Chapter 5

Edge Detection

The early stages of vision processing identify features in images that are relevant to estimating the structure and properties of objects in a scene. Edges are one such feature. Edges are significant local changes in the image and are important features for analyzing images. Edges typically occur on the boundary between two different regions in an image. Edge detection is frequently the first step in recovering information from images. Due to its importance, edge detection continues to be an active research area. This chapter covers only the detection and localization of edges. Basic concepts in edge detection will be discussed. Several common edge detectors will be used to illustrate the basic issues in edge detection. Algorithms for combining edges into contours are discussed in Chapter 6.

An edge in an image is a significant local change in the image intensity, usually associated with a discontinuity in either the image intensity or the first derivative of the image intensity. Discontinuities in the image intensity can be either (1) *step* discontinuities, where the image intensity abruptly changes from one value on one side of the discontinuity to a different value on the opposite side, or (2) *line* discontinuities, where the image intensity abruptly changes value but then returns to the starting value within some short distance. However, step and line edges are rare in real images. Because of low-frequency components or the smoothing introduced by most sensing devices, sharp discontinuities rarely exist in real signals. Step edges become *ramp* edges and line edges become *roof* edges, where intensity changes are not instantaneous but occur over a finite distance. Illustrations of these edge profiles are shown in Figure 5.1.

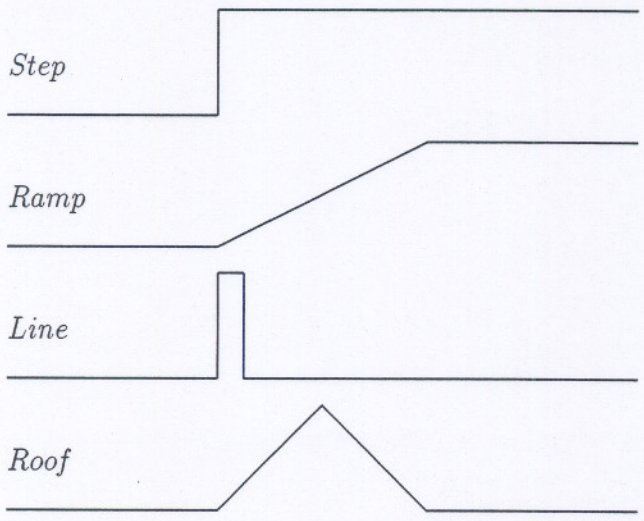


Figure 5.1: One-dimensional edge profiles.

It is also possible for an edge to have both step and line characteristics. For example, a surface that changes orientation from one flat surface to another will produce a step edge; but if the surface has a specular component of reflectance and if the surface corner is rounded, there can be a highlight due to the specular component as the surface orientation of the rounded corner passes the precise angle for specular reflection. The edge profile generated by such a situation looks like a step edge with a superimposed line edge. There are also edges associated with changes in the first derivative of the image intensity. For example, mutual reflection from the sides of a concave corner generate roof edges. Edges are important image features since they may correspond to significant features of objects in the scene. For example, the boundary of an object usually produces step edges because the image intensity of the object is different from the image intensity of the background.

This chapter will deal almost exclusively with step edges, although many of the ideas can be adapted to other types of image intensity changes. The profile of an ideal step edge and the profile of a real image that provides some examples of step edges are displayed in Figure 5.2. By definition, an edge is a significant local change in the image intensity. The plot shows step

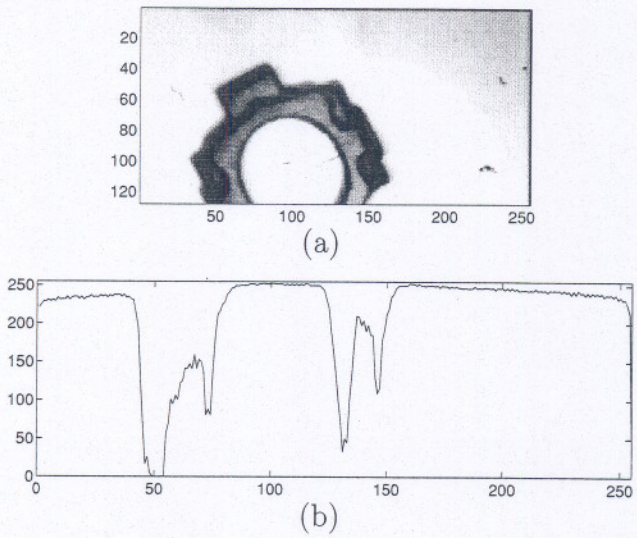


Figure 5.2: (a) The top half of the connecting rod image. (b) The profile of an ideal step change in image intensity is plotted to show that the edges are not perfectly sharp and the image is corrupted by noise. The plot is a horizontal slice through the circular portion of the connecting rod corresponding to the bottom edge of the partial rod image shown above.

changes in image intensity that are edges because the changes are significant and local. The plot also shows changes that are not edges, because they violate part of the definition. The changes due to noise are not edges even though the changes are local, because the changes are not significant. The changes due to shading, such as the ramp on the right side of the plot, are not edges even though the changes are significant, because the changes are not local. Real images are very noisy. It is difficult to develop an edge detection operator that reliably finds step edges and is immune to noise.

Before we discuss important considerations in edge detection operators, some terms must be carefully defined.

Definition 5.1 *An edge point is a point in an image with coordinates* $[i, j]$ *at the location of a significant local intensity change in the image.*

Definition 5.2 *An edge fragment corresponds to the* i *and* j *coordinates of an edge and the edge orientation* θ*, which may be the gradient angle.*

Definition 5.3 *An edge detector is an algorithm that produces a set of edges (edge points or edge fragments) from an image.*

Definition 5.4 *A contour is a list of edges or the mathematical curve that models the list of edges.*

Definition 5.5 *Edge linking is the process of forming an ordered list of edges from an unordered list. By convention, edges are ordered by traversal in a clockwise direction.*

Definition 5.6 *Edge following is the process of searching the (filtered) image to determine contours.*

The coordinates of an edge point may be the integer row and column indices of the pixel where the edge was detected, or the coordinates of the edge location at subpixel resolution. The edge coordinates may be in the coordinate system of the original image, but more likely are in the coordinate system of the image produced by the edge detection filter since filtering may translate or scale image coordinates. An edge fragment may be conceptualized as a small line segment about the size of a pixel, or as a point with an orientation attribute. The term *edge* is commonly used for either edge points or edge fragments.

The edge set produced by an edge detector can be partitioned into two subsets: correct edges, which correspond to edges in the scene, and false edges, which do not correspond to edges in the scene. A third set of edges can be defined as those edges in the scene that should have been detected. This is the set of missing edges. The false edges are called false positives, and the missing edges are called false negatives.

The difference between edge linking and edge following is that edge linking takes as input an unordered set of edges produced by an edge detector and forms an ordered list of edges. Edge following takes as input an image and produces an ordered list of edges. Edge detection uses local information to decide if a pixel is an edge, while edge following can use global information.

5.1 Gradient

Edge detection is essentially the operation of detecting significant local changes in an image. In one dimension, a step edge is associated with a local peak in

the first derivative. The gradient is a measure of change in a function, and an image can be considered to be an array of samples of some continuous function of image intensity. By analogy, significant changes in the gray values in an image can be detected by using a discrete approximation to the gradient. The gradient is the two-dimensional equivalent of the first derivative and is defined as the *vector*

$$\mathbf{G}[f(x,y)] = \begin{bmatrix} G_x \\ \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \\ \frac{\partial f}{\partial y} \end{bmatrix}. \tag{5.1}$$

There are two important properties associated with the gradient: (1) the vector $\mathbf{G}[f(x,y)]$ points in the direction of the maximum rate of increase of the function $f(x,y)$, and (2) the magnitude of the gradient, given by

$$G[f(x,y)] = \sqrt{G_x^2 + G_y^2}, \tag{5.2}$$

equals the maximum rate of increase of $f(x,y)$ per unit distance in the direction G. It is common practice, however, to approximate the gradient magnitude by absolute values:

$$G[f(x,y)] \approx |G_x| + |G_y| \tag{5.3}$$

or

$$G[f(x,y)] \approx \max(|G_x|, |G_y|). \tag{5.4}$$

From vector analysis, the *direction* of the gradient is defined as:

$$\alpha(x,y) = \tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{5.5}$$

where the angle α is measured with respect to the x axis.

Note that the magnitude of the gradient is actually independent of the direction of the edge. Such operators are called *isotropic operators.*

Numerical Approximation

For digital images, the derivatives in Equation 5.1 are approximated by differences. The simplest gradient approximation is

$$G_x \cong f[i, j+1] - f[i,j] \tag{5.6}$$

$$G_y \cong f[i,j] - f[i+1,j]. \tag{5.7}$$

Remember that j corresponds to the x direction and i to the negative y direction. These can be implemented with simple convolution masks as shown below:

$$G_x = \boxed{-1 \mid 1} \qquad G_y = \begin{array}{|c|}\hline 1 \\ \hline -1 \\ \hline \end{array} \tag{5.8}$$

When computing an approximation to the gradient, it is critical that the x and y partial derivatives be computed at exactly the same position in space. However, using the above approximations, G_x is actually the approximation to the gradient at the interpolated point $[i, j+\frac{1}{2}]$ and G_y at $[i+\frac{1}{2}, j]$. For this reason, 2×2 first differences, rather than 2×1 and 1×2 masks, are often used for the x and y partial derivatives:

$$G_x = \begin{array}{|c|c|}\hline -1 & 1 \\ \hline -1 & 1 \\ \hline \end{array} \qquad G_y = \begin{array}{|c|c|}\hline 1 & 1 \\ \hline -1 & -1 \\ \hline \end{array} \tag{5.9}$$

Now, the positions about which the gradients in the x and y directions are calculated are the same. This point lies between all four pixels in the 2×2 neighborhood at the interpolated point $[i+\frac{1}{2}, j+\frac{1}{2}]$. This fact may lead to some confusion. Therefore, an alternative approach is to use a 3×3 neighborhood and calculate the gradient about the center pixel. These methods are discussed in Section 5.2.

5.2 Steps in Edge Detection

Algorithms for edge detection contain three steps:

Filtering: Since gradient computation based on intensity values of only two points are susceptible to noise and other vagaries in discrete computations, filtering is commonly used to improve the performance of an edge detector with respect to noise. However, there is a trade-off between edge strength and noise reduction. More filtering to reduce noise results in a loss of edge strength.

Enhancement: In order to facilitate the detection of edges, it is essential to determine changes in intensity in the neighborhood of a point. Enhancement emphasizes pixels where there is a significant change in local intensity values and is usually performed by computing the gradient magnitude.

Detection: We only want points with strong edge content. However, many points in an image have a nonzero value for the gradient, and not all of these points are edges for a particular application. Therefore, some method should be used to determine which points are edge points. Frequently, thresholding provides the criterion used for detection.

Examples at the end of this section will clearly illustrate each of these steps using various edge detectors. Many edge detection algorithms include a fourth step:

Localization: The location of the edge can be estimated with subpixel resolution if required for the application. The edge orientation can also be estimated.

It is important to note that detection merely indicates that an edge is present near a pixel in an image, but does not necessarily provide an accurate estimate of edge location or orientation. The errors in edge detection are errors of misclassification: false edges and missing edges. The errors in edge estimation are modeled by probability distributions for the location and orientation estimates. We distinguish between edge detection and estimation because these steps are performed by different calculations and have different error models.

Many edge detectors have been developed in the last two decades. Here we will discuss some commonly used edge detectors. As will be clear, edge detectors differ in use of the computational approach in one or more of the above three steps. We will discuss the implications of these steps after we have discussed the edge detectors.

5.2.1 Roberts Operator

The Roberts cross operator provides a simple approximation to the gradient magnitude:

$$G[f[i,j]] = |f[i,j] - f[i+1,j+1]| + |f[i+1,j] - f[i,j+1]|. \qquad (5.10)$$

Using convolution masks, this becomes

$$G[f[i,j]] = |G_x| + |G_y| \tag{5.11}$$

where G_x and G_y are calculated using the following masks:

$$G_x = \begin{array}{|c|c|}\hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array} \qquad G_y = \begin{array}{|c|c|}\hline 0 & -1 \\ \hline 1 & 0 \\ \hline \end{array} \tag{5.12}$$

As with the previous 2×2 gradient operator, the differences are computed at the interpolated point $[i + \frac{1}{2}, j + \frac{1}{2}]$. The Roberts operator is an approximation to the continuous gradient at that point and not at the point $[i, j]$ as might be expected. The results of Roberts edge detector are shown in the figures at the end of this section.

5.2.2 Sobel Operator

As mentioned previously, a way to avoid having the gradient calculated about an interpolated point between pixels is to use a 3×3 neighborhood for the gradient calculations. Consider the arrangement of pixels about the pixel $[i, j]$ shown in Figure 5.3. The Sobel operator is the magnitude of the gradient computed by

$$M = \sqrt{s_x^2 + s_y^2}, \tag{5.13}$$

where the partial derivatives are computed by

$$s_x = (a_2 + ca_3 + a_4) - (a_0 + ca_7 + a_6) \tag{5.14}$$

$$s_y = (a_0 + ca_1 + a_2) - (a_6 + ca_5 + a_4) \tag{5.15}$$

with the constant $c = 2$.

Like the other gradient operators, s_x and s_y can be implemented using convolution masks:

$$s_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad s_y = \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -2 & -1 \\ \hline \end{array} \tag{5.16}$$

Note that this operator places an emphasis on pixels that are closer to the center of the mask. The figures at the end of this section show the performance of this operator. The Sobel operator is one of the most commonly used edge detectors.

a_0	a_1	a_2
a_7	$[i, j]$	a_3
a_6	a_5	a_4

Figure 5.3: The labeling of neighborhood pixels used to explain the Sobel and Prewitt operators [186].

5.2.3 Prewitt Operator

The Prewitt operator uses the same equations as the Sobel operator, except that the constant $c = 1$. Therefore:

$$s_x = \begin{array}{|c|c|c|}\hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \qquad s_y = \begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 0 & 0 & 0 \\ \hline -1 & -1 & -1 \\ \hline \end{array} \tag{5.17}$$

Note that, unlike the Sobel operator, this operator does not place any emphasis on pixels that are closer to the center of the masks. The performance of this edge detector is also shown in the figures at the end of this section.

5.2.4 Comparison

We now compare the different edge detectors discussed so far. The comparisons will be presented according to the first three steps described at the beginning of this section: *filtering*, *enhancement*, and *detection*. The estimation step will not be shown here. In addition, we will give results of edge detection on noisy images for two specific cases—one utilizing the filtering step and one omitting the filtering step. Results of edge detection using varying amounts of filtering will also be given.

For each of the following four figures, the sum of the absolute values of the x and y components of the gradient was used as the gradient magnitude (Equation 5.3). The filter used was the 7×7 Gaussian filter described in the

previous chapter. The threshold values used for detection are given in the captions.

Figure 5.4 shows results of all the edge detection methods discussed so far, from the simple 1×2 gradient approximation up to the Prewitt operator. Figure 5.5 shows the results of the edge detectors when the filtering step is omitted. The next set of images (Figure 5.6) shows the results of edge detection on the same image now with additive Gaussian noise, $\sigma = 12$. The filter used was the same Gaussian filter as used in the previous figure. The final series of images (Figure 5.7) shows the results of edge detection on the same noisy image. However, for these images the filtering step was again omitted. Note the many false edges detected as a result of the noise.

5.3 Second Derivative Operators

The edge detectors discussed earlier computed the first derivative and, if it was above a threshold, the presence of an edge point was assumed. This results in detection of too many edge points. (Notice the thick lines after thresholding in Figures 5.4 to 5.7.) A better approach would be to find only the points that have local maxima in gradient values and consider them edge points, as shown in Figure 5.8. This means that at edge points, there will be a peak in the first derivative and, equivalently, there will be a zero crossing in the second derivative. Thus, edge points may be detected by finding the zero crossings of the second derivative of the image intensity.

There are two operators in two dimensions that correspond to the second derivative: the Laplacian and second directional derivative.

5.3.1 Laplacian Operator

The second derivative of a smoothed step edge is a function that crosses zero at the location of the edge (see Figure 5.8). The Laplacian is the two-dimensional equivalent of the second derivative. The formula for the Laplacian of a function $f(x, y)$ is

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}. \tag{5.18}$$

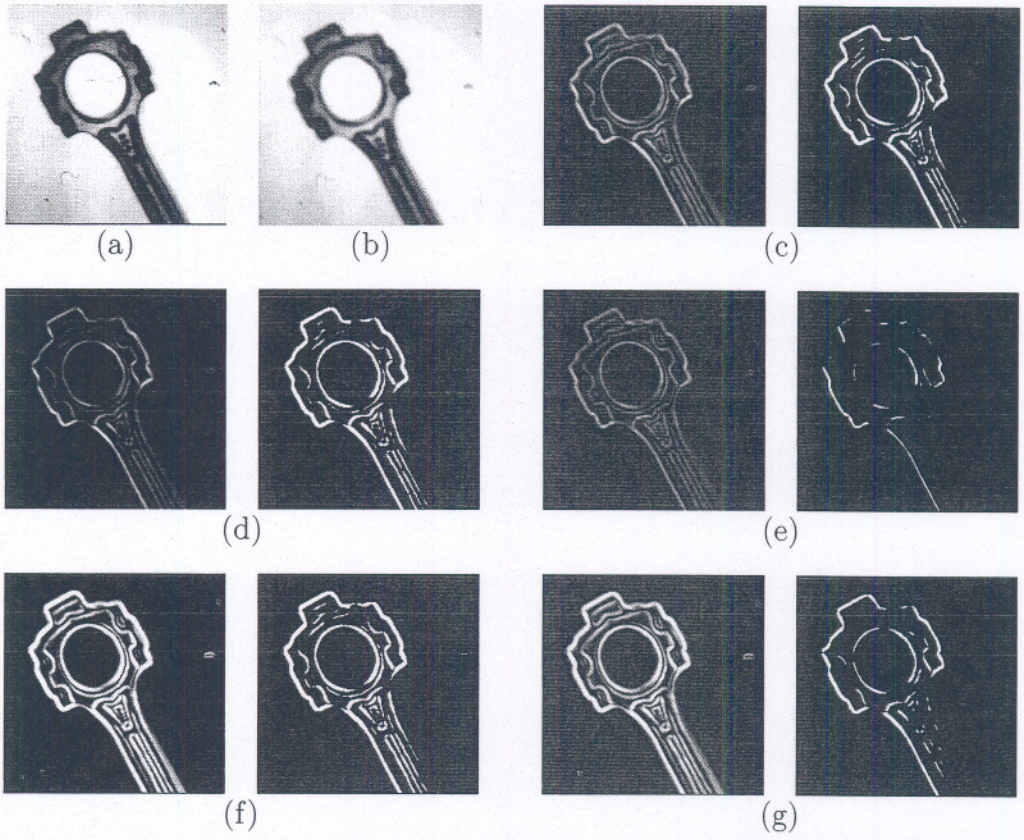


Figure 5.4: A comparison of various edge detectors. (a) Original image. (b) Filtered image. (c) Simple gradient using 1×2 and 2×1 masks, $T = 32$. (d) Gradient using 2×2 masks, $T = 64$. (e) Roberts cross operator, $T = 64$. (f) Sobel operator, $T = 225$. (g) Prewitt operator, $T = 225$.

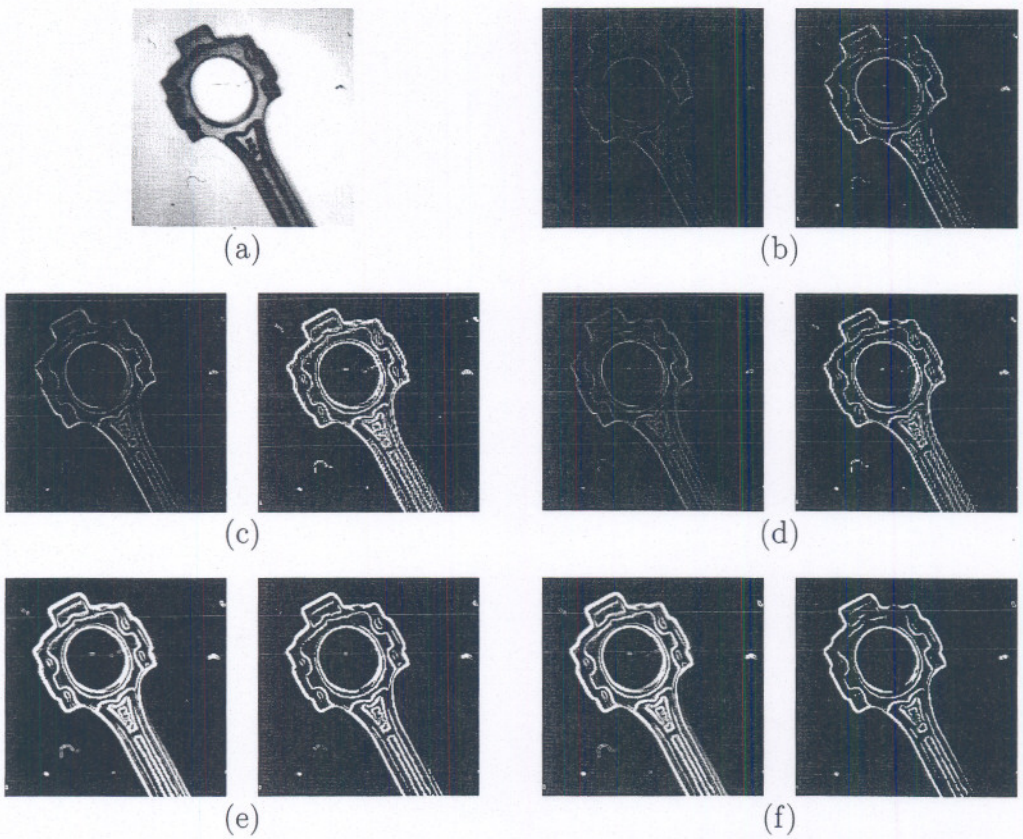


Figure 5.5: A comparison of various edge detectors without filtering. (a) Original image. (b) Simple gradient using 1×2 and 2×1 masks, $T = 64$. (c) Gradient using 2×2 masks, $T = 64$. (d) Roberts cross operator, $T = 64$. (e) Sobel operator, $T = 225$. (f) Prewitt operator, $T = 225$.

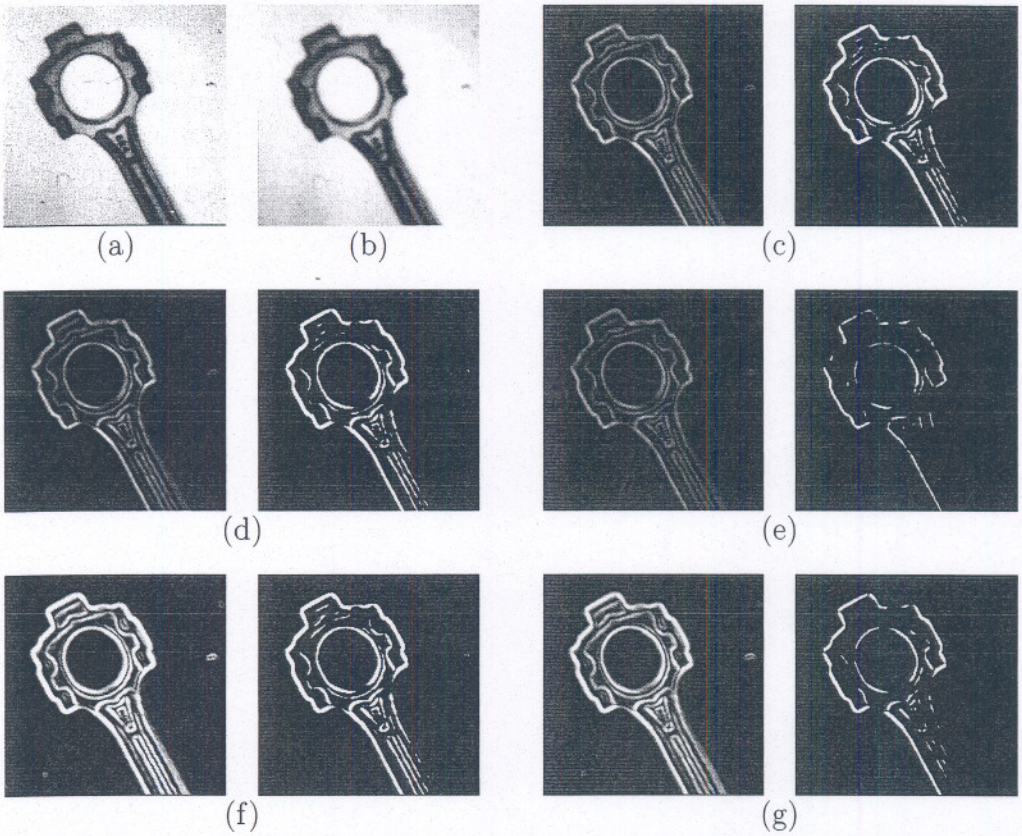


Figure 5.6: A comparison of various edge detectors on a noisy image. (a) Noisy image. (b) Filtered image. (c) Simple gradient using 1×2 and 2×1 masks, $T = 32$. (d) Gradient using 2×2 masks, $T = 64$. (e) Roberts cross operator, $T = 64$. (f) Sobel operator, $T = 225$. (g) Prewitt operator, $T = 225$.

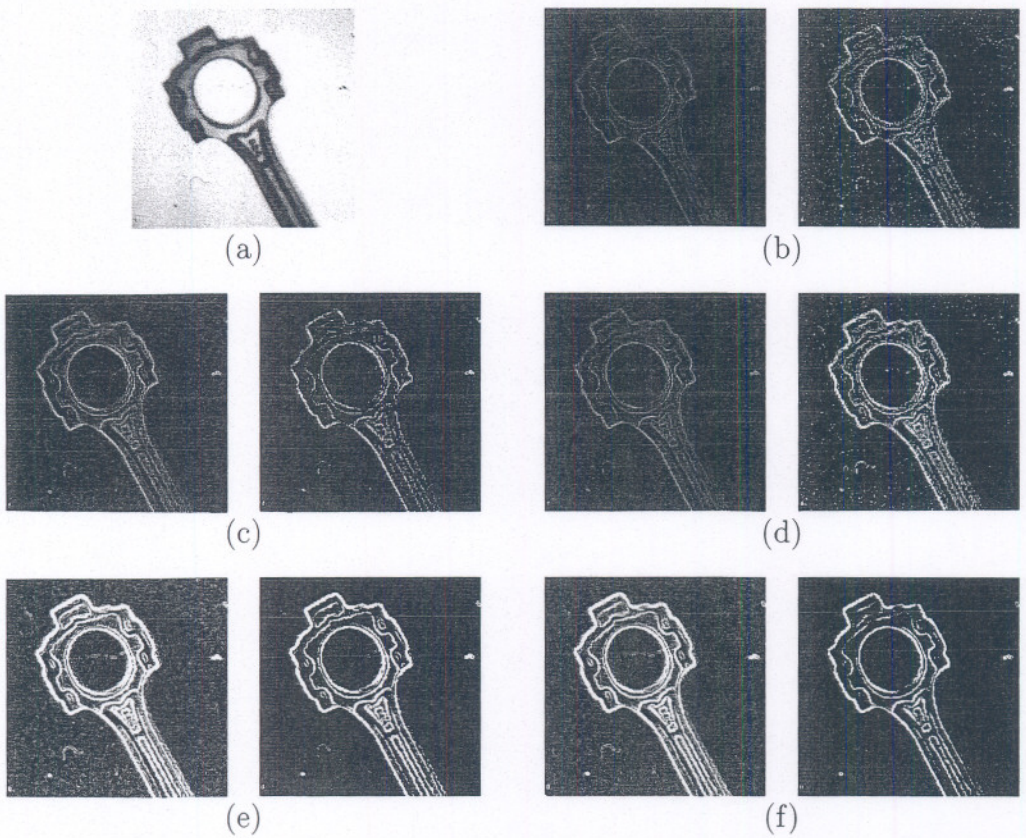


Figure 5.7: A comparison of various edge detectors on a noisy image without filtering. (a) Noisy image. (b) Simple gradient using 1×2 and 2×1 masks, $T = 64$. (c) Gradient using 2×2 masks, $T = 128$. (d) Roberts cross operator, $T = 64$. (e) Sobel operator, $T = 225$. (f) Prewitt operator, $T = 225$.

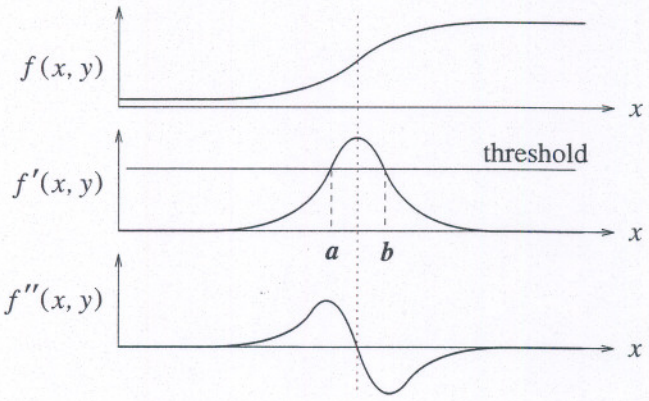


Figure 5.8: If a threshold is used for detection of edges, all points between a and b will be marked as edge pixels. However, by removing points that are *not* a local maximum in the first derivative, edges can be detected more accurately. This local maximum in the first derivative corresponds to a zero crossing in the second derivative.

The second derivatives along the x and y directions are approximated using difference equations:

$$\begin{aligned}
\frac{\partial^2 f}{\partial x^2} &= \frac{\partial G_x}{\partial x} && (5.19)\\
&= \frac{\partial \left(f[i,j+1] - f[i,j]\right)}{\partial x} && (5.20)\\
&= \frac{\partial f[i,j+1]}{\partial x} - \frac{\partial f[i,j]}{\partial x} && (5.21)\\
&= \left(f[i,j+2] - f[i,j+1]\right) - \left(f[i,j+1] - f[i,j]\right) && (5.22)\\
&= f[i,j+2] - 2f[i,j+1] + f[i,j]. && (5.23)
\end{aligned}$$

However, this approximation is centered about the pixel $[i, j+1]$. Therefore, by replacing j with $j-1$, we obtain

$$\frac{\partial^2 f}{\partial x^2} = f[i,j+1] - 2f[i,j] + f[i,j-1], \qquad (5.24)$$

which is the desired approximation to the second partial derivative centered about $[i, j]$. Similarly,

$$\frac{\partial^2 f}{\partial y^2} = f[i+1, j] - 2f[i, j] + f[i-1, j]. \tag{5.25}$$

By combining these two equations into a single operator, the following mask can be used to approximate the Laplacian:

$$\nabla^2 \approx \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array} \tag{5.26}$$

Sometimes it is desired to give more weight to the center pixels in the neighborhood. An approximation to the Laplacian which does this is

$$\nabla^2 \approx \begin{array}{|c|c|c|}\hline 1 & 4 & 1 \\ \hline 4 & -20 & 4 \\ \hline 1 & 4 & 1 \\ \hline \end{array} \tag{5.27}$$

The Laplacian operator signals the presence of an edge when the output of the operator makes a transition through zero. Trivial zeros (uniform zero regions) are ignored. In principle, the zero crossing location can be estimated to subpixel resolution using linear interpolation, but the result may be inaccurate due to noise.

Consider the example shown in Figure 5.9. This figure shows the result of the Laplacian on an image with a simple step edge. A single row of the resulting image is:

0	0	0	6	−6	0	0	0

In this example, the zero crossing, corresponding to the edge in the original image, lies halfway between the two center pixels. The edge should be marked at either the pixel to the left or the pixel to the right of the edge, as long as it is marked consistently throughout the image. In most cases, however, the zero crossing rarely lies exactly between two pixels, and the actual edge

2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8
2	2	2	2	2	8	8	8	8	8

A sample image containing a vertical step edge.

0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0
0	0	0	6	−6	0	0	0

Figure 5.9: The response of the Laplacian to a vertical step edge.

location must be determined by interpolating the pixel values on either side of the zero crossing.

Now consider the example in Figure 5.10. This figure shows the response of the Laplacian to a ramp edge. A single row of the output of the Laplacian is

0	0	0	3	0	−3	0	0

The zero crossing directly corresponds to a pixel in the image. Again, this is an ideal situation, and the actual edge location should be determined by interpolation.

5.3.2 Second Directional Derivative

The second directional derivative is the second derivative computed in the direction of the gradient. The operator is implemented using the formula

$$\frac{\partial^2}{\partial n^2} = \frac{f_x^2 f_{xx} + 2 f_x f_y f_{xy} + f_y^2 f_{yy}}{f_x^2 + f_y^2} \tag{5.28}$$

2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8
2	2	2	2	2	5	8	8	8	8

A sample image containing a vertical ramp edge.

0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0
0	0	0	3	0	−3	0	0

Figure 5.10: The response of the Laplacian to a vertical ramp edge.

The Laplacian and second directional derivative operators are not used frequently in machine vision since any operator involving two derivatives is affected by noise more than an operator involving a single derivative. Even very small local peaks in the first derivative will result in zero crossings in the second derivative. To avoid the effect of noise, powerful filtering methods must be used. In the following section, we discuss an approach which combines Gaussian filtering with the second derivative for edge detection.

5.4 Laplacian of Gaussian

As mentioned above, edge points detected by finding the zero crossings of the second derivative of the image intensity are very sensitive to noise. Therefore, it is desirable to filter out the noise before edge enhancement. To do this, the *Laplacian of Gaussian* (LoG), due to Marr and Hildreth [164], combines Gaussian filtering with the Laplacian for edge detection.

The fundamental characteristics of the Laplacian of Gaussian edge detector are

1. The smoothing filter is a Gaussian.
2. The enhancement step is the second derivative (Laplacian in two dimensions).
3. The detection criterion is the presence of a zero crossing in the second derivative with a corresponding large peak in the first derivative.
4. The edge location can be estimated with subpixel resolution using linear interpolation.

In this approach, an image should first be convolved with a Gaussian filter. (We discuss Gaussian filtering in more detail in Section 5.6.) This step smooths an image and reduces noise. Isolated noise points and small structures will be filtered out. Since the smoothing will result in *spreading* of edges, the edge detector considers as edges only those pixels that have locally maximum gradient. This is achieved by using zero crossings of the second derivative. The Laplacian is used as the approximation of the second derivative in 2-D because it is an isotropic operator. To avoid detection of insignificant edges, only the zero crossings whose corresponding first derivative is above some threshold are selected as edge points.

The output of the LoG operator, $h(x, y)$, is obtained by the convolution operation

$$h(x, y) = \nabla^2 \left[(g(x, y) \star f(x, y) \right]. \tag{5.29}$$

Using the derivative rule for convolution,

$$h(x, y) = \left[\nabla^2 g(x, y) \right] \star f(x, y), \tag{5.30}$$

where

$$\nabla^2 g(x, y) = \left(\frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} \right) e^{-\frac{(x^2+y^2)}{2\sigma^2}} \tag{5.31}$$

is commonly called the *Mexican hat* operator (shown in Figure 5.11). Thus, the following two methods are mathematically equivalent:

1. Convolve the image with a Gaussian smoothing filter and compute the Laplacian of the result.

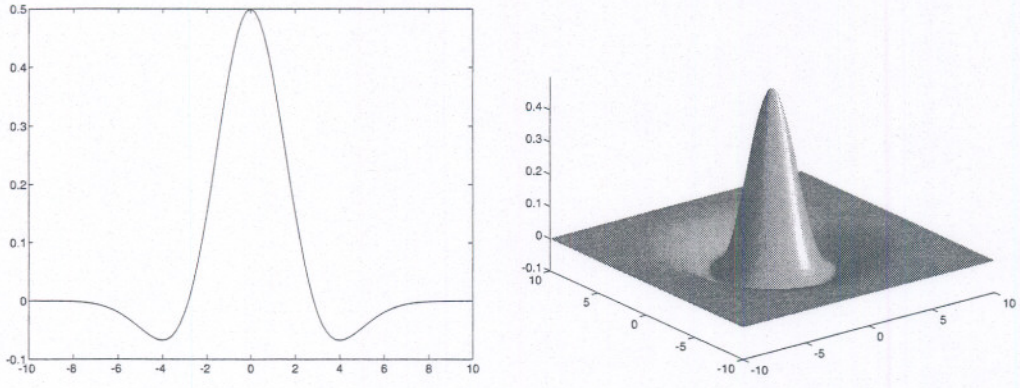


Figure 5.11: The *inverted* Laplacian of Gaussian function, $\sigma = 2$, in one and two dimensions.

2. Convolve the image with the linear filter that is the Laplacian of the Gaussian filter.

If the first method is adopted, Gaussian smoothing masks such as those described in Section 4.5.5 may be used. Typical masks to directly implement the LoG are given in Figure 5.12. In Figure 5.13 we show the result of applying the Laplacian of Gaussian operator and detection of zero crossings. For a discussion on efficient methods to implement the Laplacian of Gaussian, see [117].

At the beginning of Section 5.2, we stated that filtering (usually smoothing), enhancement, and detection were the three steps in edge detection. This is still true for edge detection using the Laplacian of Gaussian. Smoothing is performed with a Gaussian filter, enhancement is done by transforming edges into zero crossings, and detection is done by detecting the zero crossings.

It can be shown that the slope of the zero crossing depends on the contrast of the change in image intensity across the edge. The problem of combining edges obtained by applying different-size operators to images remains. In the above approach, edges at a particular resolution are obtained. To obtain real edges in an image, it may be necessary to combine information from operators at several filter sizes.

5×5 Laplacian of Gaussian mask

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

17×17 Laplacian of Gaussian mask

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
0	−1	−1	−2	−3	−3	−3	−2	−3	−2	−3	−3	−3	−2	−1	−1	0
0	−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	−2	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
−1	−1	−3	−3	−2	2	10	18	21	18	10	2	−2	−3	−3	−1	−1
−1	−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	−2	−1	0
0	−1	−1	−2	−3	−3	−3	−2	−3	−2	−3	−3	−3	−2	−1	−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	0	0	0
0	0	0	0	0	0	−1	−1	−1	−1	−1	0	0	0	0	0	0

Figure 5.12: Some useful Laplacian of Gaussian masks [146].

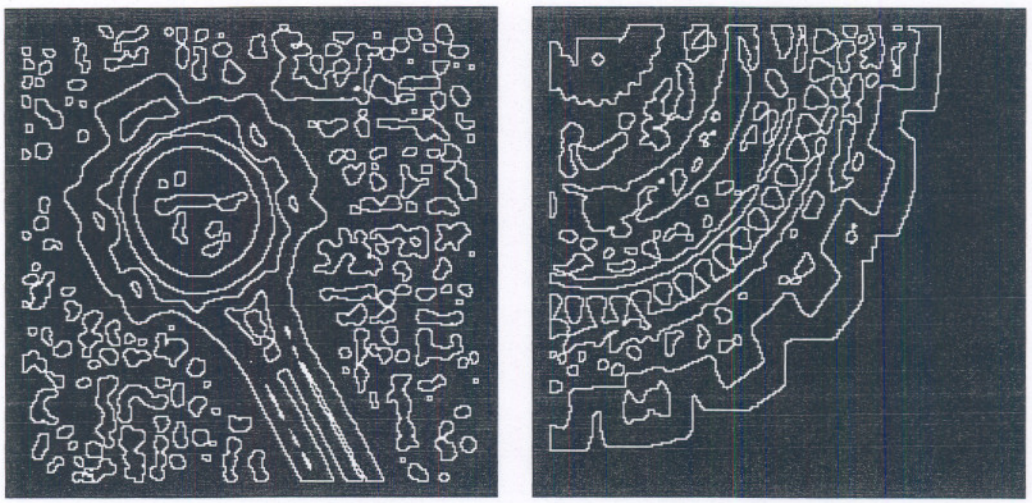


Figure 5.13: Results of the Laplacian of Gaussian edge detector.

Scale Space

The Gaussian smoothing operation results in the blurring of edges and other sharp discontinuities in an image. The amount of blurring depends on the value of σ. A larger σ results in better noise filtering but at the same time loses important edge information, which affects the performance of an edge detector. If a small filter is used, there is likely to be more noise due to insufficient averaging. For large filters, edges which are close to each other may get merged by smoothing and may be detected as only a single edge. In general, small filters result in too many noise points and large filters result in dislocation of edges and even false edges. The exact size of the filter cannot be determined without knowing the size and location of objects in an image.

Many approaches are being developed which apply filtering masks of multiple sizes and then analyze the behavior of edges at these different *scales* of filtering. The basic idea in these approaches is to exploit the fact that at higher scales, larger filtering masks result in robust but displaced edges. The location of these edges can be determined at smaller scales.

5.5 Image Approximation

An image is an array of samples of a continuous function. Most ideas about images are discussed in the continuous domain and then the desired properties are computed using discrete approximations. If we can estimate the continuous function from which the image samples were taken, then we may be able to compute image properties from the estimated function. This may allow the computation of edge location to subpixel precision.

Let

$$z = f(x, y) \tag{5.32}$$

be a continuous image intensity function like the one shown in Figure 5.14. The task is to reconstruct this continuous function from the sampled gray values. For complex images, the continuous image intensity function may contain extremely high powers of x and y. This makes reconstruction of the original function extremely difficult, if not impossible. Therefore, we try to model the image as a simple piecewise analytical function. Now the task becomes the reconstruction of the individual piecewise functions, or *facets*. In other words, try to find simple functions which best approximate the intensity values *only* in the local neighborhood of each pixel. This is illustrated in Figure 5.15. This approximation is called the *facet model* [102]. Figure 5.16 shows the coordinate system for the facet model using 5×5 neighborhoods.

The continuous image intensity function is approximated locally at every pixel in the image. For an $n \times m$ image, you will obtain $n \cdot m$ approximating functions, each valid only about a specific pixel in the image. These functions, and not the pixel values, are used to locate edges in the image.

A variety of analytical functions of varying complexity can be used to approximate image intensities. Often for simple images, piecewise constant or piecewise bilinear functions are adequate approximations of the intensity values. However, for images with more complex regions, biquadratic, bicubic, and even higher-power functions are used. For this example, we will model the image neighborhood as the following bicubic polynomial:

$$\begin{aligned} f(x, y) = {} & k_1 + k_2 x + k_3 y + k_4 x^2 + k_5 xy + k_6 y^2 \\ & + k_7 x^3 + k_8 x^2 y + k_9 x y^2 + k_{10} y^3 \end{aligned} \tag{5.33}$$

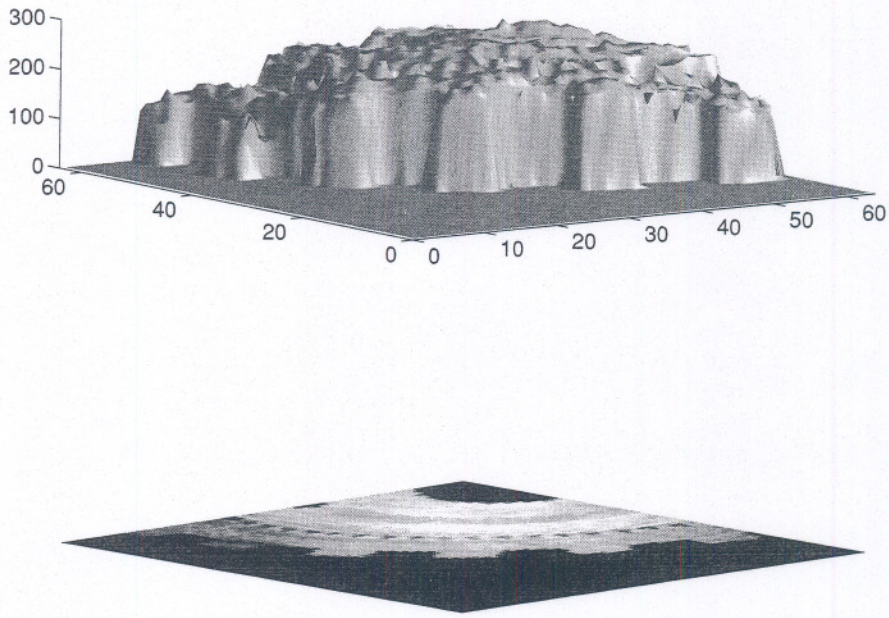


Figure 5.14: A graphical representation of the continuous image intensity function.

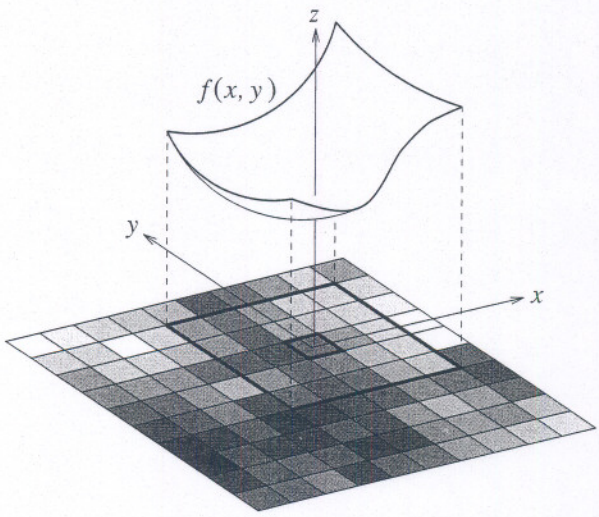


Figure 5.15: An illustration of an approximated function within a 5×5 neighborhood.

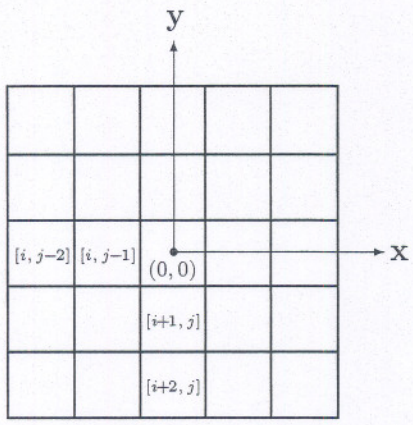


Figure 5.16: An example of the coordinate system for the facet model using a 5×5 neighborhood. The continuous intensity function is approximated at every pixel location *only* within this neighborhood. The array indices are marked on the pixels for clarity. Note that pixel $[i, j]$ lies in the center of the neighborhood.

where x and y are coordinates with respect to $(0, 0)$, the center of the image plane neighborhood that is being approximated (see Figure 5.16).

The goal now is to calculate the coefficients k_i in Equation 5.33 for each approximating function. Use least-squares methods to compute the coefficients k_i using singular-value decomposition, or, if you are using a 5×5 neighborhood, use the masks shown in Figure 5.17 to directly compute the coefficients for the bicubic approximation.[1]

To detect edges, use the fact that edge points occur at relative extrema in the first directional derivative of the function approximating the image intensity in the neighborhood of a pixel. The presence of relative extrema in the first derivative will result in a zero crossing in the second derivative in the direction of the first derivative.

The first derivative in the direction θ is given by

$$f'_\theta(x, y) = \frac{\partial f}{\partial x} \cos\theta + \frac{\partial f}{\partial y} \sin\theta. \tag{5.34}$$

[1]For a complete discussion of the construction of these masks, refer to [103, chap. 8].

−13	2	7	2	−13
2	17	22	17	2
7	22	27	22	7
2	17	22	17	2
−13	2	7	2	−13

$\left[\frac{1}{175}\right]$ k_1 1

31	−44	0	44	−31
−5	−62	0	62	5
−17	−68	0	68	17
−5	−62	0	62	5
31	−44	0	44	−31

$\left[\frac{1}{420}\right]$ k_2 x

−31	5	17	5	−31
44	62	68	62	44
0	0	0	0	0
−44	−62	−68	−62	−44
31	−5	−17	−5	31

$\left[\frac{1}{420}\right]$ k_3 y

2	−1	−2	−1	2
2	−1	−2	−1	2
2	−1	−2	−1	2
2	−1	−2	−1	2
2	−1	−2	−1	2

$\left[\frac{1}{70}\right]$ k_4 x^2

−4	−2	0	+2	+4
−2	−1	0	+1	+2
0	0	0	0	0
+2	+1	0	−1	−2
+4	+2	0	−2	−4

$\left[\frac{1}{100}\right]$ k_5 xy

2	2	2	2	2
−1	−1	−1	−2	−1
−2	−2	−2	−2	−2
−1	−1	−1	−1	−1
2	2	2	2	2

$\left[\frac{1}{420}\right]$ k_6 y^2 $\left[\frac{1}{70}\right]$

−1	2	0	−2	1
−1	2	0	−2	1
−1	2	0	−2	1
−1	2	0	−2	1
−1	2	0	−2	1

$\left[\frac{1}{60}\right]$ k_7 x^3

4	−2	−4	−2	4
2	−1	−2	−1	2
0	0	0	0	0
−2	1	2	1	−2
−4	2	4	2	−4

$\left[\frac{1}{140}\right]$ k_8 x^2y

−4	−2	0	2	4
2	1	0	−1	−2
4	2	0	−2	−4
2	1	0	−1	−2
−4	−2	0	2	4

$\left[\frac{1}{140}\right]$ k_9 xy^2

1	1	1	1	1
−2	−2	−2	−2	−2
0	0	0	0	0
2	2	2	2	2
−1	−1	−1	−1	−1

$\left[\frac{1}{60}\right]$ k_{10} y^3

Figure 5.17: Masks for computing the coefficients of the bicubic approximation [103].

The second directional derivative in the direction θ is given by

$$f''_\theta(x,y) = \frac{\partial^2 f}{\partial x^2}\cos^2\theta + 2\frac{\partial^2 f}{\partial x \partial y}\cos\theta\sin\theta + \frac{\partial^2 f}{\partial y^2}\sin^2\theta. \tag{5.35}$$

Since the local image intensity was approximated by a bicubic polynomial, the angle θ may be chosen to be the angle of the approximating plane. This will result in

$$\sin\theta = \frac{k_3}{\sqrt{k_2^2 + k_3^2}} \tag{5.36}$$

$$\cos\theta = \frac{k_2}{\sqrt{k_2^2 + k_3^2}}. \tag{5.37}$$

At a point (x_0, y_0), the second directional derivative in the direction θ is given by

$$\begin{aligned} f''_\theta(x_0,y_0) = {} & 2\left(3k_7\cos^2\theta + 2k_8\sin\theta\cos\theta + k_9\sin^2\theta\right)x_0 \\ & + 2\left(k_8\cos^2\theta + 2k_9\sin\theta\cos\theta + 3k_{10}\sin^2\theta\right)y_0 \\ & + 2\left(k_4\cos^2\theta + k_5\sin\theta\cos\theta + k_6\sin^2\theta\right). \end{aligned} \tag{5.38}$$

Since we are considering points only on the line in the direction θ, $x_0 = \rho\cos\theta$ and $y_0 = \rho\sin\theta$. Substituting this in the above, we get

$$\begin{aligned} f''_\theta(x_0,y_0) = {} & 6\left(k_{10}\sin^3\theta + k_9\sin^2\theta\cos\theta + k_8\sin\theta\cos^2\theta + k_7\cos^3\theta\right)\rho \\ & + 2\left(k_6\sin^2\theta + k_5\sin\theta\cos\theta + k_4\cos^2\theta\right) \end{aligned} \tag{5.39}$$

$$= A\rho + B. \tag{5.40}$$

Thus, there is an edge at (x_0, y_0) in the image if for some ρ, $|\rho| < \rho_0$ where ρ_0 is the length of the side of a pixel,

$$f''_\theta(x_0, y_0; \rho) = 0 \tag{5.41}$$

and

$$f'_\theta(x_0, y_0; \rho) \neq 0. \tag{5.42}$$

In other words, mark the pixel as an edge pixel if the location of the edge falls within the boundaries of the pixel (see Figure 5.18). However, do *not* mark the pixel as an edge pixel if the point lies outside the pixel boundaries as shown in Figure 5.19. The results of this operator are shown in Figure 5.20.

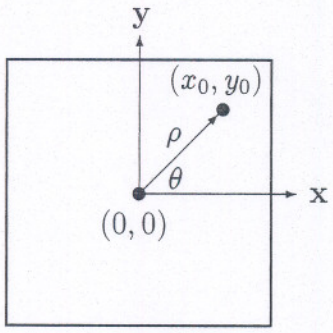


Figure 5.18: An enlarged view of the pixel at the center of the approximated function. (x_0, y_0) is the location of the edge as determined by Equations 5.41 and 5.42. This pixel will be marked as an edge pixel since the location of the edge falls within its boundaries.

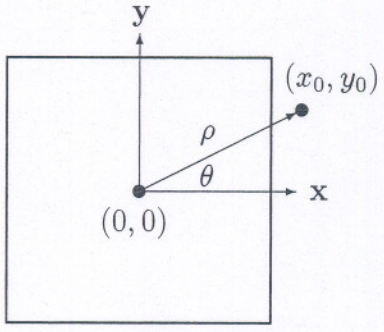


Figure 5.19: An enlarged view of the pixel at the center of the approximated function. This pixel will *not* be marked as an edge pixel since the location of the edge does not fall within the pixel boundaries.

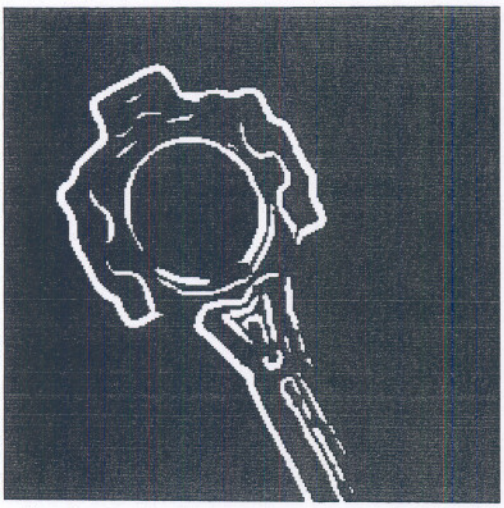


Figure 5.20: Edges obtained with facet model edge detector.

5.6 Gaussian Edge Detection

The essential idea in detecting step edges is to find points in the sampled image that have locally large gradient magnitudes. Much of the research work in step edge detection is devoted to finding numerical approximations to the gradient that are suitable for use with real images. The step edges in real images are not perfectly sharp since the edges are smoothed by the low-pass filtering inherent in the optics of the camera lens and the bandwidth limitations in the camera electronics. The images are also severely corrupted by noise from the camera and unwanted detail in the scene. An approximation to the image gradient must be able to satisfy two conflicting requirements: (1) the approximation must suppress the effects of noise, and (2) the approximation must locate the edge as accurately as possible. There is a trade-off between noise suppression and localization. An edge detection operator can reduce noise by smoothing the image, but this will add uncertainty to the location of the edge; or the operator can have greater sensitivity to the presence of edges, but this will increase the sensitivity of the operator to noise. The type of linear operator that provides the best compromise between noise immunity and localization, while retaining the advantages of Gaussian filtering, is the first derivative of a Gaussian. This operator corresponds to

smoothing an image with a Gaussian function and then computing the gradient. The gradient can be numerically approximated by using the standard finite-difference approximation for the first partial derivatives in the x and y directions listed in Section 5.1. The operator that is the combination of a Gaussian smoothing filter and a gradient approximation is not rotationally symmetric. The operator is symmetric along the edge and antisymmetric perpendicular to the edge (along the line of the gradient). This means that the operator is sensitive to the edge in the direction of steepest change, but is insensitive to the edge and acts as a smoothing operator in the direction along the edge.

5.6.1 Canny Edge Detector

The Canny edge detector is the first derivative of a Gaussian and closely approximates the operator that optimizes the product of signal-to-noise ratio and localization. The Canny edge detection algorithm is summarized by the following notation. Let $I[i,j]$ denote the image. The result from convolving the image with a Gaussian smoothing filter using separable filtering is an array of smoothed data,

$$S[i,j] = G[i,j;\sigma] \star I[i,j], \tag{5.43}$$

where σ is the spread of the Gaussian and controls the degree of smoothing.

The gradient of the smoothed array $S[i,j]$ can be computed using the 2×2 first-difference approximations (Section 5.1) to produce two arrays $P[i,j]$ and $Q[i,j]$ for the x and y partial derivatives:

$$\begin{aligned} P[i,j] &\approx (S[i,j+1] - S[i,j] \\ &\quad + S[i+1,j+1] - S[i+1,j])/2 \qquad (5.44) \\ Q[i,j] &\approx (S[i,j] - S[i+1,j] \\ &\quad + S[i,j+1] - S[i+1,j+1])/2. \qquad (5.45) \end{aligned}$$

The finite differences are averaged over the 2×2 square so that the x and y partial derivatives are computed at the same point in the image. The magnitude and orientation of the gradient can be computed from the standard formulas for rectangular-to-polar conversion:

$$\begin{aligned} M[i,j] &= \sqrt{P[i,j]^2 + Q[i,j]^2} \qquad (5.46) \\ \theta[i,j] &= \arctan(Q[i,j], P[i,j]), \qquad (5.47) \end{aligned}$$

where the arctan function takes two arguments and generates an angle over the entire circle of possible directions. These functions must be computed efficiently, preferably without using floating-point arithmetic. It is possible to compute the gradient magnitude and orientation from the partial derivatives by table lookup. The arctangent can be computed using mostly fixed-point arithmetic[2] with a few essential floating-point calculations performed in software using integer and fixed-point arithmetic [59, chap. 11]. Sedgewick [218, p. 353] provides an algorithm for an integer approximation to the gradient angle that may be good enough for many applications.

Nonmaxima Suppression

The magnitude image array $M[i,j]$ will have large values where the image gradient is large, but this is not sufficient to identify the edges, since the problem of finding locations in the image array where there is rapid change has merely been transformed into the problem of finding locations in the magnitude array $M[i,j]$ that are local maxima. To identify edges, the broad ridges in the magnitude array must be thinned so that only the magnitudes at the points of greatest local change remain. This process is called nonmaxima suppression, which in this case results in thinned edges.

Nonmaxima suppression thins the ridges of gradient magnitude in $M[i,j]$ by suppressing all values along the line of the gradient that are not peak values of a ridge. The algorithm begins by reducing the angle of the gradient $\theta[i,j]$ to one of the four sectors shown in Figure 5.21,

$$\zeta[i,j] = \text{Sector}(\theta[i,j]). \tag{5.48}$$

The algorithm passes a 3×3 neighborhood across the magnitude array $M[i,j]$. At each point, the center element $M[i,j]$ of the neighborhood is compared with its two neighbors along the line of the gradient given by the sector value $\zeta[i,j]$ at the center of the neighborhood. If the magnitude array value $M[i,j]$ at the center is not greater than both of the neighbor magnitudes along the gradient line, then $M[i,j]$ is set to zero. This process thins the broad ridges of gradient magnitude in $M[i,j]$ into ridges that are only one pixel wide. The

[2]In this context, fixed-point arithmetic is like integer arithmetic except that the number carries an implicit scale factor that assumes that the binary point is to the left of the number. Fixed-point arithmetic can be implemented using integer arithmetic on many machines.

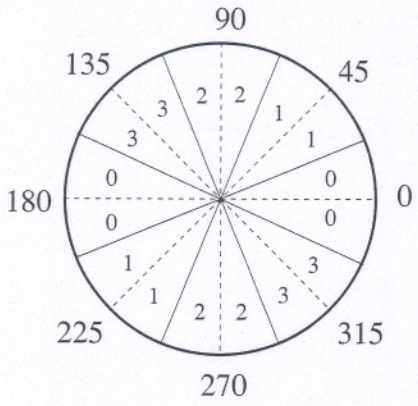


Figure 5.21: The partition of the possible gradient orientations into sectors for nonmaxima suppression is shown. There are four sectors, numbered 0 to 3, corresponding to the four possible combinations of elements in a 3×3 neighborhood that a line must pass through as it passes through the center of the neighborhood. The divisions of the circle of possible gradient line orientations are labeled in degrees.

values for the height of the ridge are retained in the nonmaxima-suppressed magnitude.

Let

$$N[i,j] = \text{nms}(M[i,j], \zeta[i,j]) \tag{5.49}$$

denote the process of nonmaxima suppression. The nonzero values in $N[i,j]$ correspond to the amount of contrast at a step change in the image intensity. In spite of the smoothing performed as the first step in edge detection, the nonmaxima-suppressed magnitude image $N[i,j]$ will contain many false edge fragments caused by noise and fine texture. The contrast of the false edge fragments is small.

Thresholding

The typical procedure used to reduce the number of false edge fragments in the nonmaxima-suppressed gradient magnitude is to apply a threshold to $N[i,j]$. All values below the threshold are changed to zero. The result of applying a threshold to the nonmaxima-suppressed magnitude is an array

of the edges detected in the image $I[i, j]$. There will still be some false edges because the threshold τ was too low (false positives), and portions of actual contours may be missing (false negatives) due to softening of the edge contrast by shadows or because the threshold τ was too high. Selecting the proper threshold is difficult and involves some trial and error. A more effective thresholding scheme uses two thresholds.

The double thresholding algorithm takes the nonmaxima-suppressed image, $N[i, j]$, and applies two thresholds τ_1 and τ_2, with $\tau_2 \approx 2\tau_1$, to produce two thresholded edge images $T_1[i, j]$ and $T_2[i, j]$. Since image T_2 was formed with a higher threshold, it will contain fewer false edges; but T_2 may have gaps in the contours (too many false negatives). The double thresholding algorithm links the edges in T_2 into contours. When it reaches the end of a contour, the algorithm looks in T_1 at the locations of the 8-neighbors for edges that can be linked to the contour. The algorithm continues to gather edges from T_1 until the gap has been bridged to an edge in T_2. The algorithm performs edge linking as a by-product of thresholding and resolves some of the problems with choosing a threshold. The Canny edge detection algorithm is outlined in Algorithm 5.1.

The edge detection algorithm presented in this section has been run on several test images. Figure 5.22 shows the image of a connecting rod. Figures 5.23 and 5.24 present the results of applying the edge detection algorithm summarized in this section to the test image in Figure 5.22. In Figure 5.23, a 7×7 Gaussian filter was used to smooth the image before computing the gradient; in Figure 5.24, a 31×31 Gaussian filter was used. The nonmaxima-suppressed gradient magnitude for the smaller filter size exhibits excellent fine detail in the edges but suffers from excessive unwanted edge fragments

Algorithm 5.1 Canny Edge Detection

1. *Smooth the image with a Gaussian filter.*
2. *Compute the gradient magnitude and orientation using finite-difference approximations for the partial derivatives.*
3. *Apply nonmaxima suppression to the gradient magnitude.*
4. *Use the double thresholding algorithm to detect and link edges.*

Figure 5.22: A test image of a connecting rod. The image was acquired by a Reticon 256 × 256 area CCD array camera.

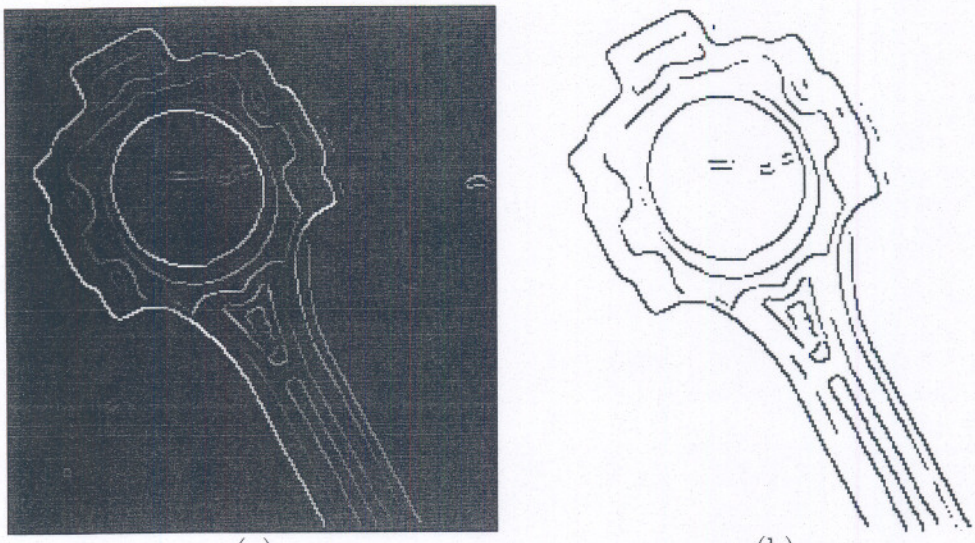
due to noise and fine texture. For the larger filter size, there are fewer unwanted edge fragments, but much of the detail in the edges has been lost. This illustrates the trade-off between edge localization and noise immunity.

5.7 Subpixel Location Estimation

In many applications, it is necessary to estimate the location of an edge to better than the spacing between pixels (subpixel resolution). The methods for obtaining subpixel resolution for gradient and second-order edge detection algorithms are very different and will be considered separately.

First, consider the output of a second-order edge detector such as the Laplacian of Gaussian. The edge is signaled by a zero crossing between pixels. In principle, the edge position can be computed to subpixel resolution by using linear interpolation. In practice, the output of second-order edge detection schemes, even with Gaussian presmoothing, is too noisy to allow any simple interpolation method to provide accurate results.

Obtaining subpixel resolution in edge location after edge detection with a gradient-based scheme is both practical and efficient. The result of applying a Gaussian smoothing filter and first derivative to an ideal step edge is a profile that is exactly the same shape as the Gaussian filter used for smoothing. If

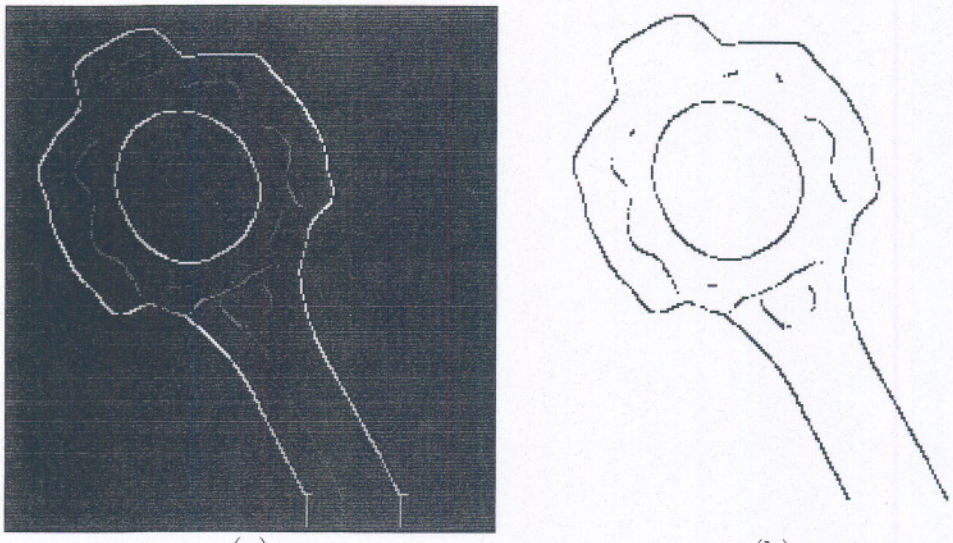


(a) (b)

Figure 5.23: The result of edge detection applied to the test image from Figure 5.22 with a 7×7 Gaussian smoothing filter. The picture in part (a) is a gray level image of the result after smoothing with a Gaussian smoothing filter, computing the gradient approximation, and suppressing nonmaxima. The weak edge fragments due to noise and fine texture do not show up clearly. Part (b) is a plot of the image from part (a) with every pixel that was greater than zero drawn in black. This plot shows the weak edge fragments that are present in the result of edge detection but do not show up clearly in a gray level display of the results.

the step edge is not ideal but makes a gradual transition from one level to another, then the result of Gaussian smoothing and a first derivative can be approximated by a broader Gaussian.

Consider a set of measurements drawn from a normal distribution. The center of the bell-shaped curve corresponds to the mean value of the normal distribution, which can be estimated by averaging the measurements. Now suppose that a histogram of the measurements, rather than the raw measurements themselves, is all the information that is available. The mean can be estimated by dividing the sum of the values for the centers of the histogram buckets, weighted by the number of entries in each bucket, by the area of the histogram. By analogy, the location of an edge can be estimated to subpixel resolution by averaging along the profile in the output of the Gaussian edge



(a) (b)

Figure 5.24: The result of edge detection applied to the test image from Figure 5.22. A 31×31 Gaussian filter was used to smooth the image, followed by computation of the gradient approximation and then nonmaxima suppression. Part (a) displays the nonmaxima-suppressed result as a gray level image. Note how the large smoothing filter has rounded the edges. Part (b) plots the results to show all of the edge fragments.

detector. To compute the edge location to subpixel resolution, take samples of the magnitude of the Gaussian edge detector output (without nonmaxima suppression) along the gradient direction to either side of the edge until the gradient magnitude falls below a threshold. Use the samples of the gradient magnitude g_i as weights to compute the weighted sum of the position d_i along the gradient. The subpixel correction to the position of the edge along the gradient direction is given by

$$\delta d = \frac{\sum_{i=1}^{n} g_i d_i}{\sum_{i=1}^{n} g_i}, \tag{5.50}$$

where d_i is the distance of a pixel along the gradient from the pixel where the edge was detected, and g_i is the gradient magnitude.

A simpler algorithm that is probably as effective is to compute the position $(\delta x, \delta y)$ of the edge to subpixel resolution, relative to the pixel where the edge was detected, by applying a first moment calculation to the magnitude of the gradient of the Gaussian edge detector. The correction can be added to the coordinates of the pixel to provide a more accurate estimate of edge location.

The algorithm for computing the average along the profile of gradient magnitude, though more complex, has the advantage that the profile can be compared with the ideal profile using statistical techniques and the results of this comparison can be used as the criterion for edge detection. If the profile is not close to a Gaussian, the edge does not correspond to the ideal step model. It may not be possible in this case to accurately estimate the edge location using the techniques presented in this section.

In Chapter 12, we will present methods for calibrating the coordinate system of the image. The integer coordinates of a pixel $[i, j]$ can be mapped to coordinates (x, y) in the image plane of the camera. The correction to the edge location provided by either of the methods for estimating edge location to subpixel resolution described above is added to the (x, y) coordinates of the pixel to obtain the precise location of the edge in the image plane. This precise, calibrated value is what is needed for measuring feature dimensions.

5.8 Edge Detector Performance

Measures for evaluating the performance of edge detectors have been formulated by Abdou and Pratt [1] and DeMicheli, Caprile, Ottonello, and Torre [66]. The criteria to consider in evaluating the performance of an edge detector include

1. Probability of false edges
2. Probability of missing edges
3. Error in estimation of the edge angle
4. Mean square distance of the edge estimate from the true edge
5. Tolerance to distorted edges and other features such as corners and junctions

The first two criteria concern the performance of an algorithm as a detector of edges. The second two criteria concern the performance of an algorithm as an estimator of the edge location and orientation. The last criterion concerns the tolerance of the edge algorithm to edges that depart from the ideal model used to formulate the algorithm.

5.8.1 Methods for Evaluating Performance

The performance of an edge detector can be evaluated in two stages: count the number of false and missing edges and measure the variance (or error distribution) for the estimated location and orientation.

For a test case, select a synthetic image where the true edges are known to lie along a contour that can be modeled by a curve with a simple mathematical formula—for example, a filled rectangle where the boundary contour can be modeled by line segments or two filled rectangles where the gap between them is known. Count the number of correct, missing, and false edges by comparing the results of the edge detector with the original (synthetic) image. This is a harder task than it appears to be. The results vary with the threshold, smoothing filter size, interactions between edges, and other factors. If you run an edge detector over the test image with no added noise, no smoothing, and no interactions between edges, then you should get a perfect set of edges (no missing or false edges). Use this set of edges as the standard for comparison.

Now consider the edges obtained from a test case that has added noise, or other distortions in the image that create missing or false edges. Compute a one-to-one match of edges in the test image to edges in the standard, based on the criterion of Euclidean distance. Ideally, we should use a proper matching algorithm such as the method for the disparity analysis of images presented in Section 14.3. Edges too far from the edges in the standard are false edges; edges that pair closely with one edge in the standard are correct. After this procedure, the edges in the standard that are not paired with one edge in the test case are missing edges.

This procedure tests an edge detector based only on its ability to indicate the presence or absence of edges, but says nothing about how accurately the edge locations or orientations are estimated. Compare the locations and orientations of edges in the set of correct edges (computed above) with the original test image. This comparison requires that the model of the test

case be available. For the filled rectangle, the model is the line segments that make up the sides of the rectangle. The edge locations and orientations must be compared with a mathematical description of the model of the scene contours. For each edge with location (x, y), how far is this location from the true location? What is the difference between the orientation of the edge and the orientation of the true curve? The edge location (x, y) could correspond to any point along the contour, but the closest point along the contour is used as the corresponding point, and the distance between the edge point and the closest point is computed. For a line segment, use the formulas in Section 6.4. Estimate the error distribution from a histogram of location errors, or tabulate the sum of the squared error and divide by $n - 1$, where n is the number of edges, to estimate the variance (refer to the formula in Appendix B). The orientation error of an edge is measured by comparing the orientation of the edge fragment with the angle of the normal to the curve that models the scene contour, evaluated at the closest point to the edge point.

5.8.2 Figure of Merit

One method to judge the performance of edge detectors is to look at an edge image and subjectively evaluate the performance. However, this does not provide an objective measure of performance. To quantitatively evaluate the performance of various edge detectors, we should formulate a criterion that may help in judging the relative performance under controlled conditions. We observe that in the response of an edge detector, there can be three types of errors:

- Missing valid edges
- Errors in localizing edges
- Classification of noise as edges

A *figure of merit* for an edge detector should consider these three errors. One such figure of merit, called Pratt's figure of merit [196], is:

$$FM = \frac{1}{\max(I_A, I_I)} \sum_{i=1}^{I_A} \frac{1}{1 + \alpha d_i^2} \tag{5.51}$$

where I_A, I_I, d, and α are detected edges, the ideal edges, the distance between the actual and ideal edges, and a design constant used to penalize displaced edges, respectively.

Note that since this figure involves missing edge points, location of edge points, and false edge points, it can be applied only to a limited class of images. One may generate known objects of controlled contrast at known location and then use the above figure of merit. It is a common practice to evaluate the performance for synthetic images by introducing random noise in images. A plot of signal-to-noise ratio against the figure of merit gives the degradation in the performance of the detector.

5.9 Sequential Methods

All the edge detectors described above are parallel in nature: they can be applied at a single pixel, using local information, independent of the result at other pixels. In practice, the performance of such edge detectors may not be acceptable due to too many missing edges. The edges detected by such detectors have to be linked to form boundaries of objects. Missing edges result in breaks in the boundaries. Several methods have been suggested to improve the performance of edge detection and linking. Such approaches include edge following, relaxation, and boundary tracking. We will discuss relaxation in a later chapter.

Edge following tries to use information from the neighborhood of an edge point to improve the performance of an edge detector. This approach may also be used to thin edges from a simple edge detector response. The basic approach is that an edge point looks at its neighbors. If the direction of an edge is compatible with that of its neighbors, the edges are linked; incompatible edges are removed and, by looking at a large neighborhood, one may fill in missing edges.

The edge following algorithm scans an image to find a strong edge used as a starting point for following the boundary of an object. Depending on the direction of the edge, the edge detector is applied to extend the edge in the proper direction. One may implement the tracking operation, shown in Figure 5.25, either without backtracking or with backtracking. Using this approach, even very weak edge segments of an object may be detected.

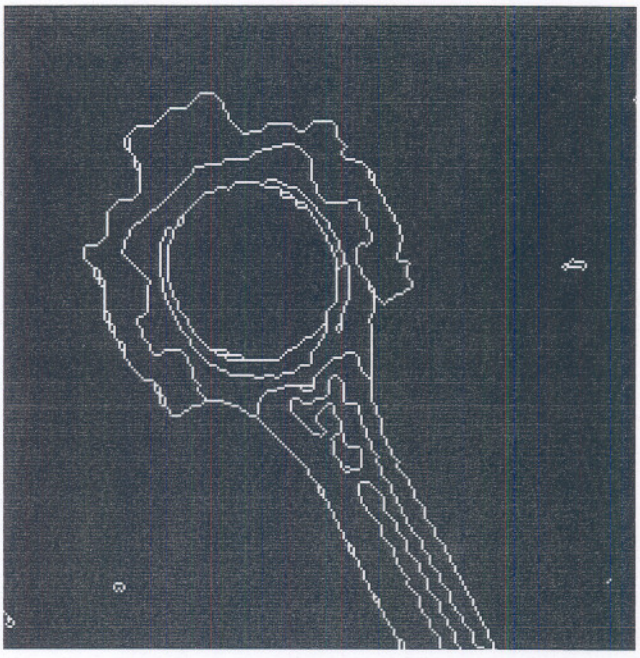


Figure 5.25: An illustration of edge following.

5.10 Line Detection

Up to this point, this chapter has discussed only how to detect step edges in images. Lines are also very important features that may have to be detected. A line can be modeled as two edges with opposite polarity that are close together, separated by less distance than the width of the smoothing filter.

Edge detection filters are designed to respond to step edges and do not provide a meaningful response to lines. A separate algorithm must be used to detect lines, and this line detection algorithm will not provide a meaningful response to step edges. One outstanding problem in machine vision is how to combine edge and line detection in a single system.

Lines can be detected with a modified version of the Canny algorithm: perform nonmaxima suppression on the smoothed image instead of the gradient magnitude. A line is the derivative of a step, so the derivative step in the Canny algorithm is unnecessary.

Further Reading

Edge detection has been one of the most popular research areas since the early days of computer vision. Roberts [202] and Sobel [225] present two early edge detectors that are still commonly used. Some other edge detectors that were popular in the early days were by Prewitt [198], Hueckel [116], and Frei and Chen [84]. Many statistical [258] and several filtering approaches have also been used for edge detection. Algorithms based on the Laplacian of Gaussian [164] and based on the gradient of Gaussian [53] were very popular in the 1980s. The Laplacian of Gaussian edge detection scheme [164] is still dominant in models of biological edge detection. Haralick [102] presented an edge detection scheme based on the second directional derivative. His scheme incorporated a form of image smoothing based on approximating the image with local surface patches.

Although edge detection using Guassian filters of different scales was introduced earlier, it began to receive considerable attention after the paper by Witkin [253]. The papers by Yuille and Poggio [260, 261] provided key theoretical results in this area. Another paper, by Hummel [118], provided additional results. Shah, Sood, and Jain [220] and Lu and Jain [159, 160] studied interaction among edges at different scales and developed a reasoning methodology for detecting edges in scale space. Edge focusing is another approach to edge detection in scale space [25].

Another class of edge detection algorithms searches the image or a filtered version of the image for patterns of image intensity that may be edges. These algorithms combine edge detection with edge linking. The analysis of the patterns of image intensity can be very elaborate, and these algorithms are usually used only in situations where it is necessary to find edges in images with very poor quality. A good treatment of this is provided in [74].

Though many edge detectors have been developed, there is still no well-defined metric to help in selecting the appropriate edge detector for an application. Lack of a performance measure makes judicious selection of an edge detector for a particular application a difficult problem. Some discussion of this may be found in [103, 196].

Exercises

5.1 What is an edge? How does it relate to the boundary of an object? How does it relate to the boundary of a region?

5.2 How can an edge be modeled in an image? Which is the most commonly used model in edge detection? Why?

5.3 What is an isotropic edge detector? How can you implement it?

5.4 What is a directional edge detector? How can you implement it? Where will you use a directional edge detector? Give edge detector masks for detecting edges oriented at 45° and −45°.

5.5 Name all the steps required in edge detection. Can you implement an edge detector by skipping one or more of these steps? How?

5.6 Why is the Laplacian not a good edge operator?

5.7 Describe the Laplacian of Gaussian edge detector. Considering different steps in edge detection, show how the Laplacian is not a good edge operator, but the Laplacian of Gaussian is.

5.8 How can you select the correct size of the operator in the LoG operators? What factors should you consider in selecting the proper size of the operator? Can you have an automatic selection algorithm?

5.9 What is the facet model of an image? How can you use it for edge detection? Can you use this model for region growing also?

5.10 Compare the Gaussian edge detector with the Laplacian of Gaussian. Use all steps in edge detection and compare what the two operators do at these steps. Where is the difference? Do you think that their performances will differ significantly? Explain clearly the difference and the effect of the difference in edge detection.

5.11 Can edges be located at subpixel resolution? How? Is there any particular approach that will be more suitable for this? Consider subpixel edge location estimation for the gradient, Laplacian, and facet models. Compare the different estimation problems.

5.12 To select a particular edge detector in a machine vision application, we should know the comparative performance of edge detectors in the type of application that we are faced with. How can we compare performance of various edge detectors in an objective manner? List all important factors in edge detection and then define a performance measure that can be effectively evaluated.

5.13 What is edge tracking? What factors must be considered in edge tracking?

5.14 The equation for the sloped planar facet model is obtained by setting all terms above k_3 to zero, yielding

$$g[i,j] = k_3 i + k_2 j + k_1.$$

The error in this approximation is given by

$$\epsilon^2 = \sum_{i=-l}^{l} \sum_{j=-l}^{l} \left(\hat{k_3} i + \hat{k_2} j + \hat{k_1} - g[i,j]\right)^2.$$

For the case $l = 1$, estimate the parameters k_1, k_2, and k_3 for the sloped facet model that best approximates the gray levels in the 3×3 neighborhood given below.

i \ j	-1	0	1
-1	5	7	9
0	3	7	7
1	1	3	5

What is the magnitude of the gradient?

5.15 Suppose that an image is smoothed with an $n \times n$ Gaussian filter. During smoothing, the square filter window is moved across the image. The pixel at position $[i,j]$ in the upper left corner of the window is replaced by the smoothed value. After smoothing, the gradient magnitude is computed using the approximations in Section 5.1. As the 2×2 operators are moved across the smoothed image, the pixel at position $[i,j]$

in the upper left corner of the window is replaced by the gradient magnitude. After edge detection, the edge location (x_{ij}, y_{ij}) for each edge pixel is computed to subpixel resolution. Where is the edge location in the coordinate system of the original (unsmoothed) image?

Computer Projects

5.1 Implement the Roberts, Sobel, and Prewitt operators. Apply these to various images. Do you get edges where you expect them? Manually identify several edge segments and note their locations. Compare the locations of edge segments given by your program with manually marked edges. Are you satisfied? Can you improve your program to do better? Change the threshold values for your detection step and see the results.

5.2 Generate a synthetic image that contains one or more known objects with clear, known intensity discontinuities. You should know the locations of these discontinuities precisely. The rest of the image should be without edges. Use any graphics technique or cut and paste using image processing programs. Use this image to do systematic experiments with different edge detectors. Apply the edge detector to the image and obtain all edges.

Define a measure to evaluate the performance of an edge detector and use that in all experiments. What factors would you consider in defining the performance measure? Repeat your experiment by adding noise. You can systematically add random noise using a random number generator. You may also want to systematically change the intensity values of the objects and background. Change the threshold values for your detection step and see the results.

5.3 Develop a test-bed to generate or acquire images for evaluating the performance measure that you defined in the above exercise. Apply each edge detector that you studied in this chapter, and plot the performance of various edge detectors by varying the parameters controlling the quality of images. These curves represent performance characteristics of edge detectors and may be useful in selecting a suitable edge detector for your application.

5.4 In many applications computational time should also be considered in selecting the proper edge detector. Compute the time requirement, using theoretical and experimental analysis, for each edge detector. Based on the performance measure you defined and determined and the time requirement, develop a general guideline for selecting an edge detector in different applications.

5.5 Facet model edge detection:

a. Using the gradient-based edge detection method with *sloped planar* facets (see Exercise 5.14), detect edges in an image of your choice. Specifically, you should do the following:

- Calculate k_1, k_2, and k_3 at each pixel.
- Calculate the gradient at each pixel.
- Identify pixels at which the gradient is above some threshold.

b. Add Gaussian noise to the above images and detect the edges as in part **a**.

5.6 Detect edges using the *cubic* facet approximation. In particular, you should do the following:

- Find the cubic fit coefficients k_1 to k_{10} at each pixel.
- Find the gradient angle α.
- Find the subpixel deviation ρ at which the second derivative is zero.
- Find if the zero crossing occurs within the boundary of the pixel.
- Confirm that the first derivative is nonzero and the third derivative is negative.
- Mark all such pixels as 255 and reset the others to zero.