✓Gradient Operators

- Gradient operators are based on the idea of using the first or second derivative of the gray level function as an edge detector
- The first derivative will mark edge points, with steeper gray level changes providing stronger edge points (larger magnitudes)

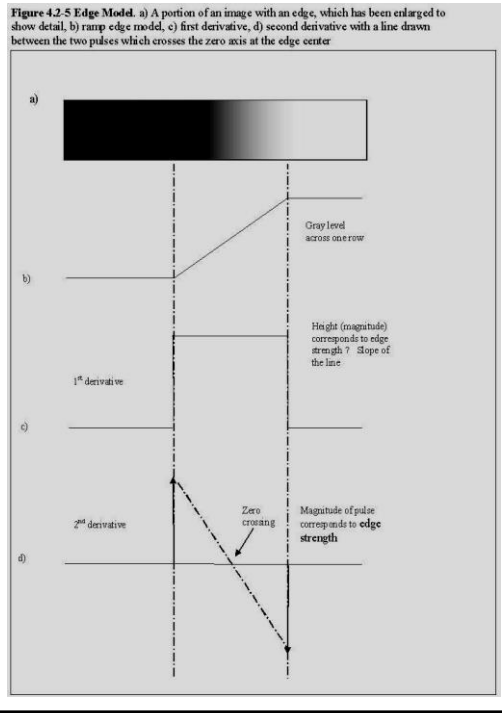


Figure 4.2-5 Edge Model. a) A portion of an image with an edge, which has been enlarged to show detail, b) ramp edge model, c) first derivative, d) second derivative with a line drawn between the two pulses which crosses the zero axis at the edge center

- The second derivative returns two impulses, one on either side of the edge, which allows to measure edge location to sub-pixel accuracy
- *Sub-pixel accuracy* refers to the fact that the zero-crossing may be at a fractional pixel distance, for example halfway between two pixels, so we could say the edge is at, for instance, $c = 75.5$

- ***Roberts operator****:*
  - A simple approximation to the first derivative
  - Marks edge points only; it does not return any information about the edge orientation
  - Simplest of the edge detection operators and will work best with binary images

- There are two forms of the Roberts operator:

1. The

2. $$\sqrt{[I(r,c) - I(r-1,c-1)]^2 + [I(r,c-1) - I(r-1,c)]^2}$$

$$|I(r,c) - I(r-1,c-1)| + |I(r,c-1) - I(r-1,c)|$$

- **The second form of the equation is often used in practice due to its computational efficiency**

- ***Sobel operator**:*

- Approximates the gradient by using a row and a column mask, which approximates the first derivative in each direction

- The Sobel edge detection masks find edges in both the horizontal and vertical directions, and then combine this information into a single metric

- The Sobel masks are as follows:

VERTICAL EDGE EDGE

HORIZONTAL

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

- The Sobel masks are each convolved with the image
- At each pixel location there are two numbers:
  1. s_1, corresponding to the result from the vertical edge mask, and
  2. s_2, from the horizontal edge mask

- s_1 and s_2 are used to compute two metrics, the edge magnitude and the edge direction, defined as follows:

  1. EDGE MAGNITUDE: $\sqrt{s_1^2 + s_2^2}$

  2. EDGE DIRECTION: $\text{Tan}^{-1}\left[\frac{s_1}{s_2}\right]$

- ***Prewitt operator****:*

- Approximates the gradient by using a row and a column mask, which approximates the first derivative in each direction

- The Prewitt edge detection masks look for edges in both the horizontal and vertical directions, and then combine this information into a single metric

- The Prewitt masks are as follows:

VERTICAL EDGE HORIZONTAL EDGE

$$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \qquad \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$$

- The Prewitt masks are each convolved with the image
- At each pixel location there are two numbers:
  1. p_1, corresponding to the result from the vertical edge mask, and
  2. p_2, from the horizontal edge mask

- p_1 and p_2 are used to compute two metrics, the edge magnitude and the edge direction, defined as follows:

1. EDGE MAGNITUDE: $\sqrt{p_1^2 + p_2^2}$

2. EDGE DIRECTION: $\text{Tan}^{-1}\left[\frac{p_1}{p_2}\right]$

- The Prewitt is simpler to calculate than the Sobel, since the only coefficients are 1's, which makes it easier to implement in hardware

- However, the Sobel is defined to place emphasis on the pixels closer to the mask center, which may desirable for some applications

- ***Laplacian operators****:*

- These are two-dimensional discrete approximations to the second derivative

- Implemented by applying *one* of the following convolution masks:

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \quad \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad \begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix}$$

- The Laplacian masks are *rotationally symmetric*, which means edges at all orientations contribute to the result

- Applied by selecting *one* mask and convolving it with the image

- The sign of the result (positive or negative) from two adjacent pixel locations provides directional information, and tells us which side of the edge is brighter

- These masks differ from the Laplacian-type previously described in Chapter 3 in that the center coefficients have been decreased by one, as we are trying to find edges, and are not interested in the image itself
- If we increase the center coefficient by one it is equivalent to adding the original image to the edge detected image

- ***Compass Masks****:*
- The *Kirsch* and *Robinson* edge detection masks are called compass masks since they are defined by taking a single mask and rotating it to the eight major compass orientations: North, Northwest, West, Southwest, South, Southeast, East, and Northeast

- The ***Kirsch compass masks*** are defined as follows:

$$k_0\begin{bmatrix}-3 & -3 & 5\\ -3 & 0 & 5\\ -3 & -3 & 5\end{bmatrix} \quad k_1\begin{bmatrix}-3 & 5 & 5\\ -3 & 0 & 5\\ -3 & -3 & -3\end{bmatrix} \quad k_2\begin{bmatrix}5 & 5 & 5\\ -3 & 0 & -3\\ -3 & -3 & -3\end{bmatrix} \quad k_3\begin{bmatrix}5 & 5 & -3\\ 5 & 0 & -3\\ -3 & -3 & -3\end{bmatrix}$$

$$k_4\begin{bmatrix}5 & -3 & -3\\ 5 & 0 & -3\\ 5 & -3 & -3\end{bmatrix} \quad k_5\begin{bmatrix}-3 & -3 & -3\\ 5 & 0 & -3\\ 5 & 5 & -3\end{bmatrix} \quad k_6\begin{bmatrix}-3 & -3 & -3\\ -3 & 0 & -3\\ 5 & 5 & 5\end{bmatrix} \quad k_7\begin{bmatrix}-3 & -3 & -3\\ -3 & 0 & 5\\ -3 & 5 & 5\end{bmatrix}$$

- The edge magnitude is defined as the maximum value found by the convolution of each of the masks with the image
- The edge direction is defined by the mask that produces the maximum magnitude
- For instance, k_0 corresponds to a horizontal edge, whereas k_5 corresponds to a diagonal edge in the Northeast/Southwest direction

- The ***Robinson compass masks*** are used in a manner similar to the Kirsch masks
- They are easier to implement, as they rely only on coefficients of 0, 1, and 2, and are symmetrical about their directional axis (the axis with the zeros which corresponds to the line direction)

- The Robinson masks are as follows:

$$r_0\begin{bmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{bmatrix} \quad r_1\begin{bmatrix}0 & 1 & 2\\ -1 & 0 & 1\\ -2 & -1 & 0\end{bmatrix} \quad r_2\begin{bmatrix}1 & 2 & 1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{bmatrix} \quad r_3\begin{bmatrix}2 & 1 & 0\\ 1 & 0 & -1\\ 0 & -1 & -2\end{bmatrix}$$

$$r_4\begin{bmatrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{bmatrix} \quad r_5\begin{bmatrix}0 & -1 & -2\\ 1 & 0 & -1\\ 2 & 1 & 0\end{bmatrix} \quad r_6\begin{bmatrix}-1 & -2 & -1\\ 0 & 0 & 0\\ 1 & 2 & 1\end{bmatrix} \quad r_7\begin{bmatrix}-2 & -1 & 0\\ -1 & 0 & 1\\ 0 & 1 & 2\end{bmatrix}$$

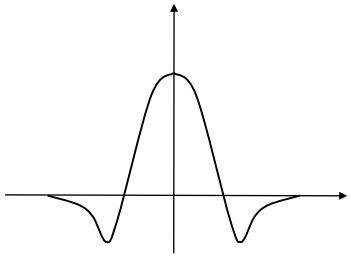
- The edge magnitude is defined as the maximum value found by the convolution of each of the masks with the image
- The edge direction is defined by the mask that produces the maximum magnitude
- Any of the edge detection masks can be extended by rotating them in a manner like the compass masks, which allows us to extract explicit information about edges in any direction

✓**Advanced Edge Detectors**

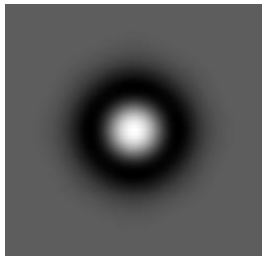
- These edge detectors are considered to be advanced because they are algorithmic in nature
- Except for the Frei-Chen masks, these algorithms begin with the idea that, most edge detectors are too sensitive to noise and by blurring the image prior to edge detection we can mitigate these noise effects

- **Laplacian of a Gaussian** (LoG):
- It consists of two steps:
  1. Convolve the image with a Gaussian smoothing filter
  2. Convolve the image with a Laplacian mask

Figure 4.2-6: Laplacian of a Gaussian



a) One-dimensional plot of the LoG function



b) The LoG as an image with white representing positive numbers, black negative numbers, and gray representing zero

- Pre-processing with a smoothing filter can mitigate noise effects and then the Laplacian operator can enhance the edges
- This operator is also called the *Mexican hat* operator, since the function resembles a sombrero
- Since the process requires the successive convolution of two masks, they can be combined into one *LoG* mask

- A common 5x5 mask that approximates the combination of the Gaussian and Laplacian into one convolution mask is as follows:

$$\begin{bmatrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{bmatrix}$$

- An interesting aspect of the *LoG* operator is that it is believed to closely model biological vision systems

- ***Canny algorithm****:*

- Developed by John Canny in 1986

- An optimal edge detection method based on a specific mathematical model for edges

- The edge model is a step edge corrupted by Gaussian noise

- Consists of four primary steps

- The Canny algorithm:

1. *Apply a Gaussian filter mask to smooth the image* to mitigate noise effects
2. *Find the magnitude and direction of the gradient* using equations similar to the Sobel or Prewitt edge detectors
3. *Apply nonmaxima suppression* which results in thinned edges
4. *Apply two thresholds* to obtain the final result – *hysteresis thresholding*

Nonmaxima suppression example

$$\begin{bmatrix} \leftarrow 50 & 112 \rightarrow & 20 \rightarrow \\ \leftarrow 40 & 100 \rightarrow & 91 \rightarrow \\ \leftarrow 88 & 95 \rightarrow & 92 \rightarrow \end{bmatrix}$$

A 3x3 subimage of the magnitude image, which consists of the magnitude results in an image grid. The arrows show the gradient directions. This particular subimage has a vertical line (a horizontal edge). To apply nonmaxima suppression we compare the center pixel magnitude along the gradient direction. Here, the 100 is compared with the 40 and the 91. Since it is a local maximum, it is retained as an edge pixel.

Hysteresis thresholding

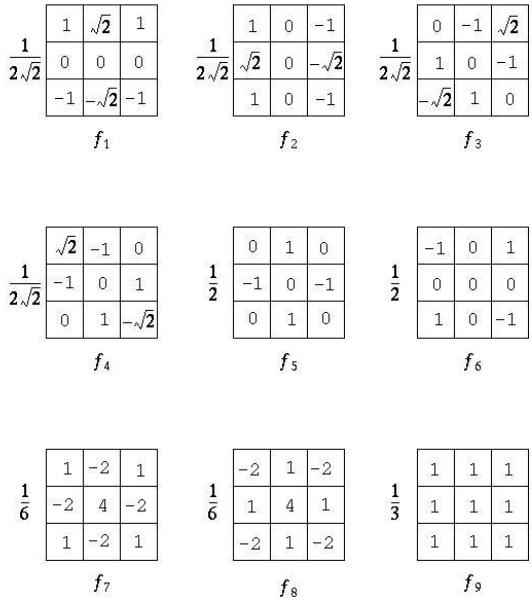
- Uses two thresholds, a high and a low
- Marks pixels above the high threshold
- Applies the low threshold to pixels connected to the pixels marked with the high threshold
- This helps to avoid false edges caused by too low a threshold value or missing edges caused by too high a value

- ***Boie-Cox algorithm****:*
  - A generalization of the Canny algorithm
  - Consists of similar steps, but uses methods to allow for a more generalized edge model (Matched & Wiener filters)
- ***Shen-Castan algorithm****:*
  - Another extension of the Canny algorithm, developed as an optimal solution to a specific mathematical model

- ***Frei-Chen masks***
  - They form a complete set of basis vectors, which means any 3x3 subimage can be represented as a weighted sum of the nine Frei-Chen masks
  - The weights are found by projecting the subimage onto each basis vector

- The projection process is similar to the convolution process in that, both overlay the mask on the image, multiply coincident terms, and sum the results – a *vector inner product*
- The Frei-Chen masks can be grouped into a set of four masks for an edge subspace, four masks for a line subspace, and one mask for an average subspace

Figure 4.2-8: Frei-Chen Masks



The first four masks comprise the edge subspace.
The next four masks comprise the line subspace.
The final mask is the average subspace.

EXAMPLE 4.2.1:

Suppose we have the following subimage, I_s:

$$I_s = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix}$$

To project this subimage onto the Frei-Chen masks, start by finding the projection onto f_1. Overlay the subimage on the mask and consider the first row. The 1 in the upper left corner of the subimage coincides with the 1 in the upper left corner of the mask, the 0 is over the $\sqrt{2}$, and the 1 on the upper right corner of the subimage coincides with the 1 in the mask. Note that all these must be summed and then multiplied by the $\frac{1}{2\sqrt{2}}$ factor to normalize the masks. The projection of I_s onto f_1 is equal to:

$$\frac{1}{2\sqrt{2}}[1(1)+0(\sqrt{2})+1(1)+1(0)+0(0)+1(0)+1(-1)+0(-\sqrt{2})+1(-1)]=0$$

If we follow this process and project the subimage, I_s, onto each of the Frei-Chen masks, we get the following:

$$f_1 \to 0,\ f_2 \to 0,\ f_3 \to 0,\ f_4 \to 0,\ f_5 \to -1,\ f_6 \to 0,\ f_7 \to 0,\ f_8 \to -1,\ f_9 \to 2.$$

- To get our image back we multiply the weights (projection values) by the basis vectors (the Frei-Chen masks)

For this example the only nonzero terms correspond to masks f_5, f_8 and f_9, and we find the following:

$$(-1)\left(\frac{1}{2}\right)\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & -1 \\ 0 & 1 & 0 \end{bmatrix} + (-1)\left(\frac{1}{6}\right)\begin{bmatrix} -2 & 1 & -2 \\ 1 & 4 & 1 \\ -2 & 1 & -2 \end{bmatrix} + (2)\left(\frac{1}{3}\right)\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{bmatrix} = I_S$$

- This illustrates what is meant by a complete set of basis vectors allowing us to represent subimage by a weighted sum

- To use the Frei-Chen masks for edge detection, select a particular subspace of interest and find the relative projection of the image onto the particular subspace

$$\cos(\Theta) = \sqrt{\frac{M}{S}}$$

Where:

$$M = \sum_{k \in \{e\}} (I_s, f_k)^2$$

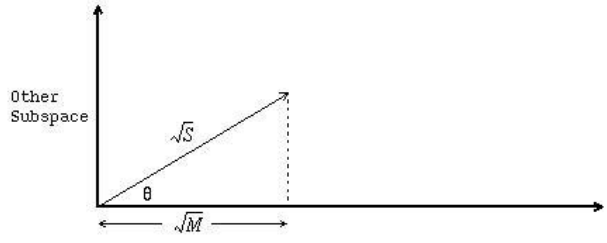
$$S = \sum_{k=1}^{9} (I_s, f_k)^2$$

The set $\{e\}$ consists of the masks of interest

The $\langle I_s, f_k \rangle$ notation refers to the process of vector inner product

- The advantage of this method is that we can select particular edge or line masks of interest, and consider the projection of those masks only

Figure 4.2-9: FREI-CHEN PROJECTION



$$\cos\theta = \sqrt{\frac{M}{S}}$$

A 2-D representation of the Frei-Chen projection concept. The actual Frei-Chen space is nine-dimensional, where each dimension is given by one of the masks, f_k

✓ Edges in Color Images

- Color images typically consist of three bands – one each for red, green and blue
- Edge detection can be performed on the original RGB data, or after mapping into another color space
- The specific method is application dependent, five example methods follow

1. Extract the luminance or brightness information and apply a monochrome edge detection method

- The brightness information obtained by:
  - Averaging the RGB components:
    $L = (R+G+B)/3$,
  - Luminance equation:
    $0.299R+0.587G+0.114B$, or
  - Vector length: $L = \text{sqrt}(R^2+G^2+B^2)$

2. Apply a monochrome edge detection method to each of the RGB bands separately and then combine the results into a composite image

3. Apply a monochrome edge detection method to each of the RGB bands separately and then retain the maximum value at each location

4. Apply a monochrome edge detection method to each of the RGB bands separately and then select specific criteria at each pixel location in order to find an edge point

5. Find the minimum given by the following two equations in order to find edges in a multispectral images:

$$\frac{\sum_{b=1}^{n}\left[I_b(r,c)-\bar{I}(r,c)\right]\left[I_b(r+1,c+1)-\bar{I}(r+1,c+1)\right]}{\sqrt{\sum_{b=1}^{n}\left[I_b(r,c)-\bar{I}(r,c)\right]^2\sum_{b=1}^{n}\left[I_b(r+1,c+1)-\bar{I}(r+1,c+1)\right]^2}}$$

$$\frac{\sum_{b=1}^{n}\left[I_b(r+1,c)-\bar{I}(r+1,c)\right]\left[I_b(r,c+1)-\bar{I}(r,c+1)\right]}{\sqrt{\sum_{b=1}^{n}\left[I_b(r+1,c)-\bar{I}(r+1,c)\right]^2\sum_{b=1}^{n}\left[I_b(r,c+1)-\bar{I}(r,c+1)\right]^2}}$$

Where:

$\bar{I}(r,c)$ is the arithmetic average of all the pixels in all bands at pixel location (r,c)

$I_b(r,c)$ is the value at location (r,c) in the b^{th} band, with a total of n bands

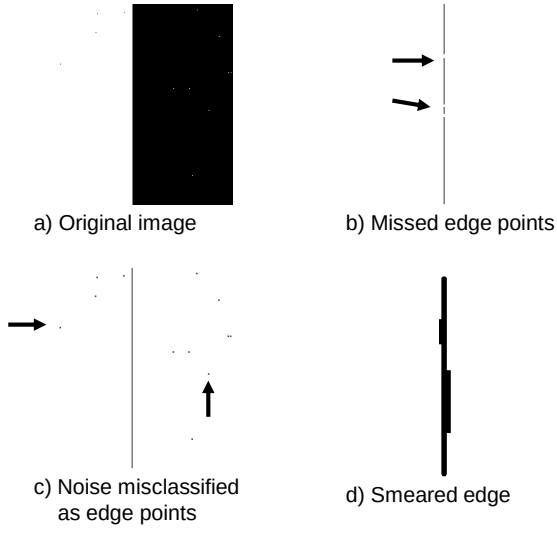
✓Edge Detector Performance

- Objective and subjective evaluations can be useful
- Objective metrics allow us to compare different techniques with fixed analytical methods
- Subjective methods often have unpredictable results

- To develop a performance metric for edge detection operators, we need to consider the types of errors that can occur and define what constitutes success
- Success criteria used in development of Canny algorithm
  - *Detection* – find all real edges, no false edges
  - *Localization* – found in correct location
  - *Single response* – no multiple edges found for single edge

- *Pratt's Figure of Merit (FOM):*
- Pratt first considered the types of errors that can occur with edge detection
- Types of errors:
  1. *classifying noise as valid edge points*
  2. *missing valid edge points*
  3. *smearing of edges*
- If these errors do not occur, we can say that we have achieved success

Figure 4.2-10: Errors in Edge Detection



The Pratt FOM, is defined as follows:

$$FOM = \frac{1}{I_N} \sum_{i=1}^{I_F} \frac{1}{1 + \alpha d_i^2}$$

I_N = the maximum of I_I and I_F

I_I = the number of ideal edge points in the image

I_F = the number of edge points found by the edge detector

α = a scaling constant that can be adjusted to adjust the penalty for offset edges

d_i = the distance of a found edge point to an ideal edge point

- For this metric, *FOM* will be 1 for a perfect edge
- Normalizing to the maximum of the ideal and found edge points guarantees a penalty for smeared edges or missing edge points
- In general, this metric assigns a better rating to smeared edges than to offset or missing edges

- The distance measure can be defined in one of three ways:

1. *City block distance*, four connectivity:

$$d = |r_1 - r_2| + |c_1 - c_2|$$

2. *Chessboard distance,* 8-connectivity:

$$d = \max(|r_1 - r_2|, |c_1 - c_2|)$$

3. *Euclidean distance*, physical distance:

$$d = \left[(r_1 - r_2)^2 + (c_1 - c_2)^2\right]^{1/2}$$

EXAMPLE 4.2.2:

Given the following image array, find the Figure of Merit for the following found edge points, designated by 1's, in a), b), and c). Let $\alpha = 0.5$, and use the city block distance measure. We assume that actual edge in the locations where the line appears, that is, at the 100's.

Image Array $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 100 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

a) $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ b) $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ c) $\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$

a) $FOM = \frac{1}{I_N} \sum_{i=1}^{I_F} \frac{1}{1+\alpha d_i^2} = \frac{1}{3}\left[\frac{1}{1+0.5(0)^2} + \frac{1}{1+0.5(0)^2} + \frac{1}{1+0.5(0)^2}\right] = 1$

EXAMPLE 4.2.2 (contd):

b) $FOM = \frac{1}{I_N} \sum_{i=1}^{I_F} \frac{1}{1+\alpha d_i^2} =$

$$\frac{1}{6}\left[\frac{1}{1+0.5(0)^2} + \frac{1}{1+0.5(0)^2} + \frac{1}{1+0.5(0)^2} + \frac{1}{1+0.5(1)^2} + \frac{1}{1+0.5(1)^2} + \frac{1}{1+0.5(1)^2}\right] \approx 0.8333$$

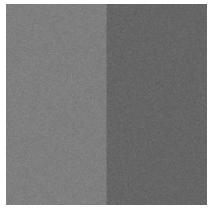
c) $FOM = \frac{1}{I_N} \sum_{i=1}^{I_F} \frac{1}{1+\alpha d_i^2} =$

$$\frac{1}{4}\left[\frac{1}{1+0.5(1)^2} + \frac{1}{1+0.5(1)^2} + \frac{1}{1+0.5(1)^2} + \frac{1}{1+0.5(2)^2}\right] \approx 0.5833$$

With result a), we find a perfect edge. In result b), we see that a smeared edge provides us with about 83%, and an offset edge in c) gives us about 58%. Note that the α parameter can be adjusted to determine the penalty for offset edges

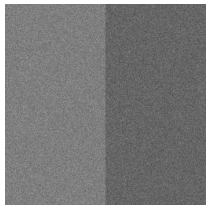
Figure 4.2-11: Pratt Figure of Merit

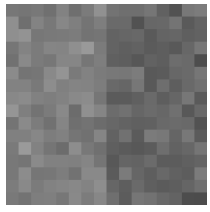
a) Original test image



b) Test image with added Gaussian noise with a variance of 25

c) Test image with added Gaussian noise, variance 100

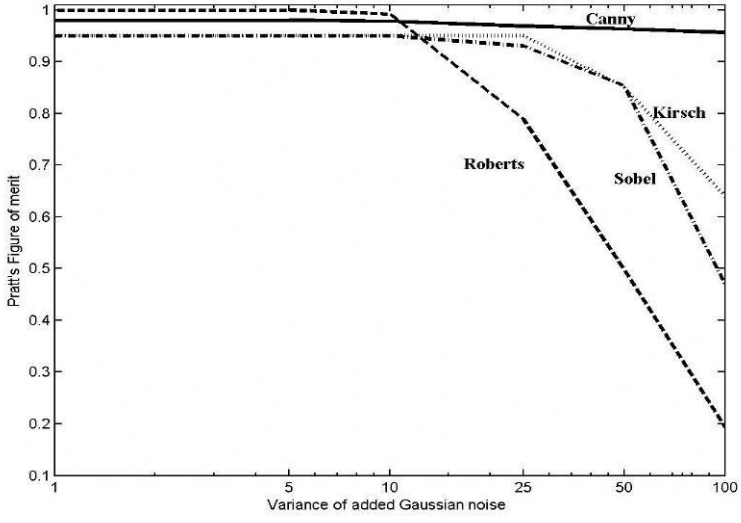




d) A 16x16 subimage cropped from image (c), enlarged to show that the edge is not as easy to find at the pixel level

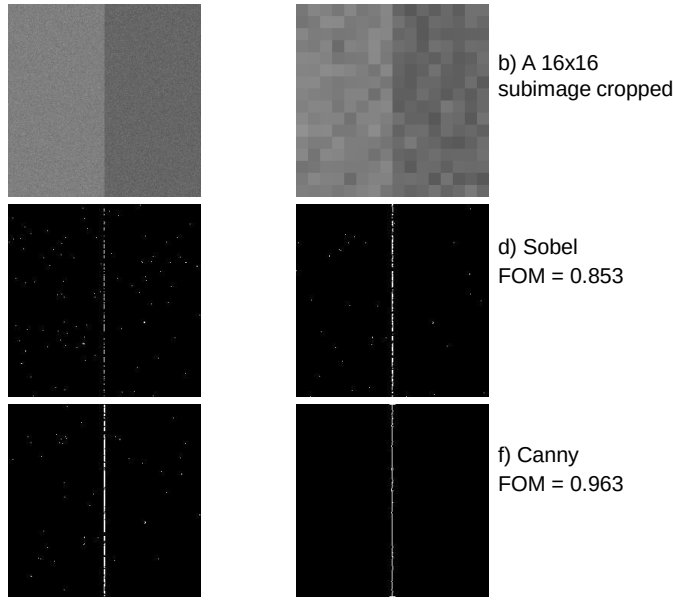
- Note: The original test image has a gray level of 127 on the left and 102 on right

Figure 4.2-11: Pratt Figure of Merit (contd)



e) This graph shows that as the noise variance increases the Canny has the best performance

Figure 4.2-12: Pratt Figure of Merit Images



a) Gaussian noise variance of 50

b) A 16x16 subimage cropped

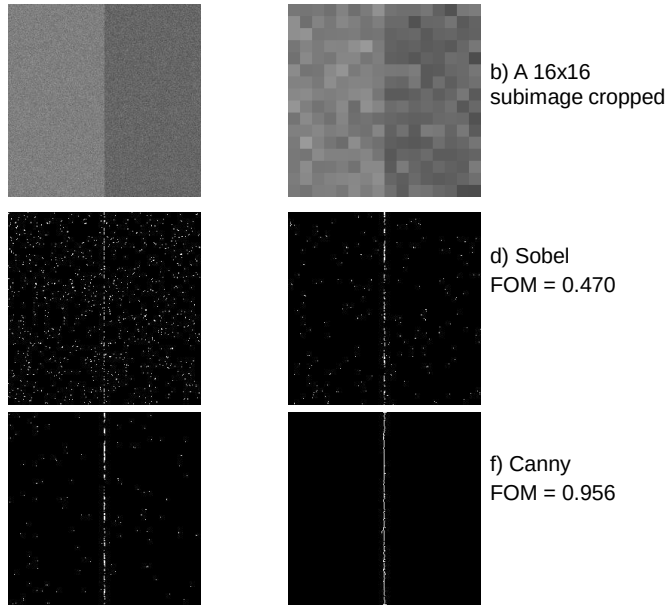
c) Roberts FOM = 0.4977

d) Sobel FOM = 0.853

e) Kirsch FOM = 0.851

f) Canny FOM = 0.963

Figure 4.2-12: Pratt Figure of Merit Images



a) Gaussian noise variance of 100

b) A 16x16 subimage cropped

c) Roberts FOM = 0.1936

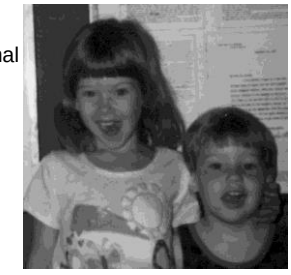
d) Sobel FOM = 0.470

e) Kirsch FOM = 0.640

f) Canny FOM = 0.956

Figure 4.2-13: Edge Detection Examples

a) Original image



b) Roberts operator

c) Sobel operator

d) Prewitt operator

Figure 4.2-13: Edge Detection Examples (contd)

e) Laplacian operator

f) Kirsch operator

g) Robinson operator

- Edge detector results are not as good when noise is added to the image
- To mitigate noise effects, preprocessing the image with mean, or averaging, spatial filters can be done
- Additionally, the size of the edge detection masks can be extended for noise mitigation

- An example of this method is to extend the Prewitt edge mask as follows:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \end{bmatrix}$$

- Can be rotated and used like the Prewitt for edge magnitude and direction
- Called *boxcar operators* and can be extended, 7x7, 9x9 and 11x11 are typical

- The Sobel operator can be extended in a similar manner:

$$\begin{bmatrix} -1 & -1 & -1 & -2 & -1 & -1 & -1 \\ -1 & -1 & -1 & -2 & -1 & -1 & -1 \\ -1 & -1 & -1 & -2 & -1 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 & 1 \\ 1 & 1 & 1 & 2 & 1 & 1 & 1 \end{bmatrix}$$

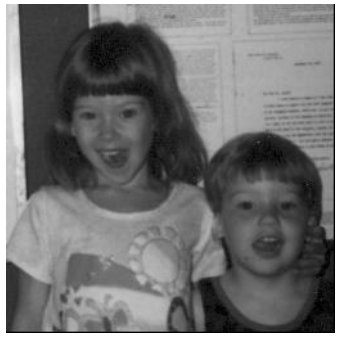
- Can be rotated and used for edge magnitude and direction as 3x3 Sobel

- *Truncated pyramid* operator can be obtained by approximating a linear distribution:

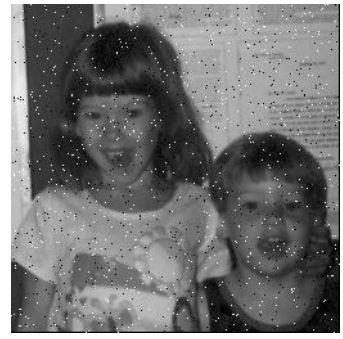
$$\begin{bmatrix} 1 & 1 & 1 & 0 & -1 & -1 & -1 \\ 1 & 2 & 2 & 0 & -2 & -2 & -1 \\ 1 & 2 & 3 & 0 & -3 & -2 & -1 \\ 1 & 2 & 3 & 0 & -3 & -2 & -1 \\ 1 & 2 & 3 & 0 & -3 & -2 & -1 \\ 1 & 2 & 2 & 0 & -2 & -2 & -1 \\ 1 & 1 & 1 & 0 & -1 & -1 & -1 \end{bmatrix}$$

- **This operator provides weights that decrease from the center pixel, which will smooth the result in a more natural manner**
- **Used like Sobel for magnitude and direction**

Figure 4.2-14: Edge Detection examples – Noise

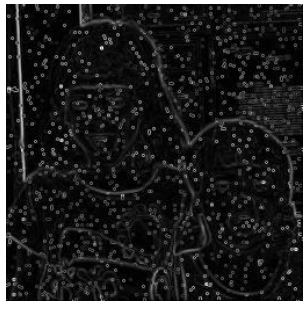


a) Original image



b) Image with added noise

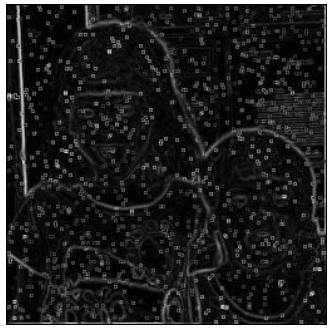
Figure 4.2-14: Edge Detection examples – Noise (contd)



c) Sobel with a 3x3 mask

d) Sobel with a 7x7 mask

Figure 4.2-14: Edge Detection examples – Noise (contd)



e) Prewitt with a 3x3 mask

f) Prewitt with a 7x7 mask

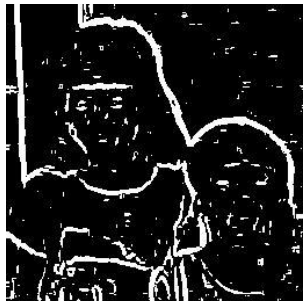
Figure 4.2-14: Edge Detection examples – Noise (contd)

g) Result from applying a threshold to the 3x3 Prewitt

h) Result from applying a threshold to the 7x7 Prewitt

Figure 4.2-14: Edge Detection examples – Noise (contd)

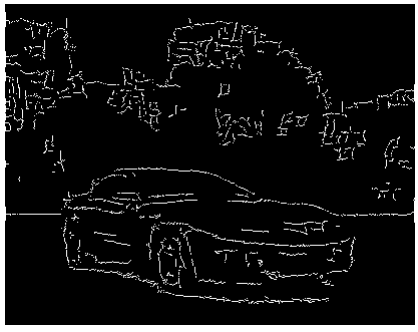
i) Truncated pyramid with a 7x7 mask



j) Results from applying a threshold to the 7x7 truncated pyramid

Figure 4.2-15: Advanced Edge Detectors with Noisy Images

a) Original image with salt-and-pepper noise added with a probability of 3% each



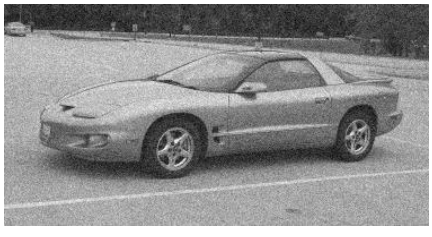
b) Canny results, parameters:
% Low Threshold = 1,
% High Threshold = 2,
Variance = 2

Figure 4.2-15: Advanced Edge Detectors with Noisy Images (contd)

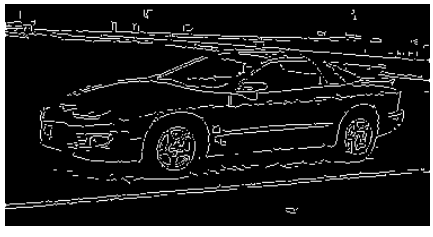
c) Frei-Chen results, parameters:
Gaussian2 prefilter,
max(edge,line),
post-threshold = 190

d) Shen-Castan results, parameters:
% Low Threshold = 1,
% High Threshold = 2,
Smooth factor = 0.9,
Window size = 7,
Thin Factor = 1

Figure 4.2-15: Advanced Edge Detectors with Noisy Images (contd)

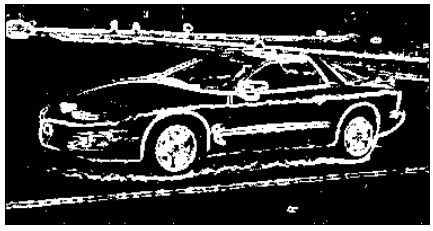


e) Original image with zero-mean Gaussian noise with a variance of 200 added

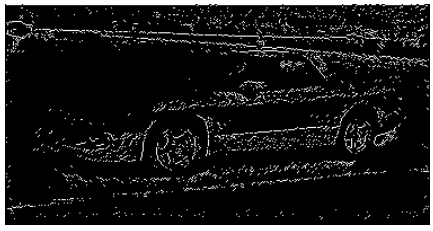


f) Canny results, parameters:
% Low Threshold = 1,
% High Threshold = 1,
Variance = 0.8

Figure 4.2-15: Advanced Edge Detectors with Noisy Images (contd)



g) Frei-Chen results, parameters:
Gaussian2 prefilter, max(edge,line),
post-threshold = 70



h) Shen-Castan results, parameters:
% Low Threshold = 1,
% High Threshold = 2,
Smooth factor = 0.9,
Window size = 7,
Thin Factor = 1

Robert Edge Detector

Sobel Edge Detector

Prewitt Detector

Laplacian Detector

Kirch Detector

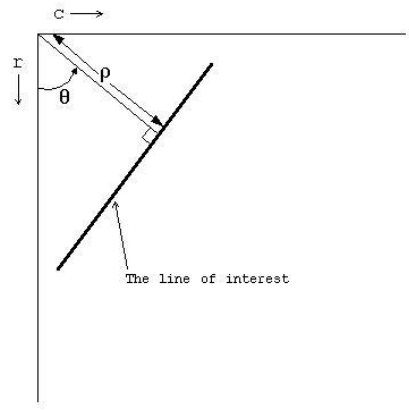
Robinson Detector

✓**Hough Transform**

- Designed specifically to find lines, where a *line* is a collection of edge points that are adjacent and have the same direction
- The Hough transform that takes a collection of *n* edge points, and efficiently finds all the lines on which the edge points lie
- Without the Hough algorithm it takes about n^3 comparisons to compare all points to all lines

- The advantage of the Hough transform is that it provides parameters to reduce the search time for finding lines, with a given set of edge points
- The parameters allow for the quantization of the line search space
- These parameters can be adjusted based on application requirements

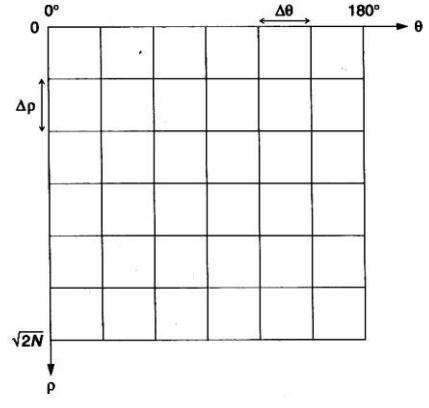
Figure 4.2-16: HOUGH TRANSFORM



The *normal* (perpendicular) representation of a line is given as:

$$\rho = r\cos(\theta) + c\sin(\theta)$$

Figure 4.2.17: Hough space



- The Hough transform works by quantizing ρ and θ

- The algorithm consists of three primary steps:

1) Define the desired increments on ρ and θ, Δ_ρ *and* Δ_θ, and quantize the space accordingly.

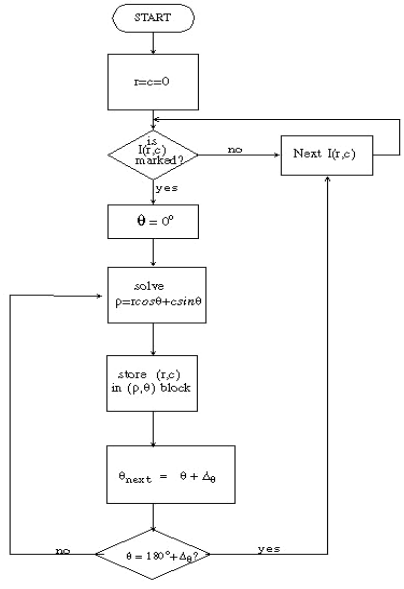
2) For every point of interest (typically points found by edge detectors that exceed some threshold value), plug the values for r and c into the line equation:

$$\rho = r\cos(\theta) + c\sin(\theta)$$

Then, for each value of θ in the quantized space, solve for ρ.

3) For each $\rho\,\theta$ pair from step 2, record the r and c pair in the corresponding block in the quantized space. This constitutes a hit for that particular block

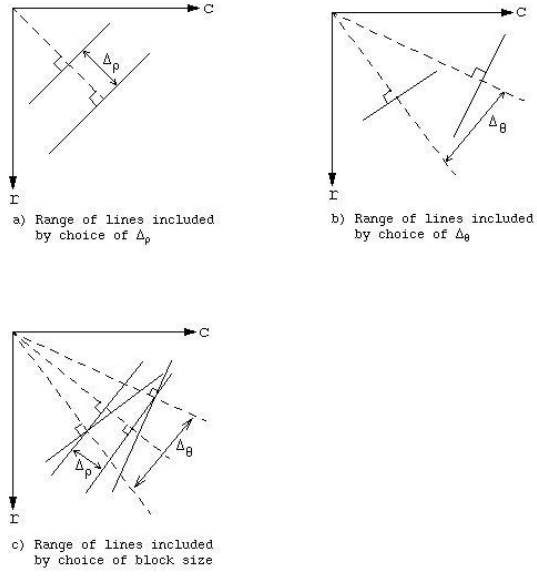
Figure 4.2-18: Hough Transform Flowchart



The flowchart is followed until all $I(r,c)$ have been examined

- When this process is completed, the number of hits in each block corresponds to the number of pixels on the line as defined by the values of ρ and θ in that block
- The advantage of large quantization blocks is that the search time is reduced, but the price paid is less line resolution in the image space

Figure 4.2-19: EFFECTS OF QUANTIZATION BLOCK SIZE FOR HOUGH TRANSFORM



a) Range of lines included by choice of Δ_ρ

b) Range of lines included by choice of Δ_θ

c) Range of lines included by choice of block size

- A threshold is selected and the quantization blocks that contain more points than the threshold are examined
- Next, line continuity is considered by searching for gaps in the line by finding the distance between points on the line (remember the points on a line correspond to points recorded in the block)
- When this process is completed, the lines are marked in the output image

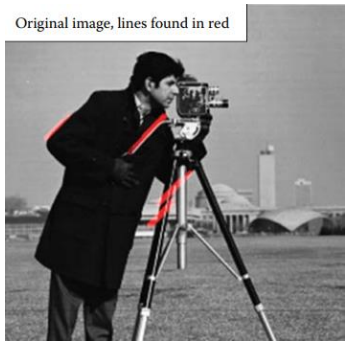
- A more advanced post-processing algorithm is implemented in CVIPtools with the Hough transform
- The algorithm works as follows:

1. Perform the Hough transform on the input image containing marked edge points, which we will call image1. The result, image2, is an image in Hough space quantized by the parameter *delta length* (ρ) and *delta angle* (fixed at one degree in CVIPtools)

2. Threshold image2 by using the parameter *line pixels*, which is the minimum number of pixels in a line (in one quantization box in Hough space), and do the inverse Hough transform. This result, image3, is a mask image with lines found in the input image at the specified angle(s), illustrated in Figure 4.2-20c. Note that these lines span the entire image
3. Perform a logical operation, image1 AND image3. The result is image4, see Figure 4.2-20d

4. Apply an *edge linking* process to image4 to connect line segments; specifically we implemented a *snake eating algorithm*. This works as follows:
   a) A line segment is considered to be a snake. It can eat another snake within *connect distance* along *line angles*, and becomes longer (see Figure 4.2-20e). This will connect disjoint line segments
   b) If a snake is too small, less than *segment length*, it will be extinct. This will remove small segments. The output from the snake eating algorithm is the final result, illustrated in Figure 4.2-20f

Hough transform post processing algorithm details

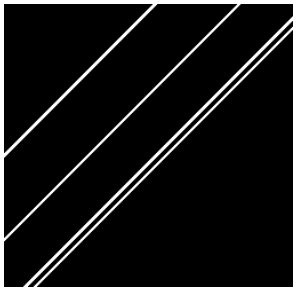


a) Original image

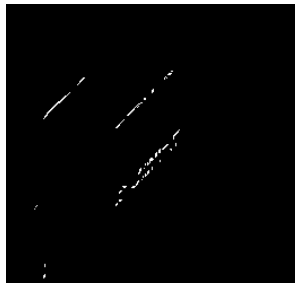
b) Image after applying the Kirsch edge operator and a threshold operation

The Hough parameters used are as follows: *Line Angles*: 45 degrees, *Line Pixels (min)*: 25, *Connect distance (max):* 5, *Delta Length*: 1, *Segment Length (min*): 15

Hough transform post processing algorithm details (contd)



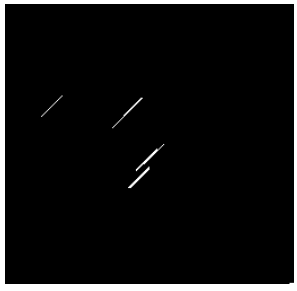
c) The mask image created from the Hough result for lines at 45 degrees



d) Result of logical AND of the images in (b) and (c)

Hough transform post processing algorithm details (contd)

e) Image (d) after snake eating, see that the camera's handle has been connected



f) The final result after snake extinction, small dashed lines are removed

Note: We have four lines, starting from the upper left: one line corresponding to the lower part of the arm above the elbow, note that the upper part of the arm is missing as it is not quite at 45 degrees; one line for the camera handle; the next line corresponds to the part of his other arm from elbow to wrist, and the last line (the lower one) that is not a true line in the image but is created by a combination of the edge detail in that area and using a connect distance of 5

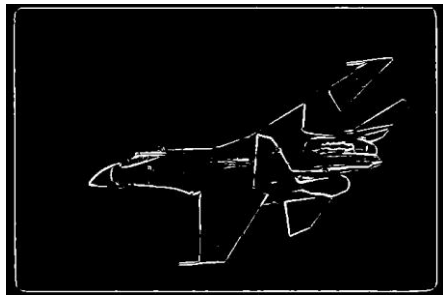
- CVIPtools parameters for the Hough transform

  - *Line Angles*: The range of angles for which the Hough transform will search. In CVIPtools Δ_θ is fixed at one degree
  - *Line Pixels (min)*: The minimum number of pixels a line must possess to be retained, also referred to as the threshold value in the Hough image
  - *Connect distance (max)*: Controls how far apart two line segments can be and still be connected (snake eating)
  - *Delta Length*: Quantizes the Hough space ρ parameter. Controls how "thick" a line can be; note that a "thick" line might consist of multiple separate lines if they are in close proximity
  - *Segment Pixels (min)*: The minimum number of pixels in a line segment for it to be retained

    Note: *Segment Pixels* controls how many pixels a solid line must have while *Line Pixels* controls how many pixels a dashed line must have

Hough Transform

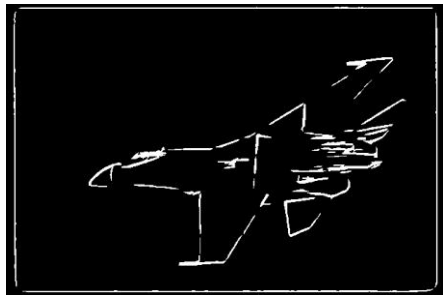
a) Original image



b) Sobel edge operator followed by a thresholding

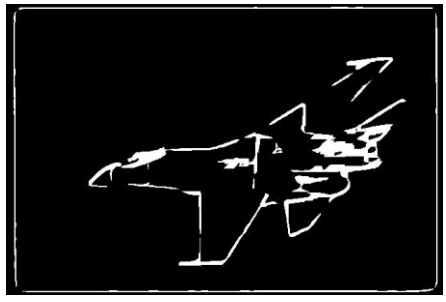
Figure 4.2-21: Hough Transform (contd)

c) Hough output with the range of line angles = 0 to 45 degrees, delta length (ρ) =1, minimum number of pixels per line = 20, maximum connect distance = 2, minimum segment size = 10

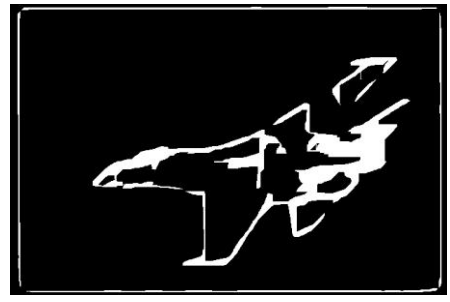


d) Hough output with same parameters as (c) except range of angles from 0 to 90 degrees

Figure 4.2-21: Hough Transform (contd)



e) Hough output with same parameters as (d) except connect distance = 5



f) Hough output with same parameters as (e) except connect distance = 10

Corner Detection

- **Edges**: image brightness changes in a specific direction
- **Lines/curves:** collection of edge points along a specific path
- **Corners**: points with high rate of change in more than one direction.

Corner Detection Uses

- Object tracking
- Object orientation
- Matching multiple images
- Object boundary cues for HVS
- Corner features are robust

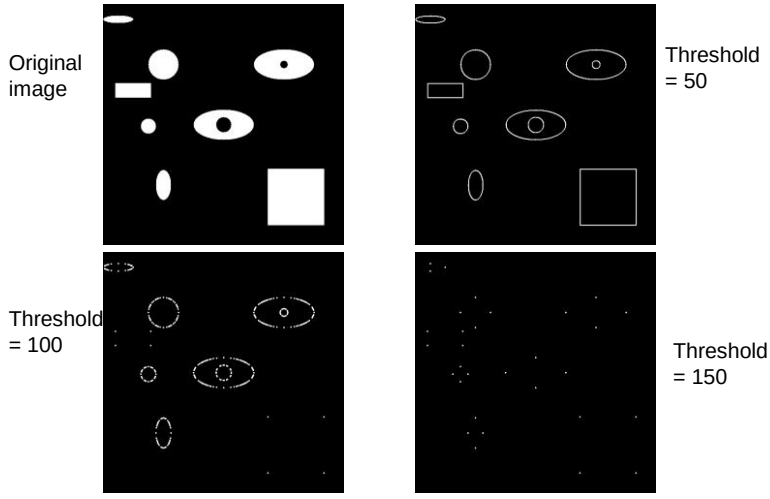
Moravec Corner Detector

- Simplest, finds points of max contrast

$$MD[I(r,c)] = \frac{1}{8}\sum_{i=r-1}^{r+1}\sum_{k=c-1}^{c+1}|I(r,c) - I(i,j)|$$

- Finds average difference in 8 directions
- Threshold after operation

Moravec Corner Detector



Harris Corner Detection Algorithm

Corner response function – CRF(r,c)

$$CRF(r,c) = [G(p_1^2)G(p_2^2) - [G(p_1p_2)]^2] - \propto [G(p_1) + G(p_2)]^2$$

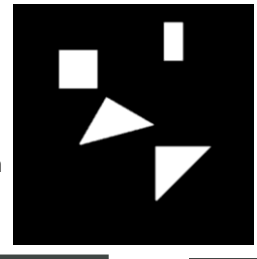
- *P1* and *P2* from Prewitt edge detector
- G(◦) represents the result after convolution with a Gaussian
- Sensitivity of detector – larger the lesser

Harris Corner Detection Algorithm

1. Blur with Gaussian
2. Find gradient (brightness) in 2 perpendicular directions, e.g. Prewitt
3. Blur results from step (2)
4. Find corner response function (CRF)
5. Threshold and apply nonmax suppression

Harris Corner Detector

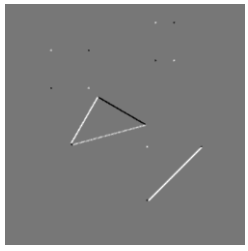
Original image After 5x5 Gaussian



$G(p_1^2)$

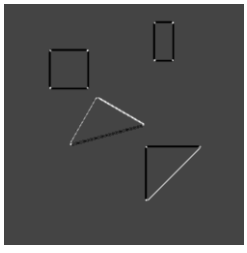
$G(P2^2)$

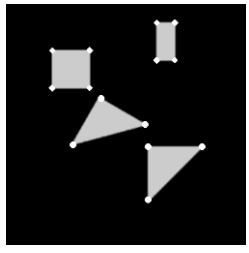
Harris Corner Detector cont.



Gaussian of product of gradients – $G(p_1p_2)$

CRF Response





Final after threshold and nonmax suppression (overlaid on original)

- The extended Hough transform can be applied to any geometric shape that can described by an equation of the following form:

$$f(r,c;\overline{p}) = 0$$

Where $f(.)$ is any function of the row and column coordinates, (r,c), and a parameter vector $\overline{p}$

In the case of the line finding Hough transform, the function is:

$\rho = r \cos(\theta) + c \sin(\theta)$

and the parameter vector is:

$$\bar{p} = \begin{bmatrix} \rho \\ \theta \end{bmatrix}$$

In the case of a circle, with the equation of a circle as follows, where a and b are the center coordinates of the circle and d is the diameter:

$$(r-a)^2 + (c-b)^2 = \left(\frac{d}{2}\right)^2$$

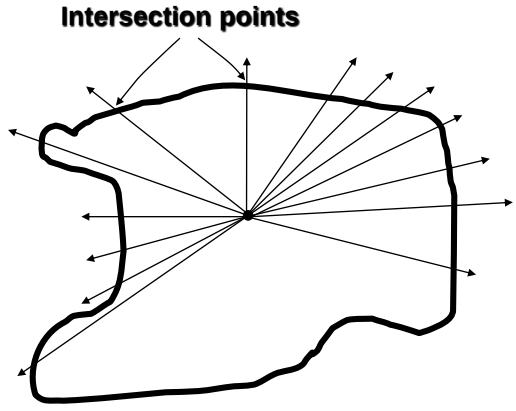
The parameter vector is:

$$\bar{p} = \begin{bmatrix} a \\ b \\ d \end{bmatrix}$$

- Application of the Hough transform to find circles follows a similar procedure as line finding, but with an increased dimensionality to the search space – it is now 3-D

- A *Generalized Hough transform* is used to find any arbitrary shape
- The generalized Hough transform works by creating a description of the shape defined by a reference point and a table of lines, called an *R-table*
- The reference point is chosen inside the sample shape, and a random line is found from the reference point to a point on the border

Generalized Hough Transform



Arbitrary shape with lines from a given point shown intersecting with the shape. The line parameters and their intersections are kept in an R-table, which is used to describe the shape.

- This intersection information is recorded in the table
- The shape is then described by a multitude of line intersection information in the R-table
- The generalized Hough algorithm is then used to search for shapes described by the R table