Lecture 3

(Segmentation and Edge/Line Detection)

Outlines:

1. Edge/Line Detection
2. Edges in Color Images
3. Edge Detector Performance
4. Hough Transform

References:

1. Chapter 4, Scott E Umbaugh, 2018, "Digital Image Processing and Analysis Applications with MATLAB and CVIPtools", 3th Ed, Taylor & Francis.

1. Introduction

The image analysis process requires us to take vast amounts of low-level pixel data and extract useful information. In this lecture, the methods to divide the image into meaningful regions which represent higher level information, edge detection, line detection, and finally image segmentation would be discussed. Image segmentation methods will look for objects that either have some measure of homogeneity within themselves or have some measure of contrast with the objects on their border, such as shape or color features.

Image segmentation techniques can be divided into three main categories (see figure 57):

a) Region growing and shrinking,
b) Clustering methods, and
c) Boundary detection.

The region growing and shrinking methods use the row and column, (r,c),-based spatial domain, whereas the clustering techniques for data can be applied to any domain, such as any N-dimensional color or feature space, whose components may even include the spatial domain's (r,c) coordinates. The region growing and shrinking category can be considered a subset of the clustering methods, they are performed by finding homogeneous regions.

The boundary detection methods are extensions of the edge detection techniques and differ from the other segmentation methods by finding the segmented regions indirectly by finding their borders. Boundary detection is often achieved using a differentiation operator to find lines or edges, followed by post processing to connect the points into borders

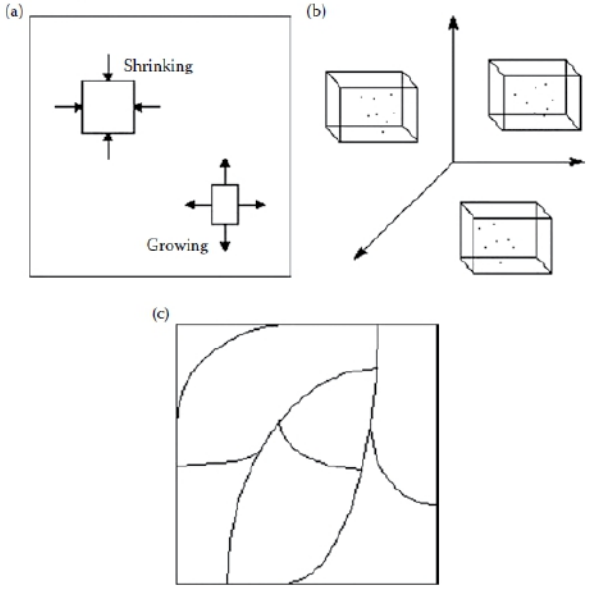


Figure 57. Image Segmentation Categories. (a) region growing/shrinking, (b) clustering, and (c) boundary detection

2. Edge/Line Detection

The edge and line detection operators presented represent the various types of operators in use today. Many are implemented with convolution masks and most are based on discrete approximations to differential operators. Differential operations measure the rate of change in a function, in this case, the image brightness function. A large change in image brightness over a short spatial distance indicates the presence of an edge. Some edge detection operators return orientation information (information about the direction of the edge), whereas others only return information about the magnitude of an edge at each point.

Edge detection methods are used as a first step in the line detection process. Edge detection is also used to find complex object boundaries by marking potential edge points corresponding to places in an image where rapid changes in brightness occur. After these edge points have been marked, they can be merged to form lines and object outlines. Often people are confused about the difference between an edge and a line. This is illustrated in figure 58 where we see that an edge occurs at a point and is perpendicular to the line. The edge direction is defined as the direction of change, so, on a curve, it will be perpendicular to the tangent line at that point. Note that a line or curve that forms a boundary can be defined as a set of connected edge points.

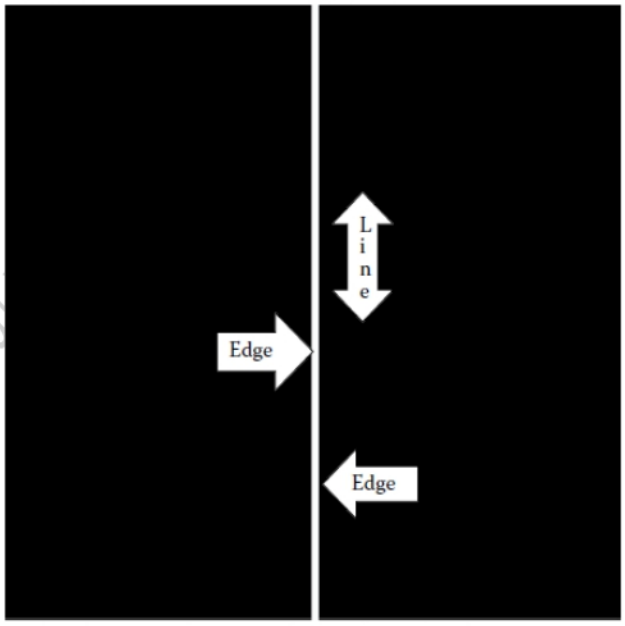


Figure 58. Edges and lines are perpendicular. The line shown here is vertical and the edge direction is horizontal. In this case, the transition from black to white occurs along a row, this is the edge direction, but the line is vertical along a column.

With many of the edge detection operators, noise in the image can create problems. That is why it is best to preprocess the image to eliminate, or at least minimize, noise effects. To deal with noise effects we must make tradeoffs

between the sensitivity and the accuracy of an edge detector. For example, if the parameters are adjusted so that the edge detector is very sensitive, it will tend to find many potential edge points that are attributable to noise. If we make it less sensitive, it may miss valid edges. The parameters that we can vary include the size of the edge detection mask and the value of the gray-level threshold. A larger mask or a higher gray-level threshold will tend to reduce noise effects, but may result in a loss of valid edge points. The tradeoff between sensitivity and accuracy is illustrated in figure 59.

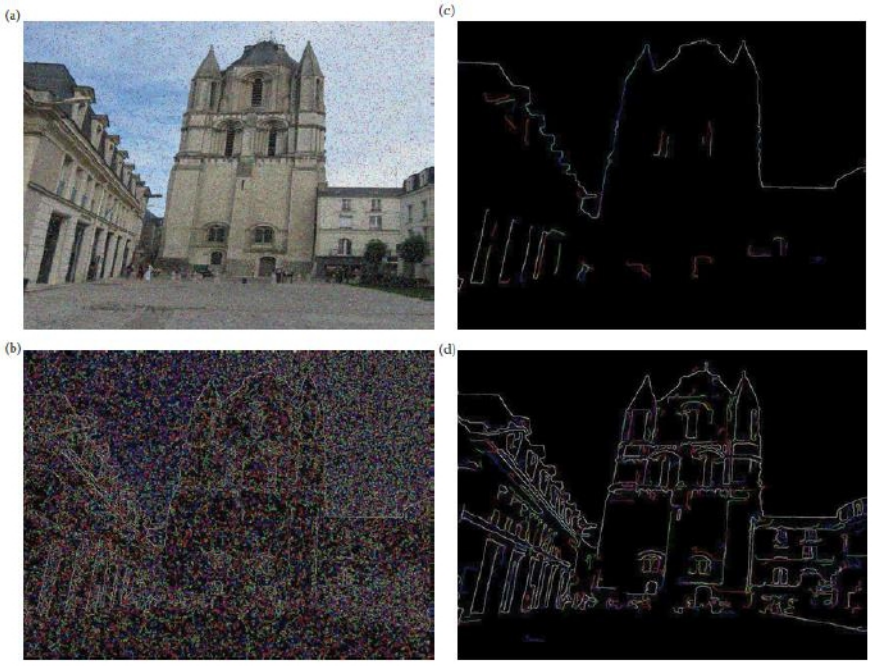


Figure 59. Noise in images requires tradeoffs between sensitivity and accuracy for edge detectors. (a) Noisy image; (b) edge detector too sensitive, many edge points found that are attributable to noise; (c) edge detector not sensitive enough, loss of valid edge points; (d) reasonable result obtained by compromise between sensitivity and accuracy.

Edge detection operators are based on the idea that edge information in an image is found by considering the relationship a pixel has with its neighbors. If a pixel's gray-level value is similar to those around it, there is probably not an edge at that point, if a pixel has neighbors with widely varying gray levels, it may represent an edge point. An edge is defined by a discontinuity in gray-level values. Ideally, an edge separates two distinct objects. In practice, apparent edges are caused by changes in color, texture, or by the specific lighting conditions present during the image acquisition process. This means that what we refer to as image objects may actually be only parts of the objects in the real world, see figure 60.

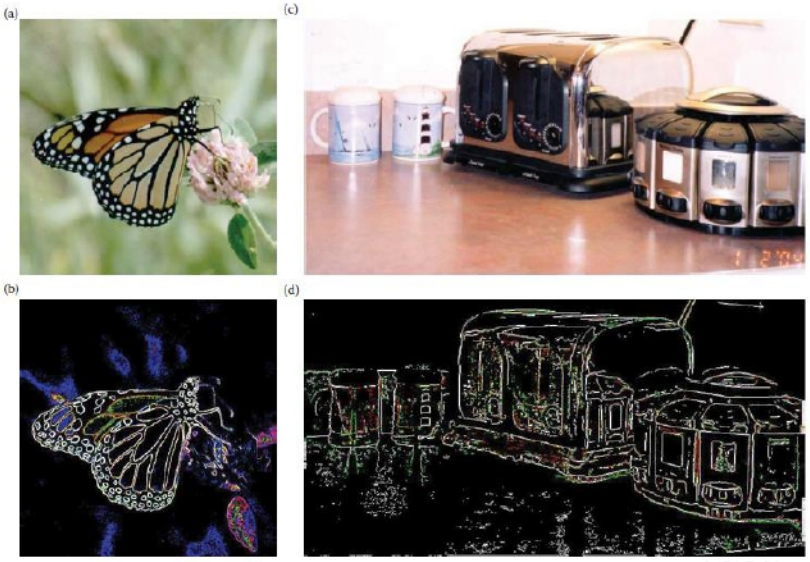


Figure 60. Image objects may be parts of real objects. (a) Butterfly image (original photo), (b) butterfly after edge detection, note that image objects are separated by color and brightness changes, (c) image of objects in kitchen corner, and (d) image after edge detection, note that some image objects are created by reflections in the image due to lighting conditions and object properties.

Figure 61 illustrates the differences between an ideal edge and a real edge, and it shows a representation of one row in an image of an ideal edge. The vertical axis represents brightness, and the horizontal axis shows the spatial coordinate. The abrupt change in brightness characterizes an ideal edge. In the corresponding image, the edge appears very distinct. In the corresponding image, the edge appears very distinct. In figure 61 b, we see the representation of a real edge, which changes gradually. The image contains the same information as does the ideal image: black on one side and white on the other, with a line down the center.

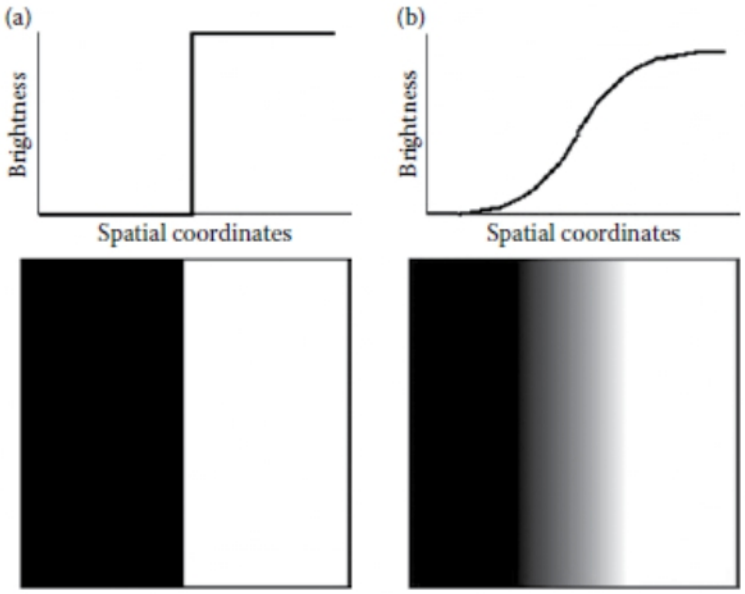
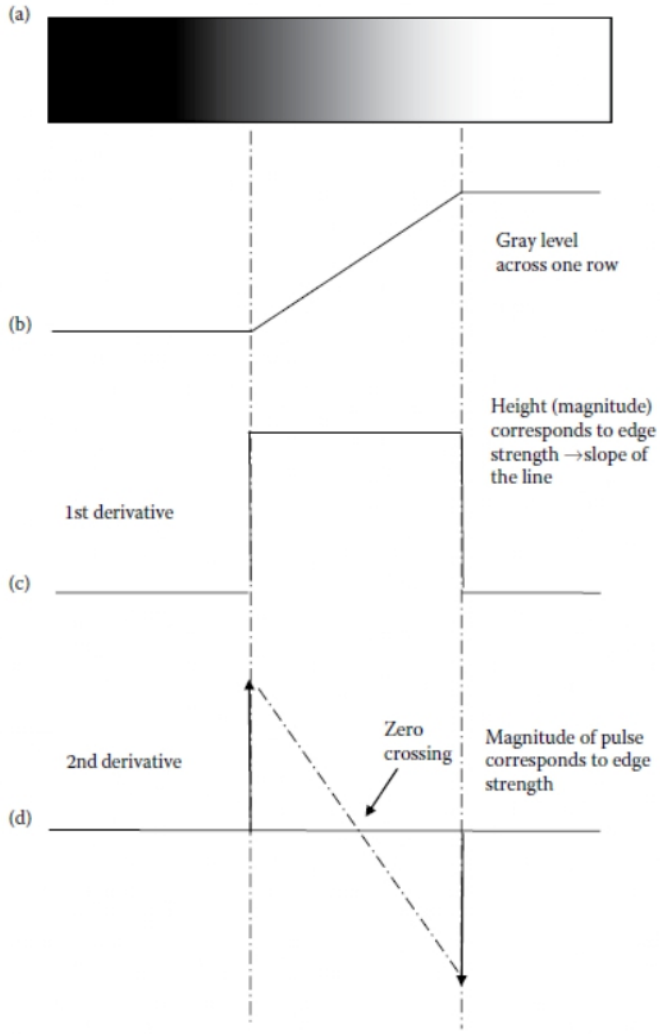


Figure 61. Ideal Edge Compared to a Real Edge. The ideal edge is an instantaneous change, whereas a real edge is typically a gradual change. (a) Ideal edge, (b) Real edge.

3. 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. Remember from calculus that the derivative measures the rate of change of a line or the slope of the line. If we model the gray-level transition of an edge by a ramp function, which is a reasonable approximation to a real edge, we can see what the first and second derivatives look like in figure 62. When the gray level is constant, the first derivative is zero, and when it is linear, it is equal to the slope of the line. With the following operators we will see that this is approximated with a difference operator, similar to the methods used to derive the definition of the derivative. The second derivative is positive at the change on the dark side of the edge, negative at the change on the light side, and zero elsewhere.



Figur 62. 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.

In figure 62 c, we can see that the magnitude of the first derivative will mark edge points, with steeper gray-level changes corresponding to stronger edges and larger magnitudes from the derivative operators. In figure 62 d, we can see that applying a second derivative operator to an edge returns two impulses, one on either side of the edge. An advantage of this is that if a line is drawn between the two impulses, the position where this line crosses the zero axis is the center of the edge, which theoretically allows us to measure edge location to subpixel accuracy. Subpixel 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.

1. Roberts operator

It is a simple approximation to the first derivative. It marks edge points only; it does not return any information about the edge orientation. It is the simplest of the edge detection operators and will work best with binary images (gray-level images can be made binary by a threshold operation). There are two forms of the Roberts operator. The first form (a) consists of the square root of the sum of the differences of the diagonal neighbors squared, as follows:

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

The second form (b) of the Roberts operator is the sum of the magnitude of the differences of the diagonal neighbors, as follows:

$$|I(r,c)-I(r-1,c-1)|+|I(r,c-1)-I(r-1,c)| \quad \text{.....b}$$

The second form of the equation is often used in practice due to its computational efficiency—it is typically faster for a computer to find an absolute value than to find square roots.

2. Prewitt operator

It is similar to the Sobel, but with different mask coefficients. The masks are defined as:

Vertical edge

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

Horizontal edge

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

These masks are each convolved with the image. At each pixel location, we find two numbers: p1, corresponding to the result from the vertical edge mask, and p2, from the horizontal edge mask. We use these results to determine two metrics, the edge magnitude and the edge direction, which are defined as follows:

$$\text{Edge magnitude} \quad \sqrt{p_1^2+p_2^2}$$

$$\text{Edge direction} \quad \tan^{-1}\left[\frac{p_1}{p_2}\right]$$

3. Sobel operator

This operator approximates the gradient by using a row and a column mask, which will approximate 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 two metrics—magnitude and direction. The masks are as follows:

Vertical edge Horizontal edge

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

These masks are each convolved with the image. At each pixel location, we now have two numbers: s_1, corresponding to the result from the vertical edge mask, and s_2, from the horizontal edge mask. We use these numbers to compute two metrics, the edge magnitude and the edge direction, defined as follows:

$$\text{Edge magnitude} \quad \sqrt{s_1^2 + s_2^2}$$

$$\text{Edge direction} \quad \tan^{-1}\left[\frac{s_1}{s_2}\right]$$

As seen in figure 2, the edge direction is perpendicular to the line (or curve), because the direction specified is the direction of the gradient, along which the gray levels are changing.

As with the **Sobel edge** detector, the direction lies 90° from the apparent direction of the line or curve. The **Prewitt** is easier to calculate than the Sobel, as 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.

4. Laplacian operators

It is similar to the spatial filters used for preprocessing. The three Laplacian masks presented below represent various practical approximations of the Laplacian, which is the two-dimensional (2D) version of the second derivative (note that these are masks used in practice and true Laplacians will have all the coefficients negated). Unlike the Sobel and Prewitt edge detection masks, the Laplacian masks are approximately rotationally symmetric, which means edges at all orientations contribute to the result. As that is the case they are applied by selecting one mask and convolving it with the image. The sign of the result (positive or negative) tells us which side of the edge is brighter.

Laplacian masks:

Type 1 Type 2 Type 3

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

These masks differ from the Laplacian type previously described in that the center coefficients have been decreased by 1. With these masks, we are trying to find edges, and are not interested in the image itself—if we increase the center coefficient by 1 it is equivalent to adding the original image to the edge detected image.

An easy way to picture the difference is to consider the effect each mask has when applied to an area of constant value. The above convolution masks return a value of 0. If we increase the center coefficients by 1, each mask returns the original gray level. Therefore, if we are only interested in edge information, the sum of the coefficients should be 0. If we want to retain most of the information that is in the original image, the coefficients should sum to a number > 0. The larger this sum, the less the processed image is changed from the original image. Consider an extreme example in which the center coefficient is very large compared with the other coefficients in the mask. The resulting pixel value will depend most heavily upon the current value, with only minimal contribution from the surrounding pixel values.

5. Compass Masks

It is called **Kirsch masks** as 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 (remember edges are perpendicular to the lines). We also see that the last four masks are actually the same as the first four, but flipped about a central axis.

6. Robinson compass masks

They are used in a manner similar to the Kirsch masks, but 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. We only need to compute the results on four of the masks; the results from the other four can be obtained by negating the results from the first four. The masks are as follows:

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

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

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. It is interesting to note that masks r_0 and r_6 are the same as the Sobel masks. We can see that any of the edge detection masks can be extended by rotating them in a manner like these compass masks, which will allow us to extract explicit information about edges in any direction.

7. Laplacian of a Gaussian (LoG)

By preprocessing with a smoothing filter we can mitigate noise effects, and then use the Laplacian to enhance the edges. By adjusting the spread, or variance, of the Gaussian, we can adjust the filter for different amounts of noise and various amounts of blurring. The combination of the Gaussian followed by a Laplacian is called a **Laplacian of a Gaussian (LoG),** or the **Mexican hat operator** as the function resembles a sombrero (see Figure 63). As the process requires the successive convolution of two masks, they can be combined into one LoG mask. Commonly used 5×5 and 17×17 masks that approximate the combination of the Gaussian and Laplacian into one convolution mask are as follows:

5×5 LoG Mask:

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

17×17 LoG Mask:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & -2 & -3 & -3 & -3 & -2 & -3 & -2 & -3 & -3 & -3 & -2 & -1 & -1 & 0 \\ -1 & -1 & -2 & -3 & -3 & -3 & 0 & 2 & 4 & 2 & 0 & -3 & -3 & -3 & -2 & -1 & 0 \\ -1 & -1 & -3 & -3 & -3 & 0 & 4 & 10 & 12 & 10 & 4 & 0 & -3 & -3 & -3 & -1 & -1 \\ -1 & -1 & -3 & -3 & -3 & 2 & 10 & 18 & 21 & 18 & 10 & 2 & -2 & -3 & -3 & -1 & -1 \\ -1 & -1 & -3 & -3 & -3 & 4 & 12 & 21 & 24 & 21 & 12 & 4 & -3 & -3 & -3 & -1 & -1 \\ 0 & -1 & -3 & -3 & -3 & 2 & 10 & 18 & 21 & 18 & 10 & 2 & -2 & -3 & -3 & -1 & -1 \\ 0 & -1 & -3 & -3 & -3 & 0 & 4 & 10 & 12 & 10 & 4 & 0 & -3 & -3 & -3 & -1 & -1 \\ 0 & -1 & -2 & -3 & -3 & -3 & 0 & 2 & 4 & 2 & 0 & -3 & -3 & -3 & -3 & -1 & 0 \\ 0 & -1 & -1 & -2 & -3 & -3 & -3 & -2 & -3 & -2 & -3 & -3 & -3 & -2 & -2 & -1 & 0 \\ 0 & 0 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & 0 & 0 \\ 0 & 0 & -1 & -1 & -1 & -2 & -3 & -3 & -3 & -3 & -3 & -2 & -1 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

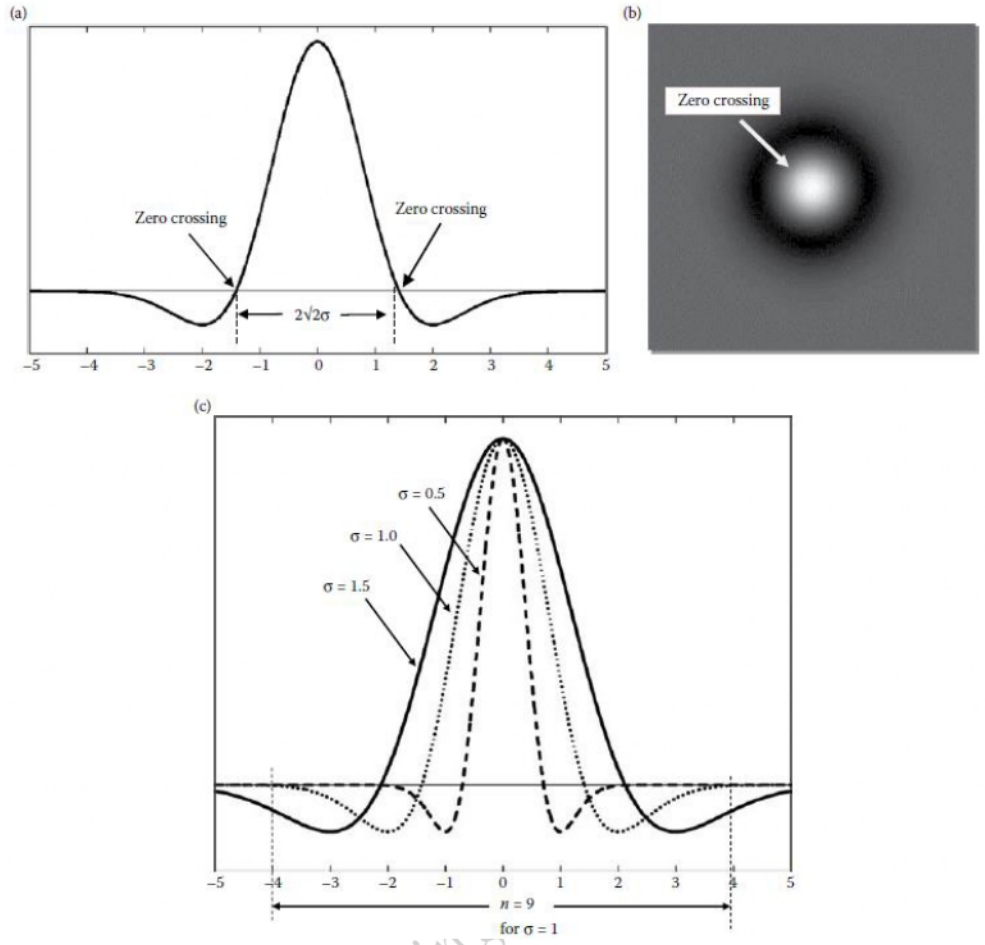
The equation for the LoG filter is:

$$LoG = \left[\frac{r^2 + c^2 - 2\sigma^2}{\sigma^4}\right] e^{-\left(\frac{r^2+c^2}{2\sigma^2}\right)}$$

Where (r,c) are the row and column coordinates and σ is the Gaussian variance. From the equation, we can see that zero-crossings occur at $(r^2 + c^2) = 2\sigma^2$ or $\sqrt{2}\sigma$ from the mean, as shown in Figure 63 a. Note that, in practice, if creating the LoG filter mask by convolving a Gaussian and a Laplacian mask, we need to be sure that the Gaussian is normalized to 1 and that the Laplacian coefficients sum to 0. This is done to avoid biasing the mask with an offset term, which will shift the zero-crossings and defeat the filter's purpose.

To determine the size of the mask to use, we consider that 99.7% of the area under a Gaussian curve is within ±3σ of the mean. Keeping in mind that the sampling grid is fixed by the pixel spacing, the variance and the mask size must be related (see Figure 4.2-6c). So we want to select a value of n for the n×n convolution mask that is an odd integer ≥6σ, or we will get only a portion of the curve with our sampled filter mask. In CVIPtools, we use the following equation to determine n, based on the variance, σ:

$$n = [2 \times \text{TRUNCATE}(3.35\sigma + 0.33) + 1].$$



Figur 63. The inverted Laplacian of a Gaussian (LoG). (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; (c) three LoG plots with σ = 0.5, 1.0, and 1.5. Note for σ = 0.5, the mask size, n, should be about 5×5; for σ = 1, 9×9, and so on. This is done so the mask covers the entire function as it goes negative and then goes back up to zero. Note this is 4σ to the left, 4σ to the right, and the center term corresponding to the term at the 0 point on the graph.

This equation assures us that we have the complete spread of the LoG filter and actually provides us with an n that corresponds to about ±4σ. Note that for positive numbers, the truncate operation is the same as the floor operation. The third step for the Laplacian of a Gaussian (LoG) algorithm is to find the zero-crossings after the LoG is performed. This can be accomplished by considering a pixel and its surrounding pixels, thus a 3×3 subimage, and looking for sign changes between two of the opposing neighbors. That is, we check the left/right, up/down, and the two diagonal neighboring pairs. Figure 64 illustrates the results from the standard Laplacian of a Gaussian (LoG) algorithm. The disadvantage of the Laplacian of a Gaussian (LoG) algorithm, or any second

derivative/zero-crossing method, is that it tends to smooth shapes too much, which has the effect of eliminating corners and creating closed loops in the resulting lines/curves. The Laplacian of a Gaussian (LoG) results are often referred to as a "plate of spaghetti," as shown in Figure 64 (c through e).

In practice, we may want to set a threshold to use before a pixel is classified as an edge. The threshold is tested against the absolute value of the difference between the two pixels that have the sign changes. If this value exceeds the threshold it is classified as an edge pixel.

Example: Applying a threshold of the Laplacian of a Gaussian (LoG) algorithm. Suppose, after the LoG, we have a 3×3 subimage as follows:

$$\begin{bmatrix} -10 & 11 & 17 \\ 18 & 2 & 15 \\ 21 & 33 & 28 \end{bmatrix}$$

The only pair that has a sign change is the NW/SE diagonal. So, the center pixel may be considered an edge pixel. If we apply a threshold, then we can calculate the absolute value of the difference of this pair:

$$|-10-28|=38$$

If this value exceeds the threshold we have set, then the center pixel is determined to be an edge pixel.

The Laplacian of a Gaussian (LoG) Laplacian of a Gaussian (LoG) has a parameter to allow the user to select single variance or dual variance. If dual variance is selected, the user specifies a sigma (σ, variance) value and a delta value. then computes the Marr–Hildreth results using two variances—the specified sigma plus the delta value and the specified sigma minus the delta value, (see Figure 64 e and f).

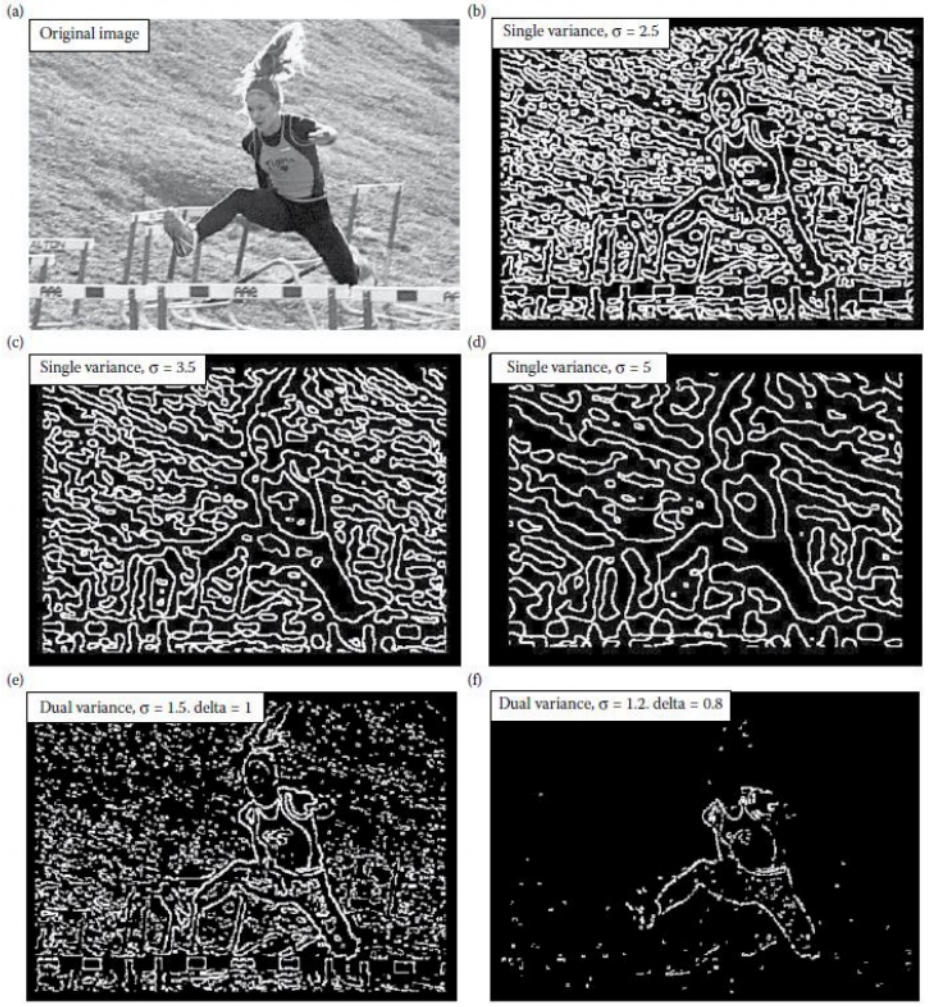


Figure 64. Results from Using Different Variance Values with the Marr–Hildreth Algorithm. Increasing the variance with a single value, which is the equivalent of using a larger mask size for the filters, the resulting edge lines are farther apart.(a) Original image; (b) results with $\sigma = 2.5$; (c) results with $\sigma = 3.5$; (d) results with $\sigma = 5$.Using a dual variance, the amount of noise retained depends on the relationship between σ and delta.(e) Dual variance with $\sigma = 1.5$ and delta = 1; (f) dual variance with $\sigma = 1.2$ and delta = 0.8.

8. Canny algorithm

The Canny algorithm, developed by John Canny in 1986, is an optimal edge detection method based on a specific mathematical model for edges.The edge model is a step edge corrupted by Gaussian noise.The algorithm consists of four primary steps:

1. Apply a Gaussian filter mask to smooth the image to mitigate noise effects.This can be performed at different scales, by varying the size of the filter mask which corresponds to the variance of the Gaussian function.A larger mask will blur the image more and will find fewer, but more prominent, edges.

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.This is done by considering small neighborhoods in the magnitude image, for example, 3×3, and comparing the center value to its neighbors in the direction of the gradient.If the center value is not larger than the neighboring pixels along the gradient direction, then set it to 0.Otherwise, it is a local maximum, so we keep it. In Figure 65, we see an example of a 3×3 neighborhood showing the magnitude at each location, and using an arrow to show the gradient direction.The center pixel has a value of 100 and the gradient direction is horizontal (corresponding to a vertical line), so it is compared to the pixels to the right and left; which are 40 and 91.As it is greater than both, it is retained as an edge pixel; if it was less than either one, then it would be removed as an edge point.Note that this will have the effect of making thick edges thinner, by selecting the "best" point along a gradient direction.
4. Apply two thresholds to obtain the final result. This technique, known as hysteresis thresholding helps to avoid false edges caused by too low a threshold value or missing edges caused by too high a value.It is a two-step thresholding method, which first marks edge pixels above a high threshold, and then applies a low threshold to pixels connected to the pixels found with the high threshold.This can be performed multiple times, as either a recursive or iterative process.

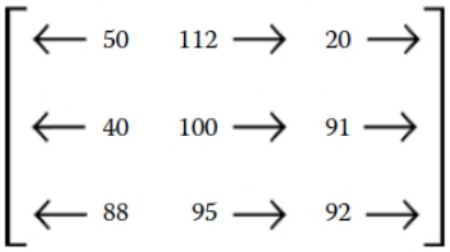


Figure 65. Nonmaxima Suppression. A 3×3 subimage of the magnitude image, which consists of the magnitude results in an image grid.

The high threshold is computed from the image by finding the value which is greater than 90% of the pixels after applying nonmaxima suppression to the magnitude images. Figures 66 shows results from varying these parameters.

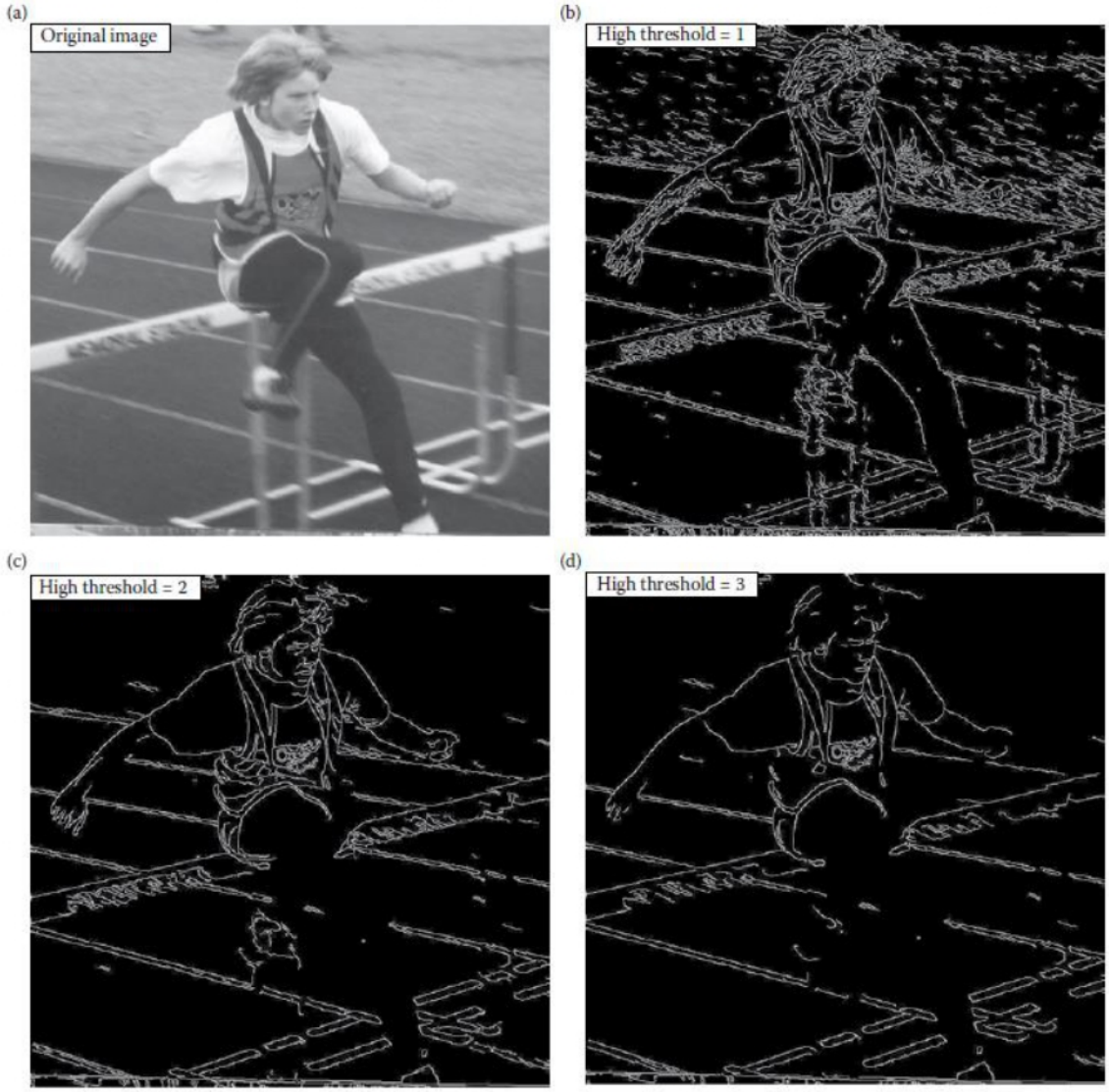


Figure 66. Results from changing high threshold with Canny Algorithm. As the high threshold in increased, small details are removed.In the results the Gaussian = 0.5 and the low threshold = 1.(a) Original image; (b) results high threshold factor = 1; (c), high threshold factor = 2; (d) high threshold factor = 3.

4. Edges in Color Images

The RGB data can be mapped into the HSV (hue/saturation/value) color space and edges are sought in the hue or saturation bands. Figure 67 illustrates this by showing that the areas of reflection are found in the saturation band, but not in the value (brightness) band. We can use any of the previously defined edge detection methods on each of the three bands individually, and can then combine the results from all three bands into a three band image (see Figure 68), as multispectral satellite images.It uses equations similar to the Roberts gradient, but is applied to all the image bands with a simple set of equations.

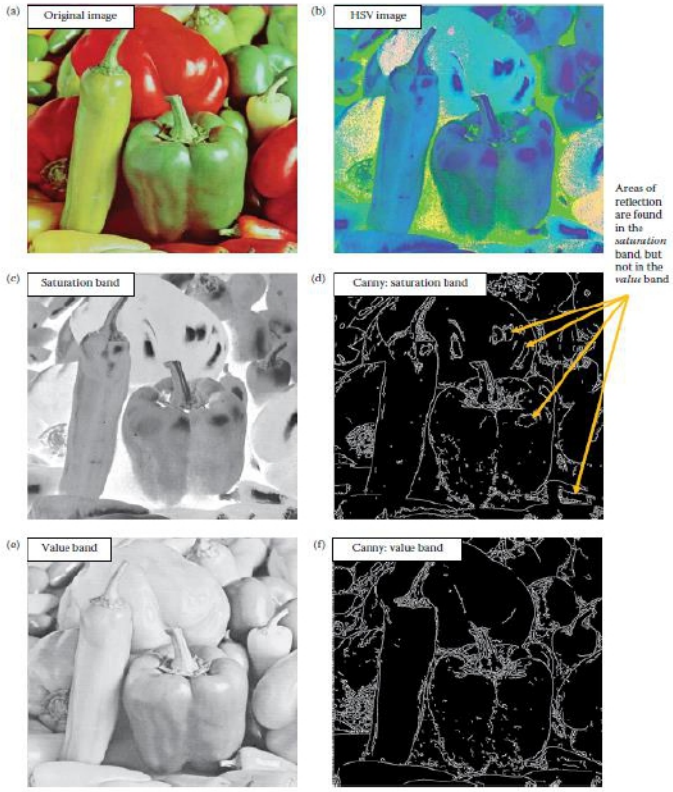


Figure 67. Color Edge Detection in HSV Space. (a) Original image, (b) original image mapped into HSV color space and displayed as an RGB image, (c) the saturation band, (d) Canny edge detection applied to the saturation band, (e) the value band, (f) Canny edge detection applied to the value band. Note that the areas of reflection, marked with the yellow arrows on image (d), are found in the saturation band, but not in the value band, image (f).

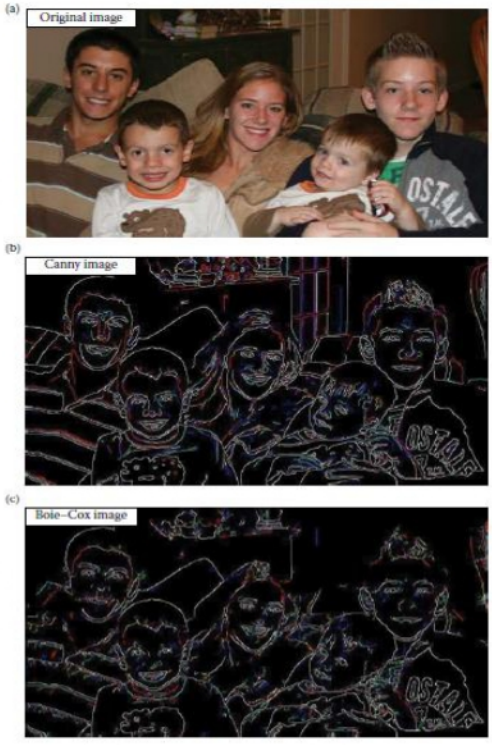


Figure 68. Color Edge Detection in RGB Space. (a) Original image in RGB space, (b) Canny edge detector, all three bands displayed, (c) Boie–Coxedge detector, all three bands displayed. The edges that appear white are in all three RGB bands. Note that some edges only appear in one or two color bands.

The result of this edge detector at pixel (r,c) is the smaller of the two values from these two equations:

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

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

Where:

$\bar{I}(r,c)$ is the arithmetic average of all the pixels in all bands at pixel location (r,c) and $I_b(r,c)$ is the value at location (r,c) in the bth band, with a total of n bands.

This edge detector has been used successfully on multispectral satellite images.

5. Edge Detector Performance

In evaluating the performance of many processes, both objective and subjective evaluation methods can be used. The objective metric allows us to compare different techniques with fixed analytical methods, whereas the subjective methods will include human evaluation as part of the process, which may lead to inconsistent results. However, for many image processing applications, the subjective measures tend to be quite useful. Therefore, in the development of an objective metric, it is advantageous to take human visual attributes into consideration.

To develop a performance metric for edge detection operators, we need to define what constitutes success.

For example, the Canny algorithm was developed considering three important edge detection success criteria:

- Detection: the edge detector should find all real edges and not find any false edges.
- Localization: the edges should be found in the correct place.
- Single response: there should not be multiple edges found for a single edge.

These correlate nicely with Pratt's Figure of Merit (FOM) defined in 1978. Pratt first considered the types of

errors that can occur with edge detection methods.

The types of errors are:
(1) Missing valid edge points,
(2) Classifying noise pulses as valid edge points, and
(3) Smearing of edges (see Figure 69).

If these errors do not occur, then a successful edge detection can be achieved.

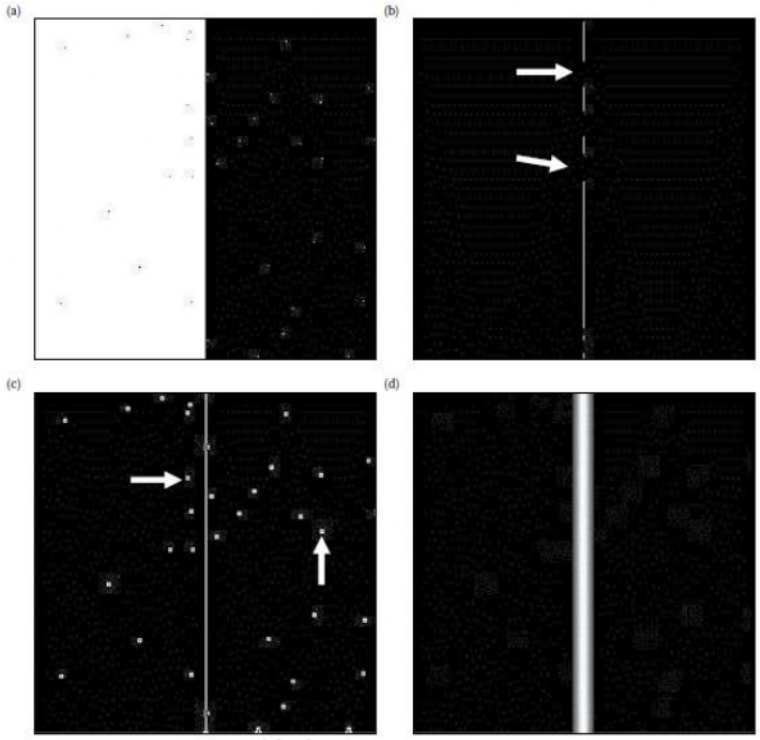


Figure 69. **Errors in Edge Detection**. (a) Original image, (b) missed edge points, examples marked with arrows, (c) noise misclassified as edge points, examples marked with arrows, (d) smeared edge.

The Pratt FOM is defined as follows:

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

Where I_N is the maximum of II and IF; II is the number of ideal edge points in the image; IF is the number of edge points found by the edge detector; α is a scaling constant that can be adjusted to adjust the penalty for offset edges, and di is 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. This is done because techniques exist to thin smeared edges, but it is difficult to determine when an edge is found in the wrong location or is completely missed. The distance, d, can be defined in more

than one way and typically depends on the connectivity definition used.The possible definitions for d are as follows:

Let the (r,c) values for two pixels be (r_1, c_1) and (r_2, c_2).

1. *City block distance*, based on four-connectivity:

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

With this distance measure, we can only move horizontally and vertically.

2. *Chessboard distance*, based on eight-connectivity:

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

With this distance measure, we can move diagonally, as well as horizontally or vertically.

3. *Euclidean distance*, based on actual physical distance:

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

EXAMPLE 4.2.3: PRATT'S FIGURE OF MERIT (FOM)

Given the following image array, find the FOM 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.

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

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

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.

6. Hough Transform

The Hough transform is designed specifically to find lines. A line is a collection of edge points that are adjacent and have the same direction. The Hough transform is an algorithm that will take a collection of n edge points, as found by an edge detector, and efficiently find all the lines on which these edge points lie. The advantage of the Hough transform is that it provides parameters to reduce the search time for finding lines based on a set of edge points, and that these parameters can be adjusted based on application requirements. To understand the Hough transform, we will first consider the normal (perpendicular) representation of a line:

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

Given a line in our row and column, (r,c) based image space, we can define that line by ρ, the distance from the origin to the line along a perpendicular to the line, and θ, the angle between the r-axis and the $\rho-$line figure 70. For each pair of values of ρ and θ, we have defined a particular line. The range on θ is 0° to 180° and ρ ranges from 0 to $\sqrt{2}$N, where N×N is the image size for a nonsquare image, it is the diagonal lengthThe algorithm consists of three primary steps:

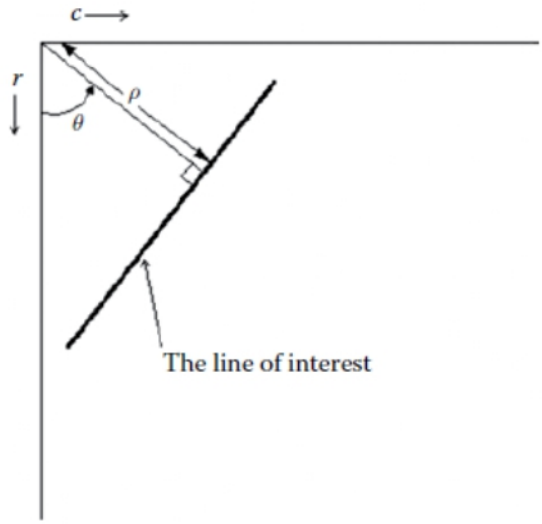


Figure 70. **The Hough Transform** can be defined by using the normal (perpendicular) representation of a line and the parameters ρ and θ.