6 Edges and Contours

Prominent image "events" originating from local changes in intensity or color, such as edges and contours, are of high importance for the visual perception and interpretation of images. The perceived amount of information in an image appears to be directly related to the distinctiveness of the contained structures and discontinuities. In fact, edge-like structures and contours seem to be so important for our human visual system that a few lines in a caricature or illustration are often sufficient to unambiguously describe an object or a scene. It is thus no surprise that the enhancement and detection of edges has been a traditional and important topic in image processing as well. In this chapter, we first look at simple methods for localizing edges and then attend to the related issue of image sharpening.

6.1 What Makes an Edge?

Edges and contours play a dominant role in human vision and probably in many other biological vision systems as well. Not only are edges visually striking, but it is often possible to describe or reconstruct a complete figure from a few key lines, as the example in Fig. 6.1 shows. But how do edges arise, and how can they be technically localized in an image?

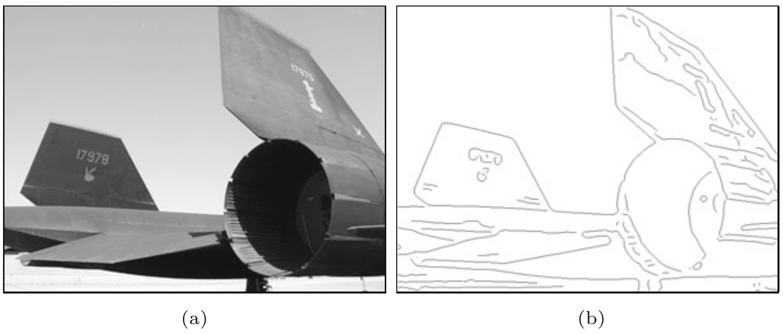


Fig. 6.1
Edges play an important role in human vision. Original image (a) and edge image (b).

Edges can roughly be described as image positions where the local intensity changes distinctly along a particular orientation. The stronger the local intensity change, the higher is the evidence for an edge at that position. In mathematics, the amount of change with respect to spatial distance is known as the first derivative of a function, and we thus start with this concept to develop our first simple edge detector.

6.2 Gradient-Based Edge Detection

For simplicity, we first investigate the situation in only one dimension, assuming that the image contains a single bright region at the center surrounded by a dark background (Fig. 6.2(a)). In this case, the intensity profile along one image line would look like the 1D function $f(x)$, as shown in Fig. 6.2(b). Taking the first derivative of the function f,

$$f'(x) = \frac{df}{dx}(x), \tag{6.1}$$

results in a positive swing at those positions where the intensity rises and a negative swing where the value of the function drops (Fig. 6.2(c)).

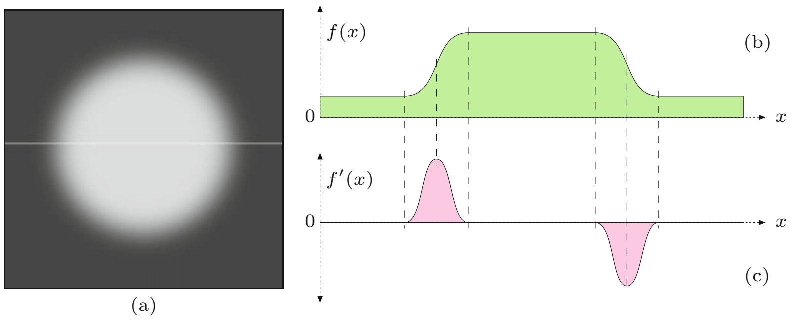


Fig. 6.2
Sample image and first derivative in one dimension: original image (a), horizontal intensity profile $f(x)$ along the center image line (b), and first derivative $f'(x)$ (c).

Unlike in the continuous case, however, the first derivative is undefined for a *discrete* function $f(u)$ (such as the line profile of a real image), and some method is needed to estimate it. Figure 6.3 shows the basic idea, again for the 1D case: the first derivative of a continuous function at position x can be interpreted as the slope of its *tangent* at this position. One simple method for roughly approximating the slope of the tangent for a *discrete* function $f(u)$ at position u is to fit a straight line through the neighboring function values $f(u–1)$ and $f(u+1)$,

$$\frac{df}{dx}(u) \approx \frac{f(u+1) - f(u-1)}{(u+1) - (u-1)} = \frac{f(u+1) - f(u-1)}{2}. \tag{6.2}$$

Of course, the same method can be applied in the vertical direction to estimate the first derivative along the y-axis, thats is, along the image columns.

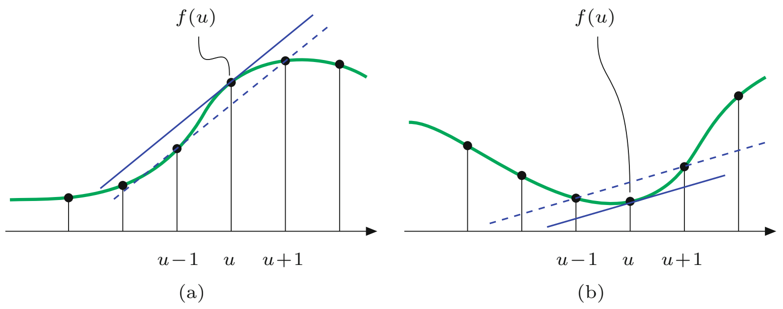


Fig. 6.3
Estimating the first derivative of a discrete function.The slope of the straight (dashed) line between the neighboring function values $f(u-1)$ and $f(u+1)$ is taken as the estimate for the slope of the tangent (i.e., the first derivative) at $f(u)$.

6.2.1 Partial Derivatives and the Gradient

A derivative of a multi-dimensional function taken along one of its coordinate axes is called a *partial derivative*; for example,

$$I_x = \frac{\partial I}{\partial x}(u,v) \qquad \text{and} \qquad I_y = \frac{\partial I}{\partial y}(u,v) \tag{6.3}$$

are the partial derivatives of the 2D image function $I(u,v)$ along the u and v axes, respectively.[1] The vector

$$\nabla I(u,v) = \begin{pmatrix} I_x(u,v) \\ I_y(u,v) \end{pmatrix} = \begin{pmatrix} \frac{\partial I}{\partial x}(u,v) \\ \frac{\partial I}{\partial y}(u,v) \end{pmatrix} \tag{6.4}$$

is called the *gradient* of the function I at position (u,v). The *magnitude* of the gradient,

$$|\nabla I| = \sqrt{I_x^2 + I_y^2}, \tag{6.5}$$

is invariant under image rotation and thus independent of the orientation of the underlying image structures. This property is important for isotropic localization of edges, and thus $|\nabla I|$ is the basis of many practical edge detection methods.

6.2.2 Derivative Filters

The components of the gradient function (Eqn. (6.4)) are simply the first derivatives of the image lines (Eqn. (6.1)) and columns along the horizontal and vertical axes, respectively. The approximation of the first horizontal derivatives (Eqn. (6.2)) can be easily implemented by a linear filter (see Sec. 5.2) with the 1D kernel

$$H_x^{\mathrm{D}} = \begin{bmatrix} -0.5 & 0 & 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}, \tag{6.6}$$

where the coefficients -0.5 and $+0.5$ apply to the image elements $I(u-1,v)$ and $I(u+1,v)$, respectively. Notice that the center pixel $I(u,v)$ itself is weighted with the zero coefficient and is thus ignored. Analogously, the vertical component of the gradient is obtained with the linear filter

[1] ∂ denotes the *partial derivative* or "del" operator.

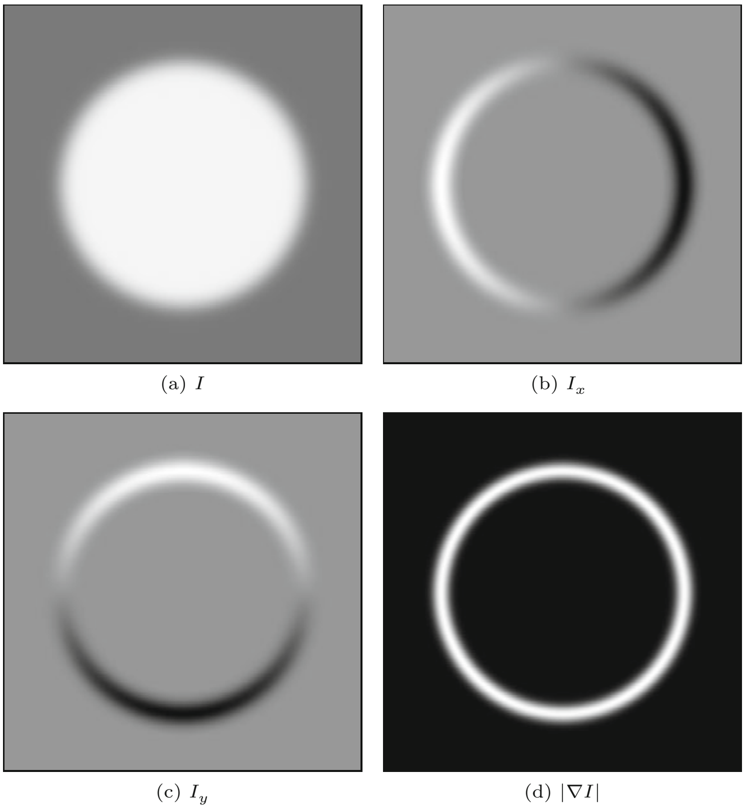


Fig. 6.4
Partial derivatives of a 2D function: synthetic image function I (a); approximate first derivatives in the horizontal direction $\partial I/\partial u$ (b) and the vertical direction $\partial I/\partial v$ (c); magnitude of the resulting gradient $|\nabla I|$ (d). In (b) and (c), the lowest (negative) values are shown black, the maximum (positive) values are white, and zero values are gray.

$$H_y^{\mathrm{D}} = \begin{bmatrix} -0.5 \\ 0 \\ 0.5 \end{bmatrix} = 0.5 \cdot \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}. \tag{6.7}$$

Figure 6.4 shows the results of applying the gradient filters defined in Eqn. (6.6) and Eqn. (6.7) to a synthetic test image.

The orientation dependence of the filter responses can be seen clearly. The horizontal gradient filter H_x^D reacts most strongly to rapid changes along the horizontal direction, (i.e., to *vertical* edges); analogously the vertical gradient filter H_y^D reacts most strongly to *horizontal* edges. The filter response is zero in flat image regions (shown as gray in Fig. 6.4(b, c)).

6.3 Simple Edge Operators

The local gradient of the image function is the basis of many classical edge-detection operators. Practically, they only differ in the type of filter used for estimating the gradient components and the way these components are combined. In many situations, one is not only interested in the *strength* of edge points but also in the local *direction* of the edge. Both types of information are contained in the gradient function and can be easily computed from the directional components. The following small collection describes some frequently used, simple edge operators that have been around for many years and are thus interesting from a historic perspective as well.

6.3.1 Prewitt and Sobel Operators

The edge operators by Prewitt [191] and Sobel [61] are two classic methods that differ only marginally in the derivative filters they use.

Gradient filters

Both operators use linear filters that extend over three adjacent lines and columns, respectively, to counteract the noise sensitivity of the simple (single line/column) gradient operators (Eqns. (6.6) and (6.7)). The Prewitt operator uses the filter kernels

$$H_x^{\mathrm{P}} = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^{\mathrm{P}} = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}, \tag{6.8}$$

which compute the average gradient components across three neighboring lines or columns, respectively. When the filters are written in separated form,

$$H_x^{\mathrm{P}} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^{\mathrm{P}} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} * \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \tag{6.9}$$

respectively, it becomes obvious that H_x^P performs a simple (box) smoothing over three lines before computing the x gradient (Eqn. (6.6)), and analogously H_y^P smooths over three columns before computing the y gradient (Eqn. (6.7)).[2] Because of the commutativity property of linear convolution, this could equally be described the other way around, with smoothing being applied *after* the computation of the gradients.

The filters for the Sobel operator are almost identical; however, the smoothing part assigns higher weight to the current center line and column, respectively:

$$H_x^{\mathrm{S}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad \text{and} \quad H_y^{\mathrm{S}} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}. \tag{6.10}$$

The estimates for the local gradient components are obtained from the filter results by appropriate scaling, that is,

$$\nabla I(u,v) \approx \frac{1}{6} \cdot \begin{pmatrix} (I * H_x^{\mathrm{P}})(u,v) \\ (I * H_y^{\mathrm{P}})(u,v) \end{pmatrix} \tag{6.11}$$

for the *Prewitt* operator and

$$\nabla I(u,v) \approx \frac{1}{8} \cdot \begin{pmatrix} (I * H_x^{\mathrm{S}})(u,v) \\ (I * H_y^{\mathrm{S}})(u,v) \end{pmatrix} \tag{6.12}$$

for the *Sobel* operator.

[2] In Eqn. (6.9), $*$ is the linear convolution operator (see Sec. 5.3.1).

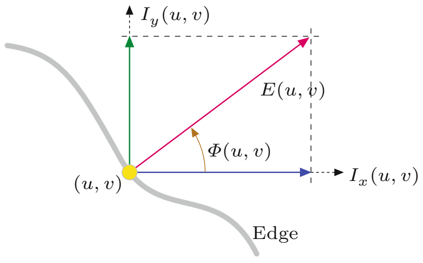


Fig. 6.5
Calculation of edge magnitude and orientation (geometry).

Edge strength and orientation

In the following, we denote the scaled filter results (obtained with either the Prewitt or Sobel operator) as

$$I_x = I * H_x \qquad \text{and} \qquad I_y = I * H_y.$$

In both cases, the local edge strength $E(u,v)$ is defined as the gradient magnitude

$$E(u,v) = \sqrt{I_x^2(u,v) + I_y^2(u,v)} \tag{6.13}$$

and the local edge orientation angle $\varPhi(u,v)$ is[3]

$$\varPhi(u,v) = \tan^{-1}\left(\frac{I_y(u,v)}{I_x(u,v)}\right) = \mathrm{ArcTan}\big(I_x(u,v), I_y(u,v)\big), \tag{6.14}$$

as illustrated in Fig. 6.5.

The whole process of extracting the edge magnitude and orientation is summarized in Fig. 6.6. First, the original image I is independently convolved with the two gradient filters H_x and H_y, and subsequently the edge strength E and orientation $\varPhi$ are computed from the filter results. Figure 6.7 shows the edge strength and orientation for two test images, obtained with the Sobel filters in Eqn. (6.10).

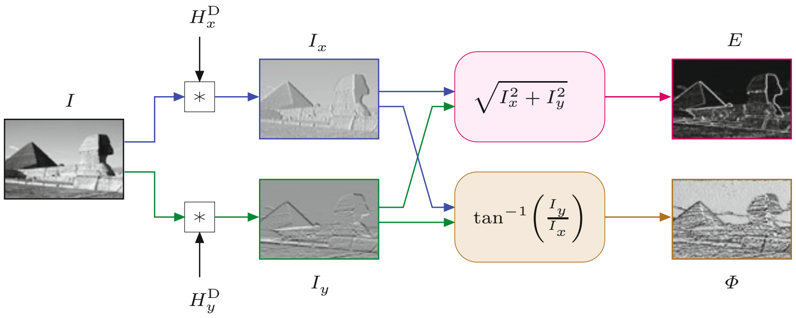


Fig. 6.6
Typical process of gradient-based edge extraction. The linear derivative filters H_x^{D} and H_y^{D} produce two gradient images, I_x and I_y, respectively. They are used to compute the edge strength E and orientation $\varPhi$ for each image position (u,v).

The estimate of the edge orientation based on the original Prewitt and Sobel filters is relatively inaccurate, and improved versions of the Sobel filters were proposed in [126, p. 353] to minimize the orientation errors:

[3] See the hints in Sec. F.1.6 in the Appendix for computing the inverse tangent $\tan^{-1}(y/x)$ with the $\mathrm{ArcTan}(x,y)$ function.

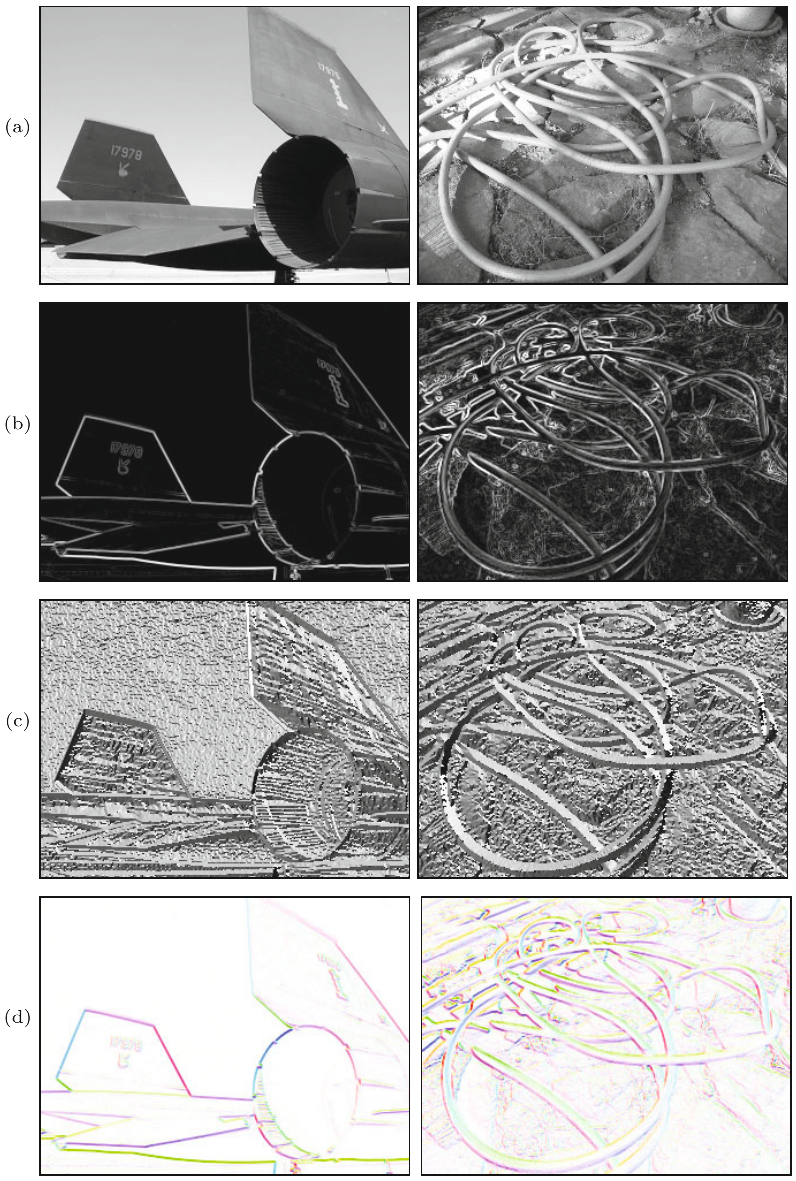


Fig. 6.7
Edge strength and orientation obtained with a Sobel operator. Original images (a), the edge strength $E(u,v)$ (b), and the local edge orientation $\Phi(u,v)$ (c). The images in (d) show the orientation angles coded as color hues, with the edge strength controlling the color saturation (see Sec. 12.2.3 for the corresponding definitions).

$$H_x^{S'} = \frac{1}{32} \cdot \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix} \quad \text{and} \quad H_y^{S'} = \frac{1}{32} \cdot \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}. \qquad (6.15)$$

These edge operators are frequently used because of their good results (see also Fig. 6.11) and simple implementation. The Sobel operator, in particular, is available in many image-processing tools and software packages (including ImageJ).

6.3.2 Roberts Operator

As one of the simplest and oldest edge finders, the Roberts operator [199] today is mainly of historic interest. It employs two extremely small filters of size 2×2 for estimating the directional gradient along

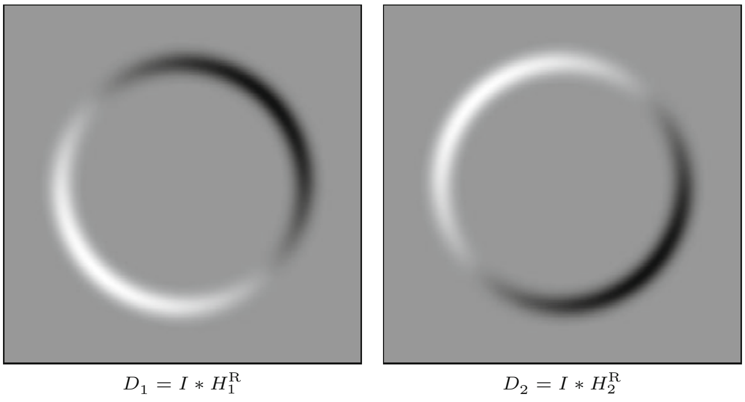


Fig. 6.8
Diagonal gradient components produced by the two Roberts filters.

the image diagonals:

$$H_1^{\mathrm{R}} = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \quad \text{and} \quad H_2^{\mathrm{R}} = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}. \tag{6.16}$$

These filters naturally respond to diagonal edges but are not highly selective to orientation; that is, both filters show strong results over a relatively wide range of angles (Fig. 6.8). The local edge strength is calculated by measuring the length of the resulting 2D vector, similar to the gradient computation but with its components rotated 45° (Fig. 6.9).

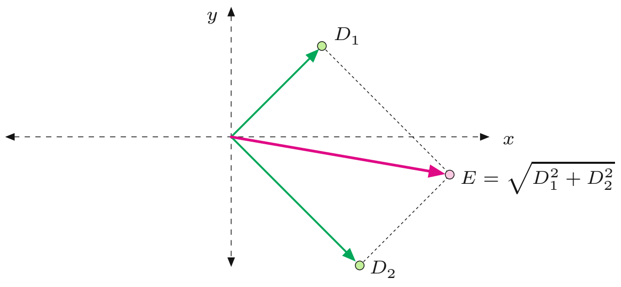


Fig. 6.9
Definition of edge strength for the Roberts operator. The edge strength $E(u, v)$ corresponds to the length of the vector obtained by adding the two orthogonal gradient components (filter results) $D_1(u, v)$ and $D_2(u, v)$.

6.3.3 Compass Operators

The design of linear edge filters involves a trade-off: the stronger a filter responds to edge-like structures, the more sensitive it is to orientation. In other words, filters that are orientation insensitive tend to respond to nonedge structures, while the most discriminating edge filters only respond to edges in a narrow range of orientations. One solution is to use not only a single pair of relatively "wide" filters for two directions (such as the Prewitt and the simple Sobel operator discussed in Sec. 6.3.1) but a larger set of filters with narrowly spaced orientations.

Extended Sobel operator

Classic examples are the edge operator proposed by *Kirsch* [136] and the "extended Sobel" or *Robinson* operator [200], which employs the following eight filters with orientations spaced at 45°:

$$H_0^{\mathrm{ES}} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad H_1^{\mathrm{ES}} = \begin{bmatrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{bmatrix}, \tag{6.17}$$

$$H_2^{\mathrm{ES}} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \qquad H_3^{\mathrm{ES}} = \begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}, \tag{6.18}$$

$$H_4^{\mathrm{ES}} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, \qquad H_5^{\mathrm{ES}} = \begin{bmatrix} 2 & 1 & 0 \\ 1 & 0 & -1 \\ 0 & -1 & -2 \end{bmatrix}, \tag{6.19}$$

$$H_6^{\mathrm{ES}} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}, \qquad H_7^{\mathrm{ES}} = \begin{bmatrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{bmatrix}. \tag{6.20}$$

Only the results of four of these eight filters $(H_0^{\mathrm{ES}}, H_1^{\mathrm{ES}}, \ldots, H_7^{\mathrm{ES}})$ must actually be computed since the remaining four are identical except for the reversed sign. For example, from the fact that $H_4^{\mathrm{ES}} = -H_0^{\mathrm{ES}}$ and the convolution being linear (Eqn. (5.22)), it follows that

$$I * H_4^{\mathrm{ES}} = I * -H_0^{\mathrm{ES}} = -(I * H_0^{\mathrm{ES}}), \tag{6.21}$$

that is, the result for filter H_4^S is simply the negative result for filter H_0^S. The directional outputs $D_0, D_1, \ldots D_7$ for the eight Sobel filters can thus be computed as follows:

$$\begin{array}{llll} D_0 \leftarrow I * H_0^{\mathrm{ES}}, & D_1 \leftarrow I * H_1^{\mathrm{ES}}, & D_2 \leftarrow I * H_2^{\mathrm{ES}}, & D_3 \leftarrow I * H_3^{\mathrm{ES}}, \\ D_4 \leftarrow -D_0, & D_5 \leftarrow -D_1, & D_6 \leftarrow -D_2, & D_7 \leftarrow -D_3. \end{array} \tag{6.22}$$

The edge strength E^S at position (u, v) is defined as the maximum of the eight filter outputs; that is,

$$\begin{aligned} E^{\mathrm{ES}}(u,v) &= \max\big(D_0(u,v), D_1(u,v), \ldots, D_7(u,v)\big) \\ &= \max\big(|D_0(u,v)|, |D_1(u,v)|, |D_2(u,v)|, |D_3(u,v)|\big), \end{aligned} \tag{6.23}$$

and the strongest-responding filter also determines the local edge orientation as

$$\Phi^{\mathrm{ES}}(u,v) = \frac{\pi}{4} j, \quad \text{with } j = \underset{0 \le i \le 7}{\operatorname{argmax}} \; D_i(u,v). \tag{6.24}$$

Kirsch operator

Another classic compass operator is the one proposed by Kirsch [136], which also employs eight oriented filters with the following kernels:

$$H_0^{\mathrm{K}} = \begin{bmatrix} -5 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & 3 & 3 \end{bmatrix}, \qquad H_4^{\mathrm{K}} = \begin{bmatrix} 3 & 3 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & -5 \end{bmatrix}, \tag{6.25}$$

$$H_1^{\mathrm{K}} = \begin{bmatrix} -5 & -5 & 3 \\ -5 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}, \qquad H_5^{\mathrm{K}} = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & -5 \\ 3 & -5 & -5 \end{bmatrix}, \tag{6.26}$$

$$H_2^{\mathrm{K}} = \begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}, \qquad H_6^{\mathrm{K}} = \begin{bmatrix} 3 & 3 & 3 \\ 3 & 0 & 3 \\ -5 & -5 & -5 \end{bmatrix}, \tag{6.27}$$

$$H_3^{\mathrm{K}} = \begin{bmatrix} 3 & -5 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & 3 \end{bmatrix}, \qquad H_7^{\mathrm{K}} = \begin{bmatrix} 3 & 3 & 3 \\ -5 & 0 & 3 \\ -5 & -5 & 3 \end{bmatrix}. \tag{6.28}$$

Again, because of the symmetries, only four of the eight filters need to be applied and the results may be combined in the same way as already described for the extended Sobel operator.

In practice, this and other "compass operators" show only minor benefits over the simpler operators described earlier, including the small advantage of not requiring the computation of square roots (which is considered a relatively "expensive" operation).

6.3.4 Edge Operators in ImageJ

The current version of ImageJ implements the Sobel operator (as described in Eqn. (6.10)) for practically any type of image. It can be invoked via the

Process ▷ Find Edges

menu and is also available through the method `void findEdges()` for objects of type `ImageProcessor`.

6.4 Other Edge Operators

One problem with edge operators based on first derivatives (as described in the previous section) is that each resulting edge is as wide as the underlying intensity transition and thus edges may be difficult to localize precisely. An alternative class of edge operators makes use of the second derivatives of the image function, including some popular modern edge operators that also address the problem of edges appearing at various levels of scale. These issues are briefly discussed in the following.

6.4.1 Edge Detection Based on Second Derivatives

The second derivative of a function measures its local curvature. The idea is that edges can be found at zero positions or—even better—at the zero crossings of the second derivatives of the image function, as illustrated in Fig. 6.10 for the 1D case. Since second derivatives generally tend to amplify image noise, some sort of presmoothing is usually applied with suitable low-pass filters.

A popular example is the "Laplacian-of-Gaussian" (LoG) operator [161], which combines gGussian smoothing and computing the second derivatives (see the *Laplace Filter* in Sec. 6.6.1) into a single linear filter. The example in Fig. 6.11 shows that the edges produced by the LoG operator are more precisely localized than the ones delivered by the Prewitt and Sobel operators, and the amount of "clutter" is comparably small. Details about the LoG operator and a comprehensive survey of common edge operators can be found in [203, Ch. 4] and [165].

6.4.2 Edges at Different Scales

Unfortunately, the results of the simple edge operators we have discussed so far often deviate from what we as humans perceive as important edges. The two main reasons for this are:

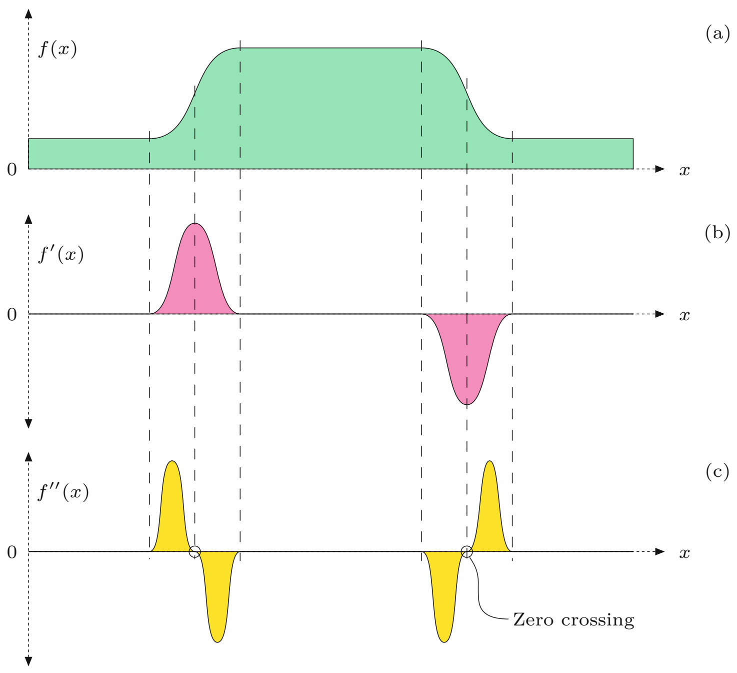


Fig. 6.10
Principle of edge detection with the second derivative: original function (a), first derivative (b), and second derivative (c). Edge points are located where the second derivative crosses through zero and the first derivative has a high magnitude.

- First, edge operators only respond to local intensity differences, while our visual system is able to extend edges across areas of minimal or vanishing contrast.
- Second, edges exist not at a single fixed resolution or at a certain scale but over a whole range of different scales.

Typical small edge operators, such as the Sobel operator, can only respond to intensity differences that occur within their 3×3 pixel filter regions. To recognize edge-like events over a greater horizon, we would either need larger edge operators (with correspondingly large filters) or to use the original (small) operators on reduced (i.e., scaled) images. This is the principal idea of "multiresolution" techniques (also referred to as "hierarchical" or "pyramid" techniques), which have traditionally been used in many image-processing applications [41, 151]. In the context of edge detection, this typically amounts to detecting edges at various scale levels first and then deciding which edge (if any) at which scale level is dominant at each image position.

6.4.3 From Edges to Contours

Whatever method is used for edge detection, the result is usually a continuous value for the edge strength for each image position and possibly also the angle of local edge orientation. How can this information be used, for example, to find larger image structures and contours of objects in particular?

Binary edge maps

In many situations, the next step after edge enhancement (by some edge operator) is the selection of edge points, a binary decision about whether an image pixel is an edge point or not. The simplest method is to apply a *threshold* operation to the edge strength delivered by the edge operator using either a fixed or adaptive threshold value, which results in a binary edge image or "edge map".

In practice, edge maps hardly ever contain perfect contours but instead many small, unconnected contour fragments, interrupted at positions of insufficient edge strength. After thresholding, the empty positions of course contain no edge information at all that could possibly be used in a subsequent step, such as for linking adjacent edge segments. Despite this weakness, global thresholding is often used at this point because of its simplicity, and some common postprocessing methods, such as the Hough transform (see Ch. 8), can cope well with incomplete edge maps.

Contour following

The idea of tracing contours sequentially along the discovered edge points is not uncommon and appears quite simple in principle. Starting from an image point with high edge strength, the edge is followed iteratively in both directions until the two traces meet and a closed contour is formed. Unfortunately, there are several obstacles that make this task more difficult than it seems at first, including the following:

- edges may end in regions of vanishing intensity gradient,
- crossing edges lead to ambiguities, and
- contours may branch into several directions.

Because of these problems, contour following usually is not applied to original images or continuous-valued edge images except in very simple situations, such as when there is a clear separation between objects (foreground) and the background. Tracing contours in segmented binary images is much simpler, of course (see Ch. 10).

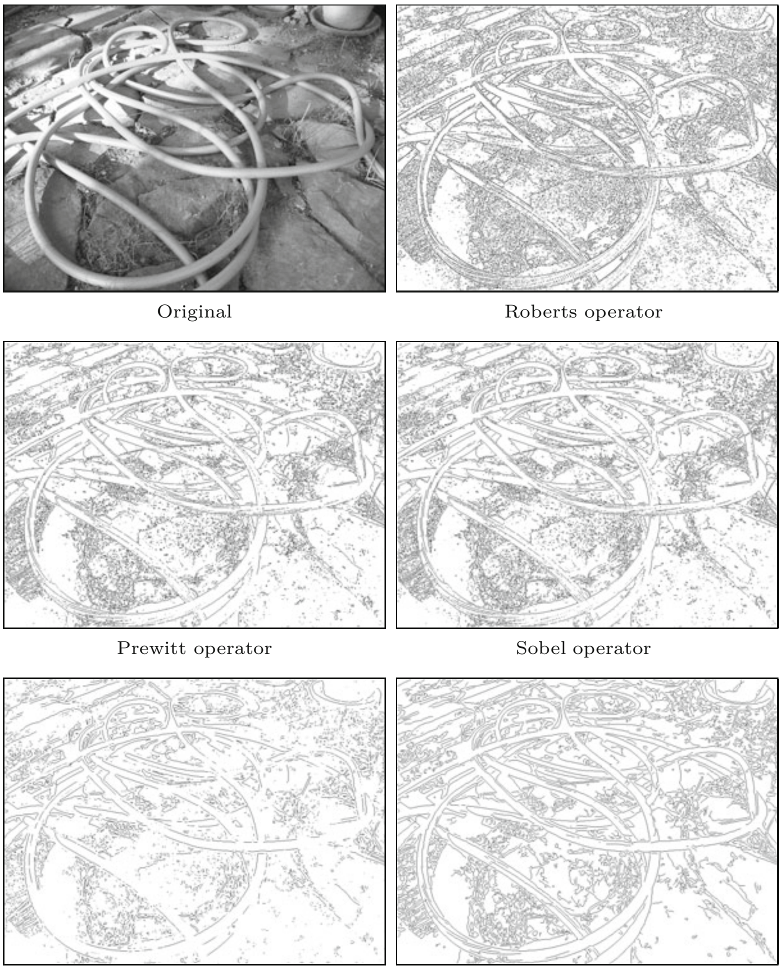
6.5 Canny Edge Operator

The operator proposed by Canny [42] is widely used and still considered "state of the art" in edge detection. The method tries to reach three main goals: (a) to minimize the number of false edge points, (b) achieve good localization of edges, and (c) deliver only a single mark on each edge. These properties are usually not achieved with simple edge operators (mostly based on first derivatives and subsequent thresholding).

At its core, the Canny "filter" is a gradient method (based on first derivatives; see Sec. 6.2), but it uses the zero crossings of second derivatives for precise edge localization.[4] In this regard, the method is similar to edge detectors that are based on the second derivatives of the image function [161].

Fully implemented, the Canny detector uses a set of relatively large, oriented filters at multiple image resolutions and merges the

[4] The zero crossings of a function's second derivative are found where the first derivates exhibit a local maximum or minimum.



Original — Roberts operator — Prewitt operator — Sobel operator — Laplacian of Gaussian — Canny operator ($\sigma = 1.0$)

Fig. 6.11
Comparison of various edge operators. Important criteria for the quality of edge results are the amount of "clutter" (irrelevant edge elements) and the connectedness of dominant edges. The Roberts operator responds to very small edge structures because of the small size of its filters. The similarity of the Prewitt and Sobel operators is manifested in the corresponding results. The edge map produced by the Canny operator is substantially cleaner than those of the simpler operators, even for a fixed and relatively small scale value σ.

individual results into a common *edge map*. It is quite common, however, to use only a single-scale implementation of the algorithm with an adjustable filter radius (smoothing parameter σ), which is nevertheless superior to most of the simple edge operators (see Fig. 6.11). In addition, the algorithm not only yields a binary edge map but connected chains of edge pixels, which greatly simplifies the subsequent processing steps. Thus, even in its basic (single-scale) form, the Canny operator is often preferred over other edge detection methods.

In its basic (single-scale) form, the Canny operator performs the following steps (stated more precisely in Algs. 6.1–6.2):

1. **Pre-processing:** Smooth the image with a Gaussian filter of width σ, which specifies the scale level of the edge detector. Calculate the x/y gradient vector at each position of the filtered image and determine the local gradient magnitude and orientation.
2. **Edge localization:** Isolate local maxima of gradient magnitude by "non-maximum suppression" along the local gradient direction.

3. **Edge tracing and hysteresis thresholding:** Collect sets of connected edge pixels from the local maxima by applying "hysteresis thresholding".

6.5.1 Pre-processing

The original intensity image I is first smoothed with a Gaussian filter kernel $H^{\mathrm{G},\sigma}$; its width σ specifies the spatial scale at which edges are to be detected (see Alg. 6.1, lines 2–10). Subsequently, first-order difference filters are applied to the smoothed image $\bar{I}$ to calculate the components $\bar{I}_x, \bar{I}_y$ of the local gradient vectors (Alg. 6.1, line 3–3).[5] Then the local magnitude E_{mag} is calculated as the norm of the corresponding gradient vector (Alg. 6.1, line 11). In view of the subsequent thresholding it may be helpful to normalize the edge magnitude values to a standard range (e.g., to $[0, 100]$).

6.5.2 Edge localization

Candidate edge pixels are isolated by local "non-maximum suppression" of the edge magnitude E_{mag}. In this step, only those pixels are preserved that represent a local maximum along the 1D profile in the direction of the gradient, that is, perpendicular to the edge tangent (see Fig. 6.12). While the gradient may point in any continuous direction, only *four* discrete directions are typically used to facilitate efficient processing. The pixel at position (u, v) is only retained as an edge candidate if its gradient magnitude is greater than both its immediate neighbors in the direction specified by the gradient vector (d_x, d_y) at position (u, v). If a pixel is not a local maximum, its edge magnitude value is set to zero (i.e., "suppressed"). In Alg. 6.1, the non-maximum suppressed edge values are stored in the map E_{nms}.

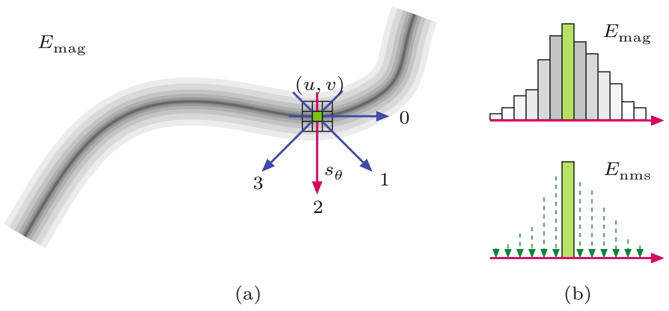


Fig. 6.12
Non-maximum suppression of gradient magnitude. The gradient direction at position (u, v) is coarsely quantized to four discrete orientations $s_\theta \in \{0, 1, 2, 3\}$ (a). Only pixels where the gradient magnitude $E_{\mathrm{mag}}(u, v)$ is a local maximum in the gradient direction (i.e., perpendicular to the edge tangent) are taken as candidate edge points (b). The gradient magnitude at all other points is set (suppressed) to zero.

The problem of finding the discrete orientation $s_\theta = 0, ..., 3$ for a given gradient vector $\boldsymbol{q} = (d_x, d_y)$ is illustrated in Fig. 6.13. This task is simple if the corresponding angle $\theta = \tan^{-1}(d_y/d_x)$ is known, but at this point the use of the trigonometric functions is typically avoided for efficiency reasons. The octant that corresponds to $\boldsymbol{q}$ can be inferred directly from the signs and magnitude of the components d_x, d_y, however, the necessary decision rules are quite complex. Much simpler rules apply if the coordinate system and gradient vector $\boldsymbol{q}$ are

[5] See also Sec. C.3.1 in the Appendix.

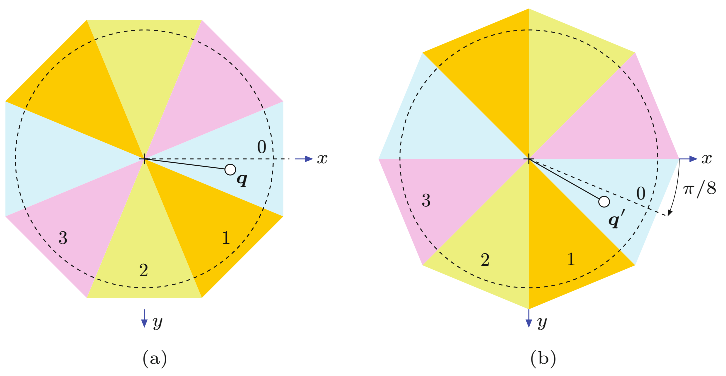


Fig. 6.13
Discrete gradient directions. In (a), calculating the octant for a given orientation vector $\boldsymbol{q} = (d_x, d_y)$ requires a relatively complex decision. Alternatively (b), if $\boldsymbol{q}$ is rotated by $\frac{\pi}{8}$ to $\boldsymbol{q}'$, the corresponding octant can be found directly from the components of $\boldsymbol{q}' = (d'_x, d'_y)$ without the need to calculate the actual angle. Orientation vectors in the other octants are mirrored to octants $s_\theta = 0, 1, 2, 3$.

Alg. 6.1
Canny edge detector for grayscale images.

```
1:  CannyEdgeDetector($I, \sigma, \mathsf{t}_{\mathrm{hi}}, \mathsf{t}_{\mathrm{lo}}$)
        Input: $I$, a grayscale image of size $M \times N$; $\sigma$, scale (radius of
        Gaussian filter $H^{\mathrm{G},\sigma}$); $\mathsf{t}_{\mathrm{hi}}, \mathsf{t}_{\mathrm{lo}}$, hysteresis thresholds ($\mathsf{t}_{\mathrm{hi}} > \mathsf{t}_{\mathrm{lo}}$).
        Returns a binary edge map of size $M \times N$.

2:      $\bar{I} \leftarrow I * H^{\mathrm{G},\sigma}$                ▷ blur with Gaussian of width $\sigma$
3:      $\bar{I}_x \leftarrow \bar{I} * [-0.5\ \ \mathbf{0}\ \ 0.5]$                      ▷ $x$-gradient
4:      $\bar{I}_y \leftarrow \bar{I} * [-0.5\ \ \mathbf{0}\ \ 0.5]^\intercal$                      ▷ $y$-gradient
5:      $(M, N) \leftarrow$ Size($\boldsymbol{I}$)
6:      Create maps:
7:          $E_{\mathrm{mag}} : M \times N \mapsto \mathbb{R}$               ▷ gradient magnitude
8:          $E_{\mathrm{nms}} : M \times N \mapsto \mathbb{R}$               ▷ maximum magnitude
9:          $E_{\mathrm{bin}} : M \times N \mapsto \{0,1\}$              ▷ binary edge pixels
10:     for all image coordinates $(u,v) \in M \times N$ do
11:         $E_{\mathrm{mag}}(u,v) \leftarrow \big[\bar{I}_x^2(u,v) + \bar{I}_y^2(u,v)\big]^{1/2}$
12:         $E_{\mathrm{nms}}(u,v) \leftarrow 0$
13:         $E_{\mathrm{bin}}(u,v) \leftarrow 0$
14:     for $u \leftarrow 1,\ldots,M-2$ do
15:         for $v \leftarrow 1,\ldots,N-2$ do
16:             $d_x \leftarrow \bar{I}_x(u,v)$,    $d_y \leftarrow \bar{I}_y(u,v)$
17:             $s_\theta \leftarrow$ GetOrientationSector($d_x, d_y$)        ▷ Alg. 6.2
18:             if IsLocalMax($E_{\mathrm{mag}}, u, v, s_\theta, \mathsf{t}_{\mathrm{lo}}$) then          ▷ Alg. 6.2
19:                 $E_{\mathrm{nms}}(u,v) \leftarrow E_{\mathrm{mag}}(u,v)$    ▷ only keep local maxima
20:     for $u \leftarrow 1,\ldots,M-2$ do
21:         for $v \leftarrow 1,\ldots,N-2$ do
22:             if $(E_{\mathrm{nms}}(u,v) \geq \mathsf{t}_{\mathrm{hi}}) \wedge (E_{\mathrm{bin}}(u,v) = 0)$ then
23:                 TraceAndThreshold($E_{\mathrm{nms}}, E_{\mathrm{bin}}, u, v, \mathsf{t}_{\mathrm{lo}}$)
                                                                 ▷ Alg. 6.2
24:     return $E_{\mathrm{bin}}$.
```

rotated by $\frac{\pi}{8}$, as illustrated in Fig. 6.13(b). This step is implemented by the function GetOrientationSector() in Alg. 6.2.[6]

6.5.3 Edge tracing and hysteresis thresholding

In the final step, sets of connected edge points are collected from the magnitude values that remained unsuppressed in the previous oper-

[6] Note that the elements of the rotation matrix in Alg. 6.2 (line 2) are constants and thus no repeated use of trigonometric functions is required.

Alg. 6.2
Procedures used in Alg. 6.1 (Canny edge detector).

```
1: GetOrientationSector($d_x, d_y$)
       Returns the discrete octant $s_\theta$ for the orientation vector $(d_x, d_y)^\intercal$.
       See Fig. 6.13 for an illustration.
2:     $\begin{pmatrix} d'_x \\ d'_y \end{pmatrix} \leftarrow \begin{pmatrix} \cos(\pi/8) & -\sin(\pi/8) \\ \sin(\pi/8) & \cos(\pi/8) \end{pmatrix} \cdot \begin{pmatrix} d_x \\ d_y \end{pmatrix}$    ▷ rotate $\begin{pmatrix} d_x \\ d_y \end{pmatrix}$ by $\pi/8$
3:     if $d'_y < 0$ then
4:         $d'_x \leftarrow -d'_x$,    $d'_y \leftarrow -d'_y$    ▷ mirror to octants $0, \ldots, 3$
5:     $s_\theta \leftarrow \begin{cases} 0 & \text{if } (d'_x \geq 0) \wedge (d'_x \geq d'_y) \\ 1 & \text{if } (d'_x \geq 0) \wedge (d'_x < d'_y) \\ 2 & \text{if } (d'_x < 0) \wedge (-d'_x < d'_y) \\ 3 & \text{if } (d'_x < 0) \wedge (-d'_x \geq d'_y) \end{cases}$
6:     return $s_\theta$.    ▷ sector index $s_\theta \in \{0, 1, 2, 3\}$

7: IsLocalMax($E_{\mathrm{mag}}, u, v, s_\theta, \mathsf{t}_{\mathrm{lo}}$)
       Determines if the gradient magnitude $E_{\mathrm{mag}}$ is a local maximum
       at position $(u, v)$ in direction $s_\theta \in \{0, 1, 2, 3\}$.
8:     $m_{\mathrm{C}} \leftarrow E_{\mathrm{mag}}(u, v)$
9:     if $m_{\mathrm{C}} < \mathsf{t}_{\mathrm{lo}}$ then
10:        return false
11:    else
12:        $m_{\mathrm{L}} \leftarrow \begin{cases} E_{\mathrm{mag}}(u-1, v) & \text{if } s_\theta = 0 \\ E_{\mathrm{mag}}(u-1, v-1) & \text{if } s_\theta = 1 \\ E_{\mathrm{mag}}(u, v-1) & \text{if } s_\theta = 2 \\ E_{\mathrm{mag}}(u-1, v+1) & \text{if } s_\theta = 3 \end{cases}$
13:        $m_{\mathrm{R}} \leftarrow \begin{cases} E_{\mathrm{mag}}(u+1, v) & \text{if } s_\theta = 0 \\ E_{\mathrm{mag}}(u+1, v+1) & \text{if } s_\theta = 1 \\ E_{\mathrm{mag}}(u, v+1) & \text{if } s_\theta = 2 \\ E_{\mathrm{mag}}(u+1, v-1) & \text{if } s_\theta = 3 \end{cases}$
14:        return $(m_{\mathrm{L}} \leq m_{\mathrm{C}}) \wedge (m_{\mathrm{C}} \geq m_{\mathrm{R}})$.

15: TraceAndThreshold($E_{\mathrm{nms}}, E_{\mathrm{bin}}, u_0, v_0, \mathsf{t}_{\mathrm{lo}}$)
       Recursively collects and marks all pixels of an edge that are 8-
       connected to $(u_0, v_0)$ and have a gradient magnitude above $\mathsf{t}_{\mathrm{lo}}$.
16:    $E_{\mathrm{bin}}(u_0, v_0) \leftarrow 1$    ▷ mark $(u_0, v_0)$ as an edge pixel
17:    $u_{\mathrm{L}} \leftarrow \max(u_0 - 1, 0)$    ▷ limit to image bounds
18:    $u_{\mathrm{R}} \leftarrow \min(u_0 + 1, M - 1)$
19:    $v_{\mathrm{T}} \leftarrow \max(v_0 - 1, 0)$
20:    $v_{\mathrm{B}} \leftarrow \min(v_0 + 1, N - 1)$
21:    for $u \leftarrow u_{\mathrm{L}}, \ldots, u_{\mathrm{R}}$ do
22:        for $v \leftarrow v_{\mathrm{T}}, \ldots, v_{\mathrm{B}}$ do
23:            if $(E_{\mathrm{nms}}(u, v) \geq \mathsf{t}_{\mathrm{lo}}) \wedge (E_{\mathrm{bin}}(u, v) = 0)$ then
24:                TraceAndThreshold($E_{\mathrm{nms}}, E_{\mathrm{bin}}, u, v, \mathsf{t}_{\mathrm{lo}}$)
25:    return
```

ation. This is done with a technique called "hysteresis thresholding" using two different threshold values , t_{lo} (with $\mathsf{t}_{\mathrm{hi}} > \mathsf{t}_{\mathrm{lo}}$). The image is scanned for pixels with edge magnitude $E_{\mathrm{nms}}(u, v) \geq \mathsf{t}_{\mathrm{hi}}$. Whenever such a (previously unvisited) location is found, a new *edge trace* is started and all connected edge pixels (u', v') are added to it as long as $E_{\mathrm{nms}}(u', v') \geq \mathsf{t}_{\mathrm{lo}}$. Only those edge traces remain that contain at least one pixel with edge magnitude greater than t_{hi} and no pixels with edge magnitude less than t_{lo}. This process (which is similar to

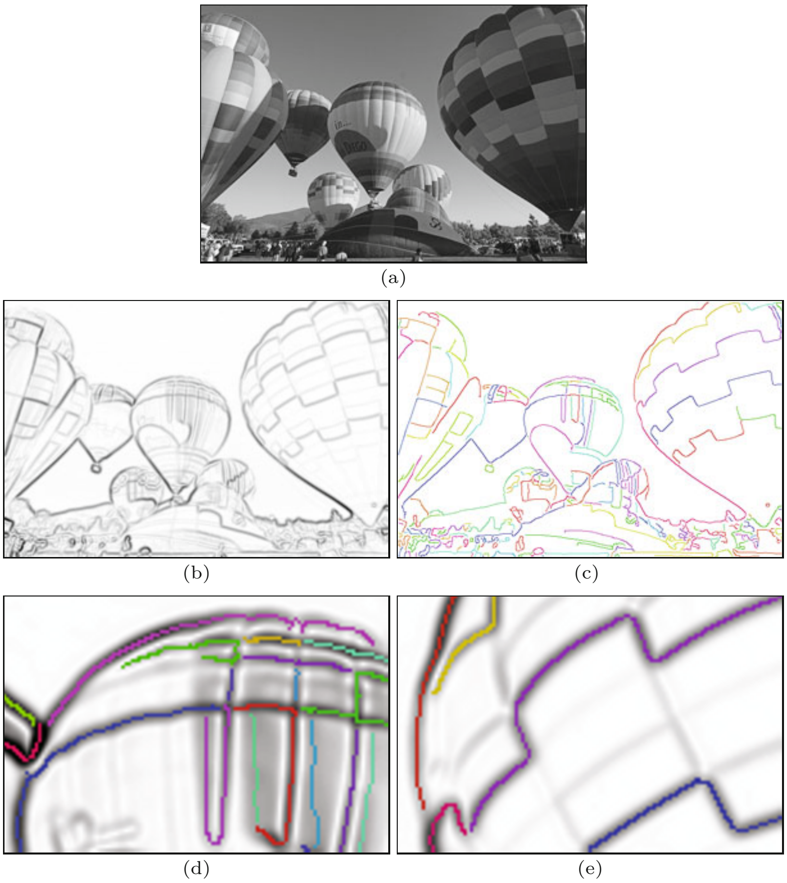


Fig. 6.14
Grayscale Canny edge operator details. Inverted gradient magnitude (a), detected edge points with connected edge tracks shown in distinctive colors (b). Details with gradient magnitude and detected edge points overlaid (c, d). Settings: $\sigma = 2.0$, $\mathsf{t}_{\mathrm{hi}} = 20\%$, $\mathsf{t}_{\mathrm{lo}} = 5\%$ (of the max. edge magnitude).

flood-fill region growing) is detailed in procedure GetOrientationSector in Alg. 6.2. Typical threshold values for 8-bit grayscale images are $\mathsf{t}_{\mathrm{hi}} = 5.0$ and $\mathsf{t}_{\mathrm{lo}} = 2.5$.

Figure 6.14 illustrates the effectiveness of non-maximum suppression for localizing the edge centers and edge-linking with hysteresis thresholding. Results from the single-scale Canny detector are shown in Fig. 6.15 for different settings of σ and fixed upper/lower threshold values $\mathsf{t}_{\mathrm{hi}} = 20\%$, $\mathsf{t}_{\mathrm{lo}} = 5\%$ (relative to the maximum gradient magnitude).

6.5.4 Additional Information

Due to the long-lasting popularity of the Canny operator, additional descriptions and some excellent illustrations can be found at various places in the literature, including [89, p. 719], [232, pp. 71–80], and [166, pp. 548–549]. An edge operator similar to the Canny detector, but based on a set of recursive filters, is described in [62]. While the Canny detector was originally designed for grayscale images, modified versions for color images exist, including the one we describe in the next section.

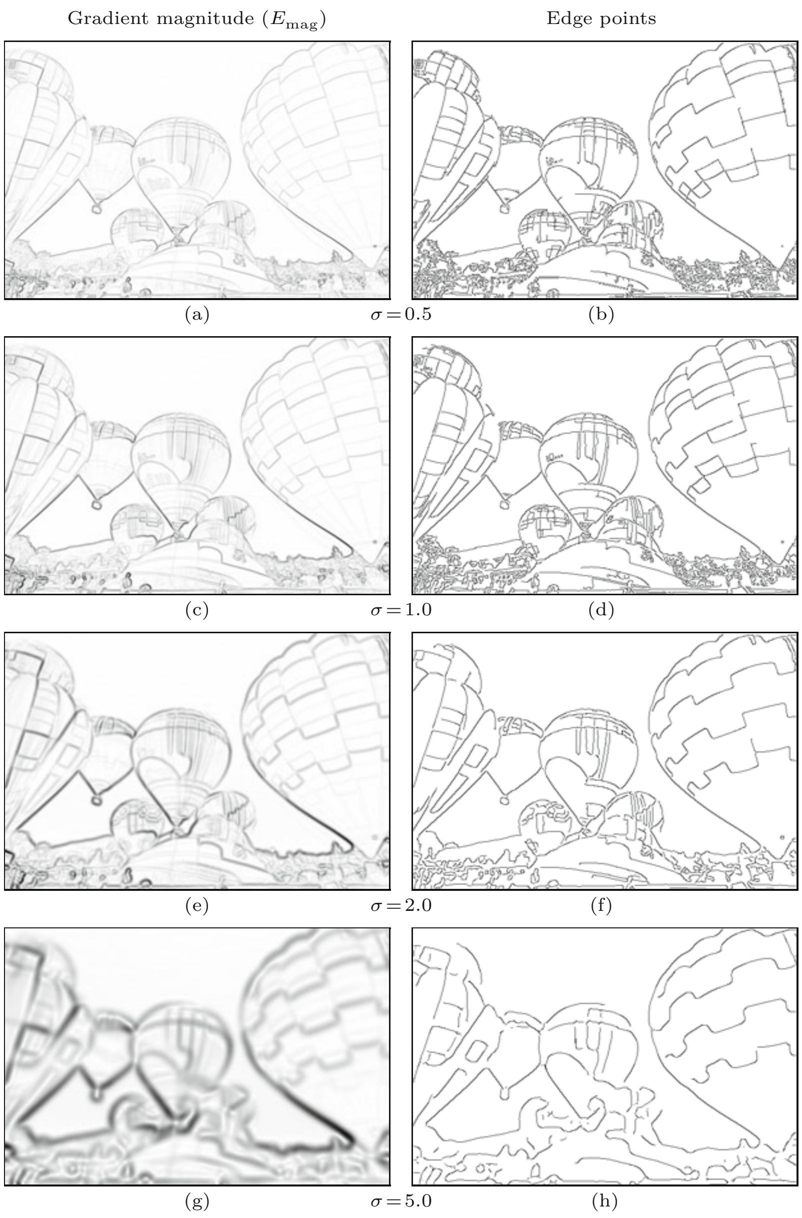


Fig. 6.15
Results from the single-scale grayscale Canny edge operator (Algs. 6.1–6.2) for different values of $\sigma = 0.5, \ldots, 5.0$. Inverted gradient magnitude (left column) and detected edge points (right column). The detected edge points (right column) are linked to connected edge chains.

6.5.5 Implementation

A complete implementation of the Canny edge detector for both grayscale and RGB color images can be found in the Java library for this book.[7] A basic usage example Prog. 16.1 is shown in Prog. 16.1 on p. 411.

[7] Class `CannyEdgeDetector` in package `imagingbook.pub.coloredge`.

6.6 Edge Sharpening

Making images look sharper is a frequent task, such as to make up for a lack of sharpness after scanning or scaling an image or to pre-compensate for a subsequent loss of sharpness in the course of printing or displaying an image. A common approach to image sharpening is to amplify the high-frequency image components, which are mainly responsible for the perceived sharpness of an image and for which the strongest occur at rapid intensity transitions. In the following, we describe two methods for artificial image sharpening that are based on techniques similar to edge detection and thus fit well in this chapter. In the following, we describe two methods for artificial image sharpening that are based on techniques similar to edge detection and thus fit well in this chapter.

6.6.1 Edge Sharpening with the Laplacian Filter

A common method for localizing rapid intensity changes are filters based on the second derivatives of the image function. Figure 6.16 illustrates this idea on a 1D, continuous function $f(x)$. The second derivative $f''(x)$ of the step function shows a positive pulse at the lower end of the transition and a negative pulse at the upper end. The edge is sharpened by subtracting a certain fraction w of the second derivative $f''(x)$ from the original function $f(x)$,

$$\hat{f}(x) = f(x) - w \cdot f''(x)\,. \tag{6.29}$$

Depending upon the weight factor $w \geq 0$, the expression in Eqn. (6.29) causes the intensity function to overshoot at both sides of an edge, thus exaggerating edges and increasing the perceived sharpness.

Laplacian operator

Sharpening of a 2D function can be accomplished with the second derivatives in the horizontal and vertical directions combined by the so-called Laplacian operator. The Laplacian operator ∇^2 of a 2D function $f(x,y)$ is defined as the sum of the second partial derivatives along the x and y directions:

$$(\nabla^2 f)(x,y) = \frac{\partial^2 f}{\partial^2 x}(x,y) + \frac{\partial^2 f}{\partial^2 y}(x,y). \tag{6.30}$$

Similar to the first derivatives (see Sec. 6.2.2), the second derivatives of a discrete image function can also be estimated with a set of simple linear filters. Again, several versions, have been proposed. For example, the two 1D filters

$$\frac{\partial^2 f}{\partial^2 x} \approx H_x^{\mathrm{L}} = \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} \quad \text{and} \quad \frac{\partial^2 f}{\partial^2 y} \approx H_y^{\mathrm{L}} = \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}, \tag{6.31}$$

for estimating the second derivatives along the x and y directions, respectively, combine to make the 2D Laplacian filter

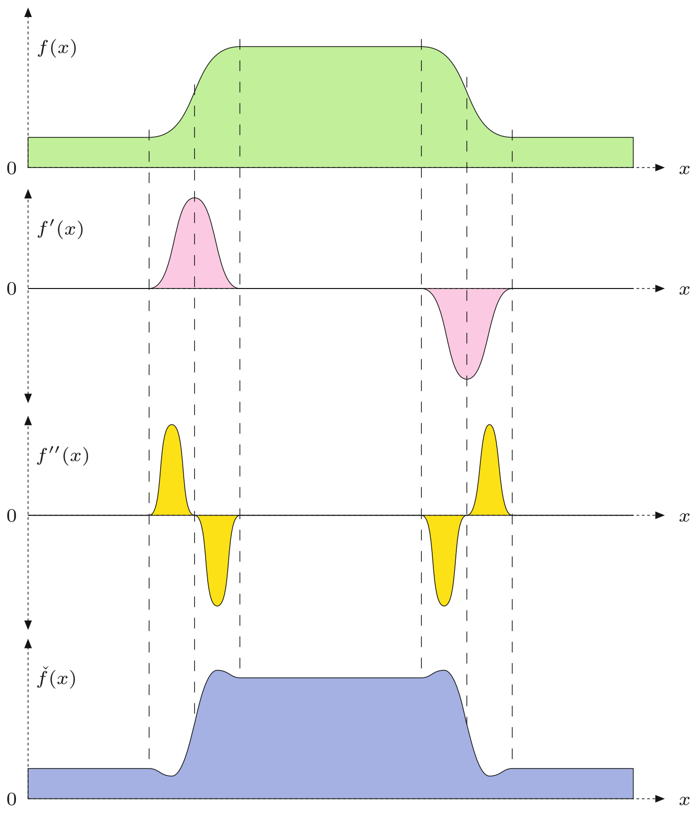


Fig. 6.16
Edge sharpening with the second derivative. The original intensity function $f(x)$, first derivative $f'(x)$, second derivative $f''(x)$, and sharpened intensity function $\hat{f}(x) = f(x) - w \cdot f''(x)$ are shown (w is a weighting factor).

$$H^{\mathrm{L}} = H_x^{\mathrm{L}} + H_y^{\mathrm{L}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}. \tag{6.32}$$

Figure 6.17 shows an example of applying the Laplacian filter H^{L} to a grayscale image, where the pairs of positive-negative peaks at both sides of each edge are clearly visible. The filter appears almost isotropic despite the coarse approximation with the small filter kernels.

Notice that H^{L} in Eqn. (6.32) is not *separable* in the usual sense (as described in Sec. 5.3.3) but, because of the linearity property of convolution (Eqns. (5.21) and (5.23)), it can be expressed (and computed) as the *sum* of two 1D filters,

$$I * H^{\mathrm{L}} = I * (H_x^{\mathrm{L}} + H_y^{\mathrm{L}}) = (I * H_x^{\mathrm{L}}) + (I * H_y^{\mathrm{L}}) = I_{xx} + I_{yy}. \tag{6.33}$$

Analogous to the gradient filters (for estimating the first derivatives), the sum of the coefficients is zero in any Laplace filter, such that its response is zero in areas of constant (flat) intensity (Fig. 6.17). Other common variants of 3×3 pixel Laplace filters are

$$H_8^{\mathrm{L}} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \text{oder} \quad H_{12}^{\mathrm{L}} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & -12 & 2 \\ 1 & 2 & 1 \end{bmatrix}. \tag{6.34}$$

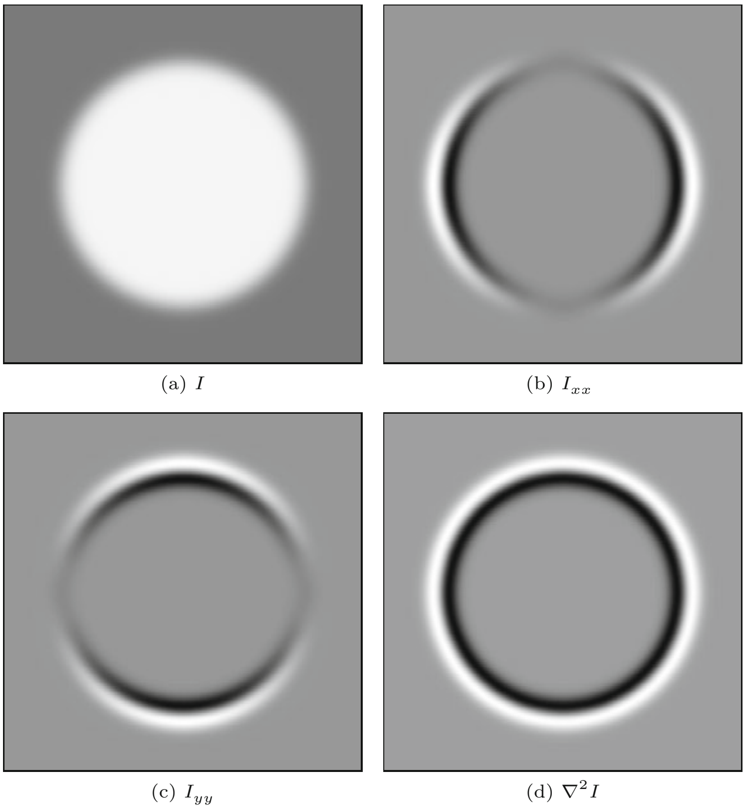


Fig. 6.17
Results of Laplace filter H^{L}: synthetic test image I (a), second partial derivative $I_{xx} = \partial^2 I/\partial^2 x$ in the horizontal direction (b), second partial derivative $I_{yy} = \partial^2 I/\partial^2 y$ in the vertical direction (c), and Laplace filter $\nabla^2 I = I_{xx} + I_{yy}$ (d). Intensities in (b–d) are scaled such that maximally negative and positive values are shown as black and white, respectively, and zero values are gray.

Sharpening

To perform the actual sharpening, as described by Eqn. (6.29) for the 1D case, we first apply a Laplacian filter H^{L} to the image I and then subtract a fraction of the result from the original image,

$$I' \leftarrow I - w \cdot (H^{\mathrm{L}} * I). \tag{6.35}$$

The factor w specifies the proportion of the Laplacian component and thus the sharpening strength. The proper choice of w also depends on the specific Laplacian filter used in Eqn. (6.35) since none of the aforementioned filters is normalized.

Figure 6.17 shows the result of applying a Laplacian filter (with the kernel given in Eqn. (6.32)) to a synthetic test image where the pairs of positive/negative peaks at both sides of each edge are clearly visible. The filter appears almost isotropic despite the coarse approximation with the small filter kernels. The application to a real grayscale image using the filter H^{L} (Eqn. (6.32)) and $w = 1.0$ is shown in Fig. 6.18.

As we can expect from second-order derivatives, the Laplacian filter is fairly sensitive to image noise, which can be reduced (as is commonly done in edge detection with first derivatives) by previous smoothing, such as with a Gaussian filter (see also Sec. 6.4.1).

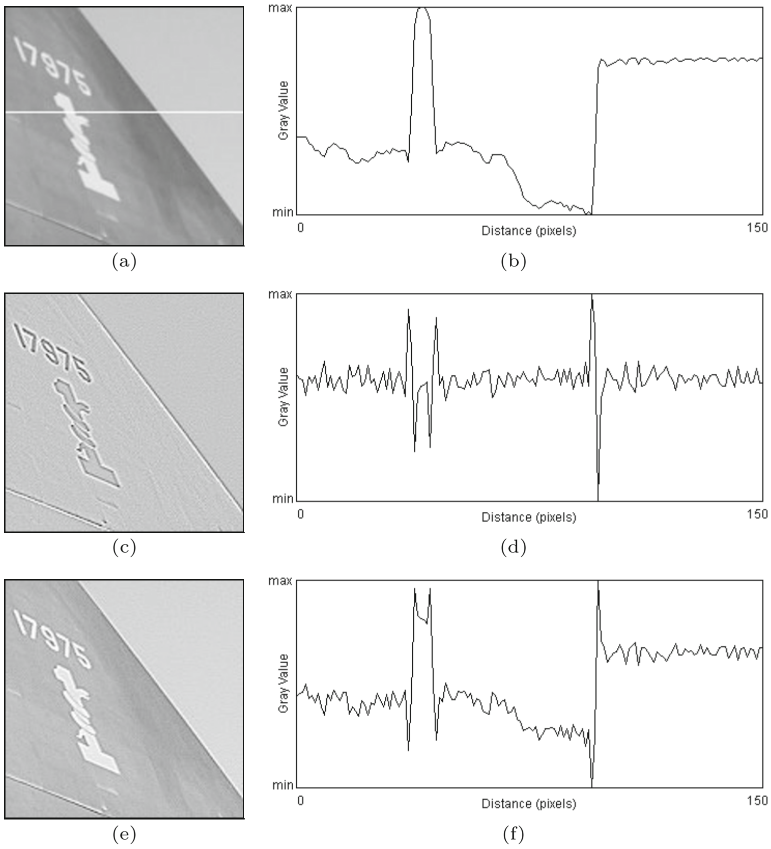


Fig. 6.18
Edge sharpening with the Laplacian filter. Original image with a horizontal profile taken from the marked line (a, b), result of Laplacian filter H^{L} (c, d), and sharpened image with sharpening factor $w = 1.0$ (e, f).

6.6.2 Unsharp Masking

"Unsharp masking" (USM) is a technique for edge sharpening that is particularly popular in astronomy, digital printing, and many other areas of image processing. The term originates from classical photography, where the sharpness of an image was optically enhanced by combining it with a smoothed ("unsharp") copy. This process is in principle the same for digital images.

Process

The first step in the USM filter is to subtract a smoothed version of the image from the original, which enhances the edges. The result is called the "mask". In analog photography, the required smoothing was achieved by simply defocusing the lens. Subsequently, the mask is again added to the original, such that the edges in the image are sharpened. In summary, the steps involved in USM filtering are:

1. The mask image M is generated by subtracting (from the original image I) a smoothed version of I, obtained by filtering with $\tilde{H}$, that is,
$$M \leftarrow I - (I * \tilde{H}) = I - \tilde{I}. \tag{6.36}$$
The kernel $\tilde{H}$ of the smoothing filter is assumed to be normalized (see Sec. 5.2.5).

2. To obtain the sharpened image $\check{I}$, the mask M is added to the original image I, weighted by the factor a, which controls the amount of sharpening,

$$\check{I} \leftarrow I + a \cdot M, \tag{6.37}$$

and thus (by inserting from Eqn. (6.36))

$$\check{I} \leftarrow I + a \cdot (I - \tilde{I}) \;=\; (1+a) \cdot I - a \cdot \tilde{I}. \tag{6.38}$$

Smoothing filter

In principle, any smoothing filter could be used for the kernel $\tilde{H}$ in Eqn. (6.36), but Gaussian filters $H^{\mathrm{G},\sigma}$with variable radius σ are most common (see also Sec. 5.2.7). Typical parameter values are 1 to 20 for σ and 0.2 to 4.0 (equivalent to 20% to 400%) for the sharpening factor a.

Figure 6.19 shows two examples of USM filters using Gaussian smoothing filters with different radii σ.

Extensions

The advantages of the USM filter over the Laplace filter are reduced noise sensitivity due to the involved smoothing and improved controllability through the parameters σ (spatial extent) and a (sharpening strength).

Of course the USM filter responds not only to real edges but to some extent to any intensity transition, and thus potentially increases any visible noise in continuous image regions. Some implementations (e.g., Adobe Photoshop) therefore provide an additional *threshold* parameter t_c to specify the *minimum local contrast* required to perform edge sharpening. Sharpening is only applied if the local contrast at position (u, v), expressed, for example, by the gradient magnitude $|\nabla I|$ (Eqn. (6.5)), is greater than that threshold. Otherwise, that pixel remains unmodified, that is,

$$\check{I}(u,v) \leftarrow \begin{cases} I(u,v) + a \cdot M(u,v) & \text{for } |\nabla I|(u,v) \geq t_c, \\ I(u,v) & \text{otherwise.} \end{cases} \tag{6.39}$$

Different to the original USM filter (Eqn. (6.37)), this extended version is no longer a *linear* filter. On color images, the USM filter is usually applied to all color channels with identical parameter settings.

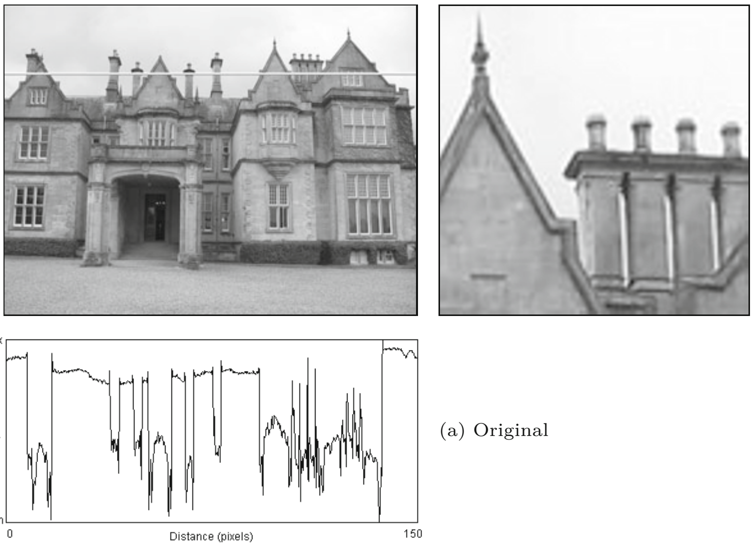
Implementation

The USM filter is available in virtually any image-processing software and, due to its simplicity and flexibility, has become an indispensable tool for many professional users. In ImageJ, the USM filter is implemented by the plugin class `UnsharpMask`[8] and can be applied through the menu

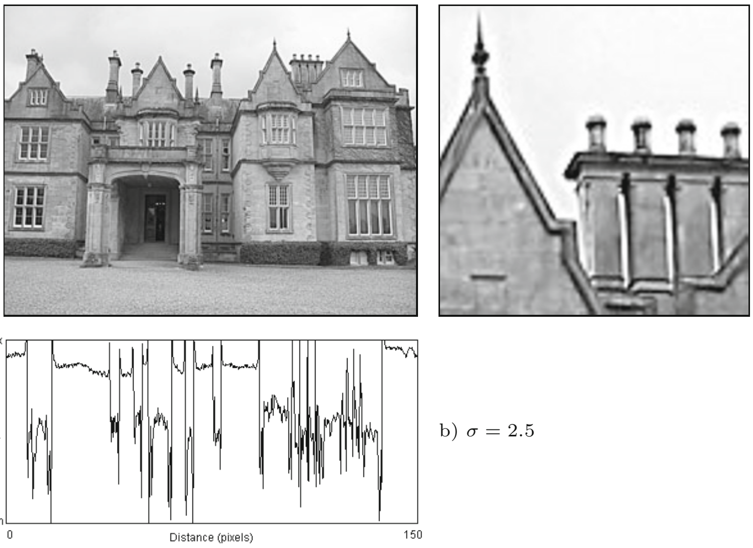
Process ▷ Filter ▷ Unsharp Mask...

[8] In package `ij.plugin.filter`.

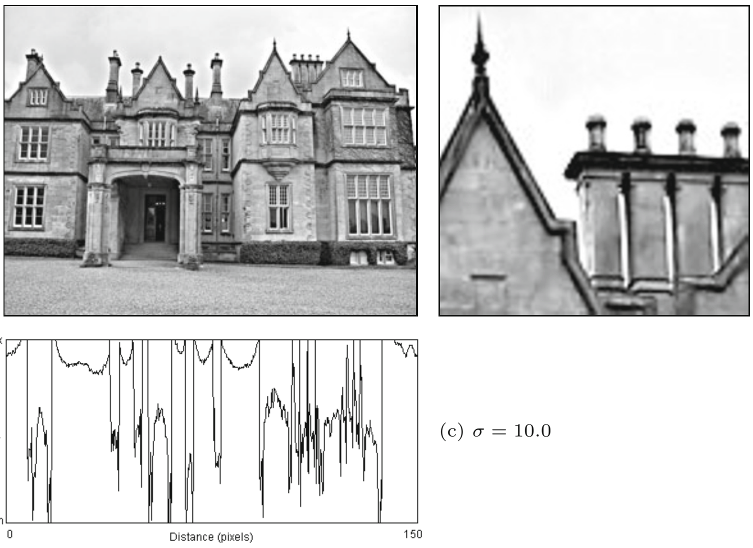
Fig. 6.19
Unsharp masking filters with varying smoothing radii $\sigma = 2.5$ and 10.0. The sharpening strength a is set to 1.0 (100%). The profiles show the intensity function for the image line marked in the original image (top-left).



(a) Original



b) $\sigma = 2.5$



(c) $\sigma = 10.0$

This filter can also be used from other plugin classes, for example, in the following way:

```
import ij.plugin.filter.UnsharpMask;
...
public void run(ImageProcessor ip) {
  UnsharpMask usm = new UnsharpMask();
  double r = 2.0; // standard settings for radius
  double a = 0.6; // standard settings for weight
  usm.sharpen(ip, r, a);
  ...
}
```

ImageJ's `UnsharpMask` implementation uses the class `GaussianBlur` for the required smoothing operation. The alternative implementation shown in Prog. 6.1 follows the definition in Eqn. (6.38) and uses Gaussian filter kernels that are created with the method `makeGaussKernel1d()`, as defined in Prog. 5.4.

```
 1  double radius = 1.0;    // radius (sigma of Gaussian)
 2  double amount = 1.0;    // amount of sharpening (1 = 100%)
 3  ...
 4  public void run(ImageProcessor ip) {
 5    ImageProcessor I = ip.convertToFloat(); // I
 6
 7    // create a blurred version of the image:
 8    ImageProcessor J = I.duplicate();            // Ĩ
 9    float[] H = GaussianFilter.makeGaussKernel1d(sigma);
10    Convolver cv = new Convolver();
11    cv.setNormalize(true);
12    cv.convolve(J, H, 1, H.length);
13    cv.convolve(J, H, H.length, 1);
14
15    I.multiply(1 + a);     // I ← (1 + a) · I
16    J.multiply(a);         // Ĩ ← a · Ĩ
17    I.copyBits(J,0,0,Blitter.SUBTRACT); //Ĩ ← (1+a) · I − a · Ĩ
18
19    //copy result back into original byte image
20    ip.insert(I.convertToByte(false), 0, 0);
21  }
```

Prog. 6.1
Unsharp masking (Java implementation). First the original image is converted to a `FloatProcessor` object `I` (I) in line 5, which is duplicated to hold the blurred image `J` ($\tilde{I}$) in line 8. The method `makeGaussKernel1d()`, defined in Prog. 5.4, is used to create the 1D Gaussian filter kernel applied in the horizontal and vertical directions (lines 12–13). The remaining calculations follow Eqn. (6.38).

Laplace vs. USM filter

A closer look at these two methods reveals that sharpening with the Laplace filter (Sec. 6.6.1) can be viewed as a special case of the USM filter. If the Laplace filter in Eqn. (6.32) is decomposed as

$$H^{\mathrm{L}} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{bmatrix} - 5 \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = 5 \cdot \bigl(\tilde{H}^{\mathrm{L}} - \delta\bigr), \quad (6.40)$$

one can see that H^L consists of a simple 3×3 pixel smoothing filter $\tilde{H}$ minus the impulse function δ. Laplace sharpening with the weight factor w as defined in Eqn. (6.35) can therefore (by a little manipulation) be expressed as

$$\begin{aligned}\check{I}_L &\leftarrow I - w\cdot(H^{\mathrm{L}} * I) \qquad = I - w\cdot\big(5(\tilde{H}^{\mathrm{L}} - \delta) * I\big)\\ &= I - 5w\cdot(\tilde{H}^{\mathrm{L}} * I - I) = I + 5w\cdot(I - \tilde{H}^{\mathrm{L}} * I) \qquad (6.41)\\ &= I + 5w\cdot M^{\mathrm{L}},\end{aligned}$$

that is, in the form of a USM filter $\check{I} \leftarrow I + a \cdot M$ (Eqn. (6.37)). Laplacian sharpening is thus a special case of a USM filter with the mask $M = M^L = (I - \tilde{H}^L * I)$, the specific smoothing filter

$$\tilde{H}^{\mathrm{L}} = \frac{1}{5}\begin{bmatrix}0 & 1 & 0\\ 1 & 1 & 1\\ 0 & 1 & 0\end{bmatrix}$$

and the sharpening factor $a = 5w$.

6.7 Exercises

Exercise 6.1. Calculate (manually) the gradient and the Laplacian (using the discrete approximations in Eqn. (6.2) and Eqn. (6.32), respectively) for the following "image":

$$I = \begin{bmatrix}14 & 10 & 19 & 16 & 14 & 12\\ 18 & 9 & 11 & 12 & 10 & 19\\ 9 & 14 & 15 & 26 & 13 & 6\\ 21 & 27 & 17 & 17 & 19 & 16\\ 11 & 18 & 18 & 19 & 16 & 14\\ 16 & 10 & 13 & 7 & 22 & 21\end{bmatrix}.$$

Exercise 6.2. Implement the Sobel edge operator as defined in Eqn. (6.10) (and illustrated in Fig. 6.6) as an ImageJ plugin. The plugin should generate two new images for the edge magnitude $E(u, v)$ and the edge orientation $\Phi(u, v)$. Come up with a suitable way to display local edge orientation.

Exercise 6.3. Express the Sobel operator (Eqn. (6.10)) in x/y-separable form analogous to the decomposition of the Prewitt operator in Eqn. (6.9).

Exercise 6.4. Implement the Kirsch operator (Eqns. (6.25)–(6.28)) analogous to the two-directional Sobel operator in Exercise 6.2 and compare the results from both methods, particularly the edge orientation estimates.

Exercise 6.5. Devise and implement a compass edge operator with more than eight (16?) differently oriented filters.

Exercise 6.6. Compare the results of the unsharp masking filters in ImageJ and Adobe Photoshop using a suitable test image. How should the parameters for σ (*radius*) and a (*weight*) be defined in both implementations to obtain similar results?