4

Edge, Line and Shape Detection

4.1 Introduction and Overview

The image analysis process requires us to take vast amounts of low-level pixel data and extract useful information. In this chapter, we will explore methods that contribute to the process of dividing the image into meaningful regions by detecting edges, lines, corners and geometric shapes. These shapes represent higher-level information, and these are often precursors to image segmentation, which is explored in the next chapter where we will see that edge and line detection are important steps for one category of image segmentation methods. Edge detection techniques are covered in Section 4.2, as well as metrics to measure edge detector performance. Section 4.3 introduces the Hough transform for line finding and discusses associated postprocessing methods. Section 4.4 will explore corner and shape detection, and this chapter concludes with a discussion of using the extended and generalized Hough transform for shape detection.

4.2 Edge Detection

The edge detection process is based on the idea that if the brightness levels are changing rapidly, there is the potential for an edge to exist at that point. Many different approaches have been explored to facilitate this process. The edge detection operators presented here are representative of the various types that have been developed. Many are implemented with small masks used in a manner similar to the convolution process; by this, we mean they are used to scan the image left to right and top to bottom, and at each pixel, replace the value with the result of a mathematical operation on the pixel's neighborhood. Most edge detectors 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 – while 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 4.2-1 where we see that an edge occurs at a point, and it 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.

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 4.2-2.

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. However, if a pixel has neighbors with widely varying gray levels, it may represent an edge point. In other words, 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

DOI: 10.1201/9781003221135-4

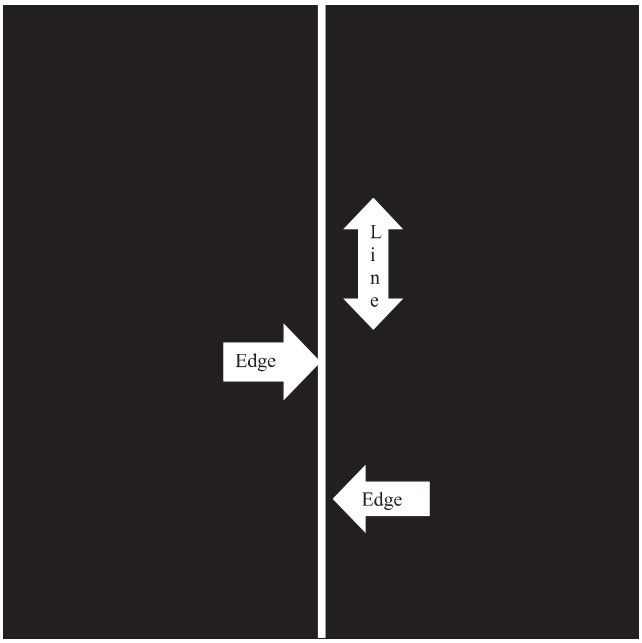

FIGURE 4.2-1
Edges and lines are perpendicular. The line shown here is vertical and the edge direction is horizontal. In this case, the transition from black to white occurs along a row, this is the edge direction, but the line is vertical along a column.

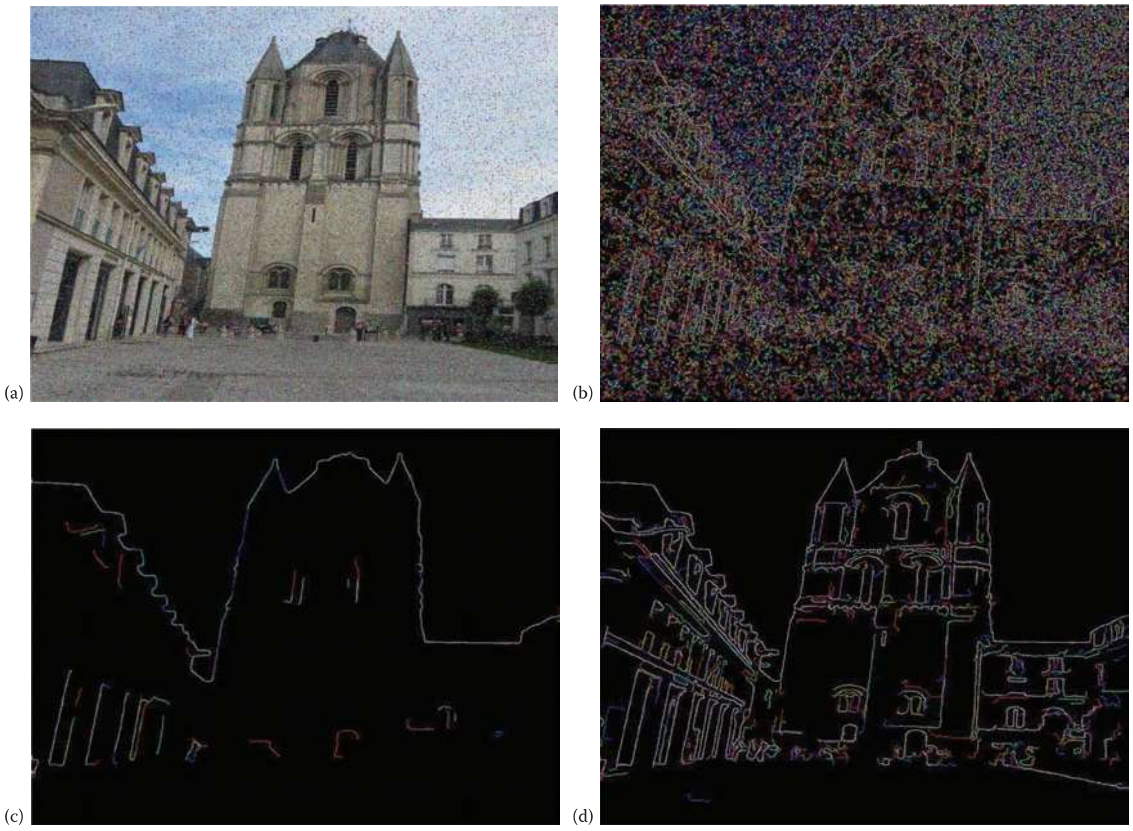

FIGURE 4.2-2
Noise in images requires tradeoffs between sensitivity and accuracy for edge detectors. (a) Noisy image, (b) edge detector too sensitive, many edge points found that are attributable to noise (c) edge detector not sensitive enough, loss of valid edge

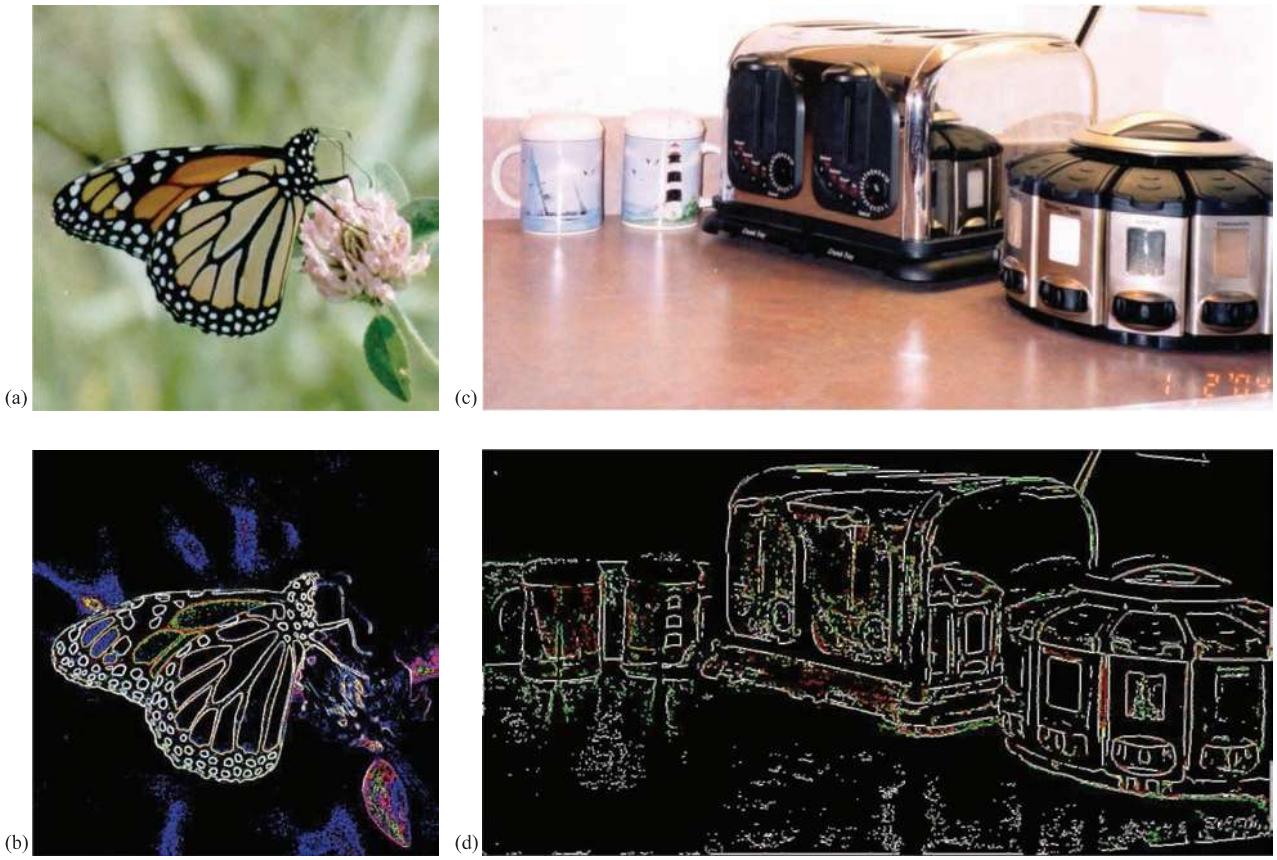

FIGURE 4.2-3
Image objects may be parts of real objects. (a) Butterfly image (original photo courtesy of Mark Zuke), (b) butterfly after edge detection, note that image objects are separated by color and brightness changes, (c) image of objects in kitchen corner, and (d) image after edge detection, note that some image objects are created by reflections in the image due to lighting conditions and object properties.

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 4.2-3.

Figure 4.2-4 illustrates the differences between an ideal edge and a real edge. Figure 4.2-4a 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 Figure 4.2-4b, we see the representation of a real edge which changes gradually. This gradual change is a minor form of blurring caused by the imaging device, the lenses, and/or the lighting and is typical for real-world (as opposed to computer generated) images. In the figure, where the edge has been exaggerated for illustration purposes, note that from a visual perspective, this 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.

4.2.1 Gradient Operators

Gradient operators are based on the concept 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 4.2-5. 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 a positive spike at the change on the dark side of the edge where the brightness is increasing, and it is a negative spike at the change on the light side where the increase stops and zero elsewhere.

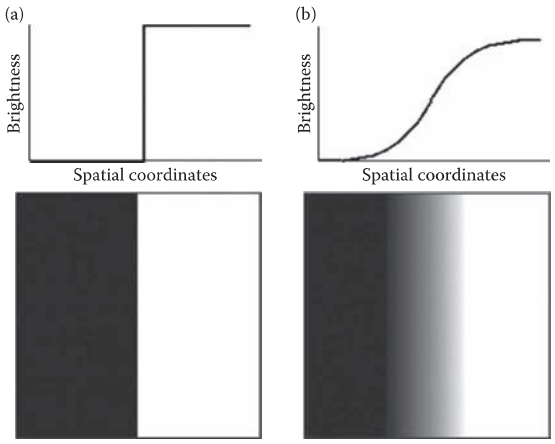

FIGURE 4.2-4
Ideal edge compared to a real edge. (a) Ideal edge and (b) real edge. The ideal edge is an instantaneous change, while a real edge is typically a gradual change. The real edge has been greatly enlarged for illustration purposes, note that from a visual perspective this image contains the same information as does the ideal image: black on one side, white on the other, with a line down the center.

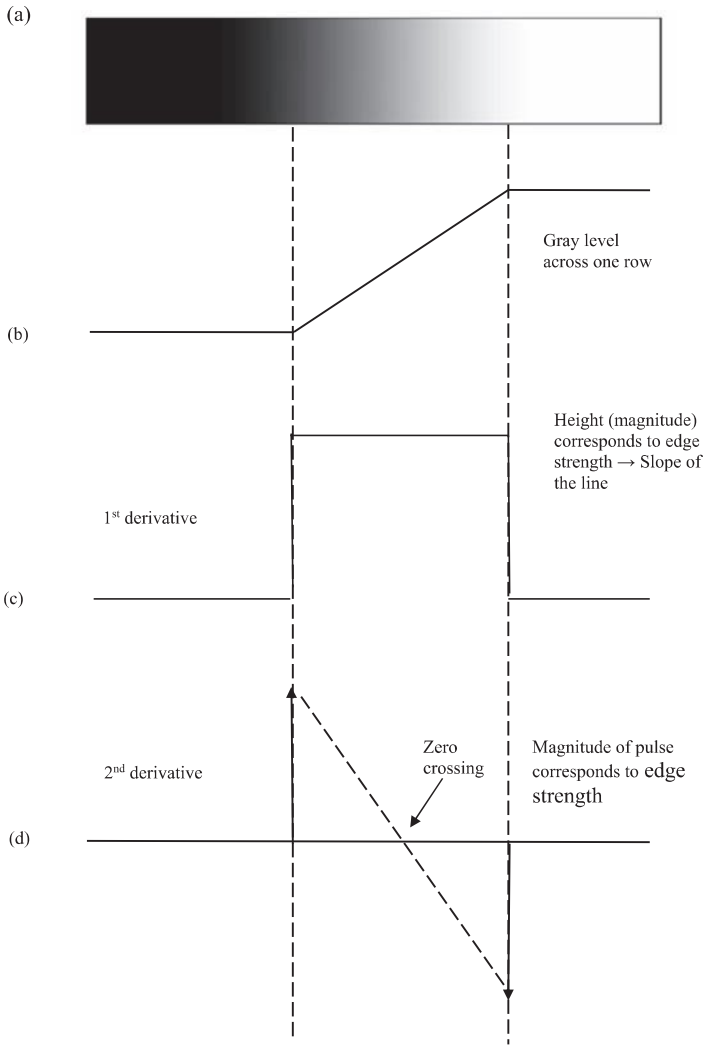

FIGURE 4.2-5
Edge model. (a) A portion of an image with an edge, which has been enlarged to show detail, (b) ramp edge model, (c) first

In Figure 4.2-5c, 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 4.2-5d, 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 sub-pixel accuracy. Sub-pixel accuracy refers to the fact that the zero crossing (*zc*) may be at a fractional pixel distance, for example, halfway between two pixels, so we could say the edge is at, for instance, *zc* = 75.5.

The **Roberts operator** 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, so is the most efficient one for binary images. There are two forms of the Roberts operator. The first consists of the square root of the sum of the differences of the diagonal neighbors squared, as follows:

$$\sqrt{[I(r,c)-I(r-1,c-1)]^2+[I(r,c-1)-I(r-1,c)]^2} \tag{4.2-1}$$

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

$$\left|I(r,c)-I(r-1,c-1)\right|+\left|I(r,c-1)-I(r-1,c)\right| \tag{4.2-2}$$

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.

The **Sobel 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

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

HORIZONTAL EDGE

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

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:

$$\text{EDGE MAGNITUDE} \sqrt{s_1^2 + s_2^2} \tag{4.2-3}$$

$$\text{EDGE DIRECTION } \text{Tan}^{-1}\left[\frac{s_1}{s_2}\right] \tag{4.2-4}$$

As seen in Figure 4.2-1, 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.

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

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: p_1, corresponding to the result from the vertical edge mask, and p_2, from the horizontal edge mask. We use these results to determine

$$\text{EDGE MAGNITUDE } \sqrt{p_1^{\,2} + p_2^{\,2}} \tag{4.2-5}$$

$$\text{EDGE DIRECTION } \text{Tan}^{-1}\left[\frac{p_1}{p_2}\right] \tag{4.2-6}$$

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, 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 be desirable for some applications.

The **Laplacian operators** described here are similar to the ones used for preprocessing as described in Section 3.2.3. The three Laplacian masks presented below represent various practical approximations of the Laplacian, which is the two-dimensional 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

Filter 1 Filter 2 Filter3

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

These masks differ from the Laplacian-type previously described in that the center coefficients have been decreased by one. With these masks, 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.

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 zero. If we increase the center coefficients by one, each mask returns the original gray level. Therefore, if we are only interested in edge information, the sum of the coefficients should be zero. If we want to retain most of the information that is in the original image, the coefficients should sum to a number greater than zero. 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.

4.2.2 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

(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.

The **Robinson compass masks** 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:

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

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

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. 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.

4.2.3 Thresholds, Noise Mitigation and Edge Linking

The gradient and compass mask edge detectors provide measures for magnitude and direction of the brightness gradient at each pixel point. This is only the first step in marking potential edge points which will be used to ultimately find object boundaries. As illustrated in Figure 4.2-2, consideration must be given to the sensitivity versus the accuracy of the edge detector. By considering any points that we do not want or need as "noise" and the valid, typically stronger edge points as the "real" edge points, the primary methods to control sensitivity and accuracy are the following: (1) control of a threshold, so that only the stronger edge points pass, (2) use of large mask sizes to mitigate noise and (3) use of a preprocessing mean filter.

We have discussed two thresholding methods in the previous chapter, the *Otsu Method* and the *Automatic Thresholding* (K-Means) – these algorithms work well with one object and high contrast with the background – a bimodal histogram as shown in Figure 4.2-6. These types of images are obtained in machine vision applications where the lighting can be controlled. This is not typically the case with the magnitude image that results from an edge detector, these images tend to have unimodal (one peak) histograms.

A method that provides reasonable results for unimodal histograms is to use the *average value* for the threshold, as in Figure 4.2-7. With very noisy images and a unimodal histogram, a good rule of thumb is to use 10%–25% of the maximum value as a threshold. An example of this is shown in Figure 4.2-8.

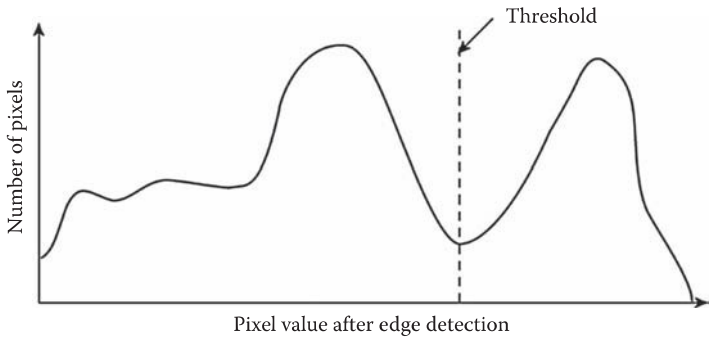

FIGURE 4.2-6
Edge detection thresholding via histogram. The histogram can be examined manually to select a good threshold. This method is easiest with a bimodal (two peaks) histogram. Alternately, the threshold can be found automatically with the Automatic Single Threshold or the Otsu algorithm described in Chapter 3.

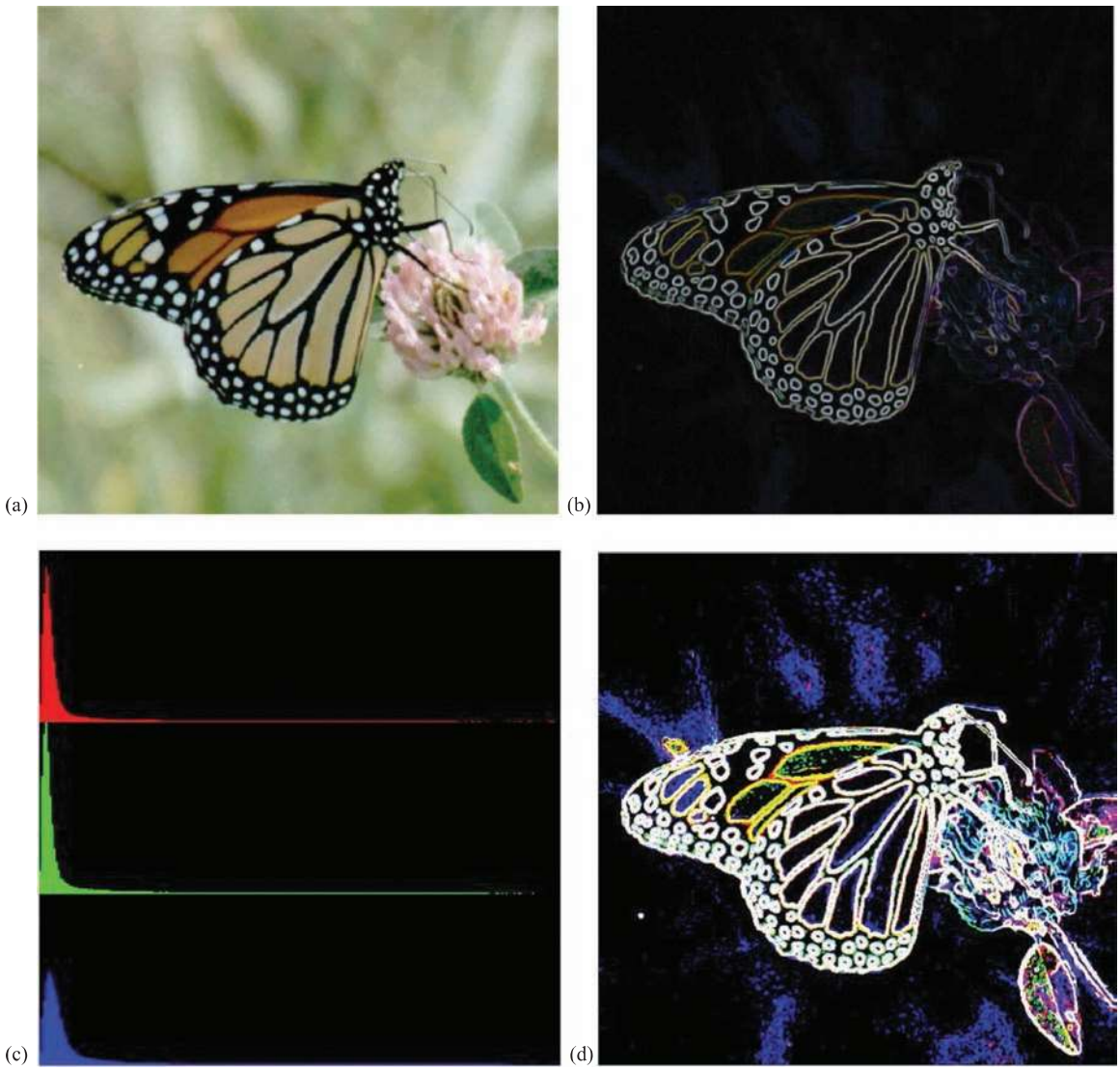

FIGURE 4.2-7
Average value thresholding. (a) Original image, (b) image after Sobel edge detector, (c) unimodal histogram of image after Sobel and (d) Sobel image after thresholding with average value.

If the images contain too much noise, most standard edge detectors perform poorly. The edge detector will tend to find more false edges as a result of the noise. We can preprocess the image with mean, or averaging, spatial filters to mitigate the effects from noise, or we can expand the edge detection operators themselves to mitigate noise effects. One way to do this is to extend the size of the edge detection masks. An example of this method is to extend the Prewitt edge mask as follows:

EXTENDED PREWITT EDGE DETECTION MASK

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

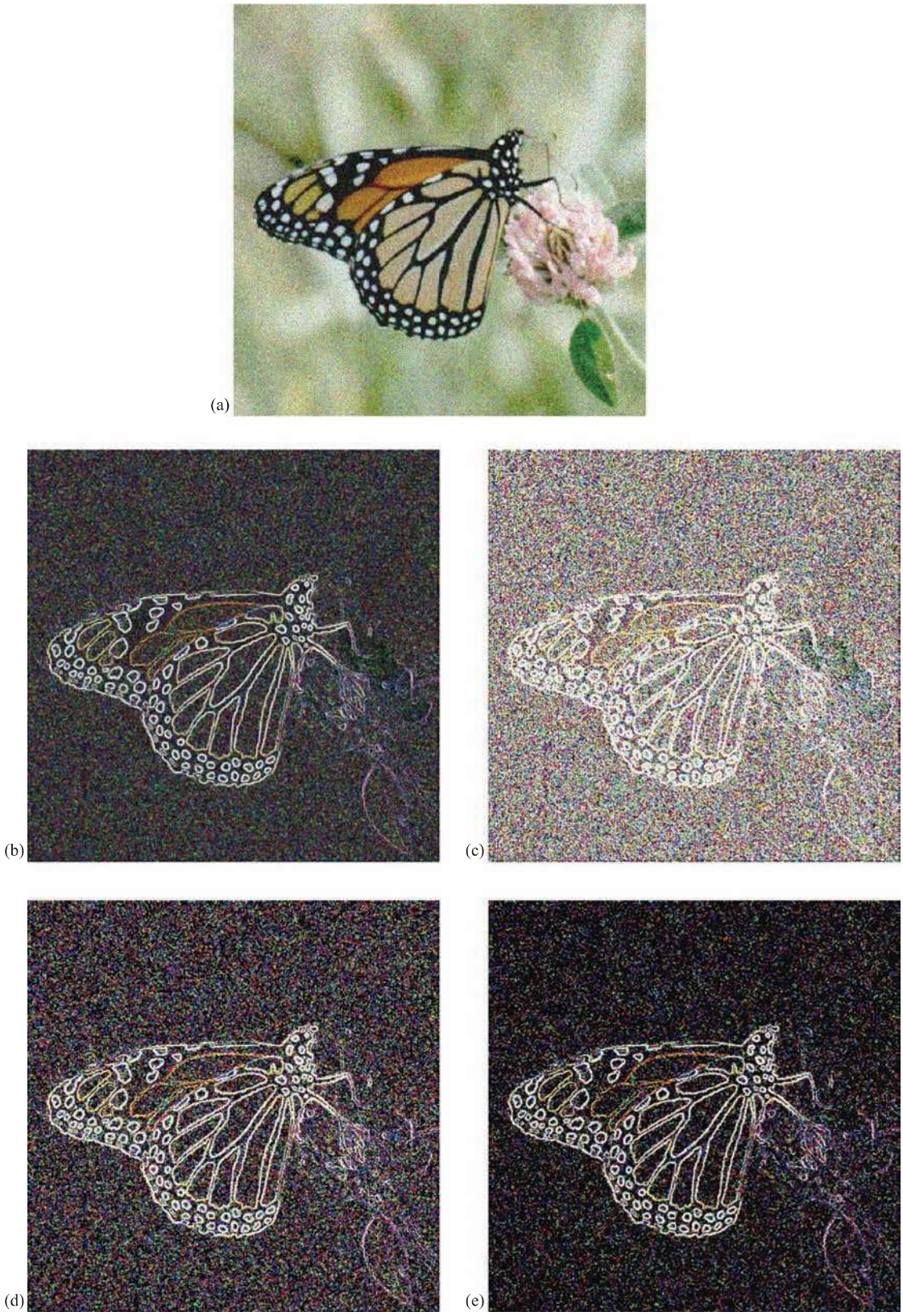

FIGURE 4.2-8
Thresholding noisy images. (a) Original image with Gaussian noise added (zero mean, variance = 800), (b) Sobel edge detector results (remapped), (c) threshold on Sobel at 10% of maximum value, (d) threshold on Sobel at 20% of maximum and (e) threshold on Sobel at 25% of maximum.

We then can rotate this by 90°, and we have both row and column masks which can be used like the Prewitt operators to return the edge magnitude and gradient. These types of operators are called boxcar operators and can be extended to any size, although 7×7, 9×9 and 11×11 are typical. The Sobel operator can be extended in a similar manner:

EXTENDED SOBEL EDGE DETECTION MASK

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

If we approximate a linear distribution, the *truncated pyramid* operator is created as follows:

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

This operator provides weights that decrease from the center pixel outward, which will smooth the result in a more natural manner. These operators are used in the same manner as the Prewitt and Sobel – we define a row and column mask and then find a magnitude and direction at each point. A comparison of applying the extended Prewitt operators with the standard 3×3 operators to an image is shown in Figure 4.2-9. We see that as the mask size increases, the lines thicken and small ones disappear. Figure 4.2-10 shows various edge detectors and the effect of larger masks on an image with Gaussian noise. Comparing Figure 4.2-10c and d and Figure 4.2-10e and f, we see that the extended operators exhibit better performance than the smaller masks. However, they require more computations and will smear the edges, which can be alleviated by postprocessing to thin the smeared edges and remove any leftover noise. Figure 4.2-11 shows results from changing Sobel mask sizes on an image with salt-and-pepper noise.

After a threshold for the edge detected image is determined, we need to merge the selected edge points into boundaries. This is done by *edge linking*. The simplest approach to edge linking involves considering each point that has passed the threshold test and connecting it to all other such points that are within a maximum distance. This method tends to connect many points and is not useful for images where too many points have been marked; it is most applicable to simple images.

Instead of thresholding and then edge linking, we can perform edge linking on the edge detected image before we threshold it. If this approach is used, we look at small neighborhoods (3×3 or 5×5) and link similar points. Similar points are defined as being spatially adjacent and having close values for both magnitude and direction. How close they need to be will be dependent on the application and the types of lines or curves we are wanting to identify. The entire image undergoes this process while keeping a list of the linked points. When the process is complete, the boundaries are determined by the linked points.

4.2.4 Advanced Edge Detectors

The advanced edge detectors are algorithms that use the basic edge detectors and combine them with the noise mitigation concepts and also introduce new ideas such as that of hysteresis thresholding. The edge detectors considered here include the Marr–Hildreth algorithm, the Canny algorithm, the Boie–Cox algorithm, the Shen–Castan

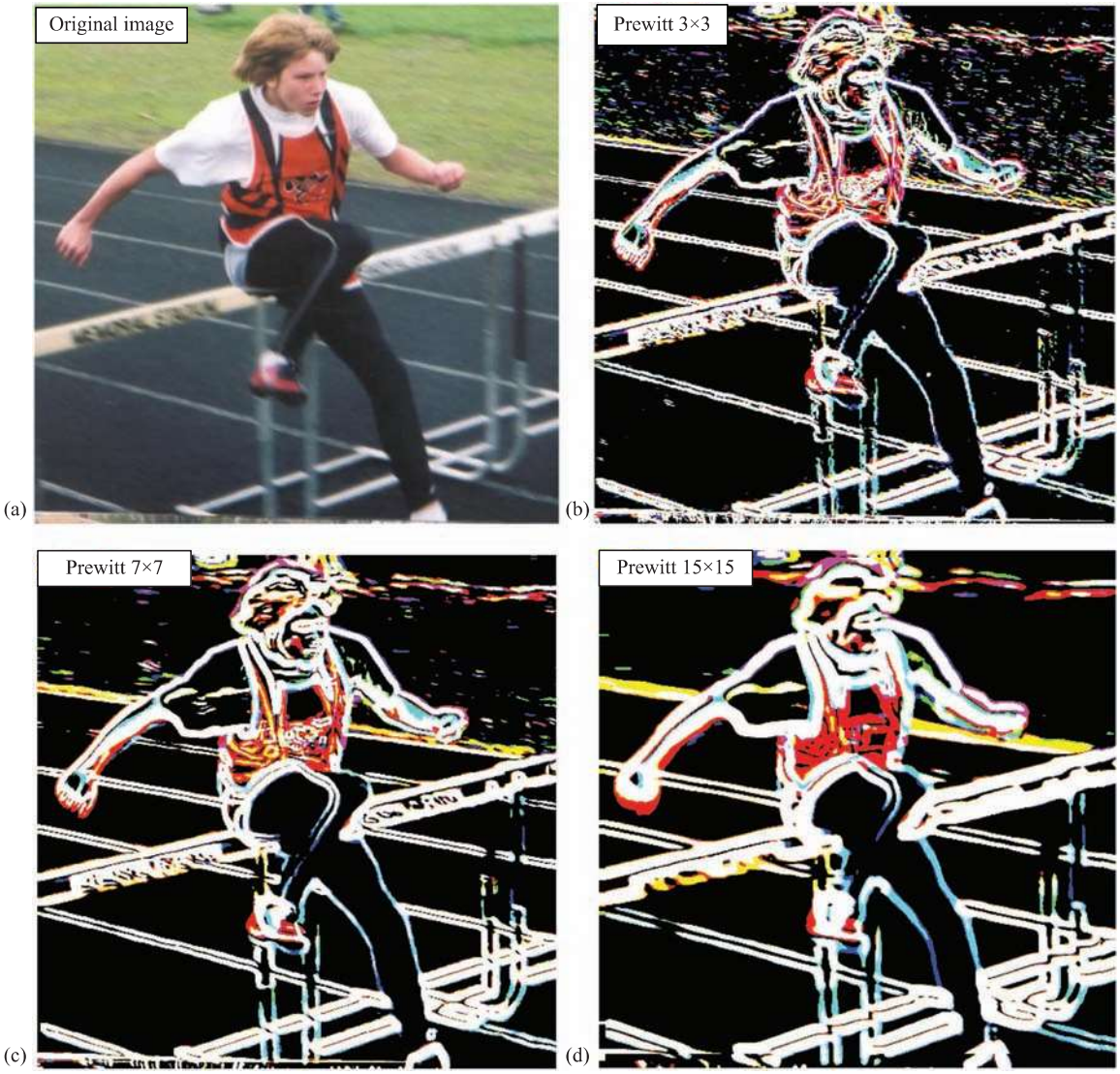

FIGURE 4.2-9
Edge detection example with various mask sizes. (a) Original image, (b) Prewitt magnitude with a 3×3 mask (c) Prewitt magnitude with a 7×7 mask and (d) Prewitt magnitude with a 15×15 mask. The images have undergone a threshold with the average value. Notice that a larger mask thickens the lines and eliminates small ones.

algorithm and the Frei–Chen masks. They are considered to be advanced because they are algorithmic in nature, which basically means they require multiple steps. Except for the Frei–Chen masks, these algorithms begin with the idea that, in general, most edge detectors are too sensitive to noise, and by blurring the image prior to edge detection, we can mitigate these noise effects. The noise considered here includes irrelevant image detail, as well as a combination of blurring from camera optics and signal corruption by camera electronics.

The simplest of these is the **Marr–Hildreth algorithm**, based on a model of the human visual system's response first developed by neuroscientist David Marr in the 1960s. The algorithm requires three steps:

1. Convolve the image with a Gaussian smoothing filter.
2. Convolve the image with a Laplacian mask.
3. Find the zero crossings of the image from Step 2.

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

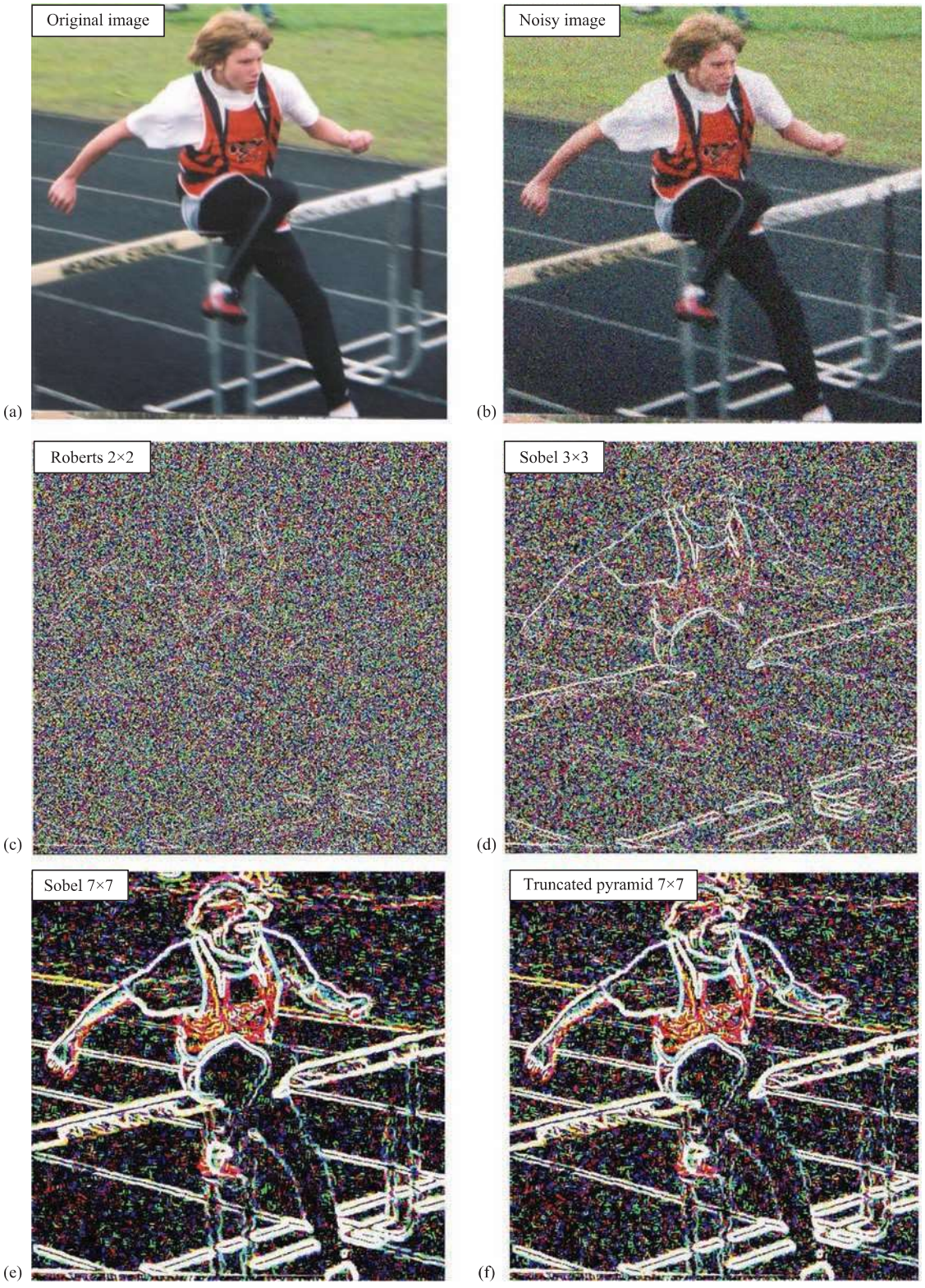

FIGURE 4.2-10
Edge detection examples with Gaussian noise – larger masks mitigate noise effects. (a) Original image, (b) image with added Gaussian noise (c) Robert's edge detection, 2×2, (d) Sobel with a 3×3 mask (e) Sobel with a 7×7 mask and (f) a 7×7 truncated pyramid. The images have undergone a threshold with the average value. In (c), with the 2×2 Roberts, the noise conceals almost all the edges. In (d), with a 3×3 Sobel mask, the edges are visible, but the resultant image is very noisy. With the 7×7 mask, shown in (e) and (f), the edges are much more prominent and the noise is much less noticeable.

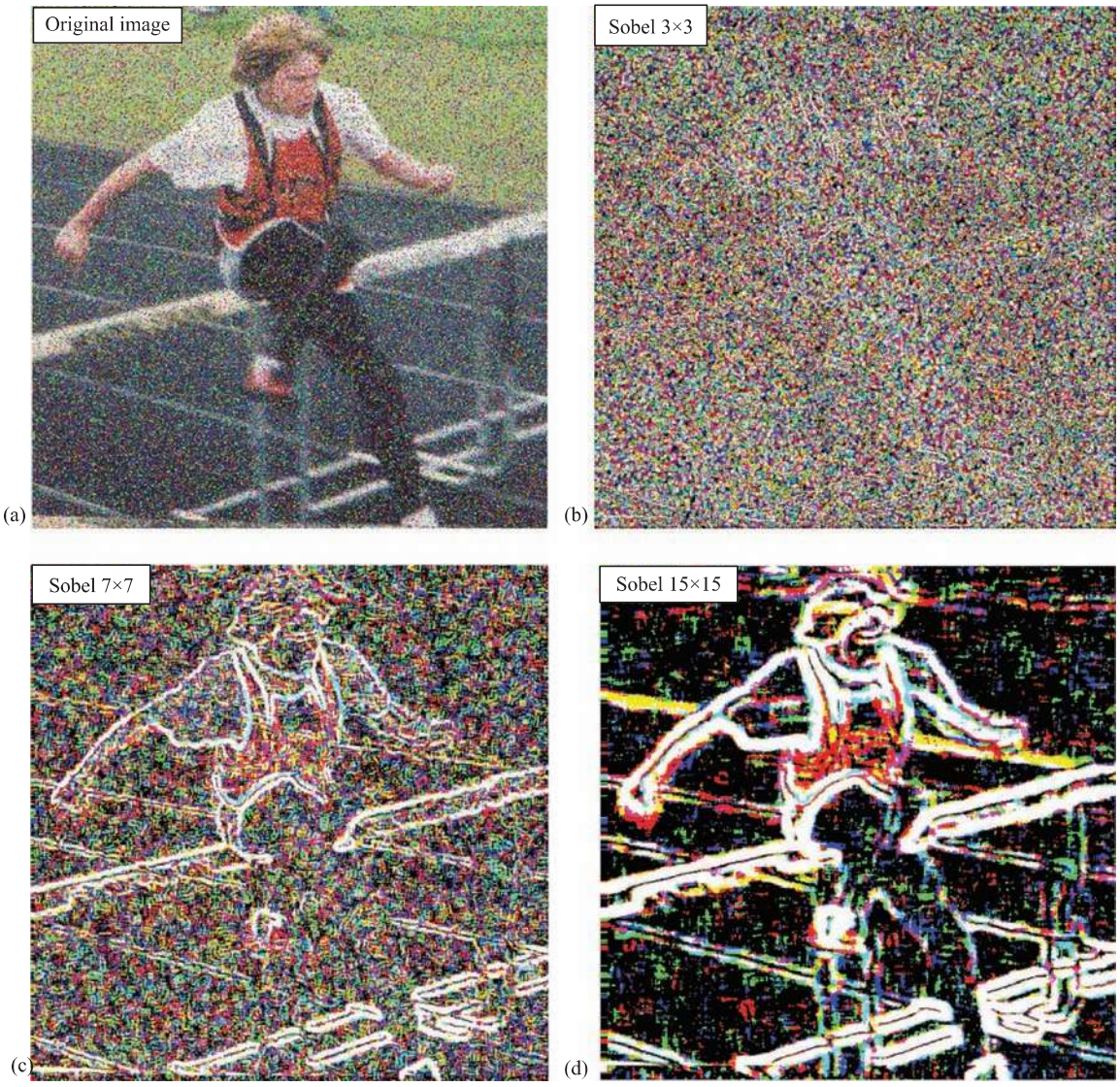

FIGURE 4.2-11
Edge detection with salt-and-pepper noise, Sobel example with various mask sizes. (a) Original image, (b) Sobel magnitude with a 3×3 mask (c) Sobel magnitude with a 7×7 mask and (d) Sobel magnitude with a 15×15 mask. The images have undergone a threshold with the average value. Notice that a larger mask helps mitigate the noise substantially, with a 3×3 the image is not even visible.

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 since the function resembles a sombrero (see Figure 4.2-12). Since 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 Laplacian of a Gaussian mask:

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

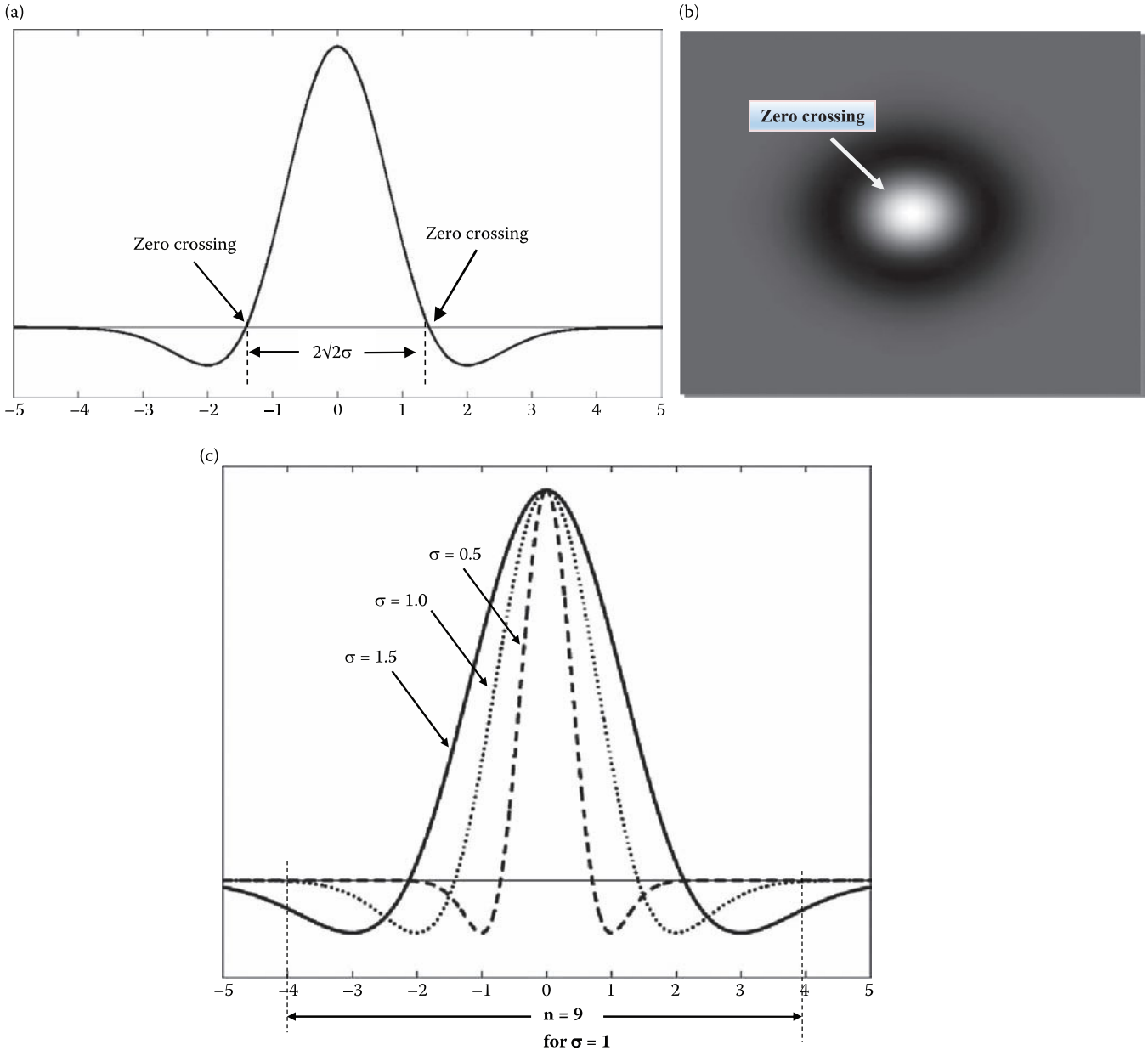

FIGURE 4.2-12

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; and (c) three *LoG* plots with $\sigma = 0.5$, 1.0 and 1.5. Note for $\sigma = 0.5$, the mask size, n, should be about 5×5; for $\sigma = 1$, 9×9, and so on. This is done so that 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.

17×17 Laplacian of a Gaussian mask:

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

The equation for the LoG filter is:

$$\text{LoG} = \left[\frac{r^2 + c^2 - 2\sigma^2}{\sigma^4}\right] e^{-\left(\frac{r^2+c^2}{2\sigma^2}\right)} \tag{4.2-7}$$

where (r, c) are the row and column coordinates, and σ^2 is the Gaussian variance and σ is the standard deviation. 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 4.2-12a. 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 one and that the Laplacian coefficients sum to zero. This is done to avoid biasing the mask with a DC 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 $\pm 3\sigma$ 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 4.2-12c). So, we want to select a value of n for the $n \times n$ convolution mask that is an odd integer greater than or equal to 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 standard deviation, σ:

$$n = \left[2 * \text{TRUNCATE}(3.35\sigma + 0.33) + 1\right] \tag{4.2-8}$$

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 $\pm 4\sigma$. Note that for positive numbers, the *truncate* operation is the same as the *floor* operation.

The third step for the Marr–Hildreth 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 4.2-13 illustrates the results from the standard Marr–Hildreth algorithm. The disadvantage of the Marr–Hildreth 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 Marr–Hildreth results are often referred to as a "plate of spaghetti", as seen in Figure 4.2-13b.

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.

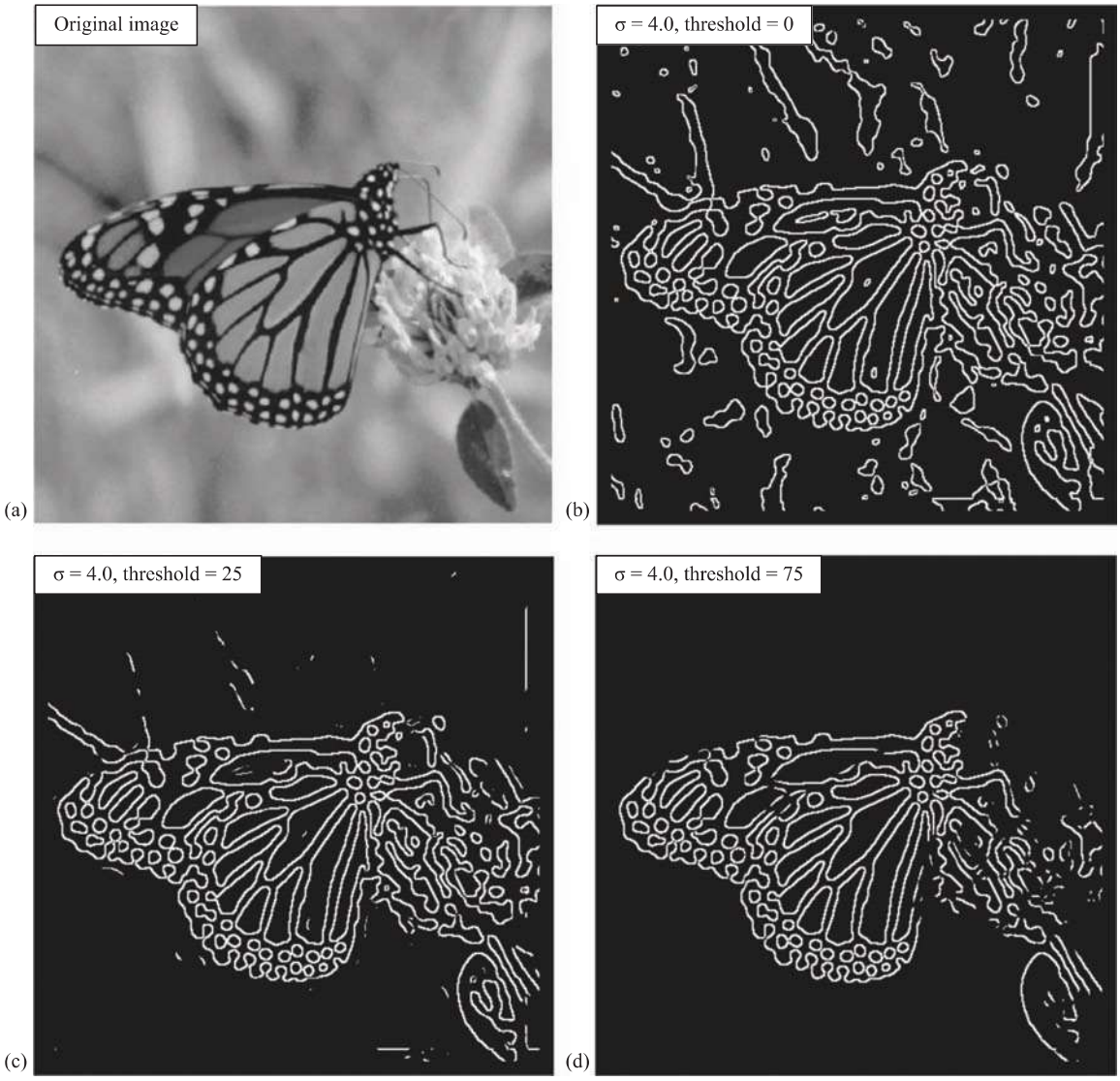

FIGURE 4.2-13
Marr–Hildreth algorithm results. (a) Original image, (b) Sigma (σ) = 4, threshold = 0, note the algorithm creating closed loops Marr–Hildreth algorithm results. (c) Sigma (σ) = 4, threshold = 25 and (d) Sigma (σ) = 4, threshold = 75; these values eliminate most of the background noise, but keep the butterfly.

EXAMPLE 4.2.1: APPLYING A THRESHOLD TO STEP 3 OF THE MARR–HILDRETH 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, we calculate the absolute value of the difference of this pair:

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

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

The Marr–Hildreth as implemented in CVIPtools 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, standard deviation) value and a *delta* value. CVIPtools then computes the Marr–Hildreth results using two variances – the specified sigma plus the delta value and the specified sigma minus the delta value. These results are then combined into a single image with a logical AND function. For color images, the user can also select to combine the bands, which performs a logical AND of the red, green and blue band results.

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, for example:

$$\begin{array}{cc} \text{VERTICAL} & \text{HORIZONTAL} \\ 1/2\begin{bmatrix} -1 & -1 \\ 1 & 1 \end{bmatrix} & 1/2\begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix} \end{array}$$

   These masks are each convolved with the image. At each pixel location, we find two numbers: c_1, corresponding to the result from the vertical edge mask, and c_2, 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:

$$\text{EDGE MAGNITUDE} \sqrt{c_1^{\,2} + c_2^{\,2}} \tag{4.2-9}$$

$$\text{EDGE DIRECTION Tan}^{-1}\left[\frac{c_1}{c_2}\right] \tag{4.2-10}$$

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 zero. Otherwise, it is a local maximum, so we keep it. In Figure 4.2-14, 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. Since it is greater than both, it is retained as an edge pixel; if it was less than either one, 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 an iterative process.

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

FIGURE 4.2-14
Nonmaxima suppression. A 3×3 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.

In CVIPtools, 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. The high threshold is multiplied with the high threshold factor to obtain the final high threshold for hysteresis. The low threshold is computed from the image by averaging the high threshold and minimum value in the image after applying the nonmaxima suppression to the magnitude images. The low threshold is then multiplied with the low threshold factor to obtain the final low threshold for hysteresis. CVIPtools also allows the variance of the Gaussian filter as an input parameter. Figures 4.2-15 and 4.2-16 show results from varying these parameters.

The **Boie–Cox algorithm**, developed in 1986 and 1987, is a generalization of the Canny algorithm. It consists of similar steps, but uses matched filters and Wiener filters (further explored in the references) to allow for a more generalized edge model. The **Shen–Castan algorithm**, developed in 1992, uses an optimal filter function they derived called an infinite symmetric exponential filter. Shen and Castan claim that their filter does better than the Canny at finding the precise location of the edge pixels. Like the Canny, it uses a smoothing filter followed by a similar multistep algorithm to find edge pixels. The search includes steps similar to the Canny, but with modifications and extensions (for more details, see the references). Figure 4.2-17 shows results from these algorithms.

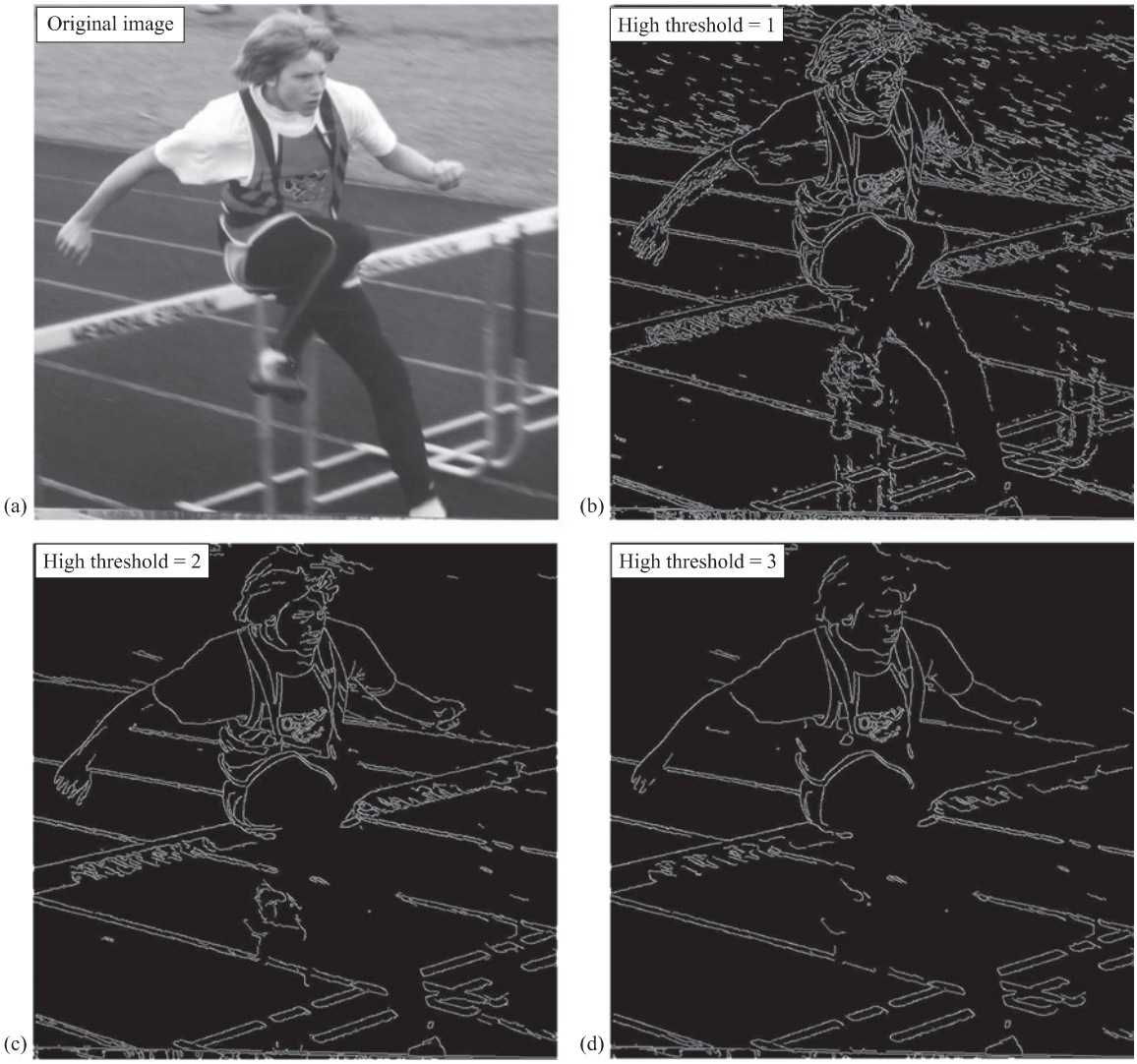

FIGURE 4.2-15
Results from changing high threshold with Canny algorithm. As the high threshold is increased, small details are removed. In the results, the Gaussian = 0.5 and the low threshold = 1. (a) Original image, (b) results high threshold factor = 1 (c) high threshold factor = 2 and (d) high threshold factor = 3.

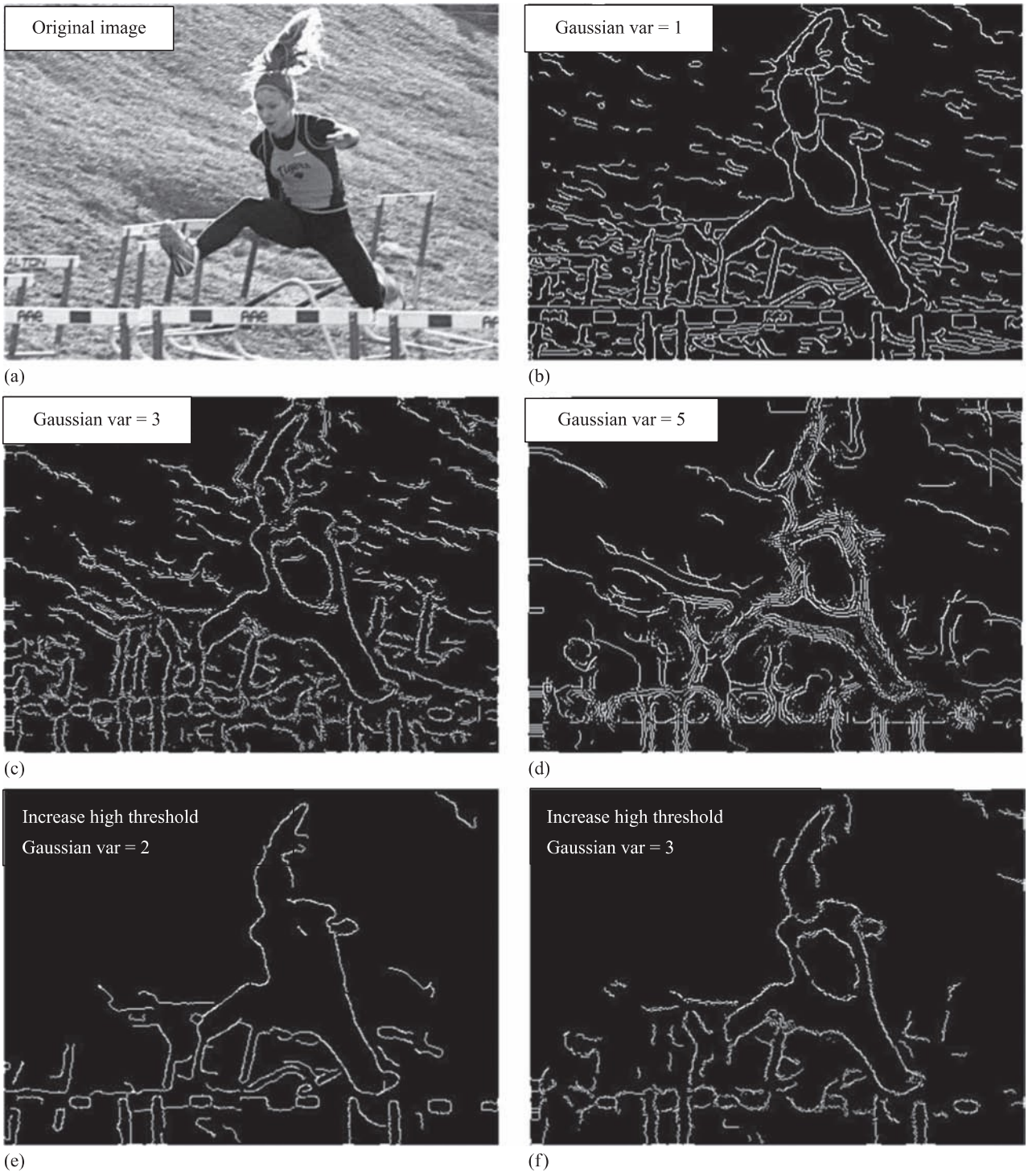

FIGURE 4.2-16

Results from changing Gaussian variance with the Canny algorithm. As the Gaussian variance is increased, the Canny should find fewer, but more prominent edges. In these results, the low and high threshold factors = 1. (a) Original image, (b) Canny results with Gaussian variance = 1, (c) Gaussian variance = 3 and (d) Gaussian variance = 5. Note that if the variance is too large, the image is blurred too much, and instead of finding "fewer, more prominent edges", we find fuzzy and then multiple edges. To avoid this, we need to also increase the high threshold. In the next two images, the low threshold is 1, but the high threshold has been increased to 2. Now we do see fewer, more prominent edges. (e) Gaussian variance = 2 and (f) Gaussian variance = 3. Results with noise in the image. Note that as the variance is increased, the false edges from the salt-and-pepper noise are mitigated.

(*Continued*)

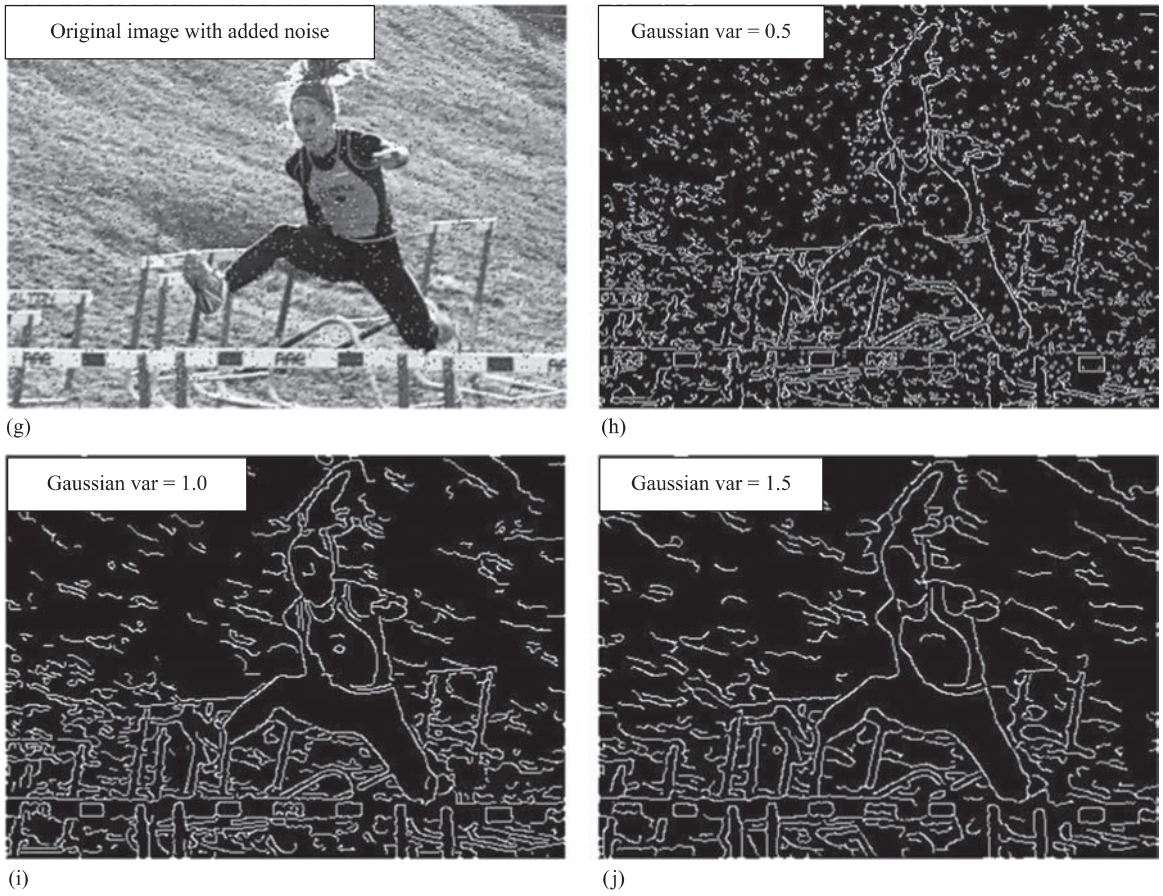

FIGURE 4.2-16 (*Continued*)
(g) original image with salt-and-pepper noise added, 2% each, (h) Canny with variance = 0.5, (i) Canny with variance = 1.0 and (j) Canny with variance = 1.5.

The **Frei–Chen masks** are unique in that they form a complete set of *basis vectors*. This means we can represent any 3×3 subimage as a weighted sum of the nine Frei–Chen masks (Figure 4.2-18). These weights are found by projecting a 3×3 subimage onto each of these masks. This projection process is similar to the convolution process in that both overlay the mask on the image, multiply coincident terms and sum the results (also called a *vector inner product*). This is best illustrated by example.

EXAMPLE 4.2.2

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:

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

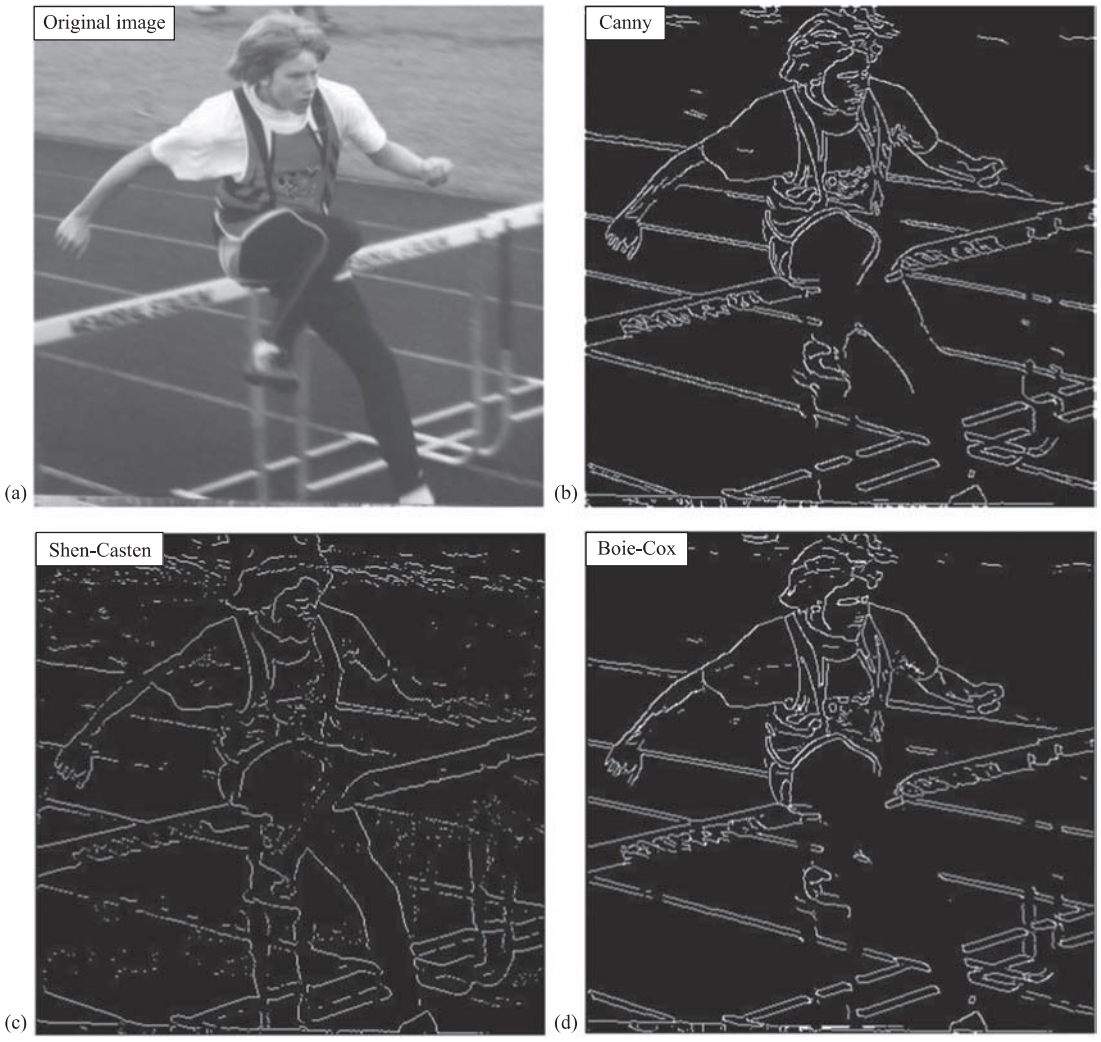

FIGURE 4.2-17
Comparison of the Canny, Shen–Castan and Boie–Cox algorithms. These results used the default parameters in CVIPtools. (a) Original image, (b) Canny (c) Shen–Castan and (d) Boie–Cox.

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

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

We can now see what is meant by a complete set of basis vectors allowing us to represent a subimage by a weighted sum. The basis vectors in this case are the Frei–Chen masks, and the weights are the projection values. Take the weights and multiply them by each mask, then sum the corresponding values. For this example, the only nonzero terms correspond to masks f_5, f_8 and f_9, and we find the following:

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

We have seen how the Frei–Chen masks can be used to represent a subimage as a weighted sum, but how are they used for edge detection? The Frei–Chen masks can be grouped into a set of four masks for an *edge* subspace, four

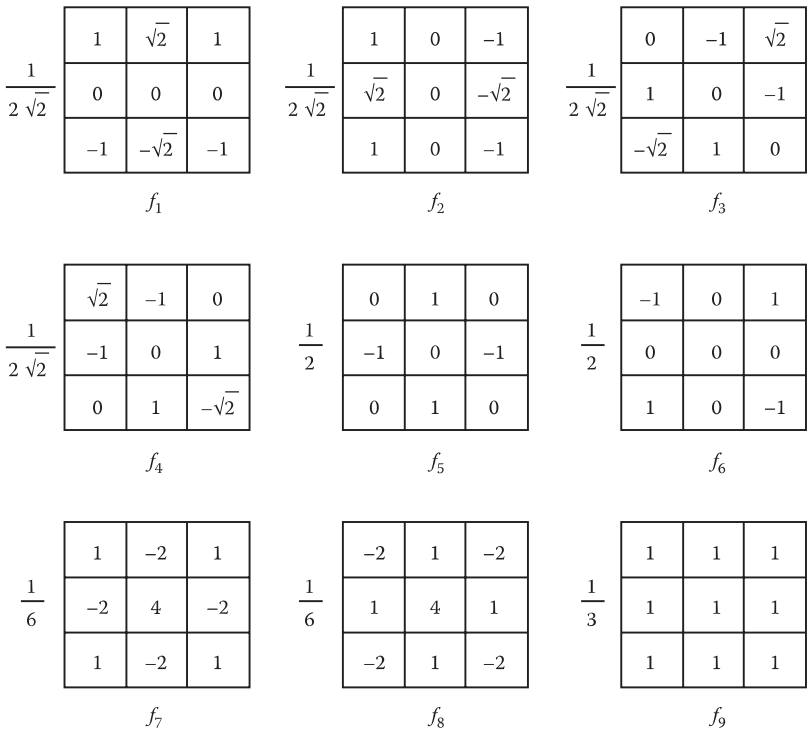

FIGURE 4.2-18
Frei–Chen masks. The first four masks, $f_1 - f_4$, comprise the edge subspace. The next four masks, $f_5 - f_8$, comprise the line subspace. The final mask, f_9, is the average subspace. More specifically, f_1 and f_2 are the *gradient* masks, f_3 and f_4 the *ripple* masks, f_5 and f_6 the *line* masks and f_7 and f_8 the *Laplacian* masks.

masks for a *line* subspace and one mask for an *average* subspace. These subspaces can be further broken down into *gradient*, *ripple*, *line* and *Laplacian* subspaces (see Figure 4.2-18). To use them for edge detection, select a particular subspace of interest and find the relative projection of the image onto the particular subspace. This is given by the following equation:

$$\cos(\Theta) = \sqrt{\frac{M}{S}} \tag{4.2-11}$$

where

$$M = \sum_{k \in \{e\}} (I_s, f_k)^2 \tag{4.2-12}$$

$$S = \sum_{k=1}^{9} (I_s, f_k)^2 \tag{4.2-13}$$

The set $\{e\}$ consists of the masks of interest. The (I_s, f_k) notation refers to the process of overlaying the mask on the subimage, multiplying coincident terms and summing the results – a vector inner product. The lengths of the vectors from the origin in the nine-dimensional Frei–Chen space are represented by $\sqrt{\boldsymbol{M}}$ and $\sqrt{\boldsymbol{S}}$, with $\boldsymbol{S}$ corresponding to the entire nine-dimensional subimage vector and $\boldsymbol{M}$ the subspace of interest. An illustration of this is shown in Figure 4.2-19. 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. To use for edge detection, we typically set a threshold on the angle to determine if a point will be considered a "hit" for the edge and/or line subspace of interest. Any pixel with a corresponding angle value below the threshold is similar enough to the subspace of interest to be considered a "hit" and is marked accordingly.

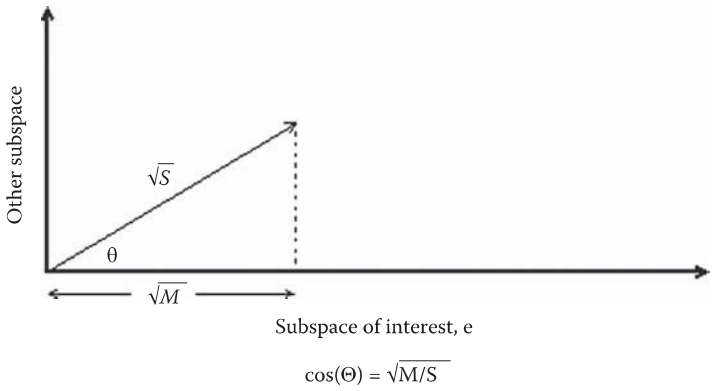

FIGURE 4.2-19
Frei–Chen projection. A 2-D representation of the Frei–Chen projection concept. The actual Frei–Chen space is nine-dimensional, where each dimension is given by one of the masks, f_k. ***S*** is the vector that represents the subimage and ***M*** is projection of the subimage vector onto the subspace of interest. The smaller the angle, the larger ***M*** is and the more the subimage is similar to the subspace of interest.

In CVIPtools, the user can select one of four different "projection" choices: (1) the edge subspace, (2) the line subspace, (3) the maximum of the edge and line subspace projection and (4) the minimum angle from the edge and line subspace projections. In all cases, CVIPtools will return an image of data type SHORT with the actual projection values. With the first two options, the user selects a threshold for the angle and only the pixel locations where the angle is smaller than the threshold will have the projection value – other pixels are set to zero. In the images shown in Figure 4.2-20, all the results have been postprocessed with a binary threshold operation, using 10% of the

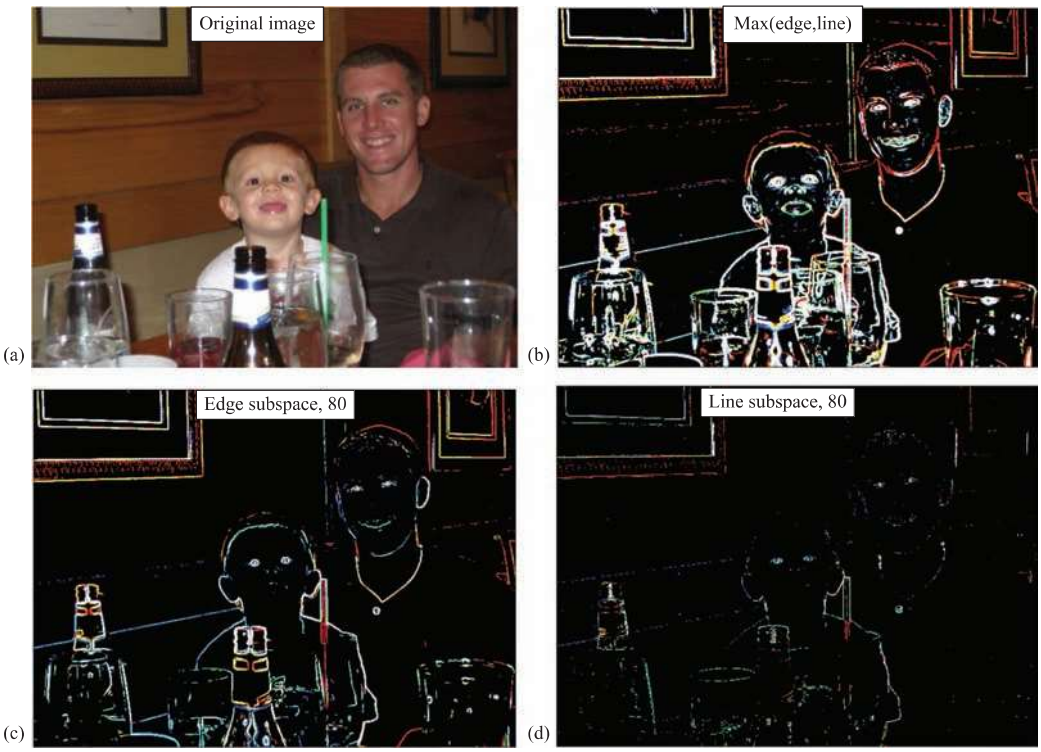

FIGURE 4.2-20
Frei–Chen results using CVIPtools. (a) Original image, (b) results from the maximum of edge and line selection (c) edge subspace angle threshold = 80 and (d) line subspace, angle threshold = 80. Note: for display purposes, the images shown have been postprocessed with a binary threshold operation using 10% of the maximum value of the output image.

maximum value in the Frei–Chen output image as the threshold. Figure 4.2-21 shows the effect of changing the binary threshold value.

The advanced edge detectors can also be used effectively in noisy images. Results from applying the Marr–Hildreth, Canny, Boie–Cox, Shen–Castan and Frei–Chen algorithms to an image with salt-and-pepper noise are shown in Figure 4.2-22. Here, we see that most of these algorithms perform well in the presence of salt-and-pepper noise. However, the Frei–Chen does not do as well as the others and the Marr–Hildreth is plagued by its usual "spaghetti effect". In Figure 4.2-23, we apply the same edge algorithms to an image with Gaussian noise. Here, we see that the Shen–Castan retains numerous spurious edges, and again, the Marr–Hildreth has the spaghetti effect. However, with Gaussian noise, the Frei–Chen, Canny and Boie–Cox appear to perform well.

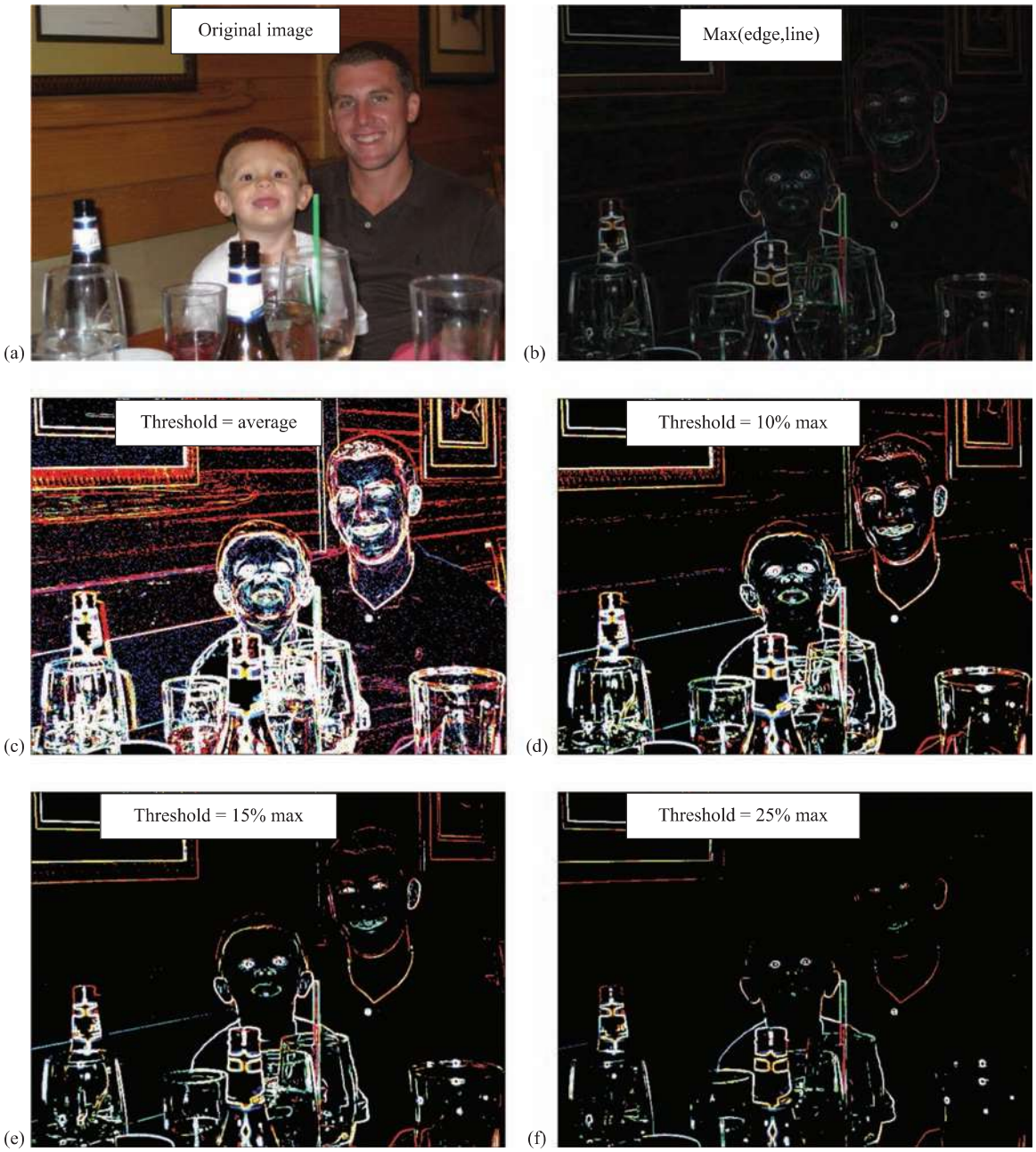

FIGURE 4.2-21

Changing the binary threshold with the Frei–Chen and maximum of edge or line subspace. (a) Original image, (b) output from the maximum value of the edge or line subspace (linearly remapped to BYTE), (c) threshold = average value, (d) threshold = 10% maximum value, (e) threshold = 15% maximum value and (f) threshold = 25% maximum value.

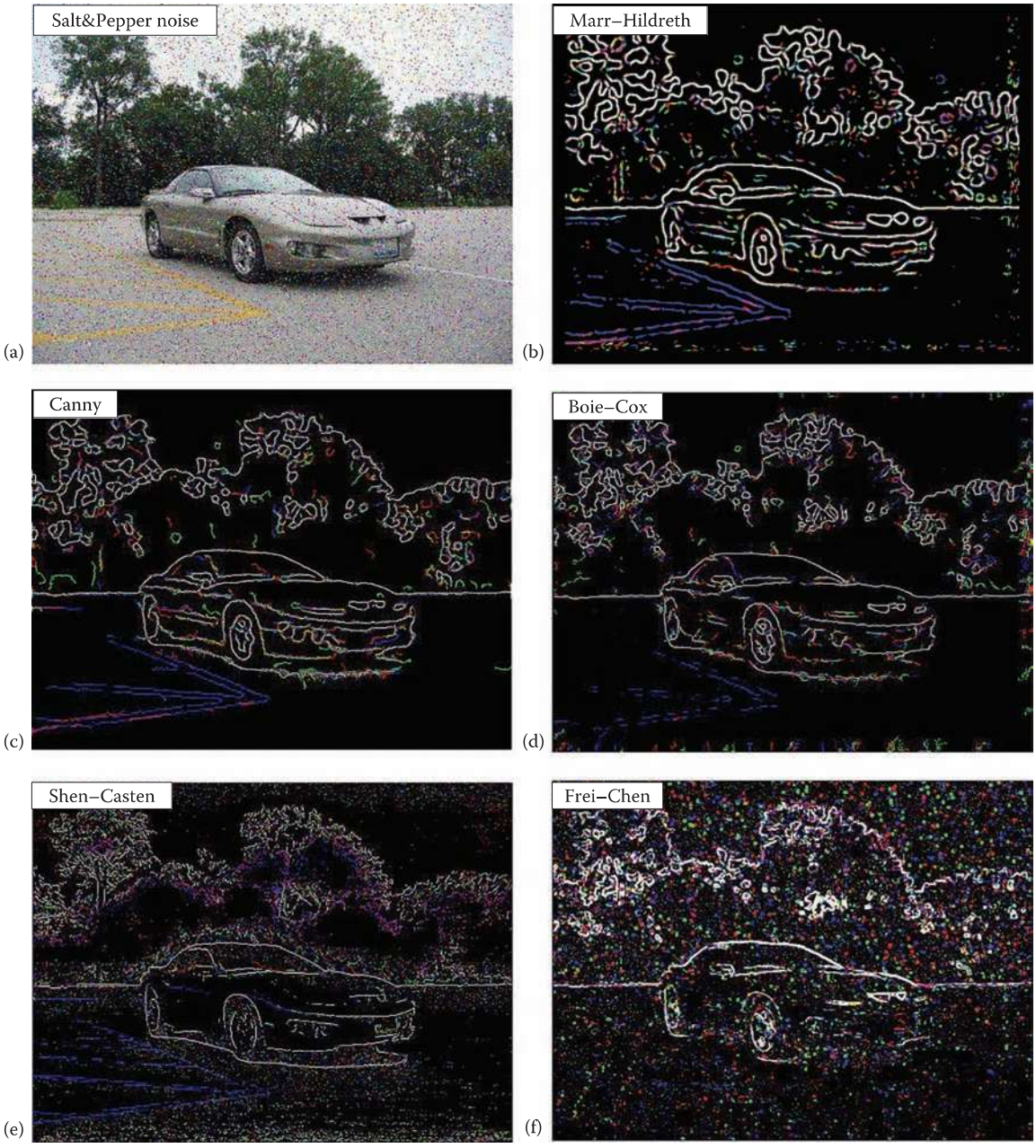

FIGURE 4.2-22
Advanced edge detectors with salt-and-pepper noise. (a) Original image with salt-and-pepper noise added with a probability of 3% each; (b) Marr–Hildreth, single variance, sigma = 4.0, threshold = 65 (c) Canny results, parameters: % Low Threshold = 1, % High Threshold = 1.5, Variance = 2; (d) Boie–Cox results, low threshold factor = 0.3, high threshold factor = 1.0, variance = 2.0 (e) Shen–Castan results, parameters: % Low Threshold = 1, % High Threshold = 2, Smooth factor = 0.9, Window size = 7, Thin Factor = 1; and (f) Frei–Chen results, parameters: Gaussian2 pre-filter, max(edge, line), post-threshold = 190. These results show that the Marr–Hildreth has the "spaghetti effect" as expected and that the Frei–Chen does not work too well with salt-and-pepper noise. The Canny, Boie–Cox and Shen–Castan work the best with salt-and-pepper noise.

4.2.5 Edges in Color Images

We saw in Chapter 1 that color images are described as three bands of monochrome image data and typical images use red, green and blue (RGB) bands. The simplest method to find edges in color images is to operate on each band separately – this is what we did in the previous figures. If color information is not important to the application, we may simply extract the luminance, or brightness, information and use the previously defined methods. Often, the

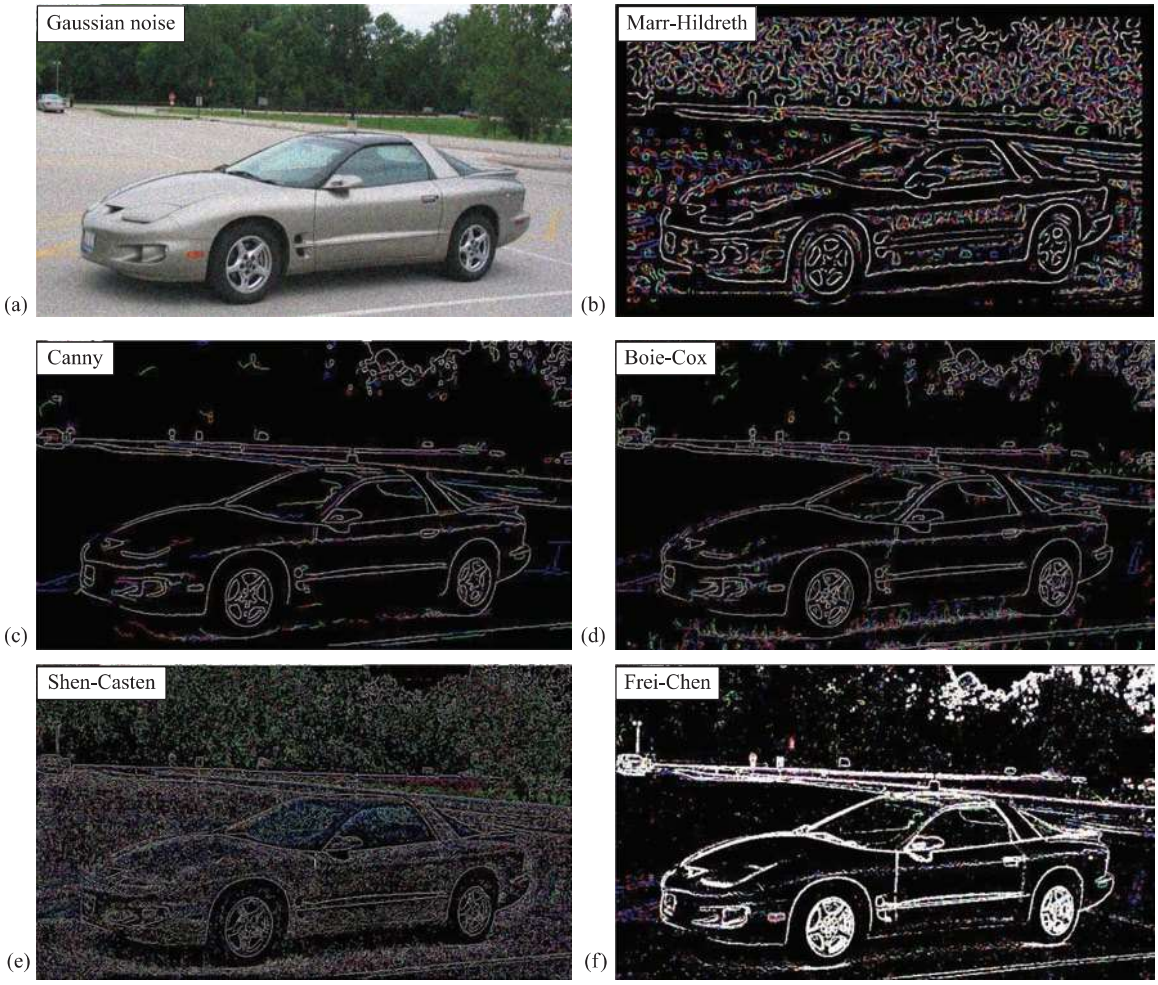

FIGURE 4.2-23

Advanced edge detectors with Gaussian noise. (a) Original image with Gaussian noise added with zero mean and a variance of 400; (b) Marr–Hildreth, dual variance, = 4.0, delta = 0.8; (c) Canny results, parameters: % Low Threshold = 1, % High Threshold = 1.5, Variance = 1.5; (d) Boie–Cox results, low threshold factor = 0.3, high threshold factor = 1.0, variance = 2.0; (e) Shen–Castan results, parameters: % Low Threshold = 1, % High Threshold = 2, Smooth factor = 0.8, Window size = 7, Thin Factor = 1; and (f) Frei–Chen results, parameters: Gaussian2 pre-filter, max(edge, line), post-threshold = 80. These results show that the Marr–Hildreth has the "spaghetti effect" as expected and that the Shen–Castan retains spurious edges with Gaussian noise. The Canny, Boie–Cox and Frei–Chen work the best with Gaussian noise.

optimal approach for finding edges in color images is application dependent and more than one possible definition of what constitutes a color edge exists.

The color transforms from Chapter 1 can be used to map the RGB images into different color spaces, and edges are found in one of the remapped bands. For example, for a particular application, we may not be interested in changes in brightness, but in changes in what we typically think of as "color"; so, the RGB data can be mapped into the HSV color space and edges are sought in the hue or saturation bands. Figure 4.2-24 illustrates this by showing that the areas of reflection are found in the saturation band, but not in the value (brightness) band.

Alternately, all three bands are used together. We can require an edge to be present in all three bands at the same location. With this scheme, any of the color spaces can be used, depending on the application, and we may want to define a quantum for "location error" and not require the edge to be at *exactly* the same pixel location in all three bands. Also, with this scheme, we can use any of the previously defined edge detection methods on each of the three bands individually. We can then combine the results from all three bands into a three-band image (as is done in CVIPtools, see Figure 4.2-25) or simply retain the maximum value at each pixel location from all three bands and output a monochrome image. With application-specific reasons, a linear combination of all three results can be used to create a monochrome image.

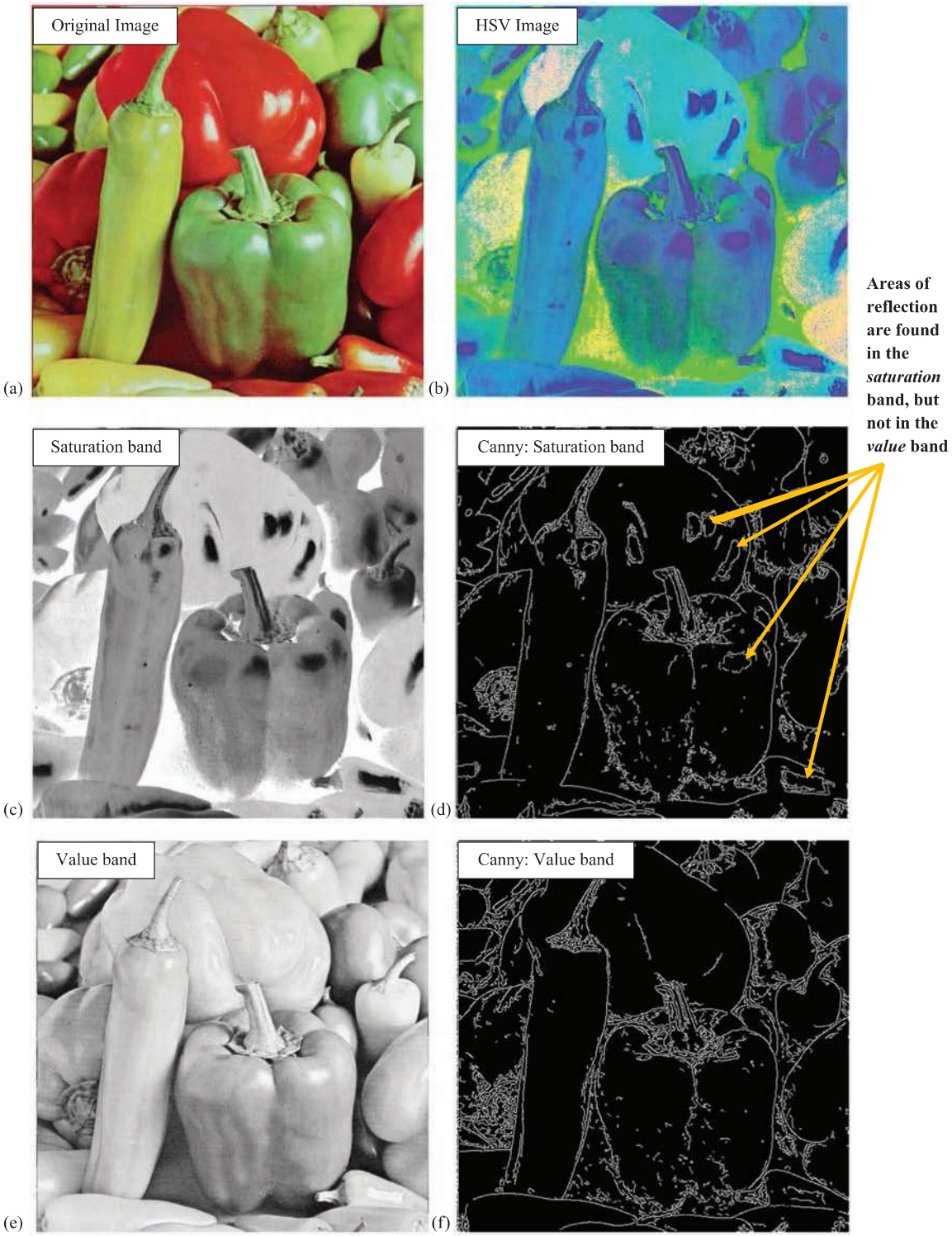

FIGURE 4.2-24

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 and (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).

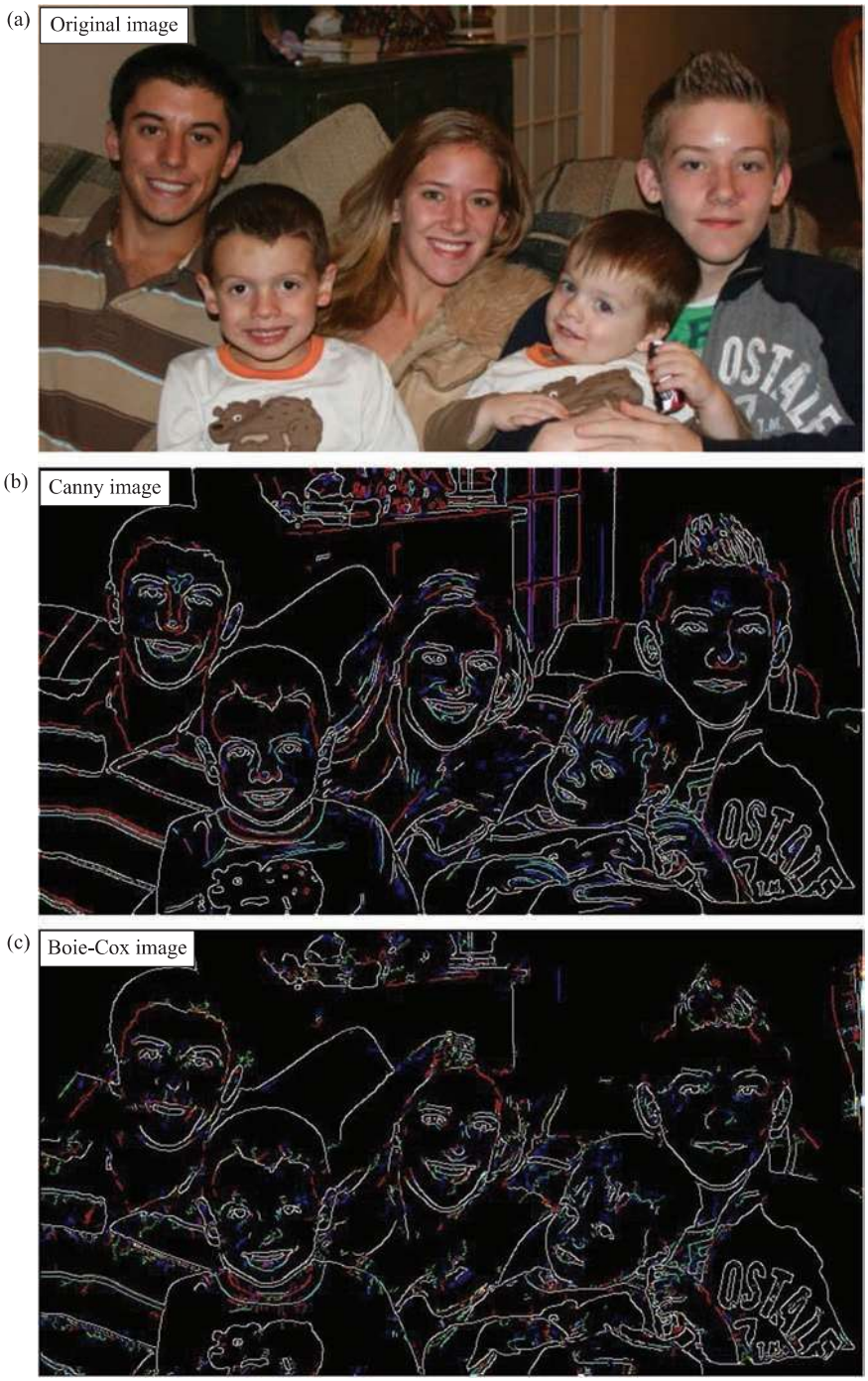

FIGURE 4.2-25
Color edge detection in RGB space. (a) Original image in RGB space, (b) Canny edge detector, all three bands displayed and (c) Boie–Cox edge detector, all three bands displayed. The edges that appear white are in all three RGB bands. Note that some edges only appear in one or two color bands.

Another method that uses all three bands simultaneously is to consider the color pixel vectors and search through the image marking edge points only if two neighboring color pixel vectors differ by some minimum distance measure. Here, we can use any vector distance measure, such as Euclidean distance (see Chapter 6 for definitions of other distance measures).

One specific method for finding edges in multispectral images, developed by Cervenka and Charvat in 1987, uses pixel values in all the image bands. This can be applied to three-band color images, as well 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. The result of this edge detector at pixel (r, c) is the smaller of the two values from these two equations:

$$\frac{\sum_{b=1}^{n}\left[I_b(r,c)-\bar{I}(r,c)\right]\left[I_b(r+1,c+1)-\bar{I}(r+1,c+1)\right]}{\sqrt{\sum_{b=1}^{n}\left[I_b(r,c)-\bar{I}(r,c)\right]^2\sum_{b=1}^{n}\left[I_b(r+1,c+1)-\bar{I}(r+1,c+1)\right]^2}} \tag{4.2-14}$$

$$\frac{\sum_{b=1}^{n}\left[I_b(r+1,c)-\bar{I}(r+1,c)\right]\left[I_b(r,c+1)-\bar{I}(r,c+1)\right]}{\sqrt{\sum_{b=1}^{n}\left[I_b(r+1,c)-\bar{I}(r+1,c)\right]^2\sum_{b=1}^{n}\left[I_b(r,c+1)-\bar{I}(r,c+1)\right]^2}} \tag{4.2-15}$$

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 bth band, with a total of n bands

This edge detector has been used successfully on multispectral satellite images. An example is shown in Figure 4.2-26. Here, we see the Cervenka and Charvat method applied and the results histogram equalized to show detail, and two different thresholds applied to the resultant image.

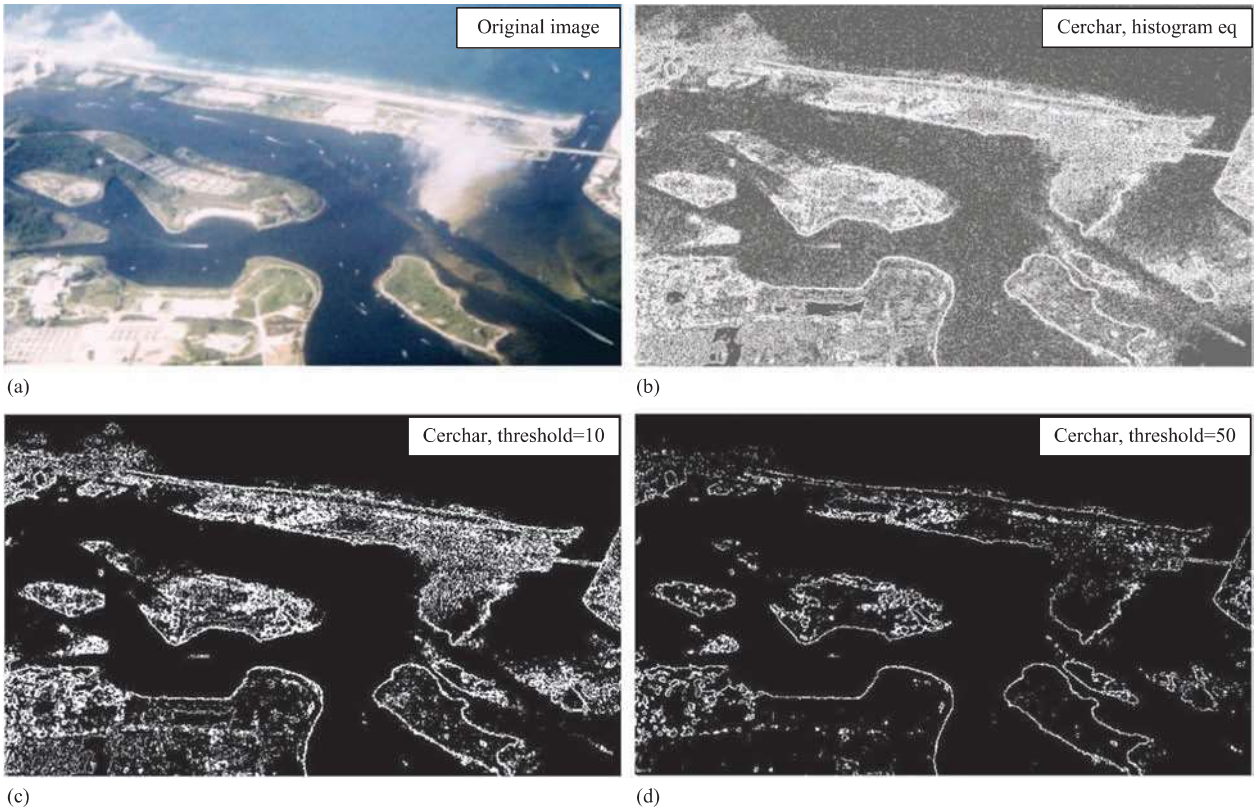

FIGURE 4.2-26
Cervenka and Charvat multispectral image detector. (a) Original image, (b) result from the *Cerchar* in CVIPtools after histogram equalization to show detail (c) result from thresholding the *Cerchar* image at 10 and (d) thresholding the *Cerchar* results at 50.

4.2.6 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. We will examine the types of errors encountered with edge detection, look at an objective measure based on these criteria and review results of the various edge detectors for our own subjective evaluation.

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 4.2-27). If these errors do not occur, a successful edge detection would have been achieved.

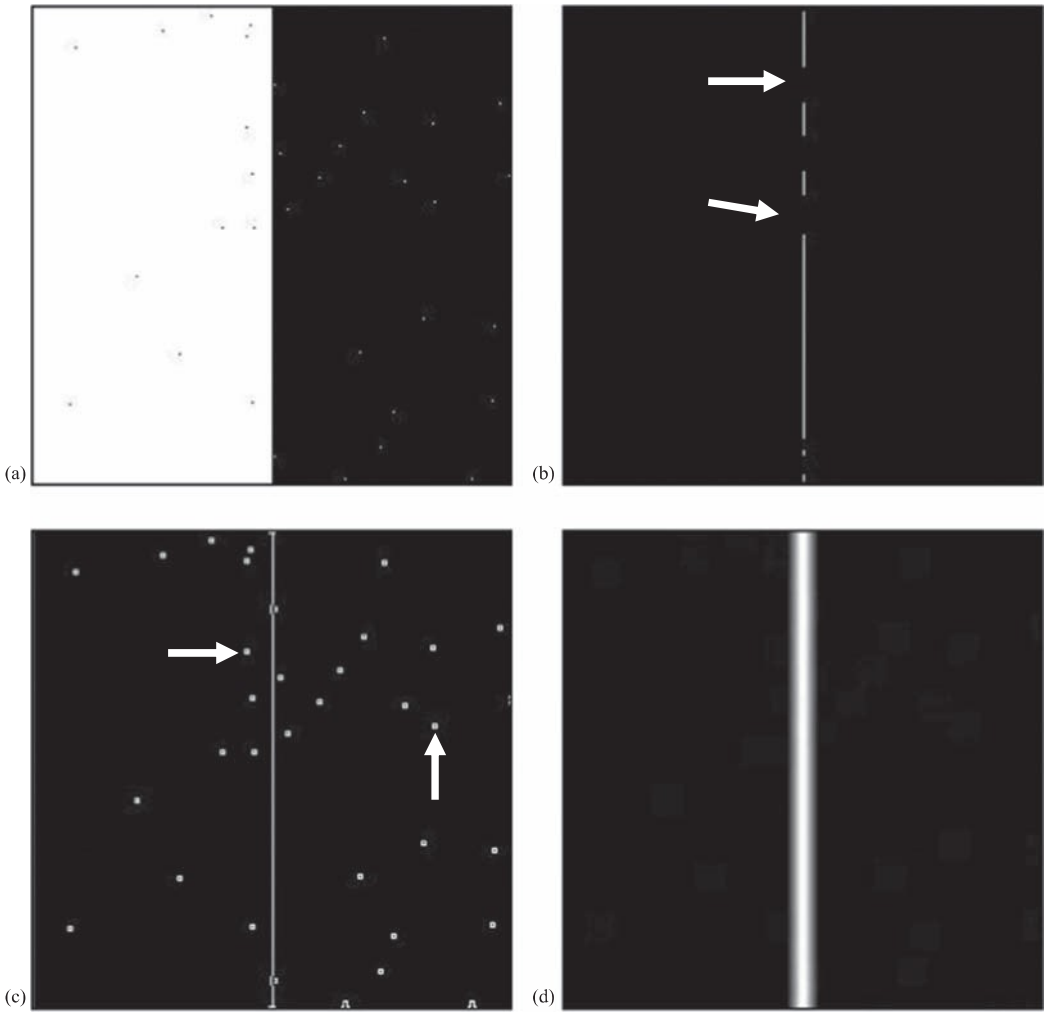

FIGURE 4.2-27
Errors in edge detection. (a) Original image, (b) missed edge points, examples marked with arrows, (c) noise misclassified as edge points, examples marked with arrows and (d) smeared edge.

The **Pratt's FOM** is defined as follows:

$$\text{FOM} = \frac{1}{I_N}\sum_{i=1}^{I_F}\frac{1}{1+\alpha d_i^2} \tag{4.2-16}$$

where
I_N is the maximum of I_I and I_F,
I_I is the number of *ideal* edge points in the image,
I_F 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 d_i 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:

$$d = |r_1 - r_2| + |c_1 - c_2| \tag{4.2-17}$$

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

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

$$d = \max\left(|r_1 - r_2|, |c_1 - c_2|\right) \tag{4.2-18}$$

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

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

$$d = \left[(r_1 - r_2)^2 + (c_1 - c_2)^2\right]^{1/2} \tag{4.2-19}$$

EXAMPLE 4.2.3

Given the following image array, find the Pratt 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:

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

$$\text{a.} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{b.} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \text{c.} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

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

b. $$\text{FOM} = \frac{1}{I_N}\sum_{i=1}^{I_F}\frac{1}{1+\alpha d_i^2} = \frac{1}{6}\left[\frac{1}{1+0.5(0)^2}+\frac{1}{1+0.5(0)^2}+\frac{1}{1+0.5(0)^2}+\frac{1}{1+0.5(1)^2}+\frac{1}{1+0.5(1)^2}+\frac{1}{1+0.5(1)^2}\right] \approx 0.8333$$

c. $$\text{FOM} = \frac{1}{I_N}\sum_{i=1}^{I_F}\frac{1}{1+\alpha d_i^2} = \frac{1}{4}\left[\frac{1}{1+0.5(1)^2}+\frac{1}{1+0.5(1)^2}+\frac{1}{1+0.5(1)^2}+\frac{1}{1+0.5(2)^2}\right] \approx 0.5833$$

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.

Applying the Pratt's FOM to selected edge detectors from each category – gradient operators, compass masks and the advanced edge detectors – results are shown in Figures 4.2-28 and 4.2-29. Figure 4.2-28 shows example test images, and the Pratt FOM results are plotted as the noise variance increases. The original test image has a gray level of 127 on the left and 102 on the right side and then Gaussian noise was added. Figure 4.2-29 shows resulting images with noise variances of 50 and 100 added to the test image. As expected, the advanced algorithms will have the best result as shown here with the Canny.

As previously mentioned, the objective metrics are often of limited use in practical applications, so we will take a subjective look at the results of the edge detectors. The human visual system is still superior, by far, to any computer vision system that has yet been devised and is often used as the final judge in application development. Figure 4.2-30 shows the magnitude images resulting from the basic edge detection operators. The magnitude images have been postprocessed with a threshold operation, using the average value for the threshold. Here, we see similar results from all the operators, but the Laplacian. This results from the Laplacian being based on the second derivative, while the others are based on the first derivative. In Figure 4.2-31, we show the magnitude and direction images from the basic gradient and compass mask edge detection operators. Here, we stretch the histogram of the magnitude images and remap the direction images from 0 to 255 (BYTE data type).

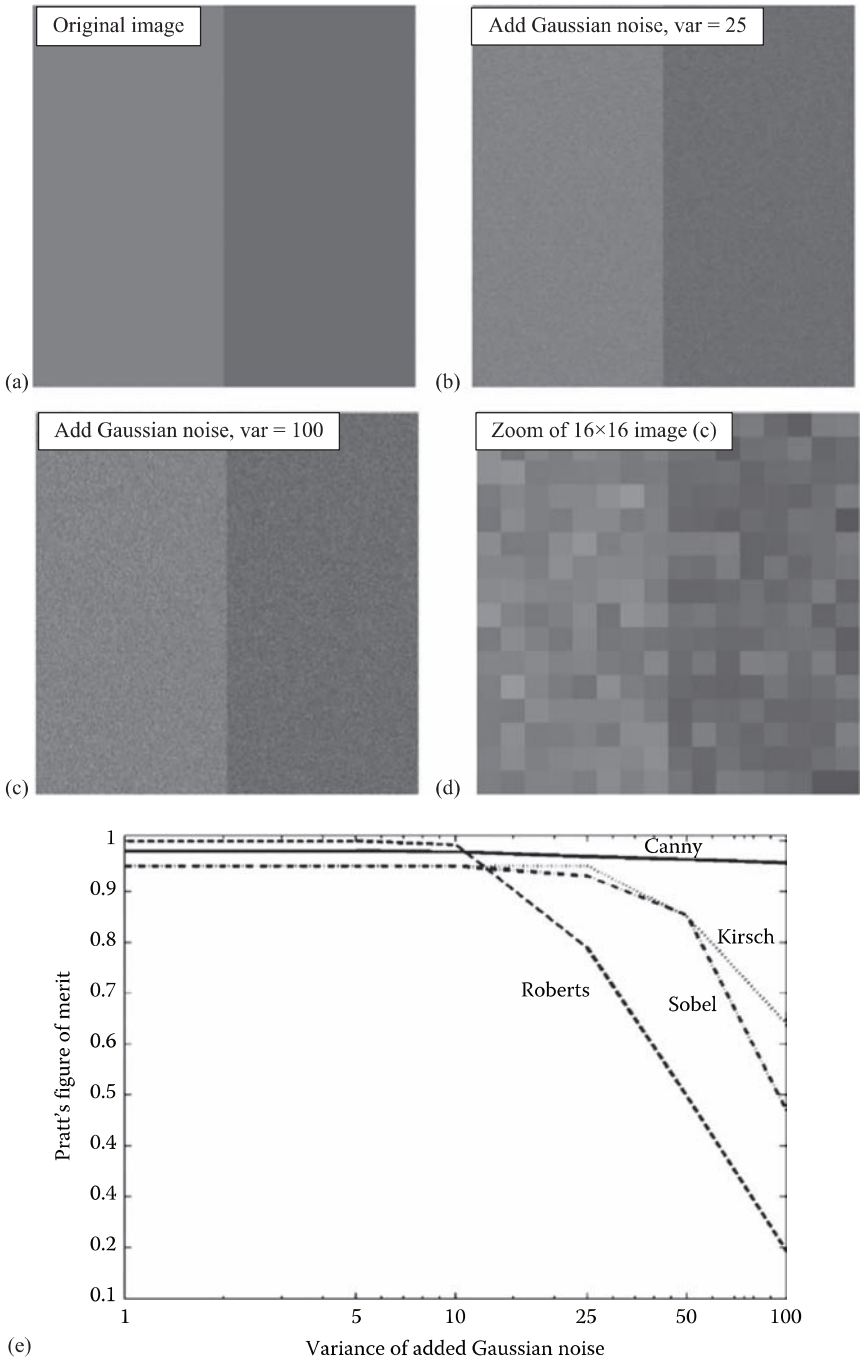

FIGURE 4.2-28

Pratt's figure of merit. (a) The original test image, 256 × 256 pixels, brightness level of 127 on the left and 102 on the right, (b) test image with added Gaussian noise with a variance of 25, (c) test image with added Gaussian noise with a variance of 100, (d) a 16 × 16 subimage cropped from image (c), enlarged to show that the edge is not as easy to find at the pixel level, and (e) this graph shows that as the noise variance increases the Canny has the best performance. We also see that the Roberts has the worst performance at high noise levels. The Roberts does poorly due to being based on a 2 × 2 mask, as opposed to the Sobel and Kirsch which are based on 3 × 3 masks. As we have seen with noisy images, a larger mask will perform better because it tends to spread the noise out – it is effectively a low-pass filter. The disadvantage of this is that fine details will be missed. This is the tradeoff that occurs with all edge detection – sensitivity versus accuracy. The test image was a step edge with Gaussian noise, so it is expected that the Canny performs the best because its development was based on this model.

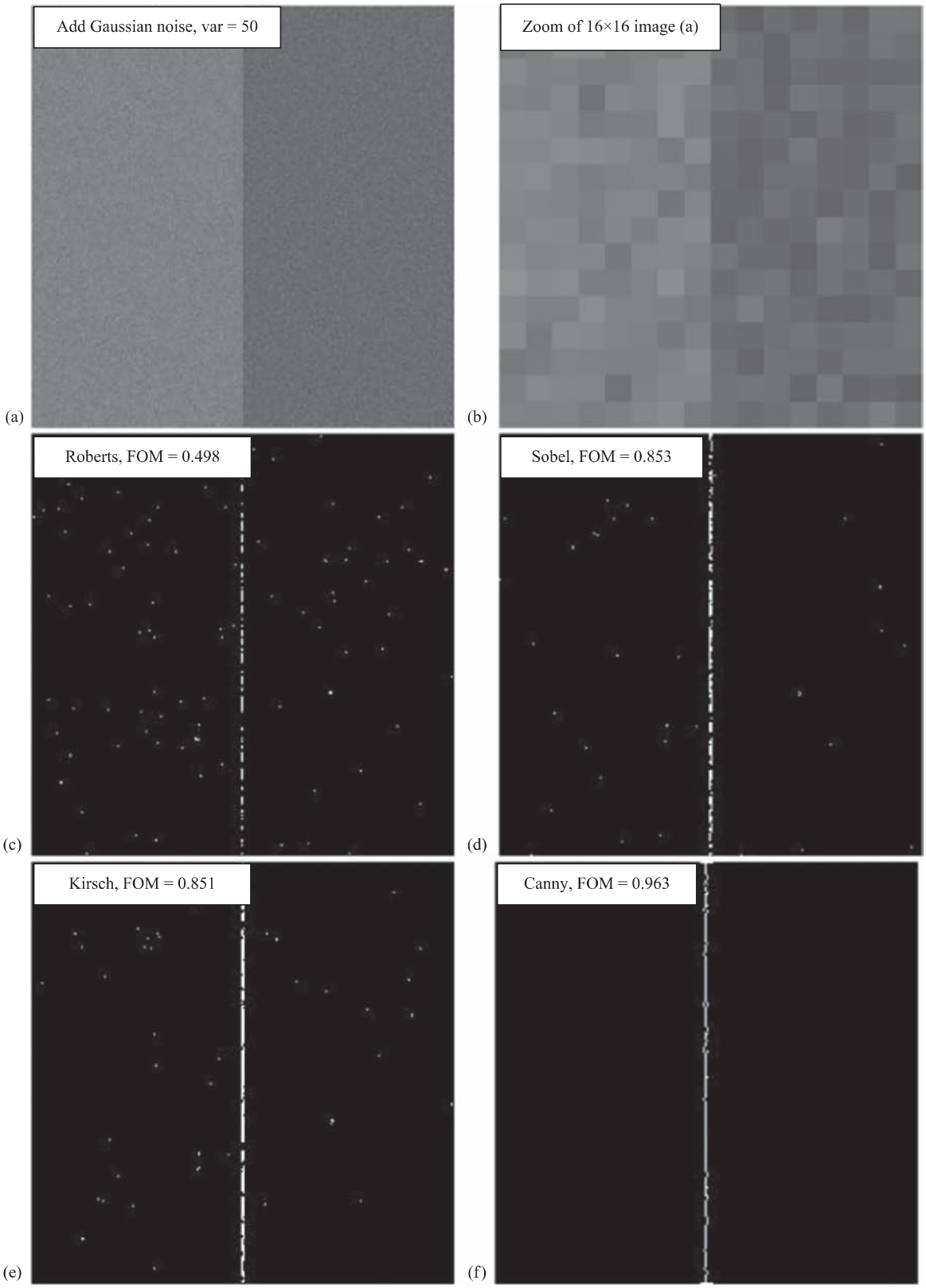

FIGURE 4.2-29

Pratt's figure of Merit images. (a) Test image with added Gaussian noise with a variance of 50, (b) a 16 × 16 subimage cropped from image (a), enlarged to show that the edge is not as easy to find at the pixel level, (c) Roberts result, FOM = 0.498, (d) Sobel result, FOM = 0.853, (e) Kirsch result, FOM = 0.851, (f) Canny result, FOM = 0.963

(*Continued*)

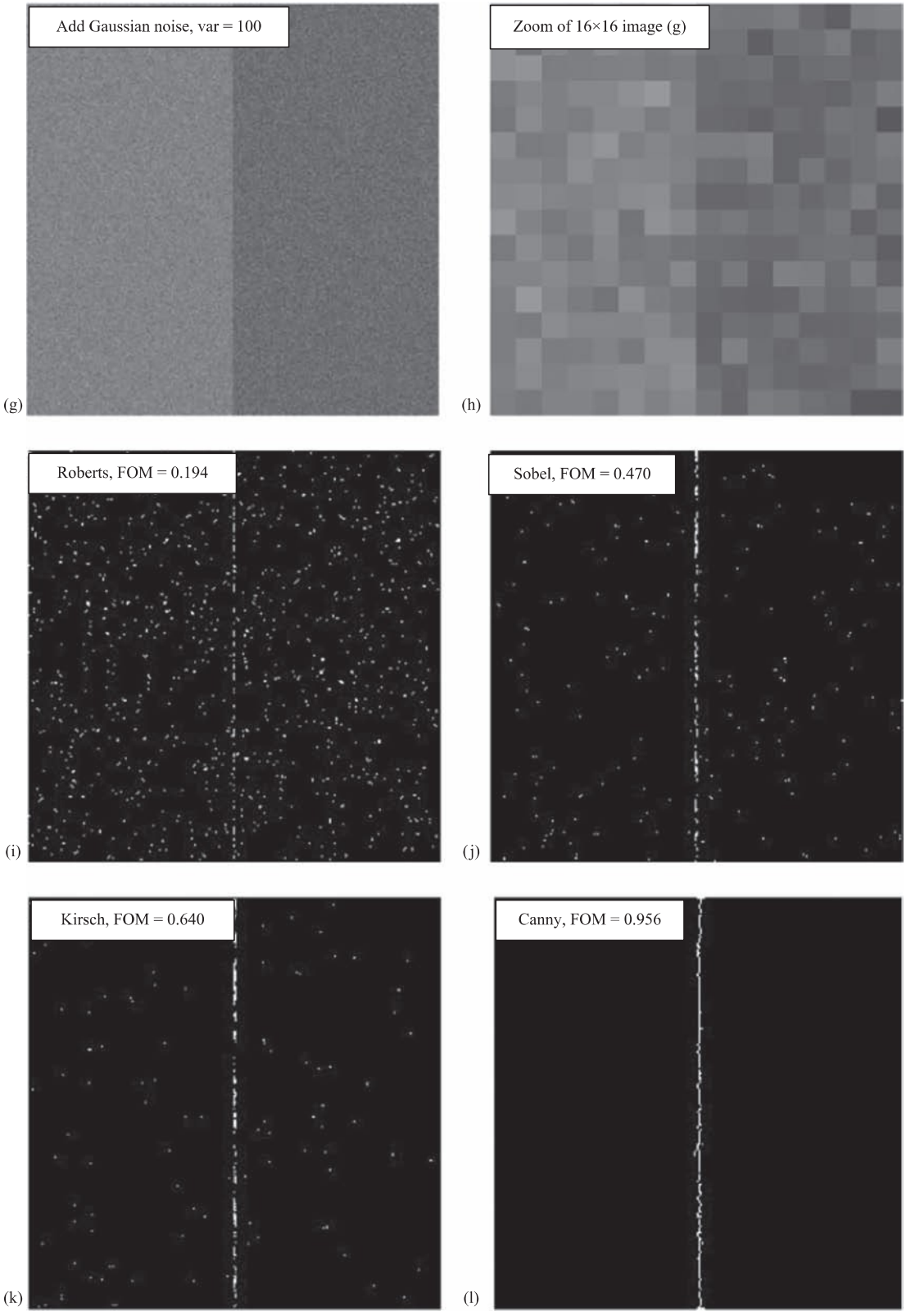

FIGURE 4.2-29 (*Continued*)
(g) test image with added Gaussian noise with a variance of 100, (h) a 16×16 subimage cropped from image (g), enlarged to show that the edge is not as easy to find at the pixel level, (i) Roberts, FOM = 0.194, (j) Sobel, FOM = 0.470, (k) Kirsch, FOM = 0.640 and (l) Canny, FOM = 0.956.

FIGURE 4.2-30
Edge detection examples. After the edge detector operator was performed, a threshold corresponding to the average value was used on the magnitude image. (a) Original image, (b) Roberts operator (c) Sobel operator, (d) Prewitt operator (e) Laplacian operator, (f) Kirsch operator (g) Robinson operator. Note that the resultant images all look similar, except for the Laplacian. The Laplacian is based on the approximation of the second derivative, unlike the others which are based on the first derivative.

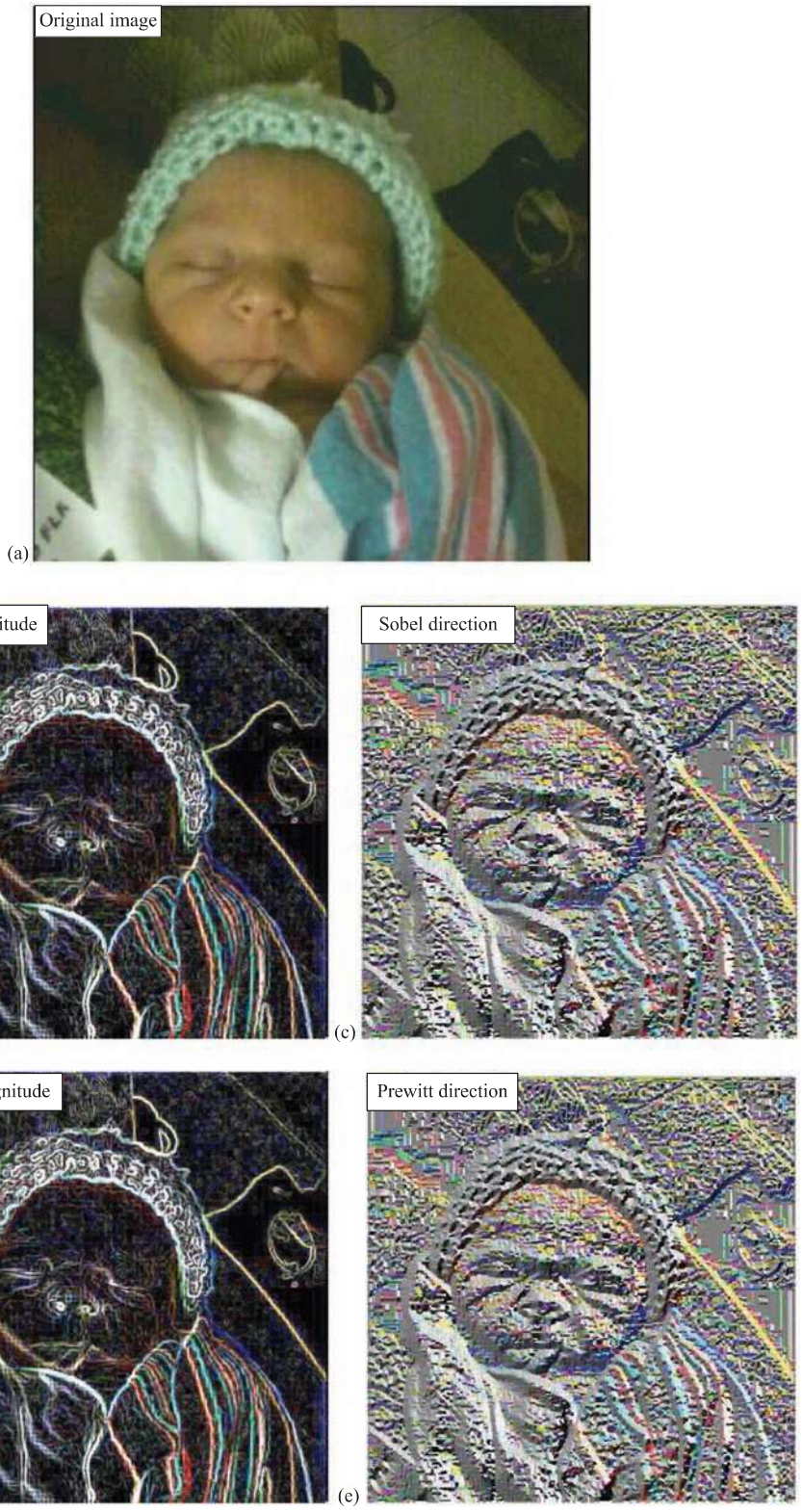

FIGURE 4.2-31
Edge detection examples with direction images. After the edge detector operator is applied, the magnitude image is remapped to BYTE and its histogram is stretched. The direction images are remapped to the BYTE range of 0–255. Note the original range on the direction images is $-\pi$ to $+\pi$. (a) Original image, (b) Sobel magnitude image, (c) Sobel direction image, (d) Prewitt magnitude, (e) Prewitt direction

(*Continued*)

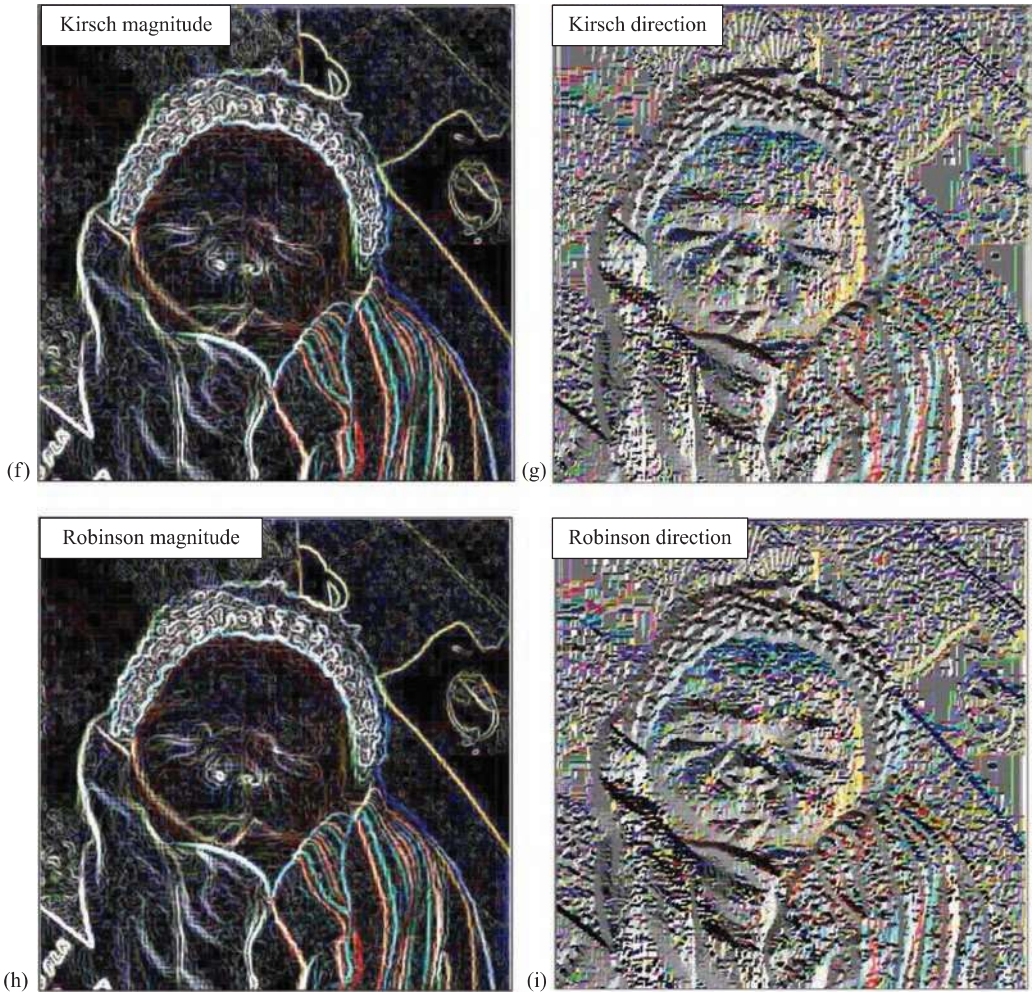

FIGURE 4.2-31 (*Continued*)
(f) Kirsch magnitude, (g) Kirsch direction, (h) Robinson magnitude and (i) Robinson direction. Note that the magnitude images all look similar, but the Sobel/Prewitt direction images differ from the Kirsch/Robinson due to the method in which they are defined.

4.3 Line Detection

4.3.1 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. Although a brute force search method can be used that will find all the lines associated with each pair of points, by checking every point with every possible line, it involves finding $n(n-1)/2$ (on the order of n^2) lines and comparing every point to all the lines which is $(n)(n(n-1))/2$ or about n^3 comparisons. This heavy computational burden is certainly not practical for real-time applications, and this provides much more information than is necessary for most applications. 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:

$$\rho = r\cos(\theta) + c\sin(\theta) \tag{4.3-1}$$

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 ρ -line (see Figure 4.3-1). Now, for each pair of values of ρ and θ, we have defined a particular line. The parameter θ ranges from 0° to 180°, and ρ ranges from 0 to $\sqrt{2}\ N$, where $N \times N$ is the image size – for a nonsquare image, it is the diagonal length. Next, we can take this $\rho \cdot \theta$ parameter space and quantize it to reduce our search time. We quantize the $\rho \cdot \theta$ parameter space, as shown in Figure 4.3-2, by dividing the space into a specific number of blocks. Each block corresponds to a line, or group of possible lines, with ρ and θ varying across the increment as defined by the size of the block. The size of these blocks corresponds to the coarseness of the quantization; bigger blocks provide less line resolution.

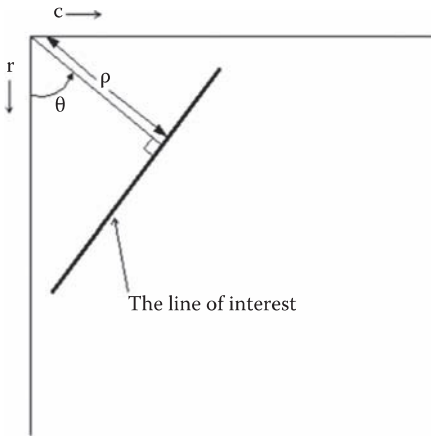

FIGURE 4.3-1
The Hough transform can be defined by using the normal (perpendicular) representation of a line and the parameters ρ and θ.

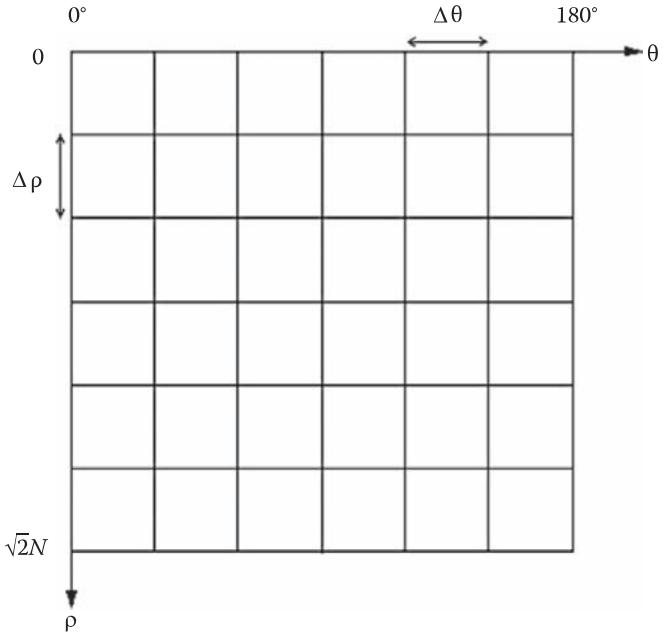

FIGURE 4.3-2
The quantized Hough space. Theta, θ, varies from 0° to 180°, and rho, ρ, varies from 0 to $\sqrt{2}N$ for a square $N \times N$ image. Each block in this quantized space represents a group of lines whose parameters can vary over one increment of ρ and θ, defined by $\Delta\rho$ and $\Delta\theta$.

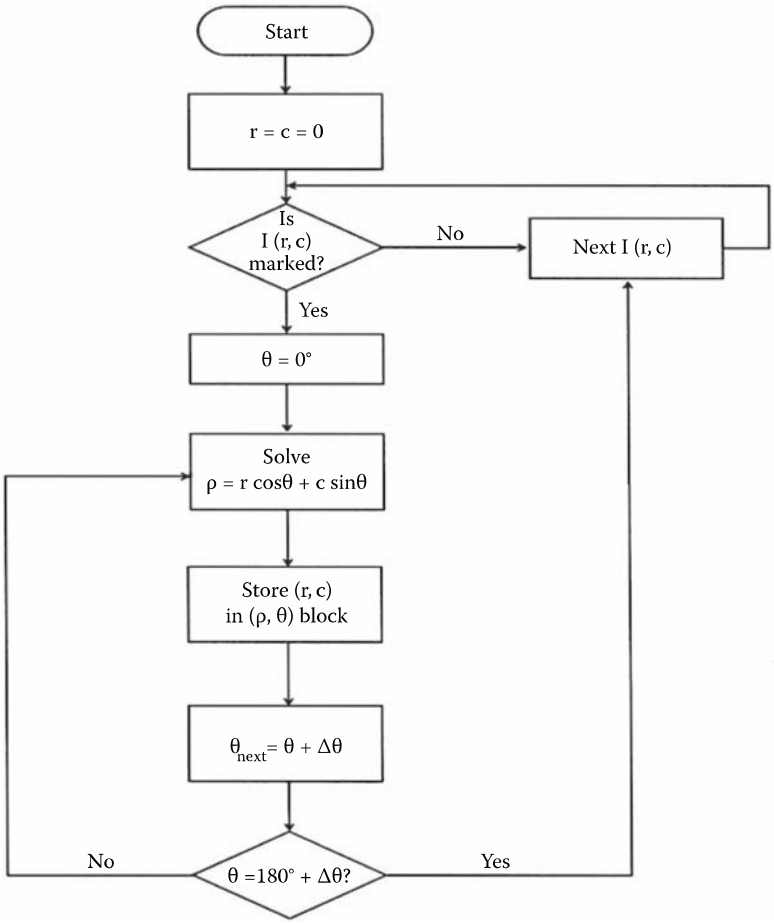

FIGURE 4.3-3
Hough transform flowchart. The flowchart is followed until all $I(r, c)$ have been examined. The entire parameter space, $\rho\theta$, is cycled by using the corresponding quantizations, $\Delta\theta$ and $\Delta\rho$, for each marked point of interest.

The algorithm used for the Hough transform (see Figure 4.3-3 for a flowchart of the process) will help understand what this means. The algorithm consists of three primary steps:

1. Define the desired increments on ρ and θ, $\Delta\rho$ and $\Delta\theta$; 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 (4.3-1):

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

   Then, for each value of θ in the quantized space, solve for ρ.
3. For each $\rho \cdot \theta$ 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.

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. Examining Figure 4.3-4, we can see that this means the line of interest in the image space can vary more. One block in the Hough space corresponds to all the solid lines in this figure – this is what we mean by reduced line resolution.

Next, select a threshold and examine the quantization blocks that contain more points than the threshold. Here, we look for continuity by searching for gaps in the line by looking at the distance between points on the

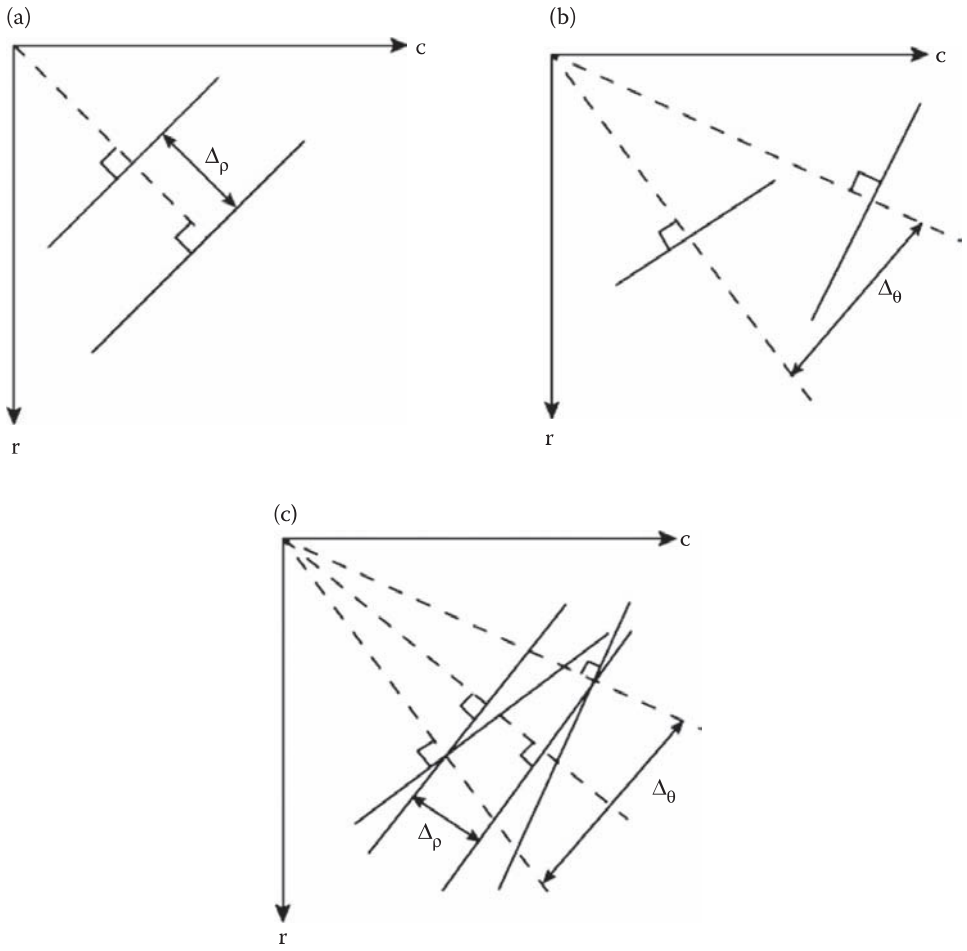

FIGURE 4.3-4
Effects of quantization block size for Hough transform. (a) Range of lines included by choice of $\Delta\rho$. (b) Range of lines included by choice of $\Delta\theta$. (c) Overall range of lines defined by $\Delta\rho$ and $\Delta\theta$, corresponding to a quantization block.

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. Note that the Hough transform will allow us to look for lines of specific orientation, if desired.

4.3.2 Postprocessing

A more advanced postprocessing algorithm is implemented in CVIPtools with the Hough transform. Images resulting from this algorithm searching for lines at 45° are shown in Figure 4.3-5, and any of these intermediate images is available as output in CVIPtools with the *Output Image* select box for 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 1° in CVIPtools).
2. Threshold image2 by using the parameter *line pixels*, which is the minimum number of pixels in a line (or 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 4.3-5c. Note that these lines span the entire image.
3. Perform a logical operation, image1 AND image3. The result is image4, see Figure 4.3-5d.

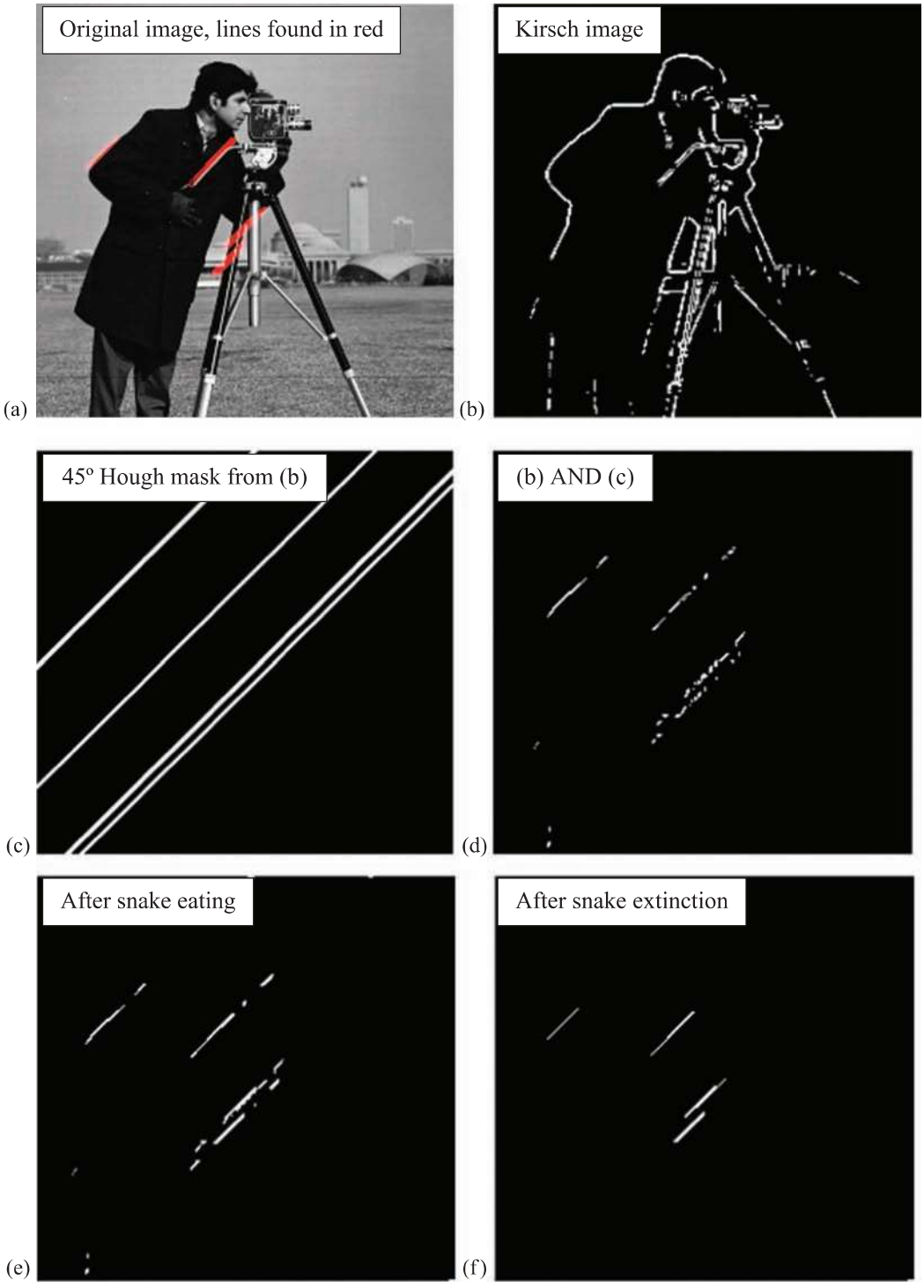

FIGURE 4.3-5

Hough transform postprocessing algorithm details. The Hough parameters used are as follows: *Line Angles*: 45°, *Line Pixels (min)*: 25, *Connect distance (max):* 5, *Delta Length*: 1 and *Segment Length (min)*: 15. (a) Original image with final lines found in red; (b) image after applying the Kirsch edge operator and a threshold operation; (c) the mask image created from the Hough result for lines at 45°; (d) result of logical AND of the images in (b) and (c); (e) image (d) after snake eating, see that the camera's handle has been connected and (f) the final result after snake extinction, small dashed lines are removed. Note that 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°; 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.

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 4.3-5e). 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, as illustrated in Figure 4.3-5f.

CVIPtools parameters for the Hough transform:

Line angles: The range of angles for which the Hough transform will search.

Line pixels (min): The minimum number of pixels must a line 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.

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 length (min): The minimum number of pixels in a line segment for it to be retained.

Segment length: controls how many pixels a solid line must have, while *line pixels* controls how many pixels a dashed line must have.

The result of applying the Hough transform to an aerial airport image to find the runway is shown in Figure 4.3-6. The Canny edge detection operator was used on the original image to provide input to the Hough transform. The

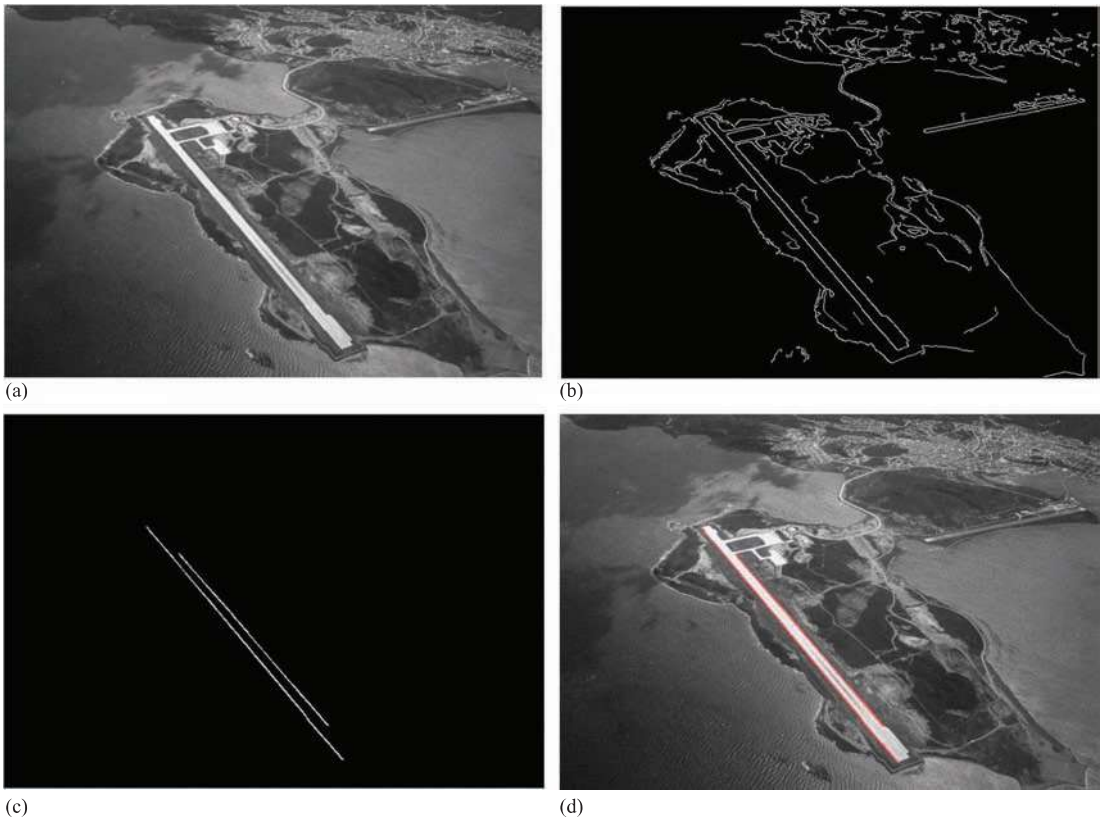

FIGURE 4.3-6
Hough transform to find airport runway. (a) Aerial image of an airport; (b) after Canny edge detection, low threshold factor = 1, high threshold factor = 2, variance = 0.5 (c) Hough output with the range of line angles = 130°–170°, delta length (ρ) = 1, minimum number of pixels per line = 35, maximum connect distance = 1, minimum segment size = 35 and (d) Hough output in red superimposed over the original image, showing the airport runway.

Canny parameters were low threshold factor = 1.0, high threshold factor = 2.0 and variance = 0.5. The Hough transform parameter *delta length (rho)* was set at 1, *line pixels* (the number-of-points threshold) was set at a minimum of 35 pixels per line and *segment length* set to 35. We see that the Hough transform provides an efficient line finding algorithm and it can also be used as a boundary detection segmentation method; these are discussed more in Chapter 5.

4.4 Corner and Shape Detection

4.4.1 Corner Detection

We have seen that edges are found by considering the rate of change, or gradient, in image brightness (gray level) in a specific direction. Lines and curves are then a collection of these edge points along a specific path. *Corners* are simply points where there is a high rate of change in more than one direction; using this definition, corners in this sense are also referred to as *points of interest*. Corner detection is useful for many applications. For example, object tracking is facilitated by the ability to delineate an object by its corners and following the movement of the corners through space. In addition to tracking the object in space, the orientation can be followed more easily with corner detection. Corners are also useful features for matching multiple images, for example, to use as match points for creation of three-dimensional models from stereo images.

Corners are used by the human visual system to provide cues about object boundaries and are also important in computer vision applications because they are robust features. We refer to corners as robust features because they can be found accurately in the presence of noise or even if image acquisition conditions, such as lighting or camera angles, vary. Note that even though the corner features themselves are robust, the corner detector may not be.

The **Moravec detector** is the simplest corner detector, but not necessarily robust. It finds points of maximum contrast, which correspond to potential corners and sharp edges. This operator is as follows:

$$\mathrm{MD}\left[I(r,c)\right] = \frac{1}{8}\sum_{i=r-1}^{r+1}\sum_{k=c-1}^{c+1}\left|I(r,c) - I(i,j)\right| \tag{4.4-1}$$

It finds the average of the sum of the absolute values of the differences between a pixel and its neighbors. Or, put more simply, finds the average difference between a pixel and its neighbors in all directions. After this operator is applied, the result can undergo a threshold operation to select only pixels above a certain value. Results of varying the threshold are shown in Figure 4.4-1. Here, we see that as the threshold is increased, we get fewer of the edge pixels and more of the corners only. We can also observe that on a digital, rectangular sampling grid, that curves have "corners".

One drawback to the Moravec corner detector is that it only considers edges oriented on the rectangular grid in 45° increments, using the eight-neighbors, so it may miss some valid corners that are not aligned on the grid. In this sense, the operator is not *isotropic*, which means it does not treat edges in all directions equally. The tradeoff is that it is simple and fast to calculate.

A more robust corner detector is the Harris corner detection algorithm, developed by Harris and Stephens in 1988. The Harris method consists of five steps: (1) Blur the image with a 2-D Gaussian convolution mask; (2) find the approximate brightness gradient in two perpendicular directions, for example, use the two Prewitt or Sobel masks; (3) blur the two brightness results with a 2-D Gaussian; (4) find the corner response function, *CRF*(*r, c*); (5) threshold the corner response function and apply nonmaxima suppression, similar to what was done with the Canny.

Now, we need to define the corner response function or *CRF*(*r, c*). To do this, we must select a gradient function for step 2; so, for simplicity, we will use the Prewitt edge detector which results in p_1 for the vertical edges (horizontal lines) and p_2 for the horizontal edges (vertical lines). Remember that p_1 and p_2 are both functions of the row and column coordinates, (*r, c*); this is implied in the below equation. Now we can define:

$$\mathrm{CRF}(r,c) = \left[G\left(p_1^2\right)G\left(p_2^2\right) - \left[G\left(p_1p_2\right)\right]^2\right] - \alpha\left[G\left(p_1\right) + G\left(p_2\right)\right]^2 \tag{4.4-2}$$

where $G(\circ)$ represents the result after convolution with a Gaussian.

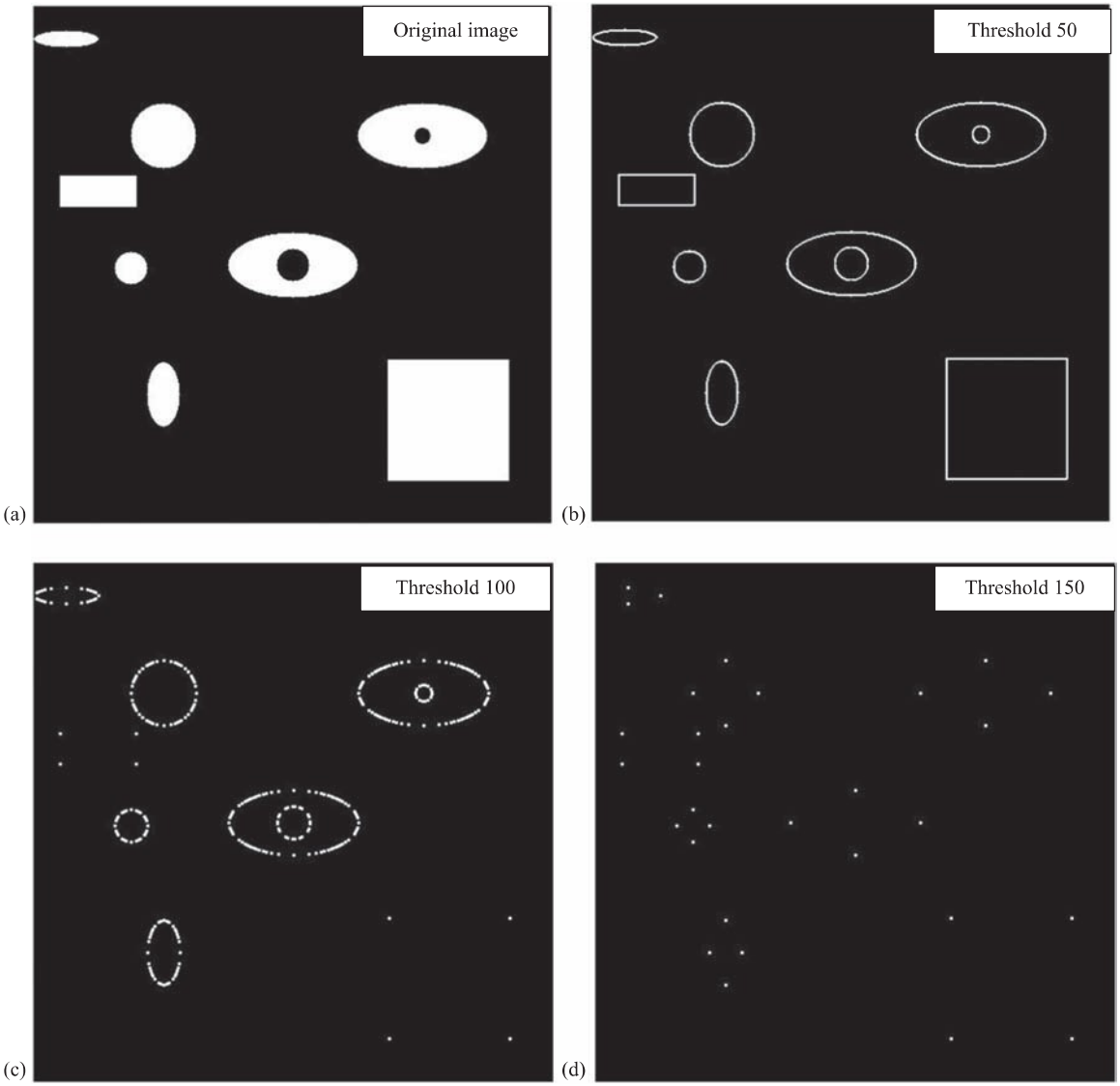

FIGURE 4.4-1
Moravec corner detector. (a) Original image, (b) resultant image from Moravec corner detector with a threshold of 50, (c) threshold of 100 and (d) threshold of 150. Note that the threshold of 150 gets the corners only, whereas lower threshold values get more edge pixels.

The larger the magnitude of CRF(r, c), the more likely a corner is at that point. The parameter α determines the sensitivity of the corner detector – a larger α makes it less sensitive and will result in fewer corners being found. The maximum value for α is 0.25, but typically values vary from 0.04 to 0.15, with 0.06 being the default value.

After the $CRF(r, c)$ is found, it must undergo a threshold process and application of nonmaxima suppression. The threshold is based on image content, and the nonmaxima suppression works by finding the largest value of the CRF within a given spatial area. Figure 4.4-2 shows application of the Harris corner detector with example intermediate images for each step in the process.

The Frei–Chen masks can also be used as a corner detector. Results are shown in Figure 4.4-3. Here, we have simple binary shapes and show the projection onto the edge subspace with a threshold angle of 50° and the projection onto the line subspace with threshold angles of 50° and 40°. Note that the projection onto the edge subspace finds the corners at a 45° angle to the corner and the projection onto the line subspace finds the corners on the corner itself. The edge subspace projection does not find the horizontal and vertical lines at the top and sides of the curves, but the projection onto the lines subspace does. Also, in (d), the line subspace with a lower angle threshold finds the "corners" of the holes only. These results are a function of the Frei–Chen masks (see Figure 4.2-18).

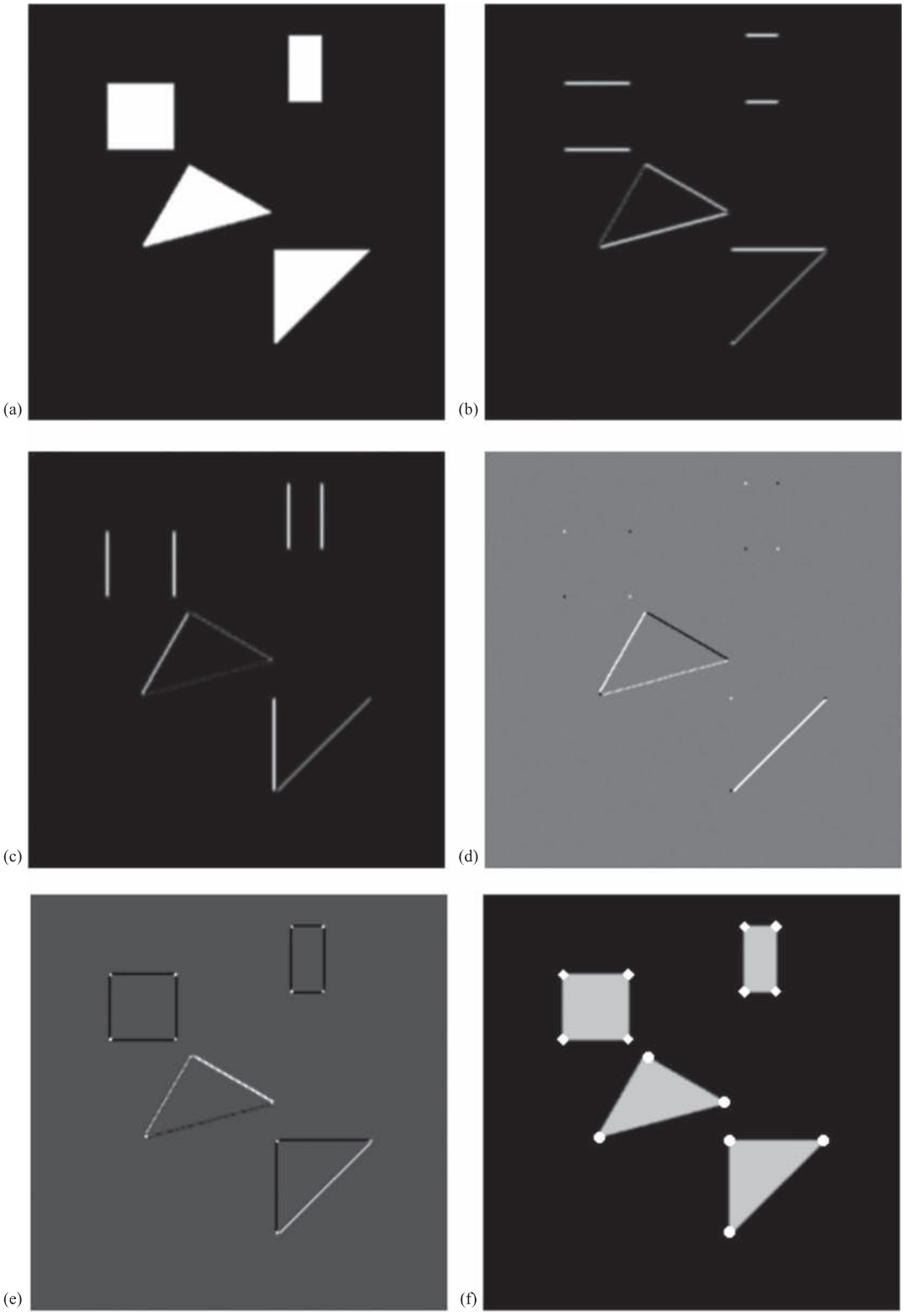

FIGURE 4.4-2
Harris corner detector. (a) Original image after application of 5×5 Gaussian mask, (b) strength of the horizontal lines, from the vertical gradient, by application of Prewitt, p_1, squared followed by a Gaussian, $G\left(p_1^2\right)$, (c) strength of the vertical lines, from the horizontal gradient, by application of Prewitt, p_2, squared followed by a Gaussian, $G\left(p_2^2\right)$, (d) Gaussian of the product of the horizontal and vertical gradient, $G\left(p_1 p_2\right)$, (e) result from the corner response function, $CRF(r, c)$, and (f) the final detected corners after thresholding and nonmaxima suppression of the $CRF(r, c)$, shown overlaid on the original shapes.

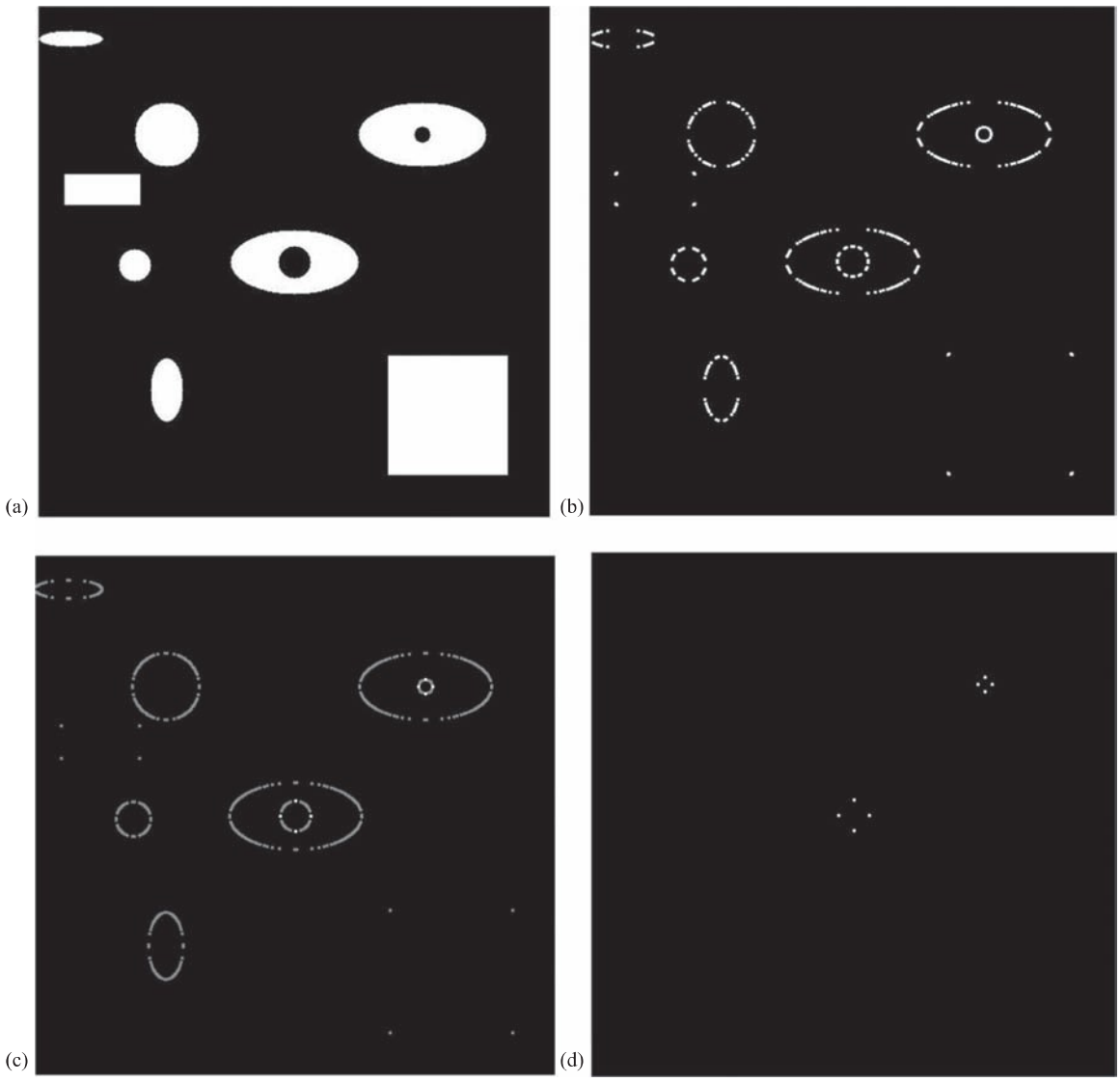

FIGURE 4.4-3
Frei–Chen masks for corner detector. (a) Original image, (b) projection onto edge subspace with a threshold angle of 50°, (c) projection onto line subspace with a threshold angle of 50° and (d) projection onto line subspace with a threshold angle of 40°. Note that the projection onto the edge subspace finds the corners at a 45° angle to the corner and the projection onto the line subspace finds the corners on the corner itself. The edge subspace projection does not find the horizontal and vertical lines at the top and sides of the curves, but the projection onto the lines subspace does. Also, in (d), the line subspace with a lower angle threshold finds the "corners" of the holes only.

4.4.2 Shape Detection with the Hough Transform

If we are searching for specific geometric shapes, we can extend the Hough transform to search for any geometric shape that can be described by mathematical equations, such as circles, ellipses or parabolas. The line finding Hough transform discussed previously was defined by quantizing the parameter space that defined the lines, specifically the mathematical space defined by the parameters ρ and θ. To extend this concept, we simply define a parameter vector and apply the Hough algorithm to this new parameter space. The *extended Hough transform* can be applied to any geometric shape that can described by an equation of the following form:

$$f\left(r,c;\overline{p}\right)=0 \tag{4.4-3}$$

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

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

$$\rho = r\cos(\theta) + c\sin(\theta) \tag{4.4-4}$$

and the parameter vector for the line is:

$$\bar{p} = \begin{bmatrix} \rho \\ \theta \end{bmatrix} \tag{4.4-5}$$

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:

$$(r-a)^2 + (c-b)^2 = \left(\frac{d}{2}\right)^2 \tag{4.4-6}$$

The parameter vector for the circle is:

$$\bar{p} = \begin{bmatrix} a \\ b \\ d \end{bmatrix} \tag{4.4-7}$$

To apply the Hough transform to find circles, we follow the same procedure as for line finding, but with an increased dimensionality to the search space – it is now a three-dimensional parameter space. This technique can be applied to any geometric shape that can be described by an equation. Another example is the ellipse, where a and b are the center, and the length of the major axis is $2h$ and the minor axis $2k$. For ellipses aligned along the row or column axis:

$$(r-a)^2/h^2 + (c-b)^2/k^2 = 1 \tag{4.4-8}$$

The parameter vector corresponding to the ellipse is:

$$\bar{p} = \begin{bmatrix} a \\ b \\ h \\ k \end{bmatrix} \tag{4.4-9}$$

Now for the ellipse, we apply the same method, but the search space is now a four-dimensional parameter space.

To search for general geometric shapes that are not readily described by parametric equations such as the circle, a *generalized Hough transform* can be used. The generalized Hough 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, such as the object center of area, is chosen inside the sample shape, and lines are defined from the reference point to a point on the border. This intersection information is recorded in the table. These lines can be defined by the angle they make with the vertical or horizontal axis or any other application-specific method that is useful. The shape is then described by the line intersection information in the *R*-table. The generalized Hough algorithm is then used to search for shapes described by this table. An illustration of how these lines and intersections are defined is shown in Figure 4.4-4.

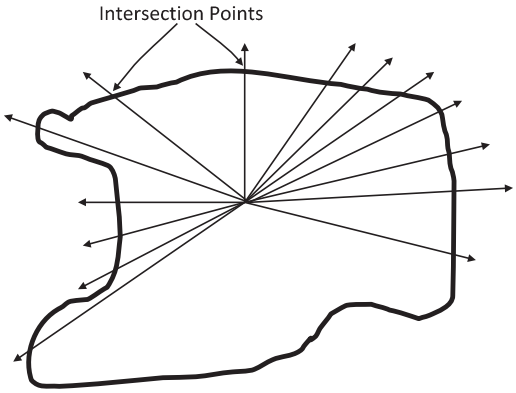

FIGURE 4.4-4
Generalized Hough transform. Arbitrary shape with lines from a given point is shown intersecting with the shape. Often the point selected is the object center of area and the lines can be defined at fixed angles depending on the application. The line parameters and their intersections are kept in an R-table, which is used to describe the shape.

4.5 Key Points

Overview: Edge, Line and Shape Detection

- Image analysis requires extracting useful information from vast amounts of low-level pixel data.
- Dividing the image into meaningful regions by detecting edges, lines, corners and geometric shapes is part of the process.
- The Hough transform is used for line finding.
- The Extended Hough is used to find regular geometric shapes such as circles and ellipses.
- The Generalized Hough is used to find arbitrary shapes.
- Corners represent points of interest, corresponding to places where the gray level is rapidly changing in multiple directions.

Edge Detection

- Edge detection operators are often implemented with convolution masks.
- Edge detection operators are often discrete approximations to differential operators.
- Edge detection operators may return magnitude and direction information, some return magnitude only.
- Edge direction and line direction are perpendicular to each other, because the edge direction is the direction of change in gray level (Figure 4.2-1).
- There is a tradeoff between sensitivity and accuracy in edge detection (Figure 4.2-2).
- Potential edge points are found by examining the relationship a pixel has with its neighbors; an edge implies a change in gray level.
- Edges may exist anywhere and be defined by color, texture, shadow, etc., and may not necessarily separate real-world objects (Figure 4.2-3).
- A real edge in an image tends to change slowly, compared to the ideal edge model which is abrupt (Figures 4.2-4 and 4.2-5).

Gradient Operators

- Gradient operators are based on the idea of using the first or second derivative of the gray level.
- The first derivative will mark edge points, with steeper gray level changes providing stronger edge points (larger magnitudes).
- The second derivative returns two impulses, one on either side of the edge.

Roberts operator: a simple approximation to the first derivative, two forms of the equations:

$$\sqrt{[I(r,c)-I(r-1,c-1)]^2+[I(r,c-1)-I(r-1,c)]^2} \tag{4.2-1}$$

$$|I(r,c)-I(r-1,c-1)|+|I(r,c-1)-I(r-1,c)| \tag{4.2-2}$$

Sobel operator: approximates the gradient with a row and column mask and returns both magnitude and direction:

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

$$\text{EDGE MAGNITUDE} \sqrt{s_1^2 + s_2^2} \tag{4.2-3}$$

$$\text{EDGE DIRECTION} \operatorname{Tan}^{-1}\left[\frac{s_1}{s_2}\right] \tag{4.2-4}$$

Prewitt operator: approximates the gradient with a row and column mask and returns both magnitude and direction; it is easier to calculate or implement in hardware than the Sobel, as it uses only 1's in the masks:

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

$$\text{EDGE MAGNITUDE} \sqrt{p_1^2 + p_2^2} \tag{4.2-5}$$

$$\text{EDGE DIRECTION} \operatorname{Tan}^{-1}\left[\frac{p_1}{p_2}\right] \tag{4.2-6}$$

Laplacian operators: these are two-dimensional discrete approximations to the second derivative, it is implemented by applying *one* of the following convolution masks:

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

Compass Masks

- The compass mask edge detectors are created by taking a single mask and rotating it to the eight major compass orientations.
- The edge magnitude is found by convolving each mask with the image and selecting the largest value at each pixel location.
- The edge direction at each point is defined by the direction of the edge mask that provides the maximum magnitude.

Kirsch Compass Masks

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

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

Robinson Compass Masks

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

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

Thresholds, Noise Mitigation and Edge Linking

- Noise includes any points not wanted or needed for the application.
- Noise mitigation requires a tradeoff between sensitivity and accuracy.
- Primary methods to mitigate noise: (1) control of a threshold, so that only the stronger edge points pass, (2) use of large mask sizes to mitigate noise and (3) use of a preprocessing mean filter.
- 10%–25% of average value good reference for threshold of unimodal histogram returned by edge detector.
- After edge points are selected, edge linking process combines points into lines and curves.
- Edge linking methods to link the edge points into segments and boundaries, including: (1) consider points that have passed the threshold test and connect them to other marked points within some maximum distance, (2) consider small neighborhoods and link points with similar magnitude and direction, then link points together to form boundaries and (3) the snake eating algorithm described in Section 4.3.2.

Advanced Edge Detectors

Marr–Hildreth algorithm: consists of three steps: (1) convolve the image with a Gaussian smoothing filter, (2) convolve the image with a Laplacian mask and (3) find the zero crossings of the image from step 2. The first two steps can be combined into one convolution filter, referred to as a Laplacian of a Gaussian, or LoG, and is given by this equation:

$$\text{LoG} = \left[\frac{r^2 + c^2 - 2\sigma^2}{\sigma^4}\right] e^{-\left(\frac{r^2+c^2}{2\sigma^2}\right)} \tag{4.2-7}$$

The LoG is implemented as an approximation with a convolution filter, such as:

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

The simplest method to find zero crossings is to examine 3×3 subimages and look for changing signs (positive/negative) in at least two opposing neighbors.

Canny algorithm: an optimal edge detector based on a specific mathematical model; it is a four-step process: (1) apply a Gaussian filter mask to smooth the image to mitigate noise effects, (2) find the magnitude and direction of the gradient, (3) apply nonmaxima suppression which results in thinned edges and (4) apply two thresholds known as hysteresis thresholding.

Hysteresis thresholding: mark pixels above a high threshold and then apply a low threshold to pixels connected to those marked by the high threshold.

Boie–Cox algorithm: a generalization of the Canny algorithm using matched filters and Wiener filters.

Shen–Castan algorithm: developed as an optimal solution to a specific mathematical model, similar to Canny, but with modifications and extensions.

Frei–Chen masks: they form a complete set of basis vectors, which means any 3×3 subimage can be represented as a weighted sum of the basis vectors. The weights are found by projecting the subimage onto each basis vector; that is, perform a vector inner product. Can be used to find edges or lines of specific orientation.

Vector inner product: found by multiplying coincident terms of two vectors and summing the results.

Edges in Color Images

Edge detection in color images is performed on the original RGB data, or after mapping into another color space, with these methods:

1. Extract the luminance or brightness information and apply a monochrome edge detection method. The brightness information can be found by averaging the RGB components: $I = \dfrac{(R+G+B)}{3}$, or by the luminance equation: $Y = 0.299R + 0.587G + 0.114B$, or by the vector length: $L = \sqrt{R^2 + G^2 + B^2}$.
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 use a linear combination of the three results at each location.
5. Apply a monochrome edge detection method to each of the RGB bands separately and then select specific criteria at each pixel location to find an edge point.
6. Equations for multispectral edges, developed by Cervenka and Charvat:

$$\frac{\sum_{b=1}^{n}\left[I_b(r,c)-\bar{I}(r,c)\right]\left[I_b(r+1,c+1)-\bar{I}(r+1,c+1)\right]}{\sqrt{\sum_{b=1}^{n}\left[I_b(r,c)-\bar{I}(r,c)\right]^2\sum_{b=1}^{n}\left[I_b(r+1,c+1)-\bar{I}(r+1,c+1)\right]^2}} \tag{4.2-14}$$

$$\frac{\sum_{b=1}^{n}\left[I_b(r+1,c)-\bar{I}(r+1,c)\right]\left[I_b(r,c+1)-\bar{I}(r,c+1)\right]}{\sqrt{\sum_{b=1}^{n}\left[I_b(r+1,c)-\bar{I}(r+1,c)\right]^2\sum_{b=1}^{n}\left[I_b(r,c+1)-\bar{I}(r,c+1)\right]^2}} \tag{4.2-15}$$

where

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

$f_b(r,c)$ 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.
- Success criteria must be defined, such as was used to develop the Canny algorithm: (1) Detection – find all real edges and not find any false edges. (2) Localization – the edges are found in the correct place. (3) Single response – no multiple edges found for a single edge.

Pratt's Figure of Merit (FOM): an objective measure developed by Pratt in 1978, which ranges from 0 (0%) for a missing edge to 1 (100%) for a perfectly found edge. It is defined as follows:

$$\text{FOM} = \frac{1}{I_N} \sum_{i=1}^{I_F} \frac{1}{1+\alpha d^2} \tag{4.2-16}$$

where I_N is the maximum of I_I and I_F; I_I is the number of ideal edge points in the image; I_F 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 d is the distance of a found edge point to an ideal edge point.

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

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

$$d = |r_1 - r_2| + |c_1 - c_2| \tag{4.2-17}$$

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

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

$$d = \max\left(|r_1 - r_2|, |c_1 - c_2|\right) \tag{4.2-18}$$

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

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

$$d = \left[(r_1 - r_2)^2 + (c_1 - c_2)^2\right]^{1/2} \tag{4.2-19}$$

Line Detection

Hough Transform

- The Hough transform is an efficient method for line finding, input is a set of marked edge points.
  Consists of three primary steps based on using the normal representation of a line, $\rho = r\cos(\theta) + c\sin(\theta)$:
  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 a threshold value, plug the values for r and c into the line equation:

$$\rho = r\cos(\theta) + c\sin(\theta) \tag{4.3-1}$$

     Then, for each value of θ in the quantized space, solve for ρ.
  3. For each $\rho \cdot \theta$ 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.

After performing the Hough transform, postprocessing must be done to extract the line information.

Postprocessing

1. Perform the Hough transform on the input image containing marked edge points.
2. Threshold by using the parameter *line pixels*, the minimum number of pixels in a line, do the inverse Hough transform.
3. Perform a logical AND operation on the original image and output image from (2).
4. Apply an *edge linking* process to connect line segments; specifically, a *snake eating algorithm* (Figure 4.3-5):
   a. A line segment is a snake. It can eat another snake within *connect distance* along *line angles*.
   b. If a snake is less than *segment length*, it is removed.

Corner and Shape Detection

Corner Detection

- *Corners* are points with a high rate of change in more than one direction, which are also known as *points of interest*.
- Corners are robust features.
- Corners are useful features in object tracking.
- Corners are useful in matching multiple images, for example, creating 3-D models from stereo images.
- Corners are used by the human visual system to provide cues about object boundaries.

Moravec corner detector: simplest, not necessarily very robust, finds points of maximum contrast (see Figure 4.4-1):

$$\mathrm{MD}\left[I(r,c)\right]=\frac{1}{8}\sum_{i=r-1}^{r+1}\sum_{k=c-1}^{c+1}\left|I(r,c)-I(i,j)\right| \tag{4.4-1}$$

Harris corner detection algorithm: (1) Blur the image with a 2-D Gaussian convolution mask, (2) find the approximate brightness gradient in two perpendicular directions, for example, use the two Prewitt or Sobel masks, (3) blur the two brightness results with a 2-D Gaussian and (4) find the corner response function:

$$\mathrm{CRF}(r,c)=\left[G\left(p_1^2\right)G\left(p_2^2\right)-\left[G\left(p_1p_2\right)\right]^2\right]-\propto\left[G\left(p_1\right)+G\left(p_2\right)\right]^2 \tag{4.4-2}$$

(5) Threshold the corner response function and apply nonmaxima suppression, similar to what was done with the Canny (see Figure 4.4-2).

Frei–Chen masks: can be used for corner detection (see Figure 4.4-3).

Shape Detection with the Hough Transform

- **Extended Hough transform** is used to find shapes and mark boundaries that can be defined by analytical equations, such as circles or ellipses. The search space is a parameter space, where the parameters are found in the equation describing the shape of interest.
- **Generalized Hough transform** is used to find any arbitrary shape (Figure 4.4-4). It works by creating a description of the shape defined by a reference point and a table of line parameters and their intersections, which is called the R-table.

4.6 References and Further Reading

Companion reading regarding edge detection includes Nixon and Aguado (2020), Davies (2018), Gonzalez and Woods (2018), Sonka, Hlavac, and Boyle (2014), Pratt (2007), Forsyth and Ponce (2011) and Shapiro and Stockman (2001). More on thresholding algorithms and edge linking can be found in Gonzalez and Woods (2018), Davies (2018), Sonka, Hlavac, and Boyle (2014), Baxes (1994) and Dougherty and Lotufo (2009).

Details on the Marr–Hildreth algorithm can be found in Marr and Hildreth (1980), and for more information on the LoG and its relationship to biological vision systems, see Marr (1982) and Shapiro and Stockman (2001). More details on implementation of the LoG and the Marr–Hildreth algorithm can be found in Parker (1997) and Haralick and Shapiro (1992).

For more details on the Canny algorithm, see Sonka, Hlavac, and Boyle (2014), Jain, Kasturi, and Schunck (1995) and Canny (1986); on the Shen–Castan algorithms, see Shen and Castan (1992) and Parker (1997); and for more on the Boie–Cox algorithm, see Boie and Cox (1987) and Seul, O'Gorman, and Sammon (2000). Shapiro and Stockman (2001) have a different, and potentially useful, approach based on energy for using the Frei–Chen masks (Frei and Chen, 1977).

The multispectral edge detection equations were found in Sonka, Hlavac, and Boyle (2014) from the original paper Cervenka and Charvat (1987). The Hough transform as described here can also be found in Gonzalez and Woods (2018), Davies (2018) and the original patent in Hough (1962). Additional line finding methods can be found in Nixon and Aguado (2020) and Davies (2018), including the RANSAC (RANdom Sampling Consensus) statistical method. Additional resources for the extended Hough transform for finding circles and ellipses include Nixon and Aguado (2020), Davies (2018) and Haidekker (2011). More details on the generalized Hough transform and corner detection can be found in Nixon and Aguado (2020) and Sonka, Hlavac, and Boyle (2014). For algorithm details and code to implement the Harris corner detection algorithm, see Burger and Burge (2008) and Harris and Stephens (1988). Companion sources for the Moravec and Harris corner detectors include Davies (2018) and Sonka, Hlavac, and Boyle (2014).

Baxes, G.A., *Digital Image Processing: Principles and Applications*, New York: Wiley, 1994.

Boie, R.A. and Cox, I., Two dimensional optimum edge recognition using matched and wiener filters for machine vision. *Proceedings of the IEEE First International Conference on Computer Vision*, New York, pp. 450–456, 1987.

Burger, W., Burge, M.J., *Digital Image Processing: An Algorithmic Introduction Using Java*, New York: Springer, 2008.

Canny, J., A computational approach for edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 8, No. 6, pp. 679–698, 1986.

Cervenka, V., and Charvat, K, Survey of Image Processing Research Applicable to the Thematic Mapping Based on Aerocosmic Data (in Czech). Technical Report A 12-346-811, Prague, Czechoslovakia: Geodetic and Cartographic Institute, 1987.

Davies, E.R, *Computer Vision: Principles, Algorithms, Applications and Learning*, 5th Edition, Cambridge, MA: Academic Press, 2018.

Dougherty, G., *Digital Image Processing for Medical Applications*, Cambridge: Cambridge University Press, 2009.

Forsyth, D.A., Ponce, J., *Computer Vision*, Upper Saddle River, NJ: Pearson, 2011.

Frei, W., Chen, C.C., Fast boundary detection: A generalization and a new algorithm. *IEEE Transactions on Computers*, Vol. C-26, No. 10, pp. 988–998, 1977.

Gonzalez, R.C., Woods, R.E., *Digital Image Processing*, 4th Edition, London: Pearson, 2018.

Haidekker, M.A., *Advanced Biomedical Image Analysis*, Hoboken, NJ: Wiley, 2011.

Haralick, R.M., Shapiro, L.G., *Computer and Robot Vision*, Reading, MA: Addison-Wesley, 1992.

Harris, C.G, Stephens, M., A combined corner and edge detector, in Taylor, C.J, editor, *Proceedings of the 4th ALVEY Vision Conference*, pp. 147–151, Manchester: University of Manchester, 1988.

Hough, P.V.C, *Methods and Means for Recognizing Complex Patterns*, Washington, DC: U.S. Patent 3,069,654, 1962.

Jain, R., Kasturi, R., Schnuck, B.G., *Machine Vision*, New York: McGraw Hill, 1995.

Marr, D., *Vision*, New York: Freeman and Company, 1982.

Marr, D., Hildreth, E., Theory of edge detection, *Proceedings of the Royal Society, B* Vol. 207, pp. 187–217, 1980.

Nixon, M.S., Aguado, A.S., *Feature Extraction and Image Processing for Computer Vision*, 4th Edition, Cambridge, MA: Academic Press, 2020.

Parker, J.R., *Algorithms for Image Processing and Computer Vision*, New York: Wiley, 1997.

Pratt, W.K., *Digital Image Processing*, New York: Wiley, 2007.

Seul, M., O'Gorman, L., Sammon, M.J., *Practical Algorithms for Image Analysis*, Cambridge: Cambridge University Press, 2000.
Shapiro, L., Stockman, G., *Computer Vision*, Upper Saddle River, NJ: Prentice Hall, 2001.
Shen, J. and Castan, S., An optimal linear operator for step edge detection, *Computer Vision, Graphics, and Image Processing: Graphical Models and Understanding*, Vol. 54, No. 2, pp. 112–133, 1992.
Sonka, M., Hlavac, V., Boyle, R., *Image Processing, Analysis and Machine Vision*, 4th Edition, Boston, MA: Cengage Learning, 2014.

4.7 Exercises

Problems

1. Many edge detectors operate in a manner similar to convolution. What exactly does this mean?
2. What does a differential operator measure and how does this relate to edge detectors?
3. In dealing with noise in edge detection, there is a tradeoff between sensitivity and accuracy. Explain what this means.
4. Compare and contrast an ideal edge and a real edge in an image. Draw a picture of both.
5. (a) Explain the idea on which gradient edge detection operators are based. (b) How do the results differ if we use a second order compared to a first order derivative operator? (c) Explain what is meant by sub-pixel accuracy and how it relates to gradient-based edge detectors.
6. Find the results of applying Robert's edge detector to the following image. Use the absolute value form of the operator. For the result, don't worry about top row and left column.

$$\begin{bmatrix} 5 & 7 & 4 & 3 \\ 4 & 0 & 0 & 0 \\ 6 & 1 & 2 & 1 \end{bmatrix}$$

7. Find the results, magnitude and direction, of applying the Prewitt edge detector to the following image. For the result, ignore the outer rows and columns.

$$\begin{bmatrix} 0 & 8 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 8 & 0 & 0 \end{bmatrix}$$

8. Two of the three Laplacian masks given are based on eight-connectivity, and the other one is based on four-connectivity – which one? Devise a Laplacian-type edge detection mask based on six-connectivity.
9. Find the results of applying the Robinson compass masks to the following image. For the result, ignore the outer rows and columns. Keep track of the maximum magnitude and which mask corresponds to it.

$$\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 5 & 5 & 5 & 5 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

10. Use CVIPtools to explore the basic edge detection operators. (a) Run CVIPtools and load test images of your choice. As one of the test images, create a binary image of a circle with *Utilities→Create→Circle.* (b) Select *Utilities→Create→Add noise.* Add noise to your test images. Use both Gaussian and salt-and-pepper noise. (c) Select *Analysis→Edge/Line Detection.* Compare thresholding the output at different levels with the

Kirsch, Pyramid and Robinson edge detection operators. In CVIPtools, select the desired edge detector, select none for pre-filtering and select the desired threshold with post-threshold option. Alternately, select none for post-threshold and use the green *Threshold at* button in the top section – this allows for the testing of different threshold levels without the need to rerun the edge detection operation. (d) For the images containing noise, compare the resultant images with and without applying a lowpass filter as a preprocessing step (use the pre-filter option in CVIPtools).

11. Use CVIPtools to explore the basic edge detection operators. (a) Run CVIPtools and load test images of your choice. As one of the test images, create a binary image of a circle with *Utilities→Create→Circle.* (b) Select *Utilities→Create→Add noise*. Add noise to your test images. Use both Gaussian and salt-and-pepper noise. (c) Select *Analysis→Edge/Line Detection*. (d) Compare using different size kernels with the Sobel and Prewitt operators. In CVIPtools, select the desired edge detection operator and select the kernel size. (e) For the images containing noise, compare the resultant images with and without applying a lowpass filter as a preprocessing step (use the pre-filter option in CVIPtools). Also, compare the results with different size kernels (masks). Does the 3×3 or 7×7 provide better results in the presence of noise?
12. Use CVIPtools to explore the basic edge detection operators. (a) Run CVIPtools and load test images of your choice. As one of the test images, create a binary image of a circle with *Utilities→Create→Circle.* (b) Select *Utilities→Create→Add noise*. Add noise to your test images. Use both Gaussian and salt-and-pepper noise. (c) Select *Analysis→Edge/Line Detection*. (d) Use the Roberts and Laplacian and compare with and without using the Add to Original checkbox. (e) For the images containing noise, compare the resultant images with and without applying a lowpass filter as a preprocessing step (use the pre-filter option in CVIPtools).
13. The Canny algorithm for edge detection was developed based on a specific edge model, what is it? What are the three criteria used to develop the algorithm? How do the four algorithmic steps relate to these three criteria?
14. Use CVIPtools to explore the Laplacian and Frei–Chen edge detection operators. Use real images and create some simple geometric images with *Utilities→Create*. Additionally, add noise to the images. (a) In *Analysis→Edge/Line Detection*, select the Frei–Chen, compare the line subspace versus the edge subspace, with the projection option in CVIPtools. Experiment with various threshold angles, as well as post-threshold values. Examine the histogram to select good threshold values. Compare the threshold values that work the best to the images with and without noise, are they the same? Why or why not? (b) Using the Laplacian in *Analysis→Edge/Line Detection*, select pre-filtering with a Gaussian – this will perform a LoG filter. Examine the histogram to select good threshold values. (c) Use *Utilities→Filter→Specify a Filter* to input the values for the 5×5 LoG filter given in the text. Do this by selecting the 5×5 and then entering the values in the box (note: the <tab> key, or the mouse, can be used to move around the box), then select OK. Compare these results to the results from (b). Are they similar? Why or why not?
15. Use CVIPtools to explore the Canny, Boie–Cox and Shen–Castan algorithms. Use *Utilities→Create* to create circle with a radius of 32 and a blur radius of 128. (a) Select the Canny algorithm. Use the Low Threshold Factor = 1, High Threshold Factor = 0.5 and compare results with a variance of 0.5, 1.5 and 3. Next, use Low Threshold Factor = 1, High Threshold Factor = 1 and compare results with a variance of 0.5, 1.5 and 3. Now apply the Canny to a complex image of your choice. Experiment with the parameters until you achieve a good result. (b) Select the Boie–Cox algorithm and apply it to your blurred circle image. Experiment with the various parameters. Next, apply it to the complex image you used with the Canny. Which settings work best for your images? (c) Select the Shen–Castan algorithm and apply it to your blurred circle image. Use the Low Threshold Factor = 1, High Threshold Factor = 1, Smoothing Factor = 0.9, Window Size = 5 and compare results with a Thin Factor of 1, 3 and 6. Next, apply it to the complex image you used with the Canny. Which settings work best for your images? (d) Repeat (a)–(c), but add noise to the images.
16. Select a color image of your choice. Use *Utilities→Convert→Color Space* to explore edge detection using various color spaces, and *Utilities→Create→Extract Band* to operate on individual bands and use *Utilities→Create→Assemble Bands* to combine resulting bands into composite images. Devise an algorithm that works best for your particular image.

17. In the following image, find the distance between the points labeled a and b in the following image:

$$\begin{bmatrix} 5 & 7 & 8 & b & 12 \\ 3 & 6 & 7 & 6 & 6 \\ 6 & a & 10 & 10 & 11 \\ 7 & 7 & 0 & 0 & 0 \\ 0 & 7 & 9 & 1 & 0 \end{bmatrix}$$

(a) Use city block distance, (b) use chessboard distance and (c) use Euclidean distance

18. Given the following image, apply a Robert's edge detector, absolute value format, to the image (do not worry about the top row and leftmost column). Next, threshold the image with the following values and find Pratt's FOM for the found edge points. Let $\alpha = 0.5$ and use the chessboard distance measure. Threshold values: (a) 5, (b) 12 and (c) 22.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 10 & 10 & 10 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

19. List all the steps in the process of using the Hough transform for line finding.
20. Use CVIPtools to explore the Hough transform and the postprocessing edge linking algorithm on an artificial image. (a) Use *Utilities→Create* to create a circle and perform an edge detector of your choice, (b) threshold the resultant image, (c) perform the Hough transform on the thresholded image, input 5 for both *line pixels* and *segment length*, then select line angles 0, (d) 90, (e) 0–20, (f) 0–45, (g) 0–90, (h) 0–150 and (i) 0–180. Do the results make sense? How are the results affected by increasing the connect distance? How are the results affected by increasing the segment length?
21. Use CVIPtools to explore the Hough transform and the postprocessing edge linking algorithm on a real image. (a) select an image of your choice and perform an edge detection, (b) threshold the resultant image, (c) perform the Hough transform on the thresholded image, input 5 for both *line pixels* and *segment length*, then select line angles 0, (d) 90, (e) 0–20, (f) 0–45, (g) 0–90, (h) 0–150 and (i) 0–180. Do the results make sense? How are the results affected by increasing the *connect distance*? Find a *connect distance* that gives you the desired results. How are the results affected by increasing the *segment length*? Find a *segment length* that gives you the desired results.

4.7.1 Programming Exercises

These exercises can be implemented with CVIPlab in MATLAB or in C/C++.

Edge Detection – Roberts and Sobel

1. Write a function to implement the Roberts edge detector. Let the user select either the square root or the absolute value form. Be careful of data type to avoid overflow problems. Compare the results from the two methods by using the *Utilities→Compare* selection in CVIPtools.
   Note: If doing this exercise in C, the following will be useful:
   - The C functions for absolute value and square root are abs() and sqrt().
   - Note that you will need to deal with potential overflow problems, as the results may be greater than 255. This may be dealt with by using a floating point image structure as an intermediate image and then remapping the image when completed. This is done as follows:

```
Image *outputimage; /*declaration of image structure pointer*/
float **image_data; /* declaration of image data pointer*/
.....
outputimage = new_Image(PGM, GRAY_SCALE, no_of_bands, no_of_rows, no_of_cols, CVIP_FLOAT,
REAL); /*creating a new image structure*/
image_data = getData_Image(outputimage, bands); /*getting the data into an array that can be
accessed as: image_data[r][c] */
.....
outputimage = remap_Image(outputimage, CVIP_BYTE, 0, 255); /*remapping a float image to byte
size, this is done before writing the image to disk with the write_Image function*/
```

2. Write a function to implement the Sobel edge detector. The function should output an image that contains the Sobel magnitude, and it is remapped as with the Roberts. Test the function on gray level images of your choice.
3. Modify the functions to handle multiband (color) images.
4. Use the *Analysis→Edge/Line detection→Edge link* selection in CVIPtools to connect the lines in the output images. Note that this requires a binary image, so be sure to apply a threshold operation to the images first. Thresholding can be performed directly in this window by typing the threshold value in the entry box and clicking on the green *Threshold at* button next to the entered value.

Edge Detection – Robinson and Kirsch Compass Masks

1. Write a function to implement the Robinson compass masks edge detector.
2. Have the function return two images: one for the magnitude and one for the direction.
3. Modify the function to handle color images.
4. Modify the function to handle non-byte images.
5. Modify the function to allow the user to choose Robinson or Kirsch masks.

Otsu and K-means Thresholding

1. Write a function to implement the Otsu thresholding method described in Chapter 3.
2. Apply the Otsu function to magnitude images that have undergone Roberts, Sobel, Prewitt and Laplacian edge detection. Compare and contrast the results, especially comparing the first derivative based to the second derivative based models. Compare results to CVIPtools.
3. Write a function to implement the K-means thresholding method described in Chapter 3.
4. Apply the K-means function to magnitude images that have undergone Roberts, Sobel, Prewitt and Laplacian edge detection. Compare and contrast the results, especially comparing the first derivative based to the second derivative based models. Compare results to CVIPtools.

Edge Linking

1. Write a function to implement the simple edge linking process described in Section 4.2.3. Consider each point that has passed the threshold test and connect it to all other such points that are within a maximum distance. Let the user specify a maximum connect distance.
2. Test this function on various edge detected images. Do you think it works very well?
3. Implement a function for edge linking before thresholding using small neighborhoods as described in Section 4.2.3. Let the user specify maximum range for magnitude and direction to consider points as similar.
4. Test this function on various edge detected images. Do you think it works better than the first function?

4.8 Supplementary Exercises

Supplementary Problems

1. (a) Apply the Prewitt to the following image, keep both magnitude and direction. Do not calculate Prewitt for outer rows and columns. (b) Which points are connected if the magnitude can vary by 0.5 and the angle must be an exact match (assume eight-connectivity)?

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

2.

a. Apply nonmaxima suppression to the pixels in the center column:

$$\begin{bmatrix} \leftarrow 70 & 116 \rightarrow & 220 \rightarrow \\ \leftarrow 20 & 90 \rightarrow & 51 \rightarrow \\ \leftarrow 88 & 95 \rightarrow & 127 \rightarrow \end{bmatrix}$$

b. Apply hysteresis thresholding to the following image with a high threshold of 200 and a low threshold of 150. Assume four-connectivity:

$$\begin{bmatrix} 22 & 34 & 32 & 44 & 45 & 45 & 43 & 43 \\ 27 & 11 & 12 & 11 & 12 & 11 & 12 & 18 \\ 22 & 221 & 223 & 226 & 222 & 199 & 10 & 9 \\ 2 & 3 & 3 & 178 & 4 & 4 & 5 & 4 \\ 7 & 7 & 7 & 216 & 7 & 7 & 7 & 7 \\ 18 & 23 & 23 & 166 & 23 & 66 & 65 & 65 \\ 188 & 76 & 75 & 198 & 199 & 187 & 184 & 123 \\ 222 & 177 & 188 & 222 & 222 & 14 & 14 & 13 \end{bmatrix}$$

3. (a) Project the following subimage onto the Frei–Chen masks. (b) Find the projection angles onto the line subspace, the edge subspace and the average subspace. (c) Use the Frei–Chen weights to get the subimage back. Did you get it back exactly the same? Why or why not?

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

4. Given the following image, apply the Otsu method to find the best threshold:

$$\begin{bmatrix} 23 & 34 & 34 & 45 & 45 & 45 & 45 & 45 \\ 23 & 12 & 77 & 12 & 12 & 77 & 12 & 12 \\ 23 & 222 & 222 & 222 & 222 & 199 & 77 & 77 \\ 3 & 3 & 3 & 199 & 3 & 3 & 3 & 3 \\ 77 & 77 & 77 & 222 & 77 & 77 & 77 & 77 \\ 23 & 23 & 23 & 199 & 23 & 65 & 65 & 65 \\ 199 & 77 & 77 & 199 & 199 & 199 & 199 & 199 \\ 222 & 222 & 222 & 222 & 222 & 12 & 12 & 12 \end{bmatrix}$$

5. (a) Explain how the extended Hough transform could be used to find ellipses that are aligned along the row or column axis. (b) How many dimensions is the search space? What are they? (c) How can this be extended to find ellipses of any orientation?
6. (a) If the Laplacian masks as defined in this chapter are applied to an image, such as in CVIPtools, the sign (positive or negative) on the results seems to be the opposite of what is shown in Figure 4.2-5. Why is that? (b) Is the Type 1 Laplacian mask actually isotropic in all directions?
7. Explain how the Hough transform can be used to find gaps between line segments on a given line. Allow for different connectivity and distance definitions.

4.8.1 Supplementary Programming Exercises

These exercises can be implemented with CVIPlab in MATLAB or in C/C++.

Canny Edge Detection

1. Write a function to implement the Canny edge detection algorithm. Use a fixed 3×3 Gaussian approximation mask for step one and the gradient edge detector given in the chapter for step 2.
2. Modify your function to allow for any size Gaussian up to a 9×9 in step one and a 3×3 Prewitt or Robinson for step two.
3. Compare your results to CVIPtools. Are they the same? Why or why not?

Boie–Cox Edge Detection

1. Use the references to research the Boie–Cox algorithm.
2. Write a function to implement the algorithm.
3. Compare your results to CVIPtools. Are they the same? Why or why not?

Shen–Castan Edge Detection

1. Use the references to research the Shen–Castan algorithm.
2. Write a function to implement the algorithm.
3. Compare your results to CVIPtools. Are they the same? Why or why not?

Hough Transform

1. Write a function to implement a Hough transform. Let the user enter the line angles of interest and the minimum number of pixels per line.
2. Compare your results to those obtained in CVIPtools.
3. Extend your Hough transform function to find circles.
4. Extend your Hough transform function to find ellipses.

Moravec Corner Detection

1. Write a function to implement the Moravec corner detector.
2. Compare your results to CVIPtools. Are they the same? Why or why not?

Harris Corner Detection

1. Write a function to implement the Harris corner detection algorithm. Write a separate function for each step and output the intermediate images.
2. Compare your results to CVIPtools. Are they the same? Why or why not?

Generalized Hough

1. Write a function to implement the generalized Hough transform.
2. Note that it is helpful to define a standard size and orientation. Use a size of 32×32, and orientation so that the axis of least second moment is horizontal, and the center of area is used as the reference point.
3. Experiment with random shapes created in CVIPtools with *Utilities→Create→Black Image* and *Utilities→Create→Border Mask*.

5

Segmentation

5.1 Introduction and Overview

Image segmentation is one of the most important processes for many computer vision and image analysis applications. Segmenting the image correctly into real objects of interest is critical for object classification which will determine the success or failure of the algorithm. For example, in a robotic control application that needs to remove "bad" parts from an assembly line, incorrect classification will result in bad parts being shipped to the customer or good parts being removed. In either case, the cost to the manufacturer is increased. In a medical diagnostic application, a poor classification process can be even more costly in terms of both dollars and human life.

In many applications, the illumination system is critical to acquiring an image that has a reasonable chance for a successful segmentation. The lighting system in industrial inspection applications can be controlled and designed to make the segmentation process easier. Special purpose lighting is often used in these types of applications where the designer can control the environment. Other applications may not allow for environmental control, such as satellite, surveillance or automated driving applications, but the imaging system designer can specify the types of sensors to be used. For example, infrared imaging sensors are a natural choice for image acquisition in applications where the objects of interest radiate heat, such as people, engines or factories. Satellites typically acquire data across a wide range of the electromagnetic spectrum, so they require sensors that will respond to all these frequencies and thus allow for the collection of image data that can be used for many different types of applications. These examples all serve to illustrate the importance of the application feedback loop in the image analysis process (see Figure 3.1-3).

5.1.1 Segmentation System Model and Preprocessing

The segmentation system model discussed here is a three-stage process. As shown in Figure 5.1-1, the three stages are (1) preprocessing, (2) segmentation and (3) postprocessing. During the *Preprocessing* stage, the image is simplified to facilitate the segmentation process itself. The simplification is usually done by filtering or grouping small areas or brightness values of pixels together. Typical operations that may be performed during this stage include gray level quantization, spatial quantization, median filtering, Kuwahara filtering, anisotropic diffusion filters and superpixel extraction. The second and primary stage of the process is the *Segmentation* method itself, which will be discussed in the following sections and stage three, *Postprocessing*, will be further explored in the section on morphological filters.

Figure 5.1-2 shows output images from preprocessing operations, and we can see that this simplification of the image itself can aid in the segmentation process. In some cases, it may be difficult to distinguish the change visually, so we have included the labeled number of connected objects with each image. The superpixel algorithm, referred to as the *simple linear iterative clustering* (SLIC) method, is essentially the k-means clustering algorithm but using a 5-dimensional vector containing the red, green, blue components and the row and column coordinates. For some applications, the SLIC algorithm can be used as a segmentation method itself. With k-means clustering, we can specify the value for k, which in this case is the number of superpixels in the output image. In Figure 5.1-3,

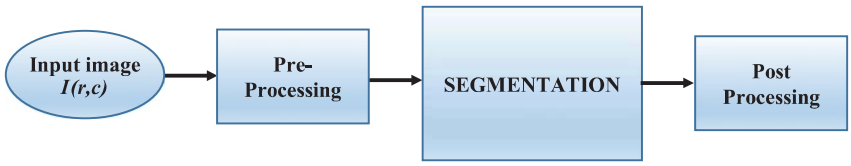

FIGURE 5.1-1
Image segmentation process. Image segmentation can be modeled as a three-stage process: (1) preprocessing to simply the image, (2) the segmentation method itself and (3) postprocessing, usually in the form of morphology filtering, to further refine the segmentation.

DOI: 10.1201/9781003221135-5