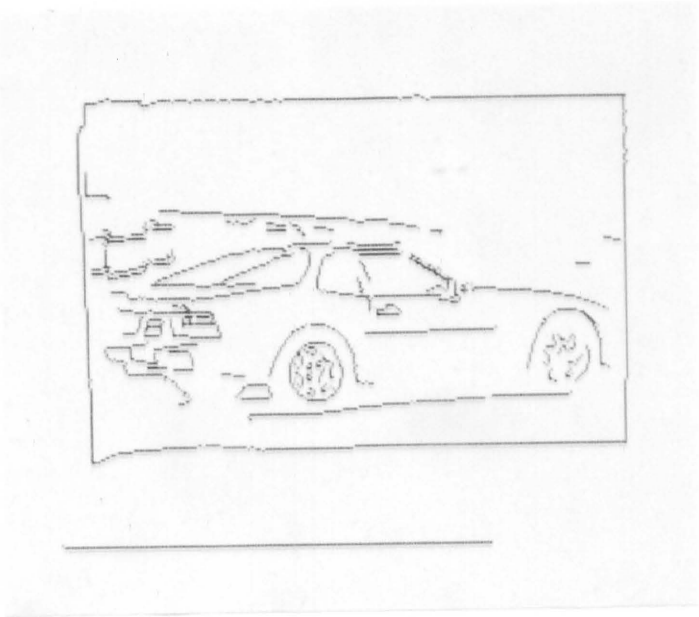


(b)

(c)



(d)



(e)

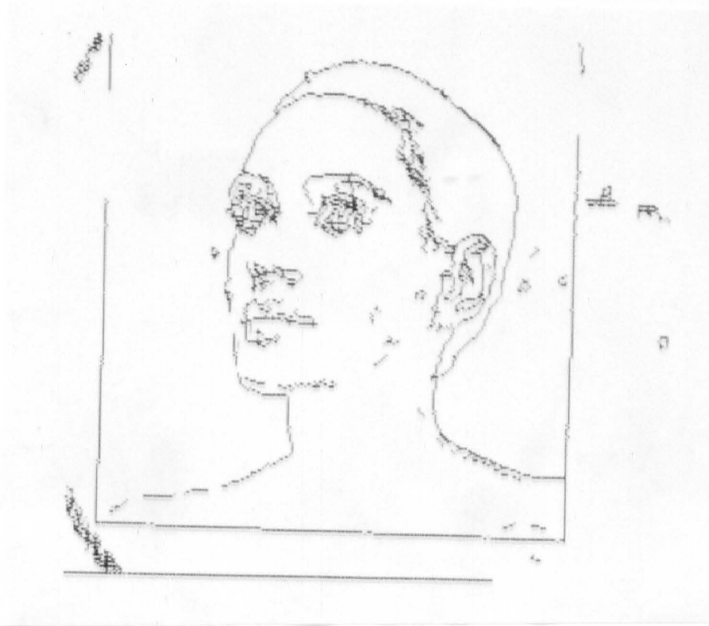
Figure (5.2) (a) Image
Edge results from :
(b) Canny's Method with $\sigma = 2.0$ and TFD $= 50$
(c) LoG with $\sigma = 2.5$ and TLEC $= 10$
(d) Eigen Vector method with No of pixels $= 7$,
TD $= 25$ and TLEC $= 3$
(e) First Derivative with Threshold $= 20$ and TLEC $= 5$

(a)

(b)

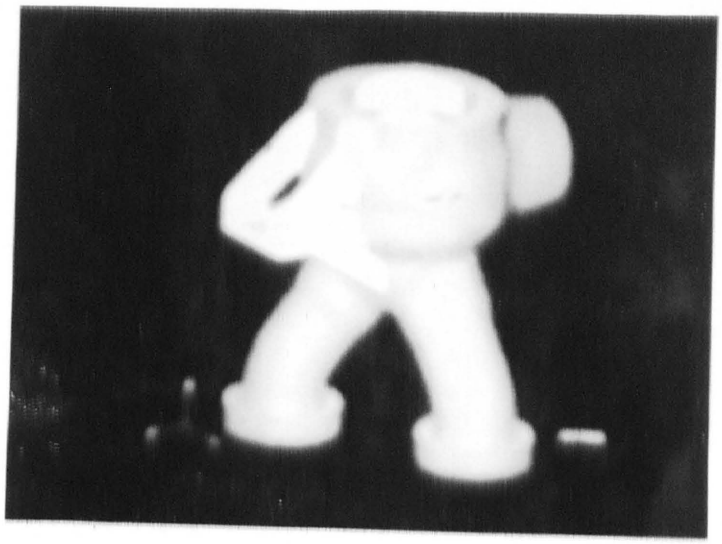
(c)

(d)

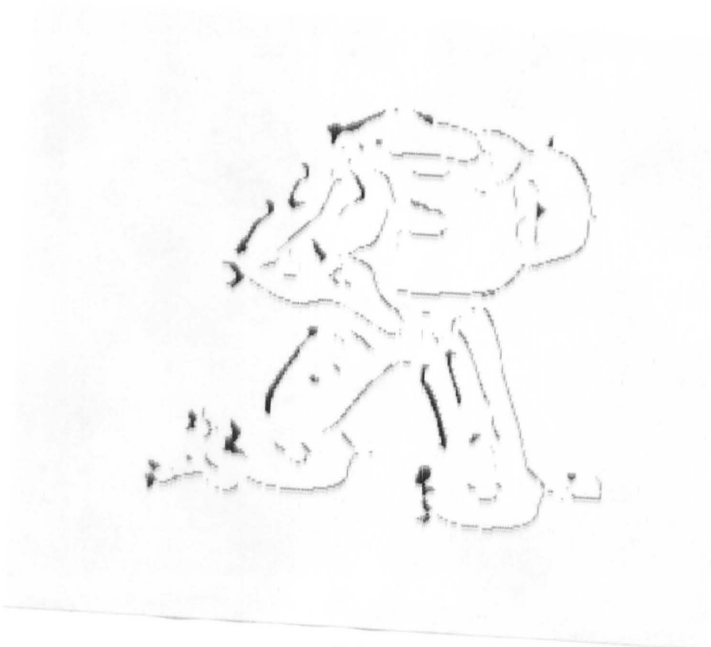


(e)

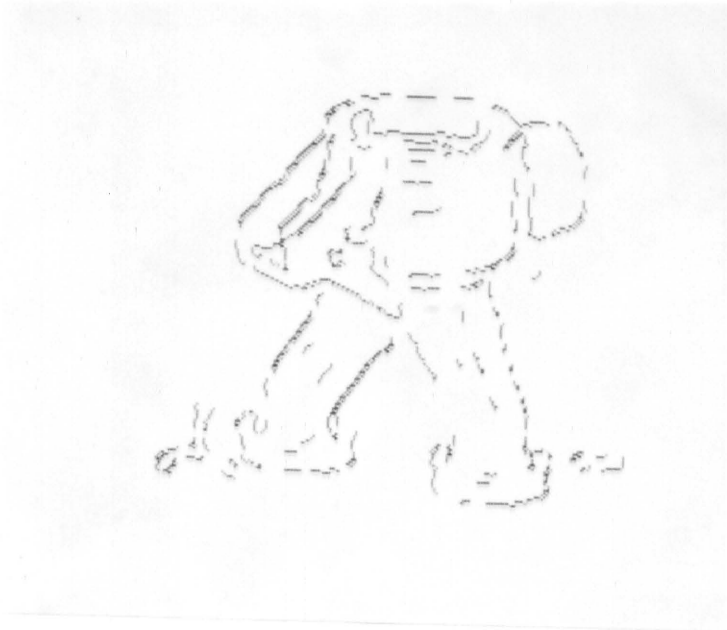
Figure (5.3) (a) Image
Edge results from :
(b) Canny's Method with $\sigma = 1.0$ and TFD $= 150$
(c) LoG with $\sigma = 1.5$ and TLEC $= 5$
(d) Eigen Vector method with No of pixels $= 7$,
TD $= 30$ and TLEC $= 3$
(e) First Derivative with Threshold $= 10$ and TLEC $= 3$



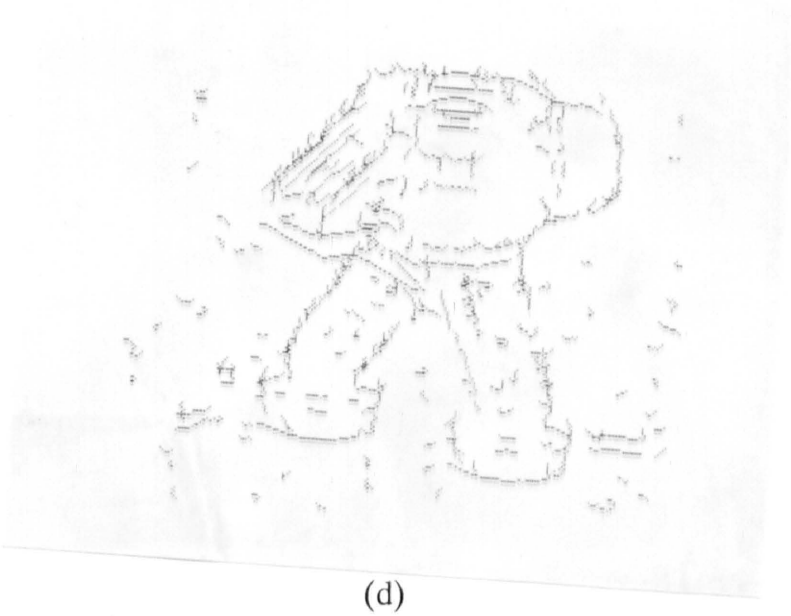
(a)



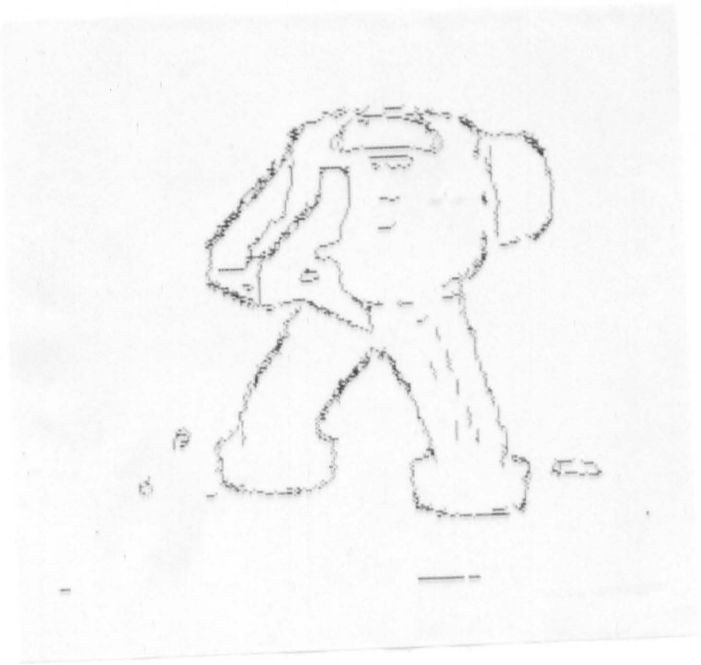
(b)



(c)

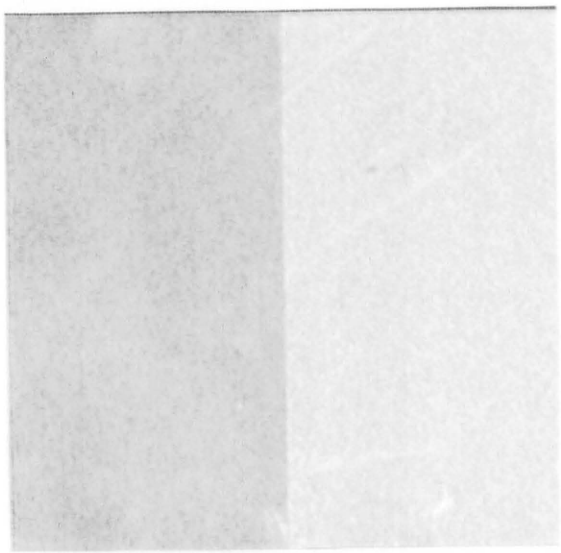


(d)



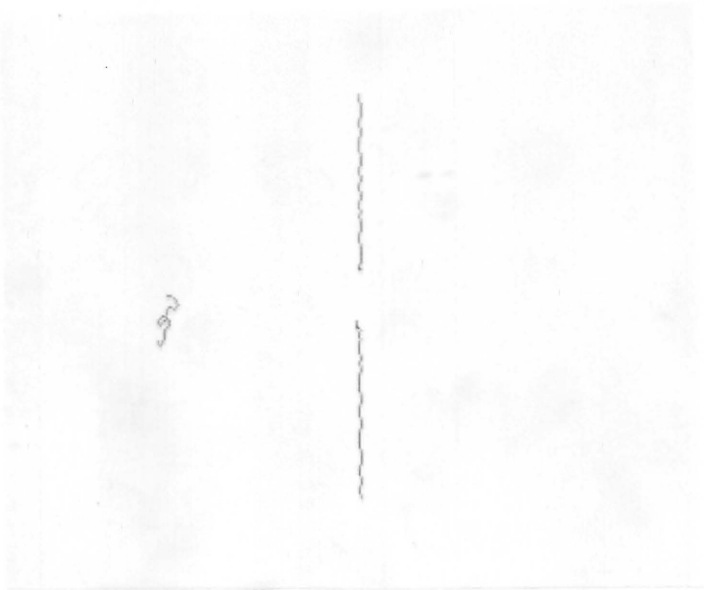
(e)

Figure (5.4) (a) Image
Edge results from :
(b) Canny's Method with σ = 1.5 and TFD = 200
(c) LoG with σ = 3.0 and TLEC = 4
(d) Eigen Vector method with No of pixels = 10,
TD = 20 and TLEC = 3
(e) First Derivative with Threshold = 10 and TLEC = 5

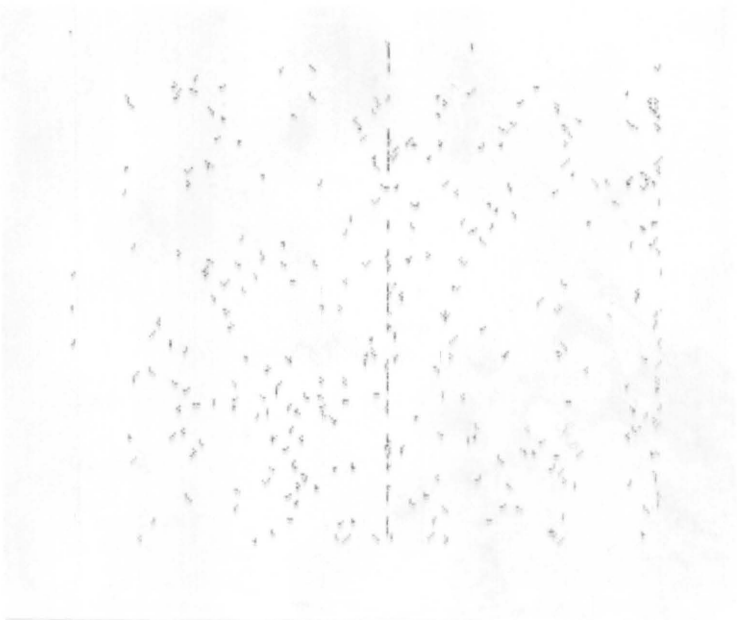


(a)

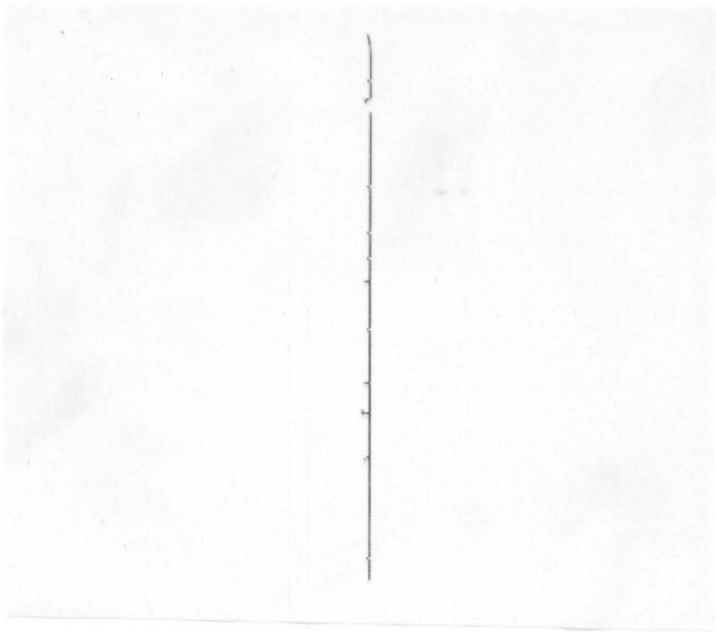
(b)



(c)



(d)



(e)

Figure (5.5) (a) Image
Edge results from :
(b) Canny's Method with $\sigma = 1.0$ and TFD $= 400$
(c) LoG with $\sigma = 5.0$ and TLEC $= 30$
(d) Eigen Vector method with No of pixels $= 14$,
TD $= 40$ and TLEC $= 1$
(e) First Derivative with Threshold $= 50$ and TLEC $= 0$

The results from Eigenvector line fitting method are more noisy because this method is not rotationaly invariant as it is applied only along X-axis and Y-axis.

In Figure 5.4 Canny's method and LoG have given better results than other two methods because there is very high variation in intensity level in the image. Here again result from Canney's method is better than LoG because of a smaller convolution mask size.

In Figure 5.5 (image with random noise) the results from Canny's method and First derivative are better than other methods because in Canny's method averaging is not as severe as LoG (small value of σ) and in First derivative method the threshold is kept equal to maximum level of random noise.

5.2 CONCLUSION:

The Canny's method works better overall than the other methods implemented, because it is rotationaly invariant as LoG and needs smaller convolution mask then LoG. Canny's method is also faster than LoG because smaller value of σ is needed as convolution matrix size depends on σ (convolution matrix size = 2 x (4.828 x σ) + 1). First derivative method is the fastest method due to less calculations as no convolution is applied in this method. This method works well if the intensity variation at edges is approximately same for all the edges in the image, and noise level is less than the difference in intensity at edges. The new approach proposed here can detect those edges

clearly which are either along X-axis or Y-axis. Therefore, it needs to be modified in order to make it rotationaly invariant so that better results can be obtained.

The major problem for all the methods is that each requires some parameters that are user input. Such parameters are usually different for different types of images. The User input variables used in above four methods are as shown in the following Table:

METHOD	VARIABLES
Canny's	Sigma (σ) and Threshold of first derivative results
LoG	Sigma (σ) and TLEC
First Derivative	Threshold of differences and TLEC
Eigen vector	No of Pixels , TD (Threshold of Distance) and TLEC

Table 5.1 Variables used in different methods, here TLEC means Threshold of Length of Edge Contures.

As can be seen from Table 5.1, all the four methods discussed so far have user input variables. Much work is left in the field of edge detection in order to eliminate such user input parameters. Better methods need to be proposed that do not need such parameters and increase the efficiency of edge detector.

The eigenvector method needs more modifications. At present, each point for eigenvector fit has the same weight. It is likely that the use of half-Gaussian type weighting factors before fitting eigenvector lines may improve the results. In some ways, this approach is similar to edge fitting methods. Therefore, a natural extension would be to try eigenvector surface fitting in a local area of interest. This may be done for example, by trying to detect two clusters in a local area, where one cluster represents one side of the edge and the other cluster represents the other side. Such approach may make this method rotationaly invariant. More research in this area is recommended.

APPENDIX

PROGRAM LISTING

```
            /**********************
             *                    *
             *    Main Program    *
             *                    *
             **********************/

#include  <stdio.h>
#include  <pixrect/pixrect_hs.h>
#include        <math.h>

#define     TRUE    1
#define     FALSE   0

FILE     *fp;
int       i,j,index,h[260][260];
Pixrect  *pr;
unsigned char red[256], green[256], blue[256];
int    res,flag;
char str[30];

main()
{
      for (i = 0; i < 256; i++)
      {
         red[i] = green[i] = blue[i] = i;
      }

      pr = pr_open("/dev/fb");
      if (!pr)
         exit(1);
      flag = TRUE;
      while (flag) {
          pr_rop(pr,0,0,900,1140,PIX_CLR,(Pixrect *)0,0,0);
          pr_putcolormap(pr, 0, 256, red, green, blue);
          printf("Pick the Edge Detector from
                              the following : \n");
          printf("\n1. Canney. \n");
          printf("2. LoG.\n");
          printf("3. First Diff.\n");
          printf("4. Eigenvector.\n");
          printf("5. Quit.\n");
          printf("Give the Response : ");
          scanf("%d",&res);
          if ((res >= 1) && (res <= 4)) {
               printf("Input Image filename : ");
               scanf("%s", str);
               fp = fopen(str, "r");
               if (fp == NULL)
               {
                   printf("File %s cannot be
                                  opened !!\n", str);
                   exit(-1);
               }
               for (i = 0; i < 240; i++)
```

```
                    for (j = 0; j < 256; j++)
                    {
                         fscanf(fp, "%d", &index);
                         h[i][j] = index;
                    }
               red[250] = 250;
               green[250] = 0;
               blue[250] = 0;
     }
     if (res == 1)
          canney();
     else if (res == 2)
          mlog();
     else if (res == 3)
          firstd();
     else if (res == 4)
          meig(h);
     else if (res == 5)
          flag = FALSE;
     else
          printf("\n\nType the Correct Response !!!\n\n");

      }

      pr_destroy(pr);
      exit(0);
}
```

```
                /*********************************
                 *                               *
                 *  First Derivative Method      *
                 *                               *
                 *********************************/

#include  <stdio.h>
#include  <pixrect/pixrect_hs.h>
#include  <math.h>
int       j, c, i, k, l, m, p, cut, thresh;
double    xmean, ymean, temp, b5, b6, sq,
              temp1, tempm, tempc, ang, rho;
double    a1[5], b1[5], a2[5], b2[5],
            r1, r2, e5, e6, s1, s2, s3;
FILE     *fp, *fp1, *fp2, *fp3;
int       i, j, g[256][256], index, count, counter;
int       x1[10000], k1[10000], code[10000];
float     values[256][3], b;
char      str[10], str1[10], junck[10];
extern Pixrect  *pr;
extern int h[260][260];
extern unsigned char red[256], green[256], blue[256];

firstd()
{
      pr_rop(pr,0,0,1100,1140,PIX_CLR,(Pixrect *)0,0,0);
      pr_putcolormap(pr, 0, 256, red, green, blue);
      for (i = 0; i < 240; i++)
         for (j = 0; j < 256; j++)
            pr_put(pr, j, i, h[i][j]);
      fird();
      printf("Threshold for length of edge contours =    ");
      scanf("%d", &cut);
      printf("Out put file name :   ");
      scanf("%s", str1);
      fp1 = fopen(str1, "w");
      counter = 1;
      while (counter)
         trace();
}

/* ---- Finding differences and thresholding the results ---
----*/

fird()
{
      int       c[3], k[3];

      printf("CUT OFF VALUE =          ");
      scanf("%d", &thresh);
      c[0] = c[1] = c[2] = k[0] = k[1] = k[2] = 0;
      for (i = 0; i < 239; i++)
         for (j = 0; j < 255; j++)
            g[i][j] = 0;
```

```
        fp3 = fopen("fn", "w");
        for (i = 0; i < 237; i++)
           for (j = 0; j < 253; j++)
           {
              c[0] = abs(h[i][j] - h[i][j + 1]);
              c[1] = abs(h[i][j + 1] - h[i][j + 2]);
              c[2] = abs(h[i][j + 2] - h[i][j + 3]);
              k[0] = abs(h[i][j] - h[i + 1][j]);
              k[1] = abs(h[i + 1][j] - h[i + 2][j]);
              k[2] = abs(h[i + 2][j] - h[i + 3][j]);
              if (c[1] >= c[0] && c[1] >= c[2])
                 if (c[1] > thresh)
                 {
                       g[i][j + 1] = 100;
                       fprintf(fp3, "%d %d ", j + 1, i);
                       pr_put(pr, j + 520, i, 255);
                 }
              if (k[1] >= k[0] && k[1] >= k[2])
                 if (k[1] > thresh)
                 {
                       g[i + 1][j] = 100;
                       pr_put(pr, j + 520, i, 255);
                       fprintf(fp3, "%d %d ", j, i + 1);
                 }
           }
}
```

```
                /**********************************
                 *                                *
                 *          LoG                   *
                 *                                *
                 **********************************/

#include  <stdio.h>
#include  <pixrect/pixrect_hs.h>
#include        <math.h>

FILE     *fp, *fp1;
extern int        h[260][260];
int       i, j, index, count, cut;
float     d, a, e, c2, c1, rr, rs;
float     sig, h1[260][260], h2[260][260],
             h3[260][260], h4[260][260];
char      str[40], str1[10];
float     filter[3][150];
int       x1[1000], k1[1000], code[1000];
int       g[650][650], counter;

extern Pixrect  *pr;
extern unsigned char red[256], green[256], blue[256];

mlog()
{
      pr_rop(pr,0,0,900,1140,PIX_CLR,(Pixrect *)0,0,0);
      pr_putcolormap(pr, 0, 256, red, green, blue);
      for (i = 0; i < 240; i++)
      {
         for (j = 0; j < 256; j++)
            pr_put(pr, j + 125, i + 125, (int) h[i][j]);
      }
      m_log();
      printf("Threshold for length of edge
                                    contures =        ");
      scanf("%d", &cut);
      printf("Out put file name :   ");
      scanf("%s", str1);
      fp1 = fopen(str1, "w");
      counter = 1;
      while (counter)
         trace();
}

m_log()
{
      cons();
      gauss();
      convolv();
      zc();
}
int       o, p, n;
```

```
cons()
{
      int       w;

      /* -------- constants ---------- */

      printf("Sigma(The blur fator) =          ");
      scanf("%f", &sig);
      w = (int) (2 * 1.414 * sig + 0.5);
      o = (int) (1.8 * w + 1);
      n = 2 * o + 1;
}

      /*------ Forming the Gaussion Filter ---------*/


gauss()
{
      for (i = 1; i <= n; i++)
      {
         rr = i - o - 1.;
         rs = rr * rr / sig;
         filter[1][i] = exp(-rs / 2);
         filter[2][i] = -(1 - rs) * filter[1][i];
      }
}

convolv()
{
      int       kk, k;

      /*----- Convolving The Image in X Direction h12 ---*/

      for (i = 0; i < 240; i++)
         for (j = o; j < 256 - o; j++)
         {
            kk = j - o;
            for (k = 1; k <= n; k++, kk++)
            {
               h1[i][j] += h[i][kk] * filter[1][k];
            }
         }

      /*------ Convolving The Image in Y Direction h12 ---*/

      for (i = o; i < 240 - o; i++)
         for (j = o; j < 256 - o; j++)
         {
            kk = i - o;
            for (k = 1; k <= n; k++, kk++)
            {
               h2[i][j] += h1[kk][j] * filter[2][k];
            }
```

```
        }

    /*---- Convolving The Image in X Direction h21 -----*/

    for (i = o; i < 240; i++)
        for (j = o; j < 256 - o; j++)
        {
            h1[i][j] = 0.;
            kk = j - o;
            for (k = 1; k <= n; k++, kk++)
            {
                h1[i][j] += h[i][kk] * filter[2][k];
            }
        }

    /*----- Convolving The Image in Y Direction h21 ----*/

    d = a = c1 = c2 = 0;
    for (i = o; i < 240 - o; i++)
    {
        for (j = o; j < 256 - o; j++)
        {
            e = 0;
            h3[i][j] = 0.;
            h4[i][j] = 0.;
            kk = i - o;
            for (k = 1; k <= n; k++, kk++)
            {
                h3[i][j] += h1[kk][j] * filter[1][k];
            }
            h4[i][j] = h2[i][j] + h3[i][j];
        }
    }
}

/* ------------ Findinr Zero Crossings -------------- */

zc()
{
    for (i = o; i < 240 - o; i++)
        for (j = o; j < 256 - o; j++)
        {
            if (h4[i][j] >= 0.)
                h1[i][j] = 10.;
            else
                h1[i][j] = 0.;
        }
    for (i = o; i < 240 - o; i++)
        for (j = o; j < 256 - o - 1; j++)
        {
            if (fabs(h4[i][j]) > 20.)
            {
                if (h1[i][j] != h1[i][j + 1])
                {
```

```
                        if (fabs(h4[i][j]) > fabs(h4[i][j +
1]))
                           g[i][j] = 100;
                        else
                           g[i][j + 1] = 100;
                     }
                  }
               }
            for (i = o; i < 240 - o - 1; i++)
               for (j = o; j < 256 - o; j++)
               {
                  if (fabs(h4[i][j]) > 20.)
                  {
                     if (h1[i][j] != h1[i + 1][j])
                     {
                           if (fabs(h4[i][j]) > fabs(h4[i +
1][j]))
                           g[i][j] = 100;
                        else
                           g[i + 1][j] = 100;
                     }
                  }
               }
            for (i = o; i < 240 - o - 1; i++)
               for (j = o; j < 256 - o; j++)
               {
                  if (fabs(h4[i][j]) > 20.)
                  {
                     if (h1[i][j] != h1[i + 1][j + 1])
                     {
                           if (fabs(h4[i][j]) > fabs(h4[i + 1][j +
1]))
                           g[i][j] = 100;
                        else
                           g[i + 1][j + 1] = 100;
                     }
                  }
               }
            fp = fopen("edge", "w");
            for (i = o; i < 240 - o - 1; i++)
               for (j = o; j < 256 - o - 1; j++)
               {
                  if (g[i][j] == 100)
                  {
                     fprintf(fp, "%d %d ", j, i);
                     pr_put(pr, j + 150, i + 450, g[i][j]);
                  }
               }
            fclose(fp);
}
```

```
          /*****************************************
           *                                       *
           *    Canny's method for edge detection  *
           *                                       *
           *****************************************/

#include  <stdio.h>
#include  <pixrect/pixrect_hs.h>
#include        <math.h>

#define     TRUE    1
#define     FALSE   0

FILE     *fp;
extern int       h[260][260];
float     h1[260][260], h2[260][260], h3[260][260];
int       g[260][260], w, w1,  o, l;
int       i, j, n, con[3][1750], index, count;
float     g1[150], s, k, temp;
int       x, y, m, c;
char      str[40];
float     a;
extern Pixrect  *pr;
extern unsigned char red[256], green[256], blue[256];


canney()
{
int  i,j;
      pr_rop(pr, 0, 0, 1100, 1140, PIX_CLR, (Pixrect *) 0,
0, 0);
      pr_putcolormap(pr,0,256,red,green,blue);
      for (i = 0;i <  240;i++)
          for (j = 0;j < 256;j++)
               pr_put(pr,j,i,h[i][j]);
      oper();
      convol();
      edge();
      nmax();
}
/* --------- Formation of Gaussian Filter ------------*/
oper()
{
      printf("Sigma (The operator width)=          ");
      scanf("%f", &s);
      w1 = (int) (2 * 1.414 * s + 0.5);
      o = (int) (1.8 * w1 + 1);
      w = (int) (2 * o + 1);
      for (i = 1; i <= w; i++)
      {
         c = i - o - 1;
         temp = -(c * c) / (2.0 * s * s);
         g1[i] = exp(temp);
      }
```

```
}

convol()
{
      float     min, max;

/*----------------- Convolution in X direction ---------- */

      for (i = 0; i < 240; i++)
         for (j = o; j < 256 - o; j++)
         {
            m = j - o;
            for (l = 1; l <= w; l++, m++)
            {
               h1[i][j] += h[i][m] * g1[l];
            }
         }
      max = 0;
      min = 10000;

/*----------------- Convolution in Y direction ---------- */

      fp = fopen("con", "w");
      for (i = o; i < 240 - o; i++)
      {
         for (j = o; j < 256 - o; j++)
         {
            m = i - o;
            for (l = 1; l <= w; l++, m++)
            {
               h2[i][j] += h1[m][j] * g1[l];
            }
            if (max < h2[i][j])
               max = h2[i][j];
            if (min > h2[i][j])
               min = h2[i][j];
         }
      }
      a = (max - min) / 256;
      for (i = o; i < 246 - o; i++)
         for (j = o; j < 256 - o; j++)
         {
            fprintf(fp, "%d ", (int) (h2[i][j] / a + 0.5));
            pr_put(pr, j + 500, i + 150, (int)
                     (h2[i][j] / a + 0.5));
         }
      fclose(fp);
}
float     k1, k2, k3, k4, k5, k6;
double    t[256][256];

edge()
{
      double    x, y, theta;
```

```
/*--------- Calculate directional derivative --------- */

      for (i = o + 1; i < 240 - o - 1; i++)
         for (j = o + 1; j < 240 - o - 1; j++)
         {
          x=(h2[i+1][j-1]+2*h2[i+1][j]+h2[i+1][j+1])
               -(h2[i-1][j-1]+2*h2[i-1][j]+h2[i-1][j+1]);
          y=(h2[i-1][j-1]+2*h2[i][j-1]+h2[i+1][j-1])
               -(h2[i-1][j+1]+2*h2[i][j+1]+h2[i+1][j+1]);
            theta = atan(y / x);
            t[i][j] = fabs(theta);
            h1[i][j] = 0;
            h1[i][j] = sqrt(x * x + y * y);
            pr_put(pr, j + 700, i + 600, (int) h1[i][j]);
         }

}

/*---- Routine to perform non-maxima suppression -----*/

nmax()
{
      double    ux, uy, g1, g2;
      float     thresh;
      FILE     *fp1;
      char      str1[10];

      printf("Threshold =    ");
      scanf("%f", &thresh);
      printf("OUT PUT FILE :   ");
      scanf("%s", str1);
      fp1 = fopen(str1, "w");
      for (i = o + 1; i < 240 - o - 1; i++)
         for (j = o + 1; j < 240 - o - 1; j++)
         {
            g[i][j] = 0;
            /*------ 0 to 45 ---------------*/
            if (t[i][j] <= 0.7857)
            {
               ux = uy = g1 = g2 = 0.;
               ux = t[i][j] / 0.7857;
               uy = 1. - ux;
               g1 = ux * h1[i-1][j+1] + uy * h1[i][j+1];
               g2 = ux * h1[i+1][j-1] + uy * h1[i][j-1];
            }
            /*------ 45 to 90 ---------------*/
            else if (t[i][j] <= 1.5714)
            {
               ux = uy = g1 = g2 = 0.;
               ux = (t[i][j] - 0.7857) / 0.7857;
               uy = 1. - ux;
               g1 = ux * h1[i-1][j] + uy * h1[i-1][j+1];
               g2 = ux * h1[i+1][j] + uy * h1[i+1][j-1];
```

```
            }
            /*------ 90 to 135 ---------------*/
            else if (t[i][j] <= 2.3571)
            {
               ux = uy = g1 = g2 = 0.;
               ux = (t[i][j] - 1.5714) / 0.7857;
               uy = 1. - ux;
               g1 = ux * h1[i-1][j-1] + uy * h1[i-1][j];
               g2 = ux * h1[i+1][j+1] + uy * h1[i+1][j];
            }
            /*------ 135 to 180 ---------------*/
            else if (t[i][j] <= 3.14285)
            {
               ux = uy = g1 = g2 = 0.;
               ux = (t[i][j] - 2.3571) / 0.7857;
               uy = 1. - ux;
               g1 = ux * h1[i][j-1] + uy * h1[i-1][j-1];
               g2 = ux * h1[i][j+1] + uy * h1[i+1][j+1];
            }
            if (h1[i][j] > g1 && h1[i][j] > g2)
               if (h1[i][j] >= thresh)
               {
                     g[i][j] = 250;
                     fprintf(fp1, "%d %d ", j, i);
                     fflush(fp1);
               }
            pr_put(pr, j + 200, i + 400, g[i][j]);
         }
      fclose(fp1);
}
```

```
                /*****************************
                 *                           *
                 *     Eigenvector Method    *
                 *                           *
                 *****************************/

#include  <stdio.h>
#include  <pixrect/pixrect_hs.h>
#include  <math.h>

#define   MAX(a,b)  ((a) > (b) ? (a) : (b))
#define MIN(a,b)        ((a) > (b) ? (b) : (a))

int       a, j, c, i, k, l, m, p, cut, r, ci, cj;
double    xmean, ymean, temp, b5, b6, sq, temp1,
               tempm, tempc, ang, rho;
double    a1[15], b1[15], a2[15], b2[15], r1, r2,
               e5, e6, s1, s2, s3;
float     s[500][2], y[500][2], thresh;
FILE     *fp, *fp1, *fp2, *fp3;
int       g[256][256], g1[260][260], h[260][260],
          index, count, counter;
char      str[10], str1[10];
int       code[10000], x1[10000], k1[10000];
extern Pixrect  *pr;
extern unsigned char red[256], green[256], blue[256];

meig(g2)
int  g2[260][260];
{
      printf("No of Pixels for Eigen Vector=       ");
      scanf("%d", &p);
      printf("Cut of distance value =     ");
      scanf("%f", &thresh);
      printf("Min no of pixels for an edge =     ");
      scanf("%d", &cut);
      printf("Out Put File :   ");
      scanf("%s", str1);
      a = 0;
      edgey(g2);
      a = 0;
      edgex(g2);
      fclose(fp);
      fp1 = fopen(str1, "w");
      fp3 = fopen("dat", "w");
      for (i = 0; i < 240; i++)
         for (j = 0; j < 256; j++)
         {
            if ((g1[i][j] == 100) || (h[i][j] == 100))
            {
               g[i][j] = 100;
               pr_put(pr, j + 650, i + 150, 255);
               fprintf(fp3, "%d %d ", j, i);
            }
```

```
            else
            {
               g[i][j] = 0;
               pr_put(pr, j + 650, i + 150, 0);
            }
         }
      counter = 1;
      while (counter)
         trace();
}

/* ---------- Eigenvector fitting in Y direction -------*/

edgey(g2)
int  g2[260][260];
{
      for (i = 0; i < 240; i++)
      {
         for (j = 0; j < 256; j++)
         {
            pr_put(pr, j, i + 150, g2[i][j]);
            s[j][1] = j;
            s[j][0] = g2[i][j];
         }
         eigeny();
      }
}

eigeny()
{
      c = 0;
      a = 0;
      for (j = 0; j < 256; j++)
      {
         l = 0;
         xmean = ymean = 0.;
         for (k = j; k < j + p; k++)
         {
            xmean += s[k][0];
            ymean += s[k][1];
            l++;
         }
         xmean /= l;
         ymean /= l;
         for (l = 0, k = j; k < j + p; k++)
         {
            y[l][0] = s[k][0] - xmean;
            y[l++][1] = s[k][1] - ymean;
         }
         s1 = s2 = s3 = 0;
         for (m = 0; m < l; m++)
         {
            s1 += y[m][0] * y[m][0];
            s2 += y[m][1] * y[m][0];
```

```
            s3 += y[m][1] * y[m][1];
         }
         temp = pow((s1 - s3), 2.0) + 4.0 * s2 * s2;
         sq = sqrt(temp);
         b5 = (s1 + s3 - sq) / 2.0;
         b6 = (s1 + s3 + sq) / 2.0;
         e6 = MAX(b5, b6);
         e5 = MIN(b5, b6);
         temp1 = s2 + s1 - e6;
         if (temp1 != 0.)
         {
            tempm = (e6 - s3 - s2) / temp1;
            tempc = ymean - tempm * xmean;
            ang = atan(tempm);
            rho = xmean * cos(ang) + ymean * sin(ang);
         }
         else
         {
            ang = 90 * 22.0 / 7.0 * 1. / 180.;
            rho = xmean;
         }
         draw_eig();
      }
}

draw_eig()
{
      double    d1, d2, d3, d4;

      d1 = y[0][0] + xmean;
      d2 = y[1 - 1][0] + xmean;
      d3 = y[0][1] + ymean;
      d4 = y[1 - 1][1] + ymean;
      if ((sin(ang) != 0.) && (cos(ang) != 0.))
      {
         r1 = (d3 * cos(ang) - d1 * sin(ang)) / cos(ang);
         r2 = (d4 * cos(ang) - d2 * sin(ang)) / cos(ang);
         r1 -= rho / sin(ang);
         r2 -= rho / sin(ang);
         r1 *= -1 * sin(ang) * cos(ang);
         r2 *= -1 * sin(ang) * cos(ang);
         b1[c] = (rho - r1 * cos(ang)) / sin(ang);
         b2[c] = (rho - r2 * cos(ang)) / sin(ang);
         a1[c] = r1;
         a2[c] = r2;
      }
      else if (sin(ang) == 0.)
      {
         a1[c] = d1;
         a2[c] = d2;
         b1[c] = b2[c] = rho;
      }
      else
      {
```

```
            a1[c] = a2[c] = rho;
            b1[c] = d3;
            b2[c] = d4;
         }

         if (c < p)
            c++;
         else
         {
            line();
            if (a < 2)
               a++;
         }
}

line()
{
         float     dis[3];

         dis[a] = (float) (sqrt((a1[c] - a2[c - p]) * (a1[c] -
              a2[c - p]) + (b1[c] - b2[c - p]) * (b1[c] - b2[c
- p])));
         if (a >= 2)
         {
            if (dis[1] >= dis[2] && dis[1] >= dis[0])
            {
               if (dis[1] > thresh)
               {
                  g1[i][j + (int) (p / 2 + 0.5)] = 100;
                  pr_put(pr, j, i + 400, 250);
               }
               else
                  g1[i][j + (int) (p / 2 + 0.5)] = 0;
            }
            dis[0] = dis[1];
            dis[1] = dis[2];
         }
         for (r = 1; r <= p; r++)
         {
            a1[r - 1] = a1[r];
            a2[r - 1] = a2[r];
            b1[r - 1] = b1[r];
            b2[r - 1] = b2[r];
         }
}

/* ----------- Eigenvector fitting in X direction -------*/

edgex(g2)
int  g2[260][260];
{
         for (j = 0; j < 256; j++)
         {
            for (i = 0; i < 240; i++)
```

```
            {
               s[i][0] = i;
               s[i][1] = g2[i][j];
            }
            eigenx();
         }
}

eigenx()
{
      c = 0;
      a = 0;
      for (i = 0; i < 240 - p; i++)
      {
         l = 0;
         xmean = ymean = 0.;
         for (k = i; k < i + p; k++)
         {
            xmean += s[k][0];
            ymean += s[k][1];
            l++;
         }
         xmean /= l;
         ymean /= l;
         for (l = 0, k = i; k < i + p; k++)
         {
            y[l][0] = s[k][0] - xmean;
            y[l++][1] = s[k][1] - ymean;
         }
         s1 = s2 = s3 = 0;
         for (m = 0; m < l; m++)
         {
            s1 += y[m][0] * y[m][0];
            s2 += y[m][1] * y[m][0];
            s3 += y[m][1] * y[m][1];
         }
         temp = pow((s1 - s3), 2.0) + 4.0 * s2 * s2;
         sq = sqrt(temp);
         b5 = (s1 + s3 - sq) / 2.0;
         b6 = (s1 + s3 + sq) / 2.0;
         e6 = MAX(b5, b6);
         e5 = MIN(b5, b6);
         temp1 = s2 + s1 - e6;
         if (temp1 != 0.)
         {
            tempm = (e6 - s3 - s2) / temp1;
            tempc = ymean - tempm * xmean;
            ang = atan(tempm);
            rho = xmean * cos(ang) + ymean * sin(ang);
         }
         else
         {
            ang = 90 * 22.0 / 7.0 * 1. / 180.;
            rho = xmean;
```

```
            }
            draw_eigx();
        }
}

draw_eigx()
{
        double    d1, d2, d3, d4;

        d1 = y[0][0] + xmean;
        d2 = y[l - 1][0] + xmean;
        d3 = y[0][1] + ymean;
        d4 = y[l - 1][1] + ymean;
        if ((sin(ang) != 0.) && (cos(ang) != 0.))
        {
            r1 = (d3 * cos(ang) - d1 * sin(ang)) / cos(ang);
            r2 = (d4 * cos(ang) - d2 * sin(ang)) / cos(ang);
            r1 -= rho / sin(ang);
            r2 -= rho / sin(ang);
            r1 *= -1 * sin(ang) * cos(ang);
            r2 *= -1 * sin(ang) * cos(ang);
            b1[c] = (rho - r1 * cos(ang)) / sin(ang);
            b2[c] = (rho - r2 * cos(ang)) / sin(ang);
            a1[c] = r1;
            a2[c] = r2;
        }
        else if (sin(ang) == 0.)
        {
            a1[c] = d1;
            a2[c] = d2;
            b1[c] = b2[c] = rho;
        }
        else
        {
            a1[c] = a2[c] = rho;
            b1[c] = d3;
            b2[c] = d4;
        }


        if (c < p)
            c++;
        else
        {
            linex();
            if (a < 2)
                a++;
        }
}

linex()
{
        float     dis[3];
```

```
      dis[a] = (float) (sqrt((a1[c] - a2[c - p]) * (a1[c] -
           a2[c - p]) + (b1[c] - b2[c - p]) * (b1[c] - b2[c
- p])));
      if (a >= 2)
      {
         if (dis[1] >= dis[2] && dis[1] >= dis[0])
         {
            if (dis[1] > thresh)
            {
               g1[i + (int) (p / 2 + 0.5)][j] = 100;
               pr_put(pr, j, i + 400, 250);
            }
            else
               g1[i + (int) (p / 2 + 0.5)][j] = 0;
         }
         dis[0] = dis[1];
         dis[1] = dis[2];
      }
      for (r = 1; r <= p; r++)
      {
         a1[r - 1] = a1[r];
         a2[r - 1] = a2[r];
         b1[r - 1] = b1[r];
         b2[r - 1] = b2[r];
      }
}
```

```
            /*************************
             *                       *
             *   Eight  Connectivity *
             *  Method to Find Length *
             *    of Edge Contures   *
             *                       *
             *************************/

#include  <stdio.h>
#include  <math.h>
#include  <pixrect/pixrect_hs.h>
extern    FILE *fp1;
extern    Pixrect   *pr;

#define   TRUE 1
#define   FALSE     0
#define MAX_BO 10000
#define MAX_BH 3000
#define d_max  10000

trace()
{
      int       c, ch1, i, n, x, y, a, b,
                        ab, err, check, peri;
      float     corr;
      extern int g[256][256], cut, counter;
      int       x1[10000], k1[10000], code[10000];
      int       x2[MAX_BH], y2[MAX_BH];
      int       count;
      char      str[10];

      count = 0;
      ch1 = TRUE;

      for (a = 0; a < 239; a++)
      {
         for (b = 0; b < 255; b++)
         {
            n = g[a][b];
            if (n == 100)
            {
               g[a][b] = 0;
               goto abc;
            }
            if ((a == 236) && (b == 252) && (n |= 100))
               counter = 0;

         }
      }

      ch1 = FALSE;
      goto lmn;
abc: ;
```

```
for (i = 0; i < MAX_BO; i++)
{
   code[i] = 0;
   x1[i] = 0;
   k1[i] = 0;
}

x1[0] = a;
k1[0] = b;
for (i = 1; i < MAX_BO; i++)
{
   check = FALSE;
   ab = 0;
   err = 0;
   while (check == FALSE)
   {
      if (check == FALSE)
      {
         if (code[i - 1] == 4 || ab == 1 ||
                  code[i - 1] == 0)
         {
               ab = 1;
               n = g[x1[i - 1] - 1][k1[i - 1] - 1];
               if (n == 100)
               {
                  x1[i] = x1[i - 1] - 1;
                  k1[i] = k1[i - 1] - 1;
                  code[i] = 1;
                  check = TRUE;
               }
         }
      }

      if (check == FALSE)
      {
         if (code[i - 1] == 5 || ab == 1)
         {
               ab = 1;
               n = g[x1[i - 1]][k1[i - 1] - 1];
               if (n == 100)
               {
                  x1[i] = x1[i - 1];
                  k1[i] = k1[i - 1] - 1;
                  code[i] = 2;
                  check = TRUE;
               }
         }
      }

      if (check == FALSE)
      {
         if (code[i - 1] == 6 || ab == 1)
         {
```

```
            ab = 1;
            n = g[x1[i - 1] + 1][k1[i - 1] - 1];
            if (n == 100)
            {
               x1[i] = x1[i - 1] + 1;
               k1[i] = k1[i - 1] - 1;
               code[i] = 3;
               check = TRUE;
            }
      }
}

if (check == FALSE)
{
      if (code[i - 1] == 7 || ab == 1)
      {
            ab = 1;
            n = g[x1[i - 1] + 1][k1[i - 1]];
            if (n == 100)
            {
               x1[i] = x1[i - 1] + 1;
               k1[i] = k1[i - 1];
               code[i] = 4;
               check = TRUE;
            }
      }
}

if (check == FALSE)
{
      if (code[i - 1] == 8 || ab == 1)
      {
            ab = 1;
            n = g[x1[i - 1] + 1][k1[i - 1] + 1];
            if (n == 100)
            {
               x1[i] = x1[i - 1] + 1;
               k1[i] = k1[i - 1] + 1;
               code[i] = 5;
               check = TRUE;
            }
      }
}

if (check == FALSE)
{
      if (code[i - 1] == 1 || ab == 1)
      {
            ab = 1;
            n = g[x1[i - 1]][k1[i - 1] + 1];
            if (n == 100)
            {
               x1[i] = x1[i - 1];
               k1[i] = k1[i - 1] + 1;
```

```
                        code[i] = 6;
                        check = TRUE;
                    }
                }
            }

            if (check == FALSE)
            {
                if (code[i - 1] == 2 || ab == 1)
                {
                    ab = 1;
                    n = gd[x1[i - 1] - 1][k1[i - 1] + 1];
                    if (n == 100)
                    {
                        x1[i] = x1[i - 1] - 1;
                        k1[i] = k1[i - 1] + 1;
                        code[i] = 7;
                        check = TRUE;
                    }
                }
            }

            if (check == FALSE)
            {
                if (code[i - 1] == 3 || ab == 1)
                {
                    ab = 1;
                    n = g[x1[i - 1] - 1][k1[i - 1]];
                    if (n == 100)
                    {
                        x1[i] = x1[i - 1] - 1;
                        k1[i] = k1[i - 1];
                        code[i] = 8;
                        check = TRUE;
                    }
                }
            }

            err += 1;
            if (err > 2)
            {
                ch1 = FALSE;
                goto lmn;
            }
        }

        if (check)
        {
            g[x1[i]][k1[i]] = 0;
            count++;
        }
    }
lmn:
    if (count > cut)
```

```
            for (a = 0; a < i + 1; a++){
               pr_put(pr, k1[a] + 500, x1[a] + 450, 255);
               fprintf(fp1, "%d %d ", k1[a], x1[a]);
            }
         fflush(fp1);
}
```

REFERENCES

1. Attneave, F., "Some information Aspects of Visual Perception," Psycological review, 61, 1954, pp. 183-193.

2. Bovik, A.C. and Munson, D. C., "Edge Detection Using Median Comparisons," Computer Vision, Graphics and Image Processings, Vol. 33, 1985, pp. 377-389.

3. Canny, J. F., " Finding Edges and Lines in Images," MIT Artificial Intelligence Laboratory Techinical Report TR-720, June 1983.

4. Greenfeld, J. S., "A Stereo Vision Approach to Automate Stereo Matching in Photogrammetry," Techinical Report, Ohio State University Columbus, Department of Geodetic Science and Surveying, TR-381, July 1987.

5. Haralick. R. M., "Digitial Step Edges from Zero Crossings of Second Directional Derivative," IEEE PAMI-30, Jan. 1971, pp. 113-125.

6 Hartly, R., " A Gaussian Weighted Multiresolution Edge Detector," Computer Vision, Graphics and Image Processing, Vol. 30, 1985, pp. 70-83.

7. Huckel, H. M., "An Operator Which Locates Edges in Digitized Pictures," Journal of ACM, Vol. 18, No. 1, Jan. 1971, pp. 113-125.

8. Jain, R. and Rheaume, D., " A two Stage Method for Fast Edge Detection," Computer Vision, Graphics, and Image Processing, Vol. 14, 1980, pp. 177-181.

9. Jadhav, R., "Development of a Low Cost PC Based Vision System," Masters Thesis, NJIT, 1987.

10. Macleod, I. D. G., " Comments on Techinques for Edge Detection," Proceedings of IEEE, Vol. 60, March 1972, pp. 344.

11. Marr, D. and Hildreth, E. C., "A Theory of Edge Detection," Phil. Trans. Roy. Soc. London B 207, 1980, pp. 187-217.

12. Medioni, G. and Nevatia, R., "Matching Images Using Linear Features," IEEE PAMI-6, No. 6, 1984, pp. 675-685.

13. Modestino, J. W. and Fries R. W., "Edge Detection in Moving Image Using Recursive Digital Filtering," Computer Graphics and Image Processing, Vol. 6, 1977, pp. 409-433.

14. Nalwa, V. S. and Binford, T. O., " On Detecting Edges," IEEE PAMI-8, No.6, Nov. 1980, pp. 699-715.

15. Nevatia, R. and Babu, K. R., " Linear Feature Extraction and Description," Computer Graphics and Image Processing, Vol. 13, 1980, pp. 699-715.

16. Nevatia, R., "Simple Polygonal Scenes," Machine Perception, Printice Hall, 1982, pp. 24-40.

17. Prewitt. J. M. S., "Object Enhancement and Extraction." Picture Processing and Psychopictorics B. lipkin & A. Rosenfeld Eds, Academic Press, New York, 1970, pp. 75-149.

18 Rosenfeld, A. and Thurston, M., " Edge and Curve Detection for Visual Scene Analysis," IEEE Transactions on Computers, Vol. 20, No. 5, 1971, pp. 562-569.

19 Robinson, G. S., " Edge Detection by Compass Gradient Masks," Computer Graphics and Image Processing, Vol. 6, 1977, pp. 492-501.

20. Shanmugam, K. F., Dickey, F. M., and Green, J. A., " An Optimal frequency Domain Filter for Edge Detection in Digital Images," IEEE PAMI-1, No. 2, 1979, pp. 37-49.

21. Torre, V. and Poggio, T. A., " On Edge Detection," IEEE PAMI-8, March 1986, pp. 147-163.