CHAPTER

Low-level feature extraction (including edge detection)

4

CHAPTER OUTLINE HEAD

Feature Extraction & Image Processing for Computer Vision.

4.1 Overview

We shall define *low-level features* to be those basic features that can be extracted automatically from an image without any shape information (information about **spatial** relationships). As such, thresholding is actually a form of low-level feature extraction performed as a point operation. Naturally, all of these approaches can be used in high-level feature extraction, where we find shapes in images. It is well known that we can recognize people from caricaturists' portraits. That is the first low-level feature we shall encounter. It is called *edge detection* and it aims to produce a **line drawing**, like one of a face in Figure 4.1(a) and (d), something akin to a caricaturist's sketch, though without the exaggeration a caricaturist would imbue. There are very basic techniques and more advanced ones and we shall look at some of the most popular approaches. The first-order detectors are equivalent to first-order differentiation and, naturally, the second-order edge-detection operators are equivalent to a one-higher level of differentiation. An alternative form of edge detection is called phase congruency and we shall again see the frequency domain used to aid analysis, this time for low-level feature extraction.

We shall also consider corner detection which can be thought of as detecting those points where lines bend very sharply with high curvature, such as the lizard's head in Figure 4.1(b) and (e). These are another low-level feature that

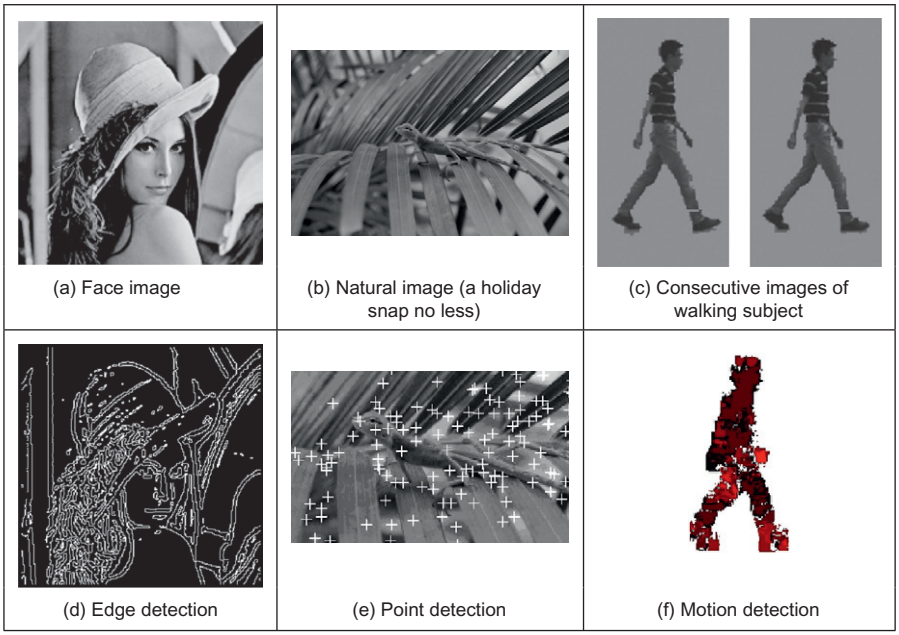


FIGURE 4.1

Low-level feature detection.

again can be extracted automatically from the image. These are largely techniques for **localized feature extraction**, in this case the curvature, and the more modern approaches extend to the detection of localized regions or **patches** of interest. Finally, we shall investigate a technique that describes **motion**, called optical flow. This is illustrated in Figure 4.1(c) and (f) with the optical flow from images of a walking man: the bits that are moving fastest are the brightest points, like the hands and the feet. All of these can provide a set of points, albeit points with different properties, but all are suitable for grouping for shape extraction. Consider a square box moving through a sequence of images. The edges are the perimeter of the box; the corners are the apices; the flow is how the box moves. All these can be collected together to find the moving box. The approaches are summarized in Table 4.1. We

Table 4.1 Overview of Chapter 4

Main Topic	Subtopics	Main Points
First-order edge detection	What is an **edge** and how we **detect** it; the **equivalence** of operators to **first-**order differentiation and the insight this brings; the need for **filtering** and more **sophisticated** first-order operators	Difference operation, *Roberts cross*, smoothing, *Prewitt*, *Sobel*, *Canny*; basis of the operators and frequency domain analysis
Second-order edge detection	Relationship between first- and **second-**order differencing operations; the **basis** of a second-order operator; the need to include **filtering** and better operations	Second-order differencing; *Laplacian*, zero-crossing detection; *Marr–Hildreth*, *Laplacian of Gaussian*, *difference of Gaussian*; *scale space*
Other edge operators	**Alternative** approaches and **performance** aspects; **comparing** different operators	Other noise models, *Spacek*; other edge models, *Petrou* and *Susan*
Phase congruency	Inverse **Fourier transform**, phase for feature extraction; alternative form of edge and feature detection	Frequency domain analysis; detecting a range of features; photometric invariance, wavelets
Localized feature extraction	Finding localized low-level features, extension from **curvature** to **patches**; nature of **curvature** and computation from: **edge** information, by **change** in intensity, and by **correlation**; motivation of **patch** detection and principles of **modern** approaches	*Planar curvature*, corners; curvature estimation by: change in *edge direction,* intensity change, *Harris* corner detector; modern feature detectors, scale space; *SIFT*, *SURF*, and *saliency* operators
Optical flow estimation	**Movement** and the nature of optical flow; estimating the optical flow by **differential** approach; need for **other** approaches (including matching regions)	Detection by differencing; *optical flow*, *aperture problem*, smoothness constraint; *differential approach*, Horn and Schunk method; *correlation*

shall start with the edge-detection techniques, with the first-order operators, which accord with the chronology of development. The first-order techniques date back by more than 30 years.

4.2 Edge detection

4.2.1 First-order edge-detection operators

4.2.1.1 Basic operators

Many approaches to image interpretation are based on edges, since analysis based on edge detection is insensitive to change in the overall illumination level. Edge detection highlights image **contrast**. Detecting contrast, which is difference in intensity, can emphasize the boundaries of features within an image, since this is where image contrast occurs. This is, naturally, how human vision can perceive the perimeter of an object, since the object is of different intensity to its surroundings. Essentially, the boundary of an object is a step change in the intensity levels. The edge is at the position of the step change. To detect the edge position we can use **first-order** differentiation since this emphasizes change; first-order differentiation gives no response when applied to signals that do not change. The first edge-detection operators to be studied here are group operators which aim to deliver an output that approximates the result of first-order differentiation.

A change in intensity can be revealed by differencing adjacent points. Differencing horizontally adjacent points will detect **vertical** changes in intensity and is often called a *horizontal edge detector* by virtue of its action. A horizontal operator will not show up **horizontal** changes in intensity since the difference is zero. (This is the form of edge detection used within the anisotropic diffusion smoothing operator in the previous chapter.) When applied to an image **P** the action of the horizontal edge detector forms the difference between two horizontally adjacent points, as such detecting the vertical edges, **Ex**, as:

$$\mathbf{Ex}_{x,y} = |\mathbf{P}_{x,y} - \mathbf{P}_{x+1,y}| \quad \forall x \in 1, N-1; \;\; y \in 1, N \tag{4.1}$$

In order to detect horizontal edges, we need a *vertical edge detector* which differences vertically adjacent points. This will determine **horizontal** intensity changes but not **vertical** ones, so the vertical edge detector detects the **horizontal** edges, **Ey**, according to:

$$\mathbf{Ey}_{x,y} = |\mathbf{P}_{x,y} - \mathbf{P}_{x,y+1}| \quad \forall x \in 1, N; \;\; y \in 1, N-1 \tag{4.2}$$

Figure 4.2(b) and (c) shows the application of the vertical and horizontal operators to the synthesized image of the square shown in Figure 4.2(a).

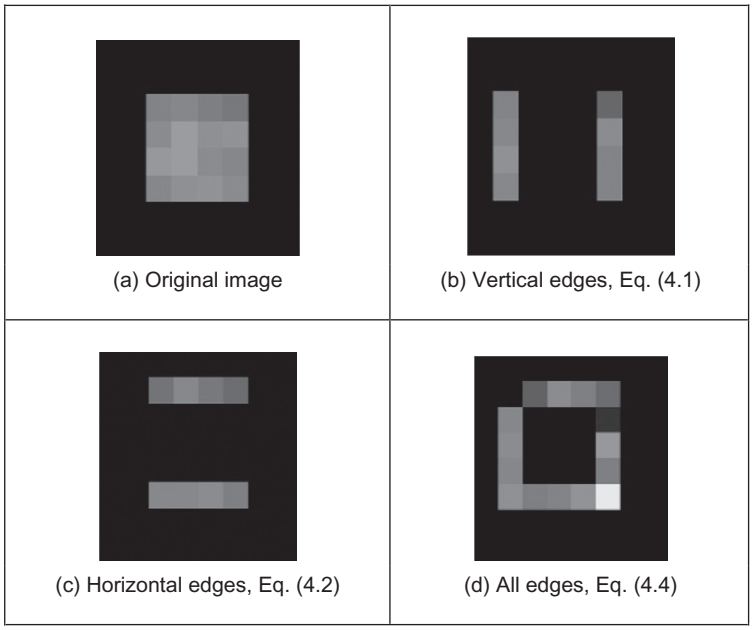


FIGURE 4.2

First-order edge detection.

The left-hand vertical edge in Figure 4.2(b) appears to be beside the square by virtue of the forward differencing process. Likewise, the upper edge in Figure 4.2(c) appears above the original square.

Combining the two gives an operator $\mathbf{E}$ that can detect vertical and horizontal edges **together**, that is,

$$\mathbf{E}_{x,y} = |\mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} + \mathbf{P}_{x,y} - \mathbf{P}_{x,y+1}| \quad \forall x, y \in 1, N-1 \tag{4.3}$$

which gives:

$$\mathbf{E}_{x,y} = |2 \times \mathbf{P}_{x,y} - \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y+1}| \quad \forall x, y \in 1, N-1 \tag{4.4}$$

Equation (4.4) gives the coefficients of a differencing template which can be convolved with an image to detect all the edge points, such as those shown in Figure 4.2(d). As in the previous chapter, the current point of operation (the position of the point we are computing a new value for) is shaded. The template shows only the weighting coefficients and not the modulus operation. Note that the bright point in the lower right corner of the edges of the square in Figure 4.2(d) is much brighter than the other points. This is because it is the only point to be detected as an edge by both the vertical and the horizontal operators and is therefore much brighter than the other edge points. In contrast, the top left-hand corner point is detected by neither operator and so does not appear in the final image.

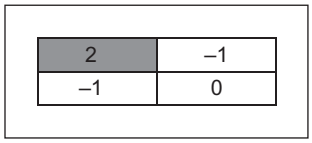


FIGURE 4.3

Template for first-order difference.

The template in Figure 4.3 is convolved with the image to detect edges. The direct implementation of this operator, i.e., using Eq. (4.4) rather than template convolution, is given in Code 4.1. Naturally, template convolution could be used, but it is unnecessarily complex in this case.

```
edge(pic):= | newpic←zero(pic)
            | for x∈0..cols(pic)-2
            |   for y∈0..rows(pic)-2
            |     newpic_{y,x}← |2·pic_{y,x}-pic_{y,x+1}-pic_{y+1,x}|
            | newpic
```

CODE 4.1

First-order edge detection.

Uniform thresholding (Section 3.3.4) is often used to select the brightest points, following application of an edge-detection operator. The threshold level controls the number of selected points; too high a level can select too few points, whereas too low a level can select too much noise. Often, the threshold level is chosen by experience or by experiment, but it can be determined automatically by considering edge data (Venkatesh and Rosin, 1995) or empirically (Haddon, 1988). For the moment, let us concentrate on the development of edge-detection operators rather than on their application.

4.2.1.2 Analysis of the basic operators

Taylor series analysis reveals that differencing adjacent points provides an estimate of the first-order derivative at a point. If the difference is taken between points separated by Δx then by Taylor expansion for $f(x + \Delta x)$ we obtain:

$$f(x + \Delta x) = f(x) + \Delta x \times f'(x) + \frac{\Delta x^2}{2!} \times f''(x) + O(\Delta x^3) \tag{4.5}$$

By rearrangement, the first-order derivative $f'(x)$ is:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x} - O(\Delta x) \tag{4.6}$$

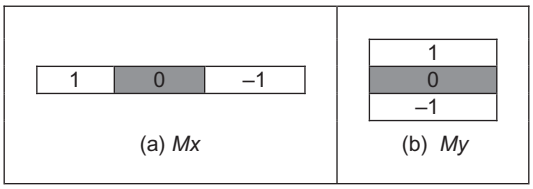


FIGURE 4.4

Templates for improved first-order difference.

This shows that the difference between adjacent points is an estimate of the first-order derivative, with error $O(\Delta x)$. This error depends on the size of the interval Δx and on the complexity of the curve. When Δx is large this error can be significant. The error is also large when the high-order derivatives take large values. In practice, the short sampling of image pixels and the reduced high-frequency content make this approximation adequate. However, the error can be reduced by spacing the differenced points by one pixel. This is equivalent to computing the first-order difference delivered by Eq. (4.1) at two adjacent points, as a new horizontal difference **Exx** where

$$\mathbf{Exx}_{x,y} = \mathbf{Ex}_{x+1,y} + \mathbf{Ex}_{x,y} = \mathbf{P}_{x+1,y} - \mathbf{P}_{x,y} + \mathbf{P}_{x,y} - \mathbf{P}_{x-1,y} = \mathbf{P}_{x+1,y} - \mathbf{P}_{x-1,y} \quad (4.7)$$

This is equivalent to incorporating spacing to detect the edges **Exx** by:

$$\mathbf{Exx}_{x,y} = |\mathbf{P}_{x+1,y} - \mathbf{P}_{x-1,y}| \quad \forall x \in 2, N-1; \;\; y \in 1, N \quad (4.8)$$

To analyze this, again by Taylor series, we expand $f(x - \Delta x)$ as:

$$f(x - \Delta x) = f(x) - \Delta x \times f'(x) + \frac{\Delta x^2}{2!} \times f''(x) - O(\Delta x^3) \quad (4.9)$$

By differencing Eq. (4.9) from Eq. (4.5), we obtain the first-order derivative as

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} - O(\Delta x^2) \quad (4.10)$$

Equation (4.10) suggests that the estimate of the first-order difference is now the difference between points separated by one pixel, with error $O(\Delta x^2)$. If $\Delta x < 1$, this error is clearly smaller than the error associated with differencing adjacent pixels, in Eq. (4.6). Again, averaging has reduced noise or error. The template for a horizontal edge-detection operator is given in Figure 4.4(a).This template gives the vertical edges detected at its center pixel. A transposed version of the template gives a vertical edge-detection operator (Figure 4.4(b)).

The *Roberts cross operator* (Roberts, 1965) was one of the earliest edge-detection operators. It implements a version of basic first-order edge detection and uses two templates that differentiate pixel values in a diagonal manner, as

opposed to along the axes' directions. The two templates are called M^+ and M^- and are given in Figure 4.5.

In implementation, the maximum value delivered by application of these templates is stored as the value of the edge at that point. The edge point $\mathbf{E}_{x,y}$ is then the maximum of the two values derived by convolving the two templates at an image point $\mathbf{P}_{x,y}$:

$$\mathbf{E}_{x,y} = \max\{|M^+ * \mathbf{P}_{x,y}|, |M^- * \mathbf{P}_{x,y}|\} \quad \forall x, y \in 1, N-1 \tag{4.11}$$

The application of the Roberts cross operator to the image of the square is shown in Figure 4.6. The results of the two templates are shown in Figure 4.6(a) and (b), and the result delivered by the Roberts operator is shown in Figure 4.6(c). Note that the corners of the square now appear in the edge image, by virtue of the diagonal differencing action, whereas they were less apparent in Figure 4.2(d) (where the top left corner did not appear).

An alternative to taking the maximum is to simply **add** the results of the two templates together to combine horizontal and vertical edges. There are of course more varieties of edges and it is often better to consider the two templates as providing components of an *edge vector*: the strength of the edge along the horizontal and vertical axes. These give components of a vector and can be added in a

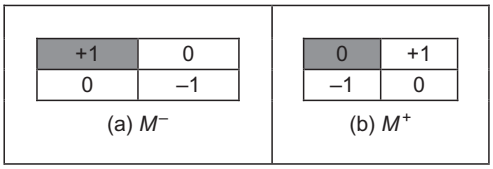


FIGURE 4.5

Templates for Roberts cross operator.

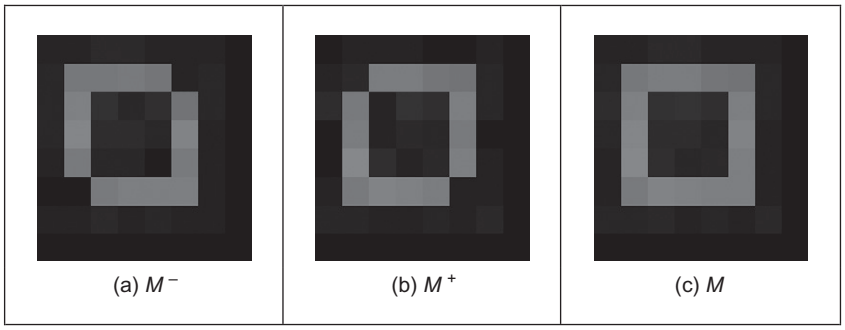


FIGURE 4.6

Applying the Roberts cross operator.

vectorial manner (which is perhaps more usual for the Roberts operator). The *edge magnitude* is the **length** of the vector and the *edge direction* is the vector's **orientation**, as shown in Figure 4.7.

4.2.1.3 Prewitt edge-detection operator

Edge detection is akin to differentiation. Since it detects change it is bound to respond to **noise**, as well as to step-like changes in image intensity (its frequency domain analog is high-pass filtering as illustrated in Figure 2.30(c)). It is therefore prudent to incorporate **averaging** within the edge-detection process. We can then extend the vertical template, Mx, along three rows, and the horizontal template, My, along three columns. These give the *Prewitt edge-detection operator* (Prewitt and Mendelsohn, 1966) that consists of two templates (Figure 4.8).

This gives two results: the rate of change of brightness along each axis. As such, this is the vector illustrated in Figure 4.7: the edge magnitude, M, is the length of the vector and the edge direction, θ, is the angle of the vector.

$$M(x,y) = \sqrt{Mx(x,y)^2 + My(x,y)^2} \tag{4.12}$$

$$\theta(x,y) = \tan^{-1}\left(\frac{My(x,y)}{Mx(x,y)}\right) \tag{4.13}$$

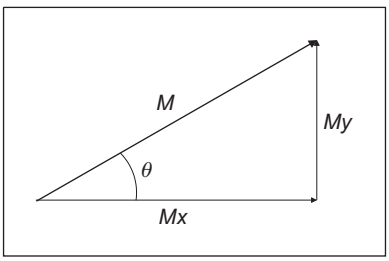


FIGURE 4.7

Edge detection in vectorial format.

1	0	−1
1	0	−1
1	0	−1

(a) Mx

1	1	1
0	0	0
−1	−1	−1

(b) My

FIGURE 4.8

Templates for Prewitt operator.

$$\text{Prewitt33\_x}(\text{pic}) := \sum_{y=0}^{2} \text{pic}_{y,0} - \sum_{y=0}^{2} \text{pic}_{y,2}$$

(a) *Mx*

$$\text{Prewitt33\_y}(\text{pic}) := \sum_{x=0}^{2} \text{pic}_{0,x} - \sum_{x=0}^{2} \text{pic}_{2,x}$$

(b) *My*

CODE 4.2

Implementing the Prewitt operator.

Again, the signs of *Mx* and *My* can be used to determine the appropriate quadrant for the edge direction. A Mathcad implementation of the two templates of Figure 4.8 is given in Code 4.2. In this code, both templates operate on a 3×3 subpicture (which can be supplied, in Mathcad, using the `submatrix` function). Again, template convolution could be used to implement this operator, but (as with direct averaging and basic first-order edge detection) it is less suited to simple templates. Also, the provision of edge magnitude and direction would require extension of the template convolution operator given earlier (Code 3.5).

When applied to the image of the square (Figure 4.9(a)) we obtain the edge magnitude and direction (Figure 4.9(b) and (d)), respectively (where Figure 4.9(d) does not include the border points but only the edge direction at processed points). The edge direction shown in Figure 4.9(d) is measured in degrees where 0° and 360° are horizontal, to the right, and 90° is vertical, upward. Though the regions of edge points are wider due to the operator's averaging properties, the edge data is clearer than the earlier first-order operator, highlighting the regions where intensity changed in a more reliable fashion (compare, for example, the upper left corner of the square which was not revealed earlier). The direction is less clear in an image format and is better exposed by Mathcad's **vector** format in Figure 4.9(c). In vector format, the edge-direction data is clearly less well defined at the corners of the square (as expected, since the first-order derivative is discontinuous at these points).

4.2.1.4 Sobel edge-detection operator

When the weight at the central pixels, for both Prewitt templates, is doubled, this gives the famous *Sobel edge-detection operator* which, again, consists of two masks to determine the edge in vector form. The Sobel operator was the most popular edge-detection operator until the development of edge-detection techniques with a theoretical basis. It proved popular because it gave, overall, a better performance than other contemporaneous edge-detection operators, such as the Prewitt operator. The templates for the Sobel operator can be found in Figure 4.10.

The Mathcad implementation of these masks is very similar to the implementation of the Prewitt operator, Code 4.2, again operating on a 3×3 subpicture. This is the standard formulation of the Sobel templates, but how do we form larger templates, say for 5×5 or 7×7? Few textbooks state its original

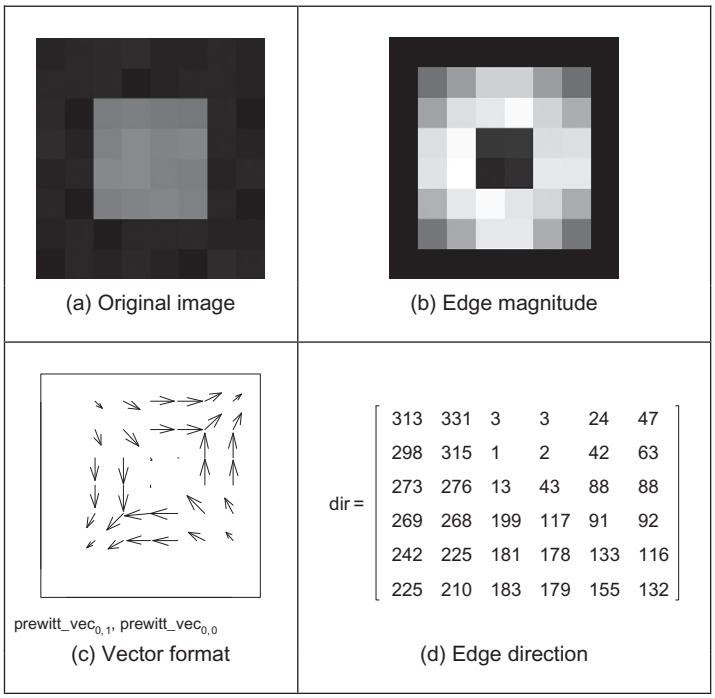


FIGURE 4.9

Applying the Prewitt operator.

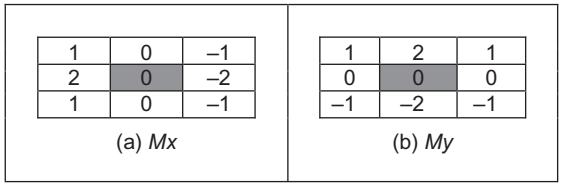


FIGURE 4.10

Templates for Sobel operator.

derivation, but it has been attributed (Heath et al., 1997) as originating from a PhD thesis (Sobel, 1970). Unfortunately a theoretical basis, that can be used to calculate the coefficients of larger templates, is rarely given. One approach to a theoretical basis is to consider the optimal forms of averaging and of differencing. Gaussian averaging has already been stated to give optimal averaging. The binomial expansion gives the integer coefficients of a series that, in the limit, approximates the normal distribution. Pascal's triangle gives sets of coefficients for a

smoothing operator which, in the limit, approaches the coefficients of a Gaussian smoothing operator. Pascal's triangle is then:

Window size							
2			1		1		
3		1		2		1	
4	1		3		3		1
5		4		6		4	

This gives the (unnormalized) coefficients of an optimal discrete smoothing operator (it is essentially a Gaussian operator with integer coefficients). The rows give the coefficients for increasing the size of template or window. The coefficients of smoothing within the Sobel operator (Figures 4.10) are those for a window size of 3. In Mathcad, by specifying the size of the smoothing window as `winsize`, the template coefficients $\texttt{smooth}_{\texttt{x\_win}}$ can be calculated at each window point `x_win` according to Code 4.3.

$$\texttt{smooth}_{\texttt{x\_win}} := \frac{\texttt{(winsize-1)!}}{\texttt{(winsize-1-x\_win)!}\cdot\texttt{x\_win!}}$$

CODE 4.3

Smoothing function.

The differencing coefficients are given by Pascal's triangle for subtraction:

Window size							
2			1		−1		
3		1		0		−1	
4	1		1		−1		−1
5		2		0		−2	

This can be implemented by subtracting the templates derived from two adjacent expansions for a smaller window size. Accordingly, we require an operator which can provide the coefficients of Pascal's triangle for arguments which are of window size `n` and a position `k`. The operator is the `Pascal(k,n)` operator in Code 4.4.

$$\texttt{Pascal(k,n)} := \begin{cases} \dfrac{\texttt{n!}}{\texttt{(n-k)!}\cdot\texttt{k!}} & \texttt{if(k}\geq\texttt{0)}\cdot\texttt{(k}\leq\texttt{n)} \\ \texttt{0} & \texttt{otherwise} \end{cases}$$

CODE 4.4

Pascal's triangle.

The differencing template, $\texttt{diff}_{\texttt{x\_win}}$, is then given by the difference between two Pascal expansions, as given in Code 4.5.

```
diff_x_win := Pascal(x_win, winsize-2)-Pascal(x_win-1, winsize-2)
```

CODE 4.5

Differencing function.

These give the coefficients of optimal differencing and optimal smoothing. This **general** form of the Sobel operator combines optimal smoothing along one axis, with optimal differencing along the other. This general form of the Sobel operator is given in Code 4.6 which combines the differencing function along one axis, with smoothing along the other.

$$\text{Sobel\_x(pic)} := \sum_{x\_win=0}^{winsize-1} \sum_{y\_win=0}^{winsize-1} \text{smooth}_{y\_win} \cdot \text{diff}_{x\_win} \cdot \text{pic}_{y\_win, x\_win}$$

(a) Mx

$$\text{Sobel\_y(pic)} := \sum_{x\_win=0}^{winsize-1} \sum_{y\_win=0}^{winsize-1} \text{smooth}_{x\_win} \cdot \text{diff}_{y\_win} \cdot \text{pic}_{y\_win, x\_win}$$

(b) My

CODE 4.6

Generalized Sobel templates.

This generates another template for the *Mx* template for a Sobel operator, given for 5×5 in Code 4.7.

$$\text{Sobel\_template\_x} = \begin{bmatrix} 1 & 2 & 0 & -2 & -1 \\ 4 & 8 & 0 & -8 & -4 \\ 6 & 12 & 0 & -12 & -6 \\ 4 & 8 & 0 & -8 & -4 \\ 1 & 2 & 0 & -2 & -1 \end{bmatrix}$$

CODE 4.7

5×5 Sobel template *Mx*.

All template-based techniques can be larger than 5×5, so, as with any group operator, there is a 7×7 Sobel and so on. The virtue of a larger edge-detection template is that it involves more smoothing to reduce noise, but edge blurring

becomes a great problem. The estimate of edge direction can be improved with more smoothing since it is particularly sensitive to noise. There are circular edge operators designed specifically to provide accurate edge-direction data.

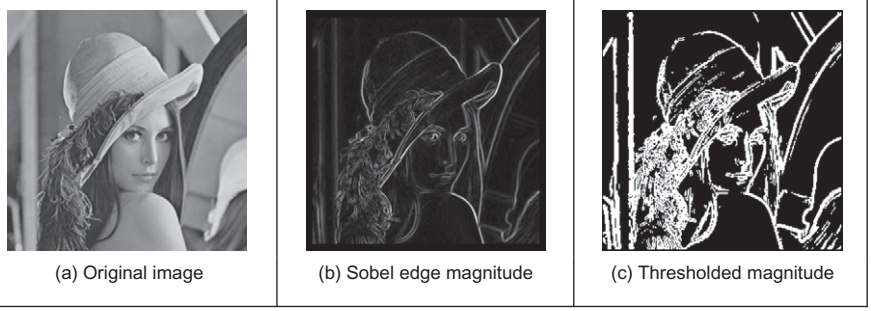
The Sobel templates can be invoked by operating on a matrix of dimension equal to the window size, from which edge magnitude and gradient are calculated. The `Sobel` function (Code 4.8) convolves the generalized Sobel template (of size chosen to be `winsize`) with the picture supplied as argument, to give outputs which are the images of edge magnitude and direction, in vector form.

```
Sobel(pic,winsize):=
                    w2←floor(winsize/2)
                    edge_mag←zero(pic)
                    edge_dir←zero(pic)
                    for x∈w2.. cols(pic)-1-w2
                      for y∈w2.. rows(pic)-1-w2
                        x_mag←Sobel_x(submatrix(pic,y-w2,y+w2,x-w2,x+w2))
                        y_mag←Sobel_y(submatrix(pic,y-w2,y+w2,x-w2,x+w2))
                        edge_mag[y,x←floor(magnitude(x_mag,y_mag)/mag_normalise)
                        edge_dir[y,x←direction(x_mag,y_mag)
                    (edge_mag edge_dir)
```

CODE 4.8

Generalized Sobel operator.

The results of applying the 3×3 Sobel operator can be seen in Figure 4.11. The original face image (Figure 4.11(a)) has many edges in the hair and in the region of the eyes. This is shown in the edge magnitude image (Figure 4.11(b)). When this is thresholded at a suitable value, many edge points are found, as



(a) Original image (b) Sobel edge magnitude (c) Thresholded magnitude

FIGURE 4.11

Applying the Sobel operator.

shown in Figure 4.11(c). Note that in areas of the image where the brightness remains fairly constant, such as the cheek and shoulder, there is little change which is reflected by low-edge magnitude and few points in the thresholded data.

The Sobel edge-direction data can be arranged to point in different ways, as can the direction provided by the Prewitt operator. If the templates are inverted to be of the form shown in Figure 4.12, the edge direction will be inverted around both the axes. If only one of the templates is inverted, the measured edge direction will be inverted about the chosen axis.

This gives **four** possible directions for measurement of the edge direction provided by the Sobel operator, two of which (for the templates that are shown in Figures 4.10 and 4.12) are illustrated in Figure 4.13(a) and (b), respectively, where inverting the *Mx* template does not highlight discontinuity at the corners. (The edge magnitude of the Sobel applied to the square is not shown but is similar to that derived by application of the Prewitt operator (Figure 4.9(b))). By swapping the Sobel templates, the measured edge direction can be arranged to be normal to the edge itself (as opposed to tangential data along the edge). This is illustrated in Figure 4.13(c) and (d) for swapped versions of the templates given in Figures 4.10 and 4.12, respectively. The rearrangement can lead to simplicity in algorithm construction when finding shapes, as to be shown later. Any algorithm which uses edge direction for finding shapes must know precisely which arrangement has been used, since the edge direction can be used to speed algorithm performance, but it must map precisely to the expected image data if used in that way.

Detecting edges by **template convolution** again has a frequency domain interpretation. The magnitude of the Fourier transform of a 5×5 Sobel template of Code 4.7 is given in Figure 4.14. The Fourier transform is given in relief in Figure 4.14(a) and as a contour plot in Figure 4.14(b). The template is for horizontal differencing action, *My*, which highlights vertical change. Accordingly, its transform reveals that it selects vertical spatial frequencies, while smoothing the horizontal ones. The horizontal frequencies are selected from a region near the origin (**low-pass** filtering), whereas the vertical frequencies are selected away from the origin (**high-pass**). This highlights the action of the Sobel operator, combining smoothing of the spatial frequencies along one axis with differencing of

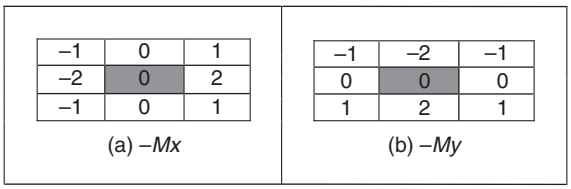


FIGURE 4.12

Inverted templates for Sobel operator.

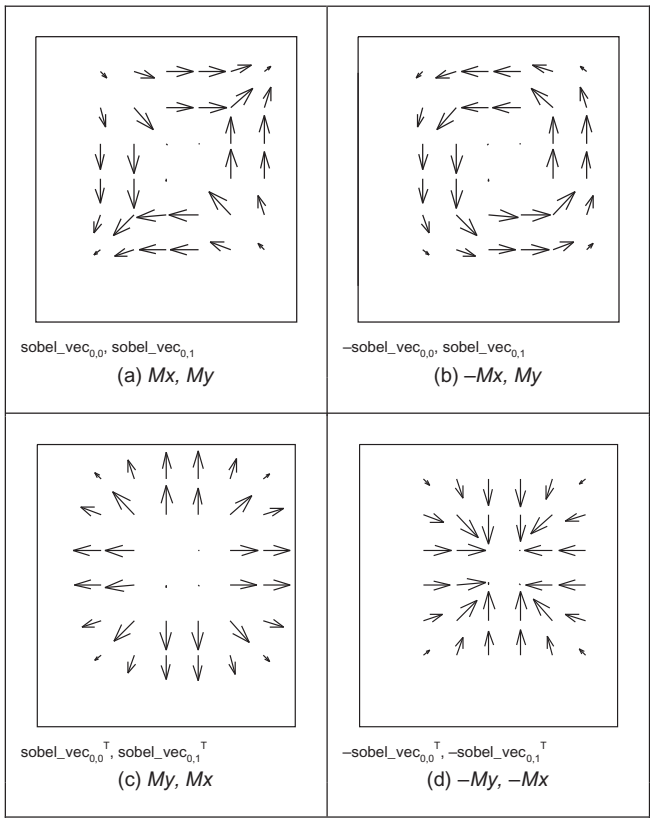


FIGURE 4.13

Alternative arrangements of edge direction.

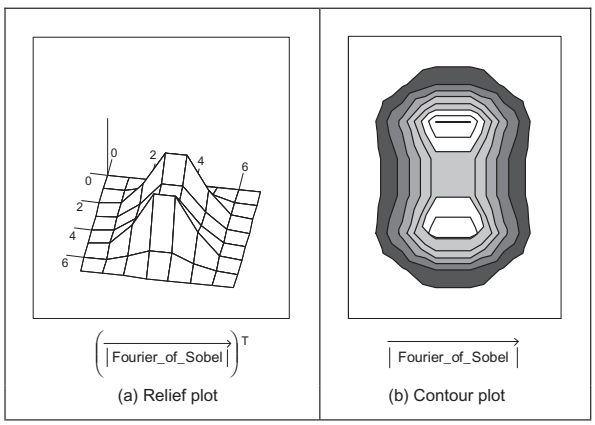


FIGURE 4.14

Fourier transform of the Sobel operator.

the other. In Figure 4.14, the smoothing is of horizontal spatial frequencies while the differencing is of vertical spatial frequencies.

An alternative frequency domain analysis of the Sobel can be derived via the *z-transform* operator. This is more than the domain of signal processing courses in electronic and electrical engineering and is included here for completeness and for linkage with signal processing. Essentially z^{-1} is a unit time-step delay operator, so z can be thought of a unit (time-step) advance, so $f(t-\tau) = z^{-1}f(t)$ and $f(t+\tau) = zf(t)$, where τ is the sampling interval. Given that we have two spatial axes x and y we can then express the Sobel operator of Figure 4.12(a) using delay and advance via the z-transform notation along the two axes as

$$S(x,y) = \begin{matrix} -z_x^{-1}z_y^{-1} + 0 + z_x z_y^{-1} \\ -2z_x^{-1} + 0 + 2z_x \\ -z_x^{-1}z_y + 0 + z_x z_y \end{matrix} \tag{4.14}$$

including zeros for the null template elements. Given that there is a standard substitution (by conformal mapping, evaluated along the frequency axis) $z^{-1} = \mathrm{e}^{-\mathrm{j}\omega t}$ to transform from the time domain (z) to the frequency domain (ω), then we have

$$\begin{aligned} \mathrm{Sobel}(\omega_x,\omega_y) &= -\mathrm{e}^{-\mathrm{j}\omega_x t}\,\mathrm{e}^{-\mathrm{j}\omega_y t} + \mathrm{e}^{\mathrm{j}\omega_x t}\,\mathrm{e}^{-\mathrm{j}\omega_y t} - 2\mathrm{e}^{-\mathrm{j}\omega_x t} + 2\mathrm{e}^{\mathrm{j}\omega_x t} - \mathrm{e}^{-\mathrm{j}\omega_x t}\,\mathrm{e}^{\mathrm{j}\omega_y t} + \mathrm{e}^{\mathrm{j}\omega_x t}\,\mathrm{e}^{\mathrm{j}\omega_y t} \\ &= (\mathrm{e}^{-\mathrm{j}\omega_y t} + 2 + \mathrm{e}^{\mathrm{j}\omega_y t})(-\mathrm{e}^{-\mathrm{j}\omega_x t} + \mathrm{e}^{\mathrm{j}\omega_x t}) \\ &= \left(\mathrm{e}^{\frac{-\mathrm{j}\omega_y t}{2}} + \mathrm{e}^{\frac{\mathrm{j}\omega_y t}{2}}\right)^2 (-\mathrm{e}^{-\mathrm{j}\omega_x t} + \mathrm{e}^{\mathrm{j}\omega_x t}) \\ &= 8\mathrm{j}\cos^2\left(\frac{\omega_y t}{2}\right)\sin(\omega_x t) \end{aligned} \tag{4.15}$$

where the transform Sobel is a function of spatial frequency, ω_x, ω_y, along the x and the y axes. This conforms rather nicely the separation between smoothing along one axis (the first part of Eq. (4.15)) and differencing along the other—here by differencing (high-pass) along the x axis and averaging (low-pass) along the y axis. This provides an analytic form of the function shown in Figure 4.14; the relationship between the DFT and this approach is evident by applying the DFT relationship (Eq. (2.15)) to the components of the Sobel operator.

4.2.1.5 The Canny edge detector

The *Canny edge-detection operator* (Canny, 1986) is perhaps the most popular edge-detection technique at present. It was formulated with three main objectives:

1. **optimal** detection with no spurious responses;
2. **good** localization with minimal distance between detected and true edge position; and
3. **single** response to eliminate multiple responses to a single edge.

The first requirement aims to **reduce** the response to noise. This can be effected by optimal smoothing; Canny was the first to demonstrate that Gaussian filtering is optimal for edge detection (within his criteria). The second criterion aims for accuracy: edges are to be detected, in the right place. This can be achieved by a process of *nonmaximum suppression* (which is equivalent to peak detection). Nonmaximum suppression retains only those points at the top of a ridge of edge data, while suppressing all others. This results in thinning: the output of nonmaximum suppression is thin lines of edge points, in the right place. The third constraint concerns location of a single edge point in response to a change in brightness. This is because more than one edge can be denoted to be present, consistent with the output obtained by earlier edge operators.

Canny showed that the Gaussian operator was optimal for image smoothing. Recalling that the Gaussian operator $g(x,y,\sigma)$ is given by:

$$g(x, y, \sigma) = \mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{4.16}$$

By differentiation, for unit vectors $U_x = [1,0]$ and $U_y = [0,1]$ along the coordinate axes, we obtain

$$\begin{aligned} \nabla g(x, y) &= \frac{\partial g(x, y, \sigma)}{\partial x} U_x + \frac{\partial g(x, y, \sigma)}{\partial y} U_y \\ &= -\frac{x}{\sigma^2} \mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}} U_x - \frac{y}{\sigma^2} \mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}} U_y \end{aligned} \tag{4.17}$$

Equation (4.17) gives a way to calculate the coefficients of a *derivative of Gaussian* template that combines first-order differentiation with Gaussian smoothing. This is a smoothed image, and so the edge will be a ridge of data. In order to mark an edge at the correct point (and to reduce multiple response), we can convolve an image with an operator which gives the first derivative in a direction normal to the edge. The maximum of this function should be the peak of the edge data, where the gradient in the original image is sharpest, and hence the location of the edge. Accordingly, we seek an operator, G_n, which is a first derivative of a Gaussian function g in the direction of the normal, $\mathbf{n}_\perp$:

$$G_n = \frac{\partial g}{\partial \mathbf{n}_\perp} \tag{4.18}$$

where $\mathbf{n}_\perp$ can be estimated from the first-order derivative of the Gaussian function g convolved with the image $\mathbf{P}$, and scaled appropriately as

$$\mathbf{n}_\perp = \frac{\nabla(\mathbf{P} * g)}{|\nabla(\mathbf{P} * g)|} \tag{4.19}$$

The location of the true edge point is at the maximum point of G_n convolved with the image. This maximum is when the differential (along $\mathbf{n}_\perp$) is zero:

$$\frac{\partial(G_n * \mathbf{P})}{\partial \mathbf{n}_\perp} = 0 \tag{4.20}$$

By substituting Eq. (4.18) in Eq. (4.20), we get

$$\frac{\partial^2(G * \mathbf{P})}{\partial {\mathbf{n}_\perp}^2} = 0 \tag{4.21}$$

Equation (4.21) provides the basis for an operator which meets one of Canny's criteria, namely that edges should be detected in the correct place. This is nonmaximum suppression, which is equivalent to retaining peaks (and thus equivalent to differentiation perpendicular to the edge), which thins the response of the edge-detection operator to give edge points which are in the right place, without multiple response and with minimal response to noise. However, it is virtually impossible to achieve an exact implementation of Canny given the requirement to estimate the normal direction.

A common approximation is, as illustrated in Figure 4.15, as follows:

1. use Gaussian smoothing (as in Section 3.4.4) (Figure 4.15(a));
2. use the Sobel operator (Figure 4.15(b));
3. use nonmaximal suppression (Figure 4.15(c)); and
4. threshold with hysteresis to connect edge points (Figure 4.15(d)).

Note that the first two stages can be combined using a version of Eq. (4.17) but are separated here so that all stages in the edge-detection process can be shown clearly. An alternative implementation of Canny's approach (Deriche, 1987) used Canny's criteria to develop 2D recursive filters, claiming performance and implementation advantage over the approximation here.

Nonmaximum suppression essentially locates the highest points in the edge magnitude data. This is performed by using edge-direction information to check that points are at the peak of a ridge. Given a 3×3 region, a point is at a

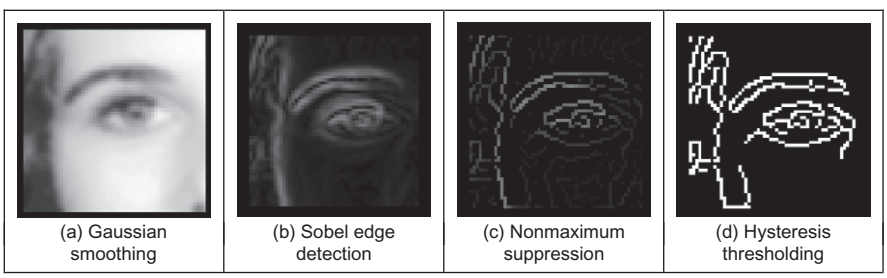


FIGURE 4.15

Stages in Canny edge detection.

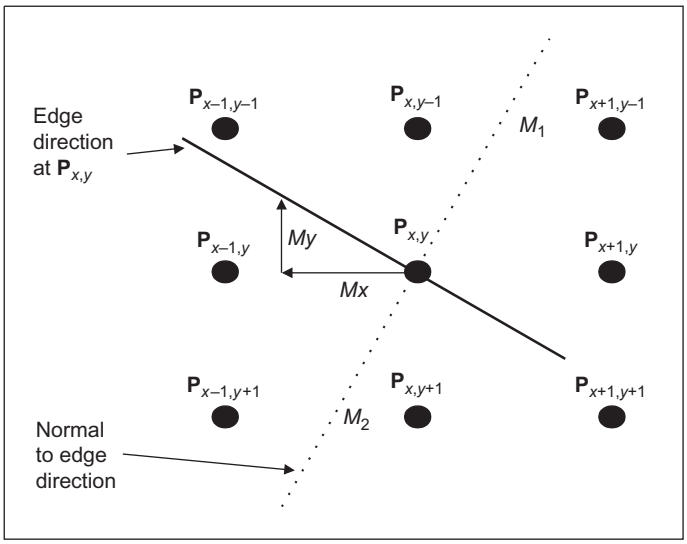


FIGURE 4.16

Interpolation in nonmaximum suppression.

maximum if the gradient at either side of it is less than the gradient at the point. This implies that we need values of gradient along a line which is normal to the edge at a point. This is illustrated in Figure 4.16, which shows the neighboring points to the point of interest, $\mathbf{P}_{x,y}$, the edge direction at $\mathbf{P}_{x,y}$ and the normal to the edge direction at $\mathbf{P}_{x,y}$. The point $\mathbf{P}_{x,y}$ is to be marked as maximum if its gradient, $M(x,y)$, exceeds the gradient at points 1 and 2, M_1 and M_2, respectively. Since we have a discrete neighborhood, M_1 and M_2 need to be interpolated. First-order interpolation using Mx and My at $\mathbf{P}_{x,y}$ and the values of Mx and My for the neighbors gives

$$M_1 = \frac{My}{Mx}M(x+1, y-1) + \frac{Mx-My}{Mx}M(x, y-1) \tag{4.22}$$

and

$$M_2 = \frac{My}{Mx}M(x-1, y+1) + \frac{Mx-My}{Mx}M(x, y+1) \tag{4.23}$$

The point $\mathbf{P}_{x,y}$ is then marked as a maximum if $M(x,y)$ exceeds both M_1 and M_2, otherwise it is set to zero. In this manner the peaks of the ridges of edge magnitude data are retained, while those not at the peak are set to zero. The implementation of nonmaximum suppression first requires a function which generates the coordinates of the points between which the edge magnitude is interpolated. This is the function `get_coords` in Code 4.9 which requires the angle of the normal to the edge direction, returning the coordinates of the points beyond and behind the normal.

```
get_coords(angle) := | δ←0.000000000000001
                     | x1←ceil[(cos(angle+π/8)·√2)-0.5-δ]
                     | y1←ceil[(-sin(angle-π/8)·√2)-0.5-δ]
                     | x2←ceil[(cos(angle-π/8)·√2)-0.5-δ]
                     | y2←ceil[(-sin(angle-π/8)·√2)-0.5-δ]
                     | (x1 y1 x2 y2)
```

CODE 4.9

Generating coordinates for interpolation.

The nonmaximum suppression operator, `non_max`, in Code 4.10 then interpolates the edge magnitude at the two points either side of the normal to the edge direction. If the edge magnitude at the point of interest exceeds these two then it

```
non_max(edges) := | for i∈1..cols(edges0,0)-2
                  |   for j∈1..rows(edges0,0)-2
                  |     Mx←(edges0,0)j,i
                  |     My←(edges0,1)j,i
                  |     o←atan(Mx/My) if My ≠ 0
                  |     (o←π/2) if (My=0)·(Mx>0)
                  |     o←-π/2 otherwise
                  |     adds←get_coords(o)
                  |     M1←[My·(edges0,2)j+adds0,1,i+adds0,0 ···
                  |         +(Mx-My)·(edges0,2)j+adds0,3,i+adds0,2]
                  |     adds←get_coords(o+π)
                  |     M2←[My·(edges0,2)j+adds0,1,i+adds0,0 ···
                  |         +(Mx-My)·(edges0,2)j+adds0,3,i+adds0,2]
                  |     isbigger←[[Mx·(edges0,2)j,i>M1]·[Mx.(edges0,2)j,i≥M2]] ···
                  |              +[[Mx·(edges0,2)j,i<M1]·[Mx.(edges0,2)j,i≤M2]]
                  |     new_edgej,i←(edges0,2)j,i if isbigger
                  |     new_edgej,i←0 otherwise
                  | new_edge
```

CODE 4.10

Nonmaximum suppression.

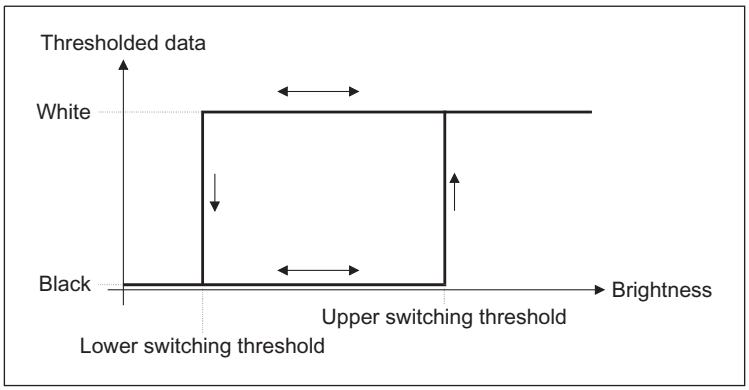


FIGURE 4.17

Hysteresis thresholding transfer function.

is retained, otherwise it is discarded. Note that the potential singularity in Eqs (4.22) and (4.23) can be avoided by use of multiplication in the magnitude comparison, as opposed to division in interpolation, as it is in Code 4.10. In practice however, this implementation, Codes 4.9 and 4.10, can suffer from numerical imprecision and ill conditioning. Accordingly, it is better to implement a handcrafted interpretation of Eqs (4.22) and (4.23) applied separately to the four quadrants. This is too lengthy to be included here, but a version is included with the worksheets for this chapter.

The transfer function associated with *hysteresis thresholding* is shown in Figure 4.17. Points are set to white once the upper threshold is exceeded and set to black when the lower threshold is reached. The arrows reflect possible movement: there is only one way to change from black to white and vice versa.

The application of nonmaximum suppression and hysteresis thresholding is illustrated in Figure 4.18. This contains a ridge of edge data, the edge magnitude. The action of nonmaximum suppression is to select the points along the top of the ridge. Given that the top of the ridge initially exceeds the upper threshold, the thresholded output is set to white until the peak of the ridge falls beneath the lower threshold. The thresholded output is then set to black until the peak of the ridge exceeds the upper switching threshold.

Hysteresis thresholding requires two thresholds, an **upper** and a **lower** threshold. The process starts when an edge point from nonmaximum suppression is found to exceed the upper threshold. This is labeled as an edge point (usually white, with a value 255) and forms the first point of a line of edge points. The neighbors of the point are then searched to determine whether or not they exceed the lower threshold, as shown in Figure 4.19. Any neighbor that exceeds the lower threshold is labeled as an edge point and its neighbors are then searched to determine whether or not they exceed the lower threshold. In this manner, the first edge point found (the one that exceeded the upper threshold) becomes a **seed**

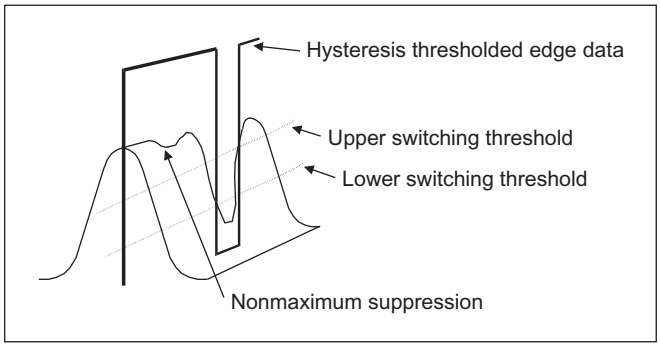


FIGURE 4.18

Action of nonmaximum suppression and hysteresis thresholding.

≥ Lower	≥ Lower	≥ Lower
≥ Lower	Seed ≥ upper	≥ Lower
≥ Lower	≥ Lower	≥ Lower

FIGURE 4.19

Neighborhood search for hysteresis thresholding.

point for a search. Its neighbors, in turn, become seed points if they exceed the lower threshold, and so the search extends, along branches arising from neighbors that exceeded the lower threshold. For each branch, the search terminates at points that have no neighbors above the lower threshold.

In implementation, hysteresis thresholding clearly requires **recursion**, since the length of any branch is unknown. Having found the initial **seed** point, the seed point is set to white and its neighbors are searched. The coordinates of each point are checked to see whether it is within the picture size, according to the operator `check`, given in Code 4.11.

```
check(xc,yc,pic):=|1 if (xc≥1)·(xc≤cols(pic)-2)·(yc≥1)·(yc≤rows(pic)-2)
                  |0 otherwise
```

CODE 4.11

Checking points are within an image.

The neighborhood (as shown in Figure 4.19) is then searched by a function `connect` (Code 4.12) that is fed with the nonmaximum suppressed edge image, the coordinates of the seed point whose connectivity is under analysis and the lower switching threshold. Each of the neighbors is searched if its value exceeds the lower threshold, and the point has not already been labeled as white

```
connect(x,y,nedg,low):= | for x1∈x−1.. x+1
                        |   for y1∈y−1.. y+1
                        |     if(nedg_{y1,x1}≥low)·(nedg_{y1,x1}≠255)·check
                        |       (x1,y1,nedg)
                        |         | nedg_{y1,x1}←255
                        |         | nedg←connect(x1,y1,nedg,low)
                        | nedg
```

CODE 4.12

Connectivity analysis after seed point location.

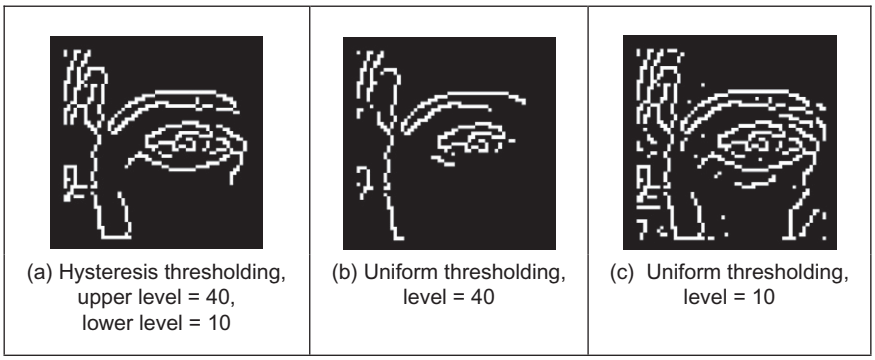
(otherwise the function would become an infinite loop). If both conditions are satisfied (and the point is within the picture), then the point is set to white and becomes a seed point for further analysis. This implementation tries to check the seed point as well, even though it has already been set to white. The operator could be arranged not to check the current seed point, by direct calculation without the `for` loops, and this would be marginally faster. Including an extra Boolean constraint to inhibit check of the seed point would only slow the operation. The `connect` routine is recursive: it is called again by the new seed point.

The process starts with the point that exceeds the upper threshold. When such a point is found, it is set to white and it becomes a seed point where connectivity analysis starts. The calling operator for the connectivity analysis, `hyst_thr`, which starts the whole process is given in Code 4.13. When `hyst_thr` is invoked, its arguments are the coordinates of the point of current interest, the nonmaximum suppressed edge image, `n_edg` (which is eventually delivered as the hysteresis thresholded image), and the upper and lower switching thresholds, `upp` and `low`, respectively. For **display** purposes, this operator requires a later operation to remove points which have not been set to white (to remove those points which are below the upper threshold and which are not connected to points above the lower threshold). This is rarely used in application since the points set to white are the only ones of interest in later processing.

```
hyst_thr(n_edg,upp,low):= | for x∈1.. cols(n_edg)−2
                          |   for y∈1.. rows(n_edg)−2
                          |     if[(n_edg_{y,x}≥upp)·(n_edg_{y,x}≠255)]
                          |         | n_edg_{y,x}←255
                          |         | n_edg←connect(x,y,n_edg,low)
                          | n_edg
```

CODE 4.13

Hysteresis thresholding operator.



(a) Hysteresis thresholding, upper level = 40, lower level = 10

(b) Uniform thresholding, level = 40

(c) Uniform thresholding, level = 10

FIGURE 4.20

Comparing hysteresis thresholding with uniform thresholding.

A comparison with the results of **uniform** thresholding is shown in Figure 4.20. Figure 4.20(a) shows the result of hysteresis thresholding of a Sobel edge-detected image of the eye with an upper threshold set to 40 brightness values and a lower threshold of 10 brightness values. Figure 4.20(b) and (c) shows the result of uniform thresholding applied to the image with thresholds of 40 and 10 brightness values, respectively. Uniform thresholding can select too few points if the threshold is too high, and too many if it is too low. Hysteresis thresholding naturally selects **all** the points as shown in Figure 4.20(b) and **some** of those as shown in Figure 4.20(c), those connected to the points in Figure 4.20(b). In particular, part of the nose is partly present in Figure 4.20(a), whereas it is absent in Figure 4.20(b) and masked by too many edge points in Figure 4.20(c). Also, the eyebrow is more complete in Figure 4.20(a) whereas it is only partial in Figure 4.20(b), and complete (but obscured) in Figure 4.20(c). Hysteresis thresholding therefore has an ability to detect major features of interest in the edge image, in an improved manner to uniform thresholding.

The action of the Canny operator on a larger image is shown in Figure 4.21, in comparison with the result of the Sobel operator. Figure 4.21(a) is the original image of a face, Figure 4.21(b) is the result of the Canny operator (using a 5×5 Gaussian operator with $\sigma = 1.0$ and with upper and lower thresholds set appropriately), and Figure 4.21(c) is the result of a 3×3 Sobel operator with uniform thresholding. The retention of major detail by the Canny operator is very clear; the face is virtually recognizable in Figure 4.21(b), whereas it is less clear in Figure 4.21(c).

4.2.2 Second-order edge-detection operators

4.2.2.1 Motivation

First-order edge detection is based on the premise that differentiation highlights change; image intensity changes in the region of a feature boundary. The process

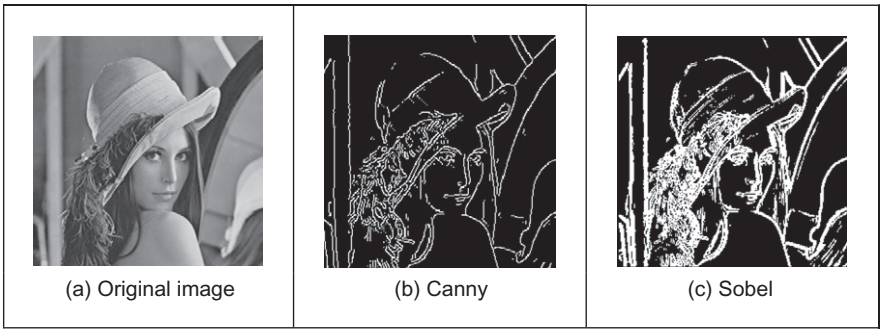


FIGURE 4.21

Comparing Canny with Sobel.

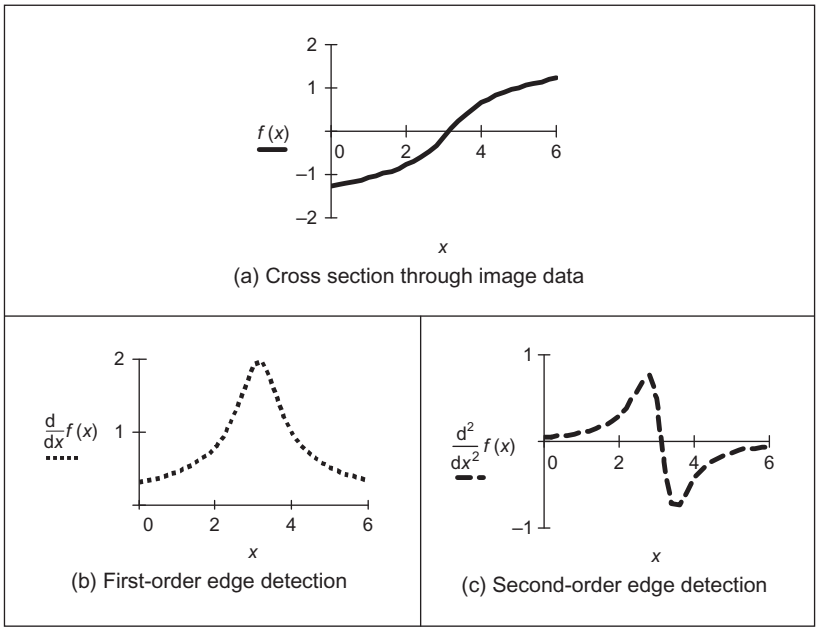


FIGURE 4.22

First- and second-order edge detection.

is illustrated in Figure 4.22 where Figure 4.22(a) is a cross section through image data. The result of **first-order** edge detection, $f'(x) = \mathrm{d}f/\mathrm{d}x$ in Figure 4.22(b), is a **peak** where the rate of change of the original signal, $f(x)$ in Figure 4.22(a), is greatest. There are of course higher order derivatives; applied to the same cross section of data, the **second-order** derivative, $f''(x) = \mathrm{d}^2f/\mathrm{d}x^2$) in Figure 4.22(c), is

FIGURE 4.23

Horizontal second-order template.

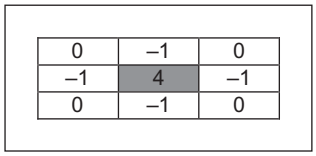


FIGURE 4.24

Laplacian edge detection operator.

greatest where the rate of change of the signal is greatest and zero when the rate of change is constant. The rate of change is constant at the peak of the first-order derivative. This is where there is a **zero crossing** in the second-order derivative, where it changes sign. Accordingly, an alternative to first-order differentiation is to apply second-order differentiation and then find zero crossings in the second-order information.

4.2.2.2 Basic operators: the Laplacian

The *Laplacian operator* is a template which implements second-order differencing. The second-order differential can be approximated by the difference between two adjacent first-order differences:

$$f''(x) \cong f'(x) - f'(x+1) \tag{4.24}$$

which, by Eq. (4.6), gives

$$f''(x+1) \cong -f(x) + 2f(x+1) - f(x+2) \tag{4.25}$$

This gives a horizontal second-order template as shown in Figure 4.23.

When the horizontal second-order operator is combined with a vertical second-order difference we obtain the full Laplacian template, as shown in Figure 4.24. Essentially, this computes the difference between a point and the average of its four direct neighbors. This was the operator used earlier in anisotropic diffusion, Section 3.5.3, where it is an approximate solution to the heat equation.

Application of the Laplacian operator to the image of the square is given in Figure 4.25. The original image is provided in numeric form in Figure 4.25(a). The detected edges are the **zero crossings** in Figure 4.25(b) and can be seen to

$$p = \begin{bmatrix} 1 & 2 & 3 & 4 & 1 & 1 & 2 & 1 \\ 2 & 2 & 3 & 0 & 1 & 2 & 2 & 1 \\ 3 & 0 & 38 & 39 & 37 & 36 & 3 & 0 \\ 4 & 1 & 40 & 44 & 41 & 42 & 2 & 1 \\ 1 & 2 & 43 & 44 & 40 & 39 & 1 & 3 \\ 2 & 0 & 39 & 41 & 42 & 40 & 2 & 0 \\ 1 & 2 & 0 & 2 & 2 & 3 & 1 & 1 \\ 0 & 2 & 1 & 3 & 1 & 0 & 4 & 2 \end{bmatrix}$$

(a) Image data

$$L = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -31 & -47 & -36 & -32 & 0 & 0 \\ 0 & -44 & 70 & 37 & 31 & 60 & -28 & 0 \\ 0 & -42 & 34 & 12 & 1 & 50 & -39 & 0 \\ 0 & -37 & 47 & 8 & -6 & 33 & -42 & 0 \\ 0 & -45 & 72 & 37 & 45 & 74 & -34 & 0 \\ 0 & 5 & -44 & -38 & -40 & -31 & -6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(b) After Laplacian operator

FIGURE 4.25

Edge detection via the Laplacian operator.

lie between the edge of the square and its background. The result highlights the boundary of the square in the original image, but there is also a slight problem: there is a small hole in the shape in the lower right. This is by virtue of second-order differentiation, which is inherently more susceptible to noise. Accordingly, to handle noise we need to introduce smoothing.

An alternative structure to the template in Figure 4.24 is one where the central weighting is 8 and the neighbors are all weighted as −1. Naturally, this includes a different form of image information, so the effects are slightly different. (Essentially, this now computes the difference between a pixel and the average of its neighboring points, including the corners.) In both structures, the central weighting can be negative and that of the four or the eight neighbors can be positive, without loss of generality. Actually, it is important to ensure that the sum of template coefficients is zero, so that the edges are not detected in areas of uniform brightness. One advantage of the Laplacian operator is that it is **isotropic** (like the Gaussian operator): it has the same properties in each direction. However, as yet it contains **no** smoothing and will again respond to noise, more so than a first-order operator since it is differentiation of a higher order. As such, the Laplacian operator is rarely used in its basic form. Smoothing can use the averaging operator described earlier but a more optimal form is Gaussian smoothing. When this is incorporated with the Laplacian, we obtain a Laplacian of Gaussian (LoG) operator which is the basis of the Marr−Hildreth approach, to be considered next. A clear disadvantage with the Laplacian operator is that edge direction is **not** available. It does however impose low computational cost, which is its main advantage. Though interest in the Laplacian operator abated with rising interest in the Marr−Hildreth approach, a nonlinear Laplacian operator was developed (Vliet and Young, 1989) and shown to have good performance, especially in low-noise situations.

4.2.2.3 The Marr–Hildreth operator

The *Marr–Hildreth* approach (Marr and Hildreth, 1980) again uses Gaussian filtering. In principle, we require an image which is the second differential ∇^2 of a Gaussian operator $g(x,y)$ convolved with an image **P**. This convolution process can be separated as

$$\nabla^2(g(x,y) * \mathbf{P}) = \nabla^2(g(x,y)) * \mathbf{P} \tag{4.26}$$

Accordingly, we need to compute a template for ∇^2 $(g(x,y))$ and convolve this with the image. By further differentiation of Eq. (4.17), we achieve a LoG operator:

$$\begin{aligned}\nabla^2 g(x,y) &= \frac{\partial^2 g(x,y,\sigma)}{\partial x^2} U_x + \frac{\partial^2 g(x,y,\sigma)}{\partial y^2} U_y \\ &= \frac{\partial \nabla g(x,y,\sigma)}{\partial x} U_x + \frac{\partial \nabla g(x,y,\sigma)}{\partial y} U_y \\ &= \left(\frac{x^2}{\sigma^2} - 1\right) \frac{\mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}}}{\sigma^2} + \left(\frac{y^2}{\sigma^2} - 1\right) \frac{\mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}}}{\sigma^2} \\ &= \frac{1}{\sigma^2}\left(\frac{(x^2+y^2)}{\sigma^2} - 2\right) \mathrm{e}^{\frac{-(x^2+y^2)}{2\sigma^2}}\end{aligned} \tag{4.27}$$

This is the basis of the Marr–Hildreth operator. Equation (4.27) can be used to calculate the coefficients of a template which, when convolved with an image, combines Gaussian smoothing with second-order differentiation. The operator is sometimes called a "Mexican hat" operator, since its surface plot is the shape of a sombrero, as illustrated in Figure 4.26.

The calculation of the LoG can be approximated by the *difference of Gaussian* where the difference is formed from the result of convolving two Gaussian filters with differing variance (Marr, 1982; Lindeberg, 1994):

$$\sigma\nabla^2 g(x,y,\sigma) = \frac{\partial g}{\partial \sigma} \approx \frac{g(x,y,k\sigma) - g(x,y,\sigma)}{k\sigma - \sigma} \tag{4.28}$$

where $g(x,y,\sigma)$ is the Gaussian function and k is a constant. Although similarly named, the *derivative of Gaussian*, Eq. (4.17), is a **first**-order operator including Gaussian smoothing, $\nabla g(x,y)$. It does actually seem counterintuitive that the difference of two smoothing operators should lead to second-order edge detection. The approximation is illustrated in Figure 4.27 where in 1D two Gaussian distributions of different variance are subtracted to form a 1D operator whose cross section is equivalent to the shape of the LoG operator (a cross section of Figure 4.26).

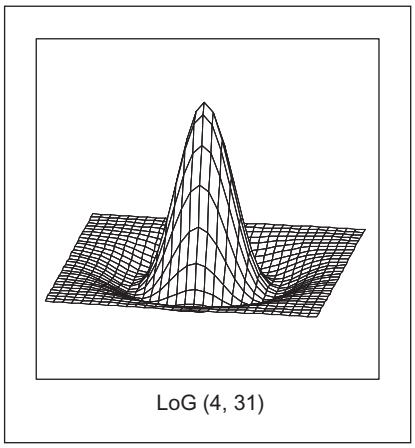


FIGURE 4.26

Shape of LoG operator.

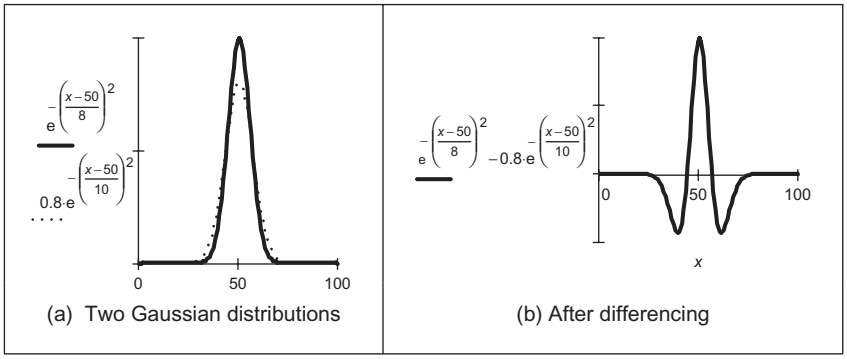


FIGURE 4.27

Approximating the LoG by difference of Gaussian.

The implementation of Eq. (4.27) to calculate template coefficients for the `LoG` operator is given in Code 4.14. The function includes a normalization function which ensures that the sum of the template coefficients is unity, so that edges are not detected in area of uniform brightness. This is in contrast with the earlier Laplacian operator (where the template coefficients summed to zero) since the LoG operator includes smoothing within the differencing action, whereas the Laplacian is pure differencing. The template generated by this function can then be used within template convolution. The Gaussian operator again suppresses the influence of points away from the center of the template, basing

differentiation on those points nearer the center; the standard deviation, σ, is chosen to ensure this action. Again, it is isotropic consistent with Gaussian smoothing.

```
LoG(σ,size):= | cx← (size-1)/2
              | cy← (size-1)/2
              | for x∈0.. size-1
              |   for y∈0.. size-1
              |     | nx←x-cx
              |     | ny←y-cy
              |     | template_{y,x}← (1/σ^2)·((nx^2+ny^2)/σ^2 - 2).e^(-((nx^2+ny^2)/(2.σ^2)))
              | template←normalize(template)
              | template
```

CODE 4.14

Implementation of the LoG operator.

Determining the zero-crossing points is a major difficulty with this approach. There is a variety of techniques which can be used, including manual determination of zero crossing or a least squares fit of a plane to local image data, which is followed by the determination of the point at which the plane crosses zero, if it does. The former is too simplistic, whereas the latter is quite complex (see Section 11.2, Appendix 2).

The approach here is much simpler: given a local 3×3 area of an image, this is split into quadrants. These are shown in Figure 4.28, where each quadrant contains the center pixel. The first quadrant contains the four points in the upper left corner and the third quadrant contains the four points in the upper right. If the average of the points in any quadrant differs in sign from the average in any other

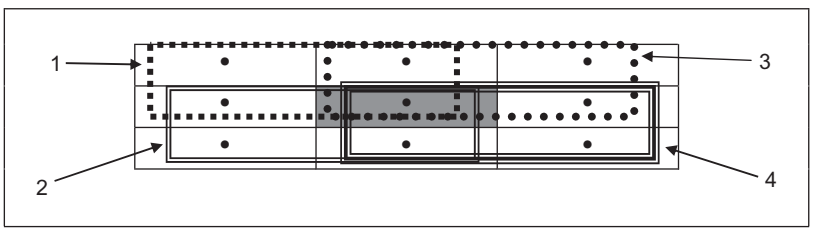


FIGURE 4.28

Regions for zero-crossing detection.

quadrant, there must be a zero crossing at the center point. In `zerox`, (Code 4.15), the average intensity in each quadrant is then evaluated, giving four values `int0`, `int1`, `int2`, and `int3`. If the maximum value of these points is positive, and the minimum value is negative, there must be a zero crossing within the neighborhood. If one exists, the output image at that point is marked as white, otherwise it is set to black.

```
zerox(pic):=|newpic←zero(pic)
            |for x∈1.. cols(pic)-2
            |   for y∈1.. rows(pic)-2
            |      int0← Σ(x1=x-1..x) Σ(y1=y-1..y) pic[y1,x1]
            |      int1← Σ(x1=x-1..x) Σ(y1=y..y+1) pic[y1,x1]
            |      int2← Σ(x1=x..x+1) Σ(y1=y-1..y) pic[y1,x1]
            |      int3← Σ(x1=x..x+1) Σ(y1=y..y+1) pic[y1,x1]
            |      maxval←max(int)
            |      minval←min(int)
            |      newpic[y,x] ←255 if (maxval>0)·(minval<0)
            |
            |newpic
```

CODE 4.15

Zero-crossing detector.

The action of the Marr−Hildreth operator is shown in Figure 4.29, applied to the face image as shown in Figure 4.21(a). The output of the `LoG` operator is hard to interpret visually and is not shown here (remember that it is the zero crossings which mark the edge points and it is hard to see them). The detected zero crossings (for a 3×3 neighborhood) are shown in Figure 4.29(b) and (c) for LoG operators of size 11×11 with $\sigma = 1.12$ and 15×15 with $\sigma = 2.3$, respectively. These show that the selection of window size and variance can be used to provide edges at differing scales. Some of the smaller regions as shown in Figure 4.29(b) join to form larger regions as shown in Figure 4.29(c). Note that one virtue of the Marr−Hildreth operator is its ability to provide **closed** edge borders which the Canny operator cannot. Another virtue is that it avoids the recursion associated with hysteresis thresholding that can require a massive stack size for large images.

The Fourier Transform of a LoG operator is shown in relief in Figure 4.30(a) and as a contour plot in Figure 4.30(b). The transform is circular−symmetric, as expected. Since the transform reveals that the LoG operator omits low and high frequencies (those close to the origin and those far away from the origin), it is

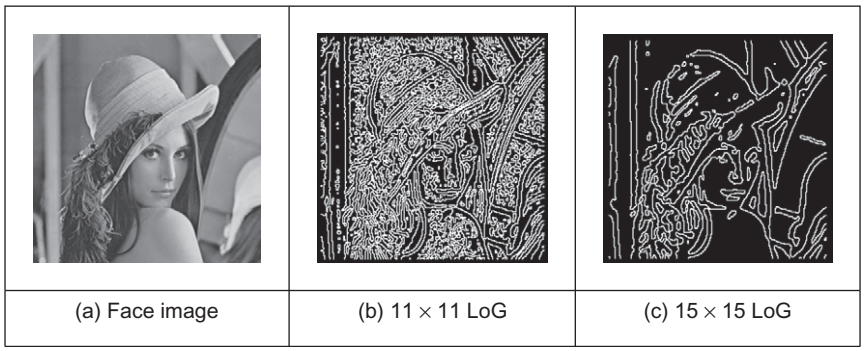


FIGURE 4.29

Marr–Hildreth edge detection.

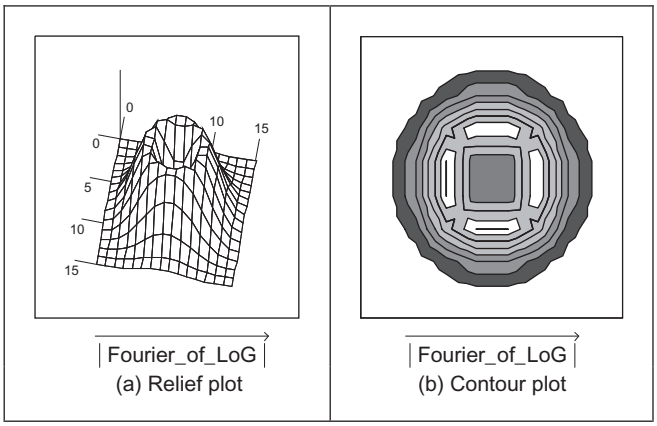


FIGURE 4.30

Fourier transform of LoG operator.

equivalent to a **band-pass filter**. Choice of the value of σ controls the spread of the operator in the spatial domain and the "width" of the band in the frequency domain: setting σ to a high value gives low-pass filtering, as expected. This differs from first-order edge-detection templates which offer a **high-pass** (differencing) filter along one axis with a **low-pass** (smoothing) action along the other axis.

The Marr–Hildreth operator has stimulated much attention, perhaps in part, because it has an appealing relationship to human vision and its ability for multi-resolution analysis (the ability to detect edges at differing scales). In fact, it has been suggested that the original image can be reconstructed from the zero crossings at different scales. One early study (Haralick, 1984) concluded that the Marr–Hildreth operator could give good performance. Unfortunately, the

implementation appeared to be different from the original LoG operator (and has actually appeared in some texts in this form) as noted by one of the Marr–Hildreth study's originators (Grimson and Hildreth, 1985). This led to a somewhat spirited reply (Haralick, 1985) not only clarifying concern but also raising issues about the nature and operation of edge-detection schemes which remain relevant today. Given the requirement for convolution of large templates, attention quickly focused on frequency domain implementation (Huertas and Medioni, 1986), and speed improvement was later considered in some detail (Forshaw, 1988). Later, schemes were developed to refine the edges produced via the LoG approach (Ulupinar and Medioni, 1990). Though speed and accuracy are major concerns with the Marr–Hildreth approach, it is also possible for zero-crossing detectors to mark as edge points ones which have no significant contrast, motivating study of their authentication (Clark, 1989). Gunn (1999) studied the relationship between mask size of the LoG operator and its error rate. Essentially, an acceptable error rate defines a truncation error which in turn gives an appropriate mask size. Gunn (1999) also observed the paucity of studies on zero-crossing detection and offered a detector slightly more sophisticated than the one here (as it includes the case where a zero crossing occurs at a boundary whereas the one here assumes that the zero crossing can only occur at the center). The similarity is not coincidental: Mark developed the one here after conversations with Steve Gunn, who he works with!

4.2.3 Other edge-detection operators

There have been many approaches to edge detection. This is not surprising since it is often the first stage in a vision process. The most popular are the Sobel, Canny, and Marr–Hildreth operators. Clearly, in any implementation, there is a **compromise** between (computational) cost and efficiency. In some cases, it is difficult to justify the extra complexity associated with the Canny and the Marr–Hildreth operators. This is in part due to the images: few images contain the adverse noisy situations that complex edge operators are designed to handle. Also, when finding shapes, it is often prudent to extract more than enough low-level information and to let the more sophisticated shape detection process use, or discard, the information as appropriate. For these reasons we will study only two more edge-detection approaches and only briefly. They are the *Spacek* and the *Petrou* operators: both are designed to be optimal and both have different properties and a different basis (the **smoothing** functional in particular) to the Canny and Marr–Hildreth approaches. The Spacek and Petrou operators are included by virtue of their optimality. Essentially, while Canny maximized the ratio of the signal-to-noise ratio with the localization, Spacek (1986) maximized the ratio of the product of the signal-to-noise ratio and the peak separation with the localization. In Spacek's work, since the edge was again modeled as a step function, the ideal filter appeared to be of the same form as Canny's. Spacek's operator can give better performance than Canny's formulation (Jia and Nixon, 1995), as such

challenging the optimality of the Gaussian operator for noise smoothing (in step-edge detection), though such advantage should be explored in application.

Petrou and Kittler (1991) questioned the validity of the step-edge model for real images. Given that the composite performance of an image acquisition system can be considered to be that of a low-pass filter, any step changes in the image will be smoothed to become a ramp. As such, a more plausible model of the edge is a ramp rather than a step. Since the process is based on ramp edges, and because of limits imposed by its formulation, the Petrou operator uses templates that are much wider in order to preserve optimal properties. As such, the operator can impose greater computational complexity but is a natural candidate for applications with the conditions for which its properties were formulated.

Of the other approaches, Korn (1988) developed a unifying operator for symbolic representation of gray level change. The *Susan* operator (Smith and Brady, 1997) derives from an approach aimed to find more than just edges since it can also be used to derive *corners* (where feature boundaries change direction sharply, as in *curvature* detection in Section 4.4.1) and structure-preserving image noise reduction. Essentially, SUSAN derives from Smallest Univalue Segment Assimilating Nucleus which concerns aggregating the difference between elements in a (circular) template centered on the nucleus. The USAN is essentially the number of pixels within the circular mask which have similar brightness to the nucleus. The edge strength is then derived by subtracting the USAN size from a geometric threshold, which is say ¾ of the maximum USAN size. The method includes a way of calculating edge direction, which is essential if nonmaximum suppression is to de applied. The advantages are in simplicity (and hence speed) since it is based on simple operations and the possibility of extension to find other feature types.

4.2.4 Comparison of edge-detection operators

Naturally, the selection of an edge operator for a particular application depends on the application itself. As has been suggested, it is not usual to require the sophistication of the advanced operators in many applications. This is reflected in analysis of the performance of the edge operators on the eye image. In order to provide a different basis for comparison, we shall consider the difficulty of low-level feature extraction in ultrasound images. As has been seen earlier (Section 3.5.5), ultrasound images are very **noisy** and require **filtering** prior to analysis. Figure 4.31(a) is part of the ultrasound image which could have been filtered using the truncated median operator (Section 3.5.2). The image contains a feature called the pitus (it's the "splodge" in the middle), and we shall see how different edge operators can be used to detect its perimeter, though without noise filtering. The median is a very popular filtering process for general (i.e., nonultrasound) applications. Accordingly, it is of interest that one study (Bovik et al., 1987) has suggested that the known advantages of median filtering (the removal

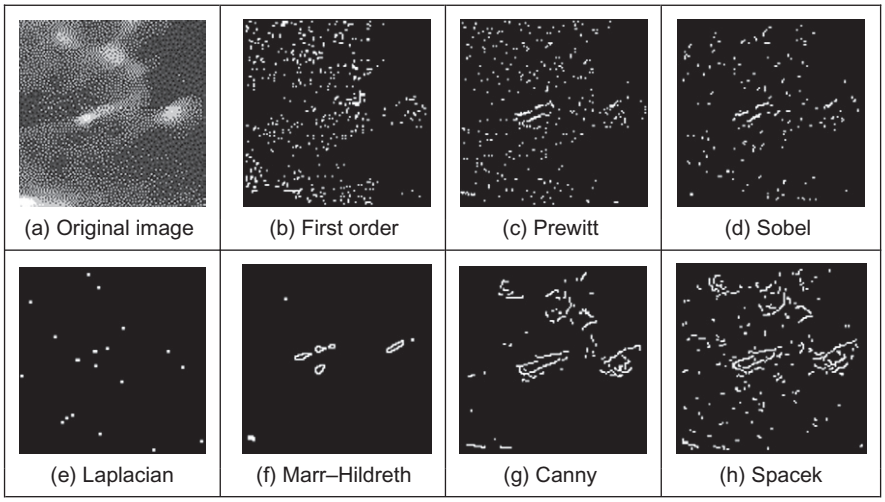


FIGURE 4.31

Comparison of edge-detection operators.

of noise with the preservation of edges, especially for salt and pepper noise) are shown to good effect if it is used as a prefilter to first- and second-order approaches, though naturally with the cost of the median filter. However, we will not consider median filtering here: its choice depends more on suitability to a particular application.

The results for all edge operators have been generated using hysteresis thresholding where the thresholds were selected manually for best performance. The basic first-order operator (Figure 4.31(b)) responds rather nicely to the noise and it is difficult to select a threshold which reveals a major part of the pitus border. Some is present in the Prewitt (Figure 4.31(c)) and Sobel (Figure 4.31(d)) operators' results, but there is still much noise in the processed image, though there is less in the Sobel. The Laplacian operator (Figure 4.31(e)) gives very little information indeed, as to be expected with such noisy imagery. However, the more advanced operators can be used to good effect. The Marr−Hildreth approach improves matters (Figure 4.31(f)), but suggests that it is difficult to choose a LoG operator of appropriate size to detect a feature of these dimensions in such noisy imagery—illustrating the compromise between the size of operator needed for noise filtering and the size needed for the target feature. However, the Canny and Spacek operators can be used to good effect, as shown in Figure 4.31(g) and (h), respectively. These reveal much of the required information, together with data away from the pitus itself. In an automated analysis system, for this application, the extra complexity of the more sophisticated operators would clearly be warranted.

4.2.5 Further reading on edge detection

Few computer vision and image processing texts omit detail concerning edge-detection operators, though few give explicit details concerning implementation. Naturally, many of the earlier texts omit the more recent techniques. Further information can be found in journal papers; Petrou's excellent study of edge detection (Petrou, 1994) highlights the study of the performance factors involved in the optimality of the Canny, Spacek, and Petrou operators with extensive tutorial support (though I suspect Petrou junior might one day be embarrassed by the frequency his youthful mugshot is used—his teeth show up very well!). There have been a number of surveys of edge detection highlighting performance attributes in comparison. For example, see Torre and Poggio (1986) that gives a theoretical study of edge detection and considers some popular edge-detection techniques in light of this analysis. One survey (Heath et al., 1997) surveys many approaches comparing them in particular with the Canny operator (and states where code for some of the techniques they compared can be found). This showed that best results can be achieved by tuning an edge detector for a particular application and highlighted good results by the Bergholm operator (Bergholm, 1987). Marr (1982) considers the Marr−Hildreth approach to edge detection in the light of human vision (and its influence on perception), with particular reference to scale in edge detection. More recently Yitzhaky and Peli (2003) suggests "a general tool to assist in practical implementations of parametric edge detectors where an automatic process is required" and uses statistical tests to evaluate edge-detector performance. Since edge detection is one of the most important vision techniques, it continues to be a focus of research interest. Accordingly, it is always worth looking at recent papers to find new techniques, or perhaps more likely performance comparison or improvement, that might help you solve a problem.

4.3 Phase congruency

The comparison of edge detectors highlights some of their innate problems: incomplete contours, the need for selective thresholding, and their response to noise. Further, the selection of a threshold is often inadequate for all the regions in an image since there are many changes in local illumination. We shall find that some of these problems can be handled at a higher level, when shape extraction can be arranged to accommodate partial data and to reject spurious information. There is though natural interest in refining the low-level feature extraction techniques further.

Phase congruency is a feature detector with two main advantages: it can detect a **broad range** of features and it is **invariant to** local (and smooth) **change in illumination**. As the name suggests, it is derived by frequency domain considerations operating on the considerations of phase (aka time). It is illustrated detecting some 1D features in Figure 4.32 where the features are the solid lines: a

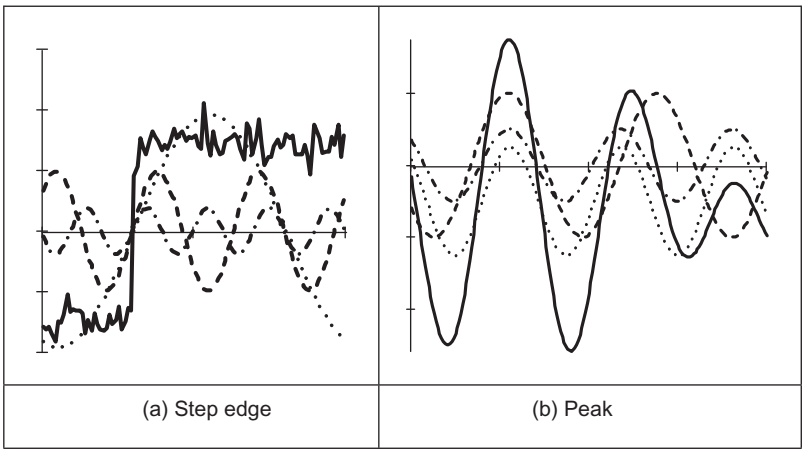


FIGURE 4.32

Low-level feature extraction by phase congruency.

(noisy) **step** function in Figure 4.32(a) and a peak (or impulse) in Figure 4.32(b). By Fourier transform analysis, any function is made up from the controlled addition of sinewaves of differing frequencies. For the step function to occur (the solid line in Figure 4.32(a)), the constituent frequencies (the dotted lines in Figure 4.32(a)) must all change at the same time, so they add up to give the edge. Similarly, for the peak to occur, the constituent frequencies must all peak at the same time; in Figure 4.32(b) the solid line is the peak and the dotted lines are some of its constituent frequencies. This means that in order to find the feature we are interested in, we can determine points where events happen at the same time: this is phase congruency. By way of generalization, a triangle wave is made of peaks and troughs: phase congruency implies that the peaks and troughs of the constituent signals should coincide.

In fact, the constituent sinewaves plotted in Figure 4.32(a) were derived by taking the Fourier transform of a step and then determining the sinewaves according to their magnitude and phase. The Fourier transform in Eq. (2.15) delivers the complex Fourier components **Fp**. These can be used to show the constituent signals *xc* by

$$xc(t) = |\mathbf{Fp}_u| e^{j\left(\frac{2\pi}{N}ut + \phi(\mathbf{FP}_u)\right)} \tag{4.29}$$

where $|\mathbf{Fp}_u|$ is again the magnitude of the uth Fourier component (Eq. (2.7)) and $\phi(\mathbf{FP}_u) = \langle \mathbf{FP}_u$ is the argument, the phase in Eq. (2.8). The (dotted) frequencies displayed in Figure 4.32 are the first four odd components (the even components for this function are zero, as shown in the Fourier transform of the step in Figure 2.11). The addition of these components is indeed the inverse Fourier transform which reconstructs the step feature.

(a) Modified cameraman image

(b) Edges by the Canny operator

(c) Phase congruency

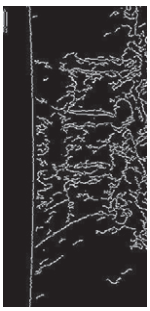
FIGURE 4.33

Edge detection by Canny and by phase congruency.

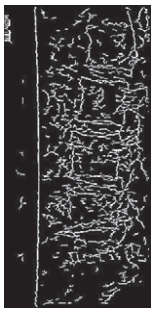
The advantages are that detection of congruency is invariant with local contrast: the sinewaves still add up so the changes are still in the same place, even if the magnitude of the step edge is much smaller. In images, this implies that we can change the contrast and still detect edges. This is illustrated in Figure 4.33. Here, a standard image processing image, the "cameraman" image from the early UCSD dataset, has been changed between the left and right sides so that the contrast changes in the two halves of the image (Figure 4.33(a)). Edges detected by Canny are shown in Figure 4.33(b) and by phase congruency in Figure 4.33(c). The basic structure of the edges detected by phase congruency is very similar to that structure detected by Canny, and the phase congruency edges appear somewhat cleaner (there is a single line associated with the tripod control in phase congruency); both detect the change in brightness between the two halves. There is a major difference though: the building in the lower right side of the image is barely detected in the Canny image whereas it can clearly be seen in phase congruency image. Its absence is due to the parameter settings used in the Canny operator. These can be changed, but if the contrast were to change again, then the parameters would need to be reoptimized for the new arrangement. This is not the case for phase congruency.

Naturally such a change in brightness might appear unlikely in practical application, but this is not the case with moving objects which interact with illumination or in fixed applications where illumination changes. In studies aimed to extract spinal information from digital videofluoroscopic X-ray images in order to provide guidance for surgeons (Zheng et al., 2004), phase congruency was found to be immune to the changes in contrast caused by slippage of the shield used to protect the patient while acquiring the image information. One such image is shown in Figure 4.34. The lack of shielding is apparent in the bloom at the side of the images. This changes as the subject is moved, so it proved difficult to optimize the parameters for Canny over the whole sequence (Figure 4.34(b)) but the

(a) Digital videofluoroscopic image of lower spine showing vertebrae



(b) Edges by the Canny operator



(c) Features by phase congruency

FIGURE 4.34

Spinal contour by phase congruency (Zheng et al., 2004).

detail of a section of the phase congruency result (Figure 4.34(c)) shows that the vertebrae information is readily available for later high-level feature extraction.

The original notions of phase congruency are the concepts of *local energy* (Morrone and Owens, 1987), with links to the human visual system (Morrone and Burr, 1988). One of the most sophisticated implementations was by Kovesi (1999), with added advantage that his Matlab implementation is available on the Web (http://www.csse.uwa.edu.au/~pk/Research/research.html) as well as much more information. Essentially, we seek to determine features by detection of points at which Fourier components are maximally in phase. By extension of the Fourier reconstruction functions in Eq. (4.29), Morrone and Owens (1987) defined a measure of phase congruency, PC, as

$$\mathrm{PC}(x) = \max_{\overline{\phi}(x)\in 0,2\pi}\left(\frac{\sum_{u}|\mathbf{Fp}_u|\cos(\phi_u(x)-\overline{\phi}(x))}{\sum_{u}|\mathbf{Fp}_u|}\right) \tag{4.30}$$

where $\phi_u(x)$ represents the local phase of the component $\mathbf{Fp}_u$ at position x. Essentially, this computes the ratio of the sum of projections onto a vector (the sum in the numerator) to the total vector length (the sum in the denominator). The value of $\overline{\phi}(x)$ that maximizes this equation is the amplitude weighted mean local phase angle of all the Fourier terms at the point being considered. In Figure 4.35 the resulting vector is made up of four components, illustrating the projection of the second onto the resulting vector. Clearly, the value of phase congruency ranges from 0 to 1, the maximum occurring when all elements point along the resulting vector. As such, the resulting phase congruency is a **dimensionless normalized measure** which is thresholded for image analysis.

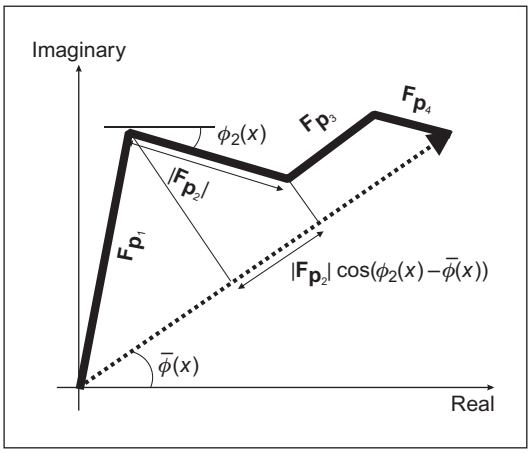


FIGURE 4.35

Summation in phase congruency.

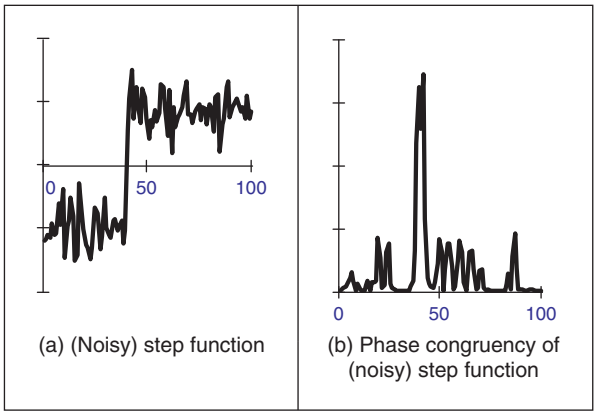


FIGURE 4.36

1D phase congruency.

In this way, we have calculated the phase congruency for the step function in Figure 4.36(a), which is shown in Figure 4.36(b). Here, the position of the step is at time step 40; this is the position of the peak in phase congruency, as required. Note that the noise can be seen to affect the result, though the phase congruency is largest at the right place.

One interpretation of the measure is that since for small angles $\cos\theta = 1 - \theta^2$ Eq. (4.30) expresses the ratio of the magnitudes weighted by the variance of the difference to the summed magnitude of the components. There is certainly

difficulty with this measure, apart from difficulty in implementation: it is sensitive to **noise**, as is any phase measure; it is not **conditioned** by the magnitude of a response (small responses are not discounted); and it is not well **localized** (the measure varies with the cosine of the difference in phase, not with the difference itself—though it does avoid discontinuity problems with direct use of angles). In fact, the phase congruency is directly proportional to the local energy (Venkatesh and Owens, 1989), so an alternative approach is to search for maxima in the local energy. The notion of local energy allows us to compensate for the sensitivity to the detection of phase in noisy situations.

For these reasons, Kovesi (1999) developed a wavelet-based measure which improved performance, while accommodating noise. In basic form, phase congruency can be determined by convolving a set of wavelet filters with an image and calculating the difference between the average filter response and the individual filter responses. The response of a (1D) signal I to a set of wavelets at scale n is derived from the convolution of the cosine and sine wavelets (discussed in Section 2.7.3) denoted as M_n^e and M_n^o, respectively,

$$[e_n(x), o_n(x)] = [I(x) * M_n^e, I(x) * M_n^o] \tag{4.31}$$

to deliver the even and odd components at the nth scale $e_n(x)$ and $o_n(x)$, respectively. The amplitude of the transform result at this scale is the local energy,

$$A_n(x) = \sqrt{e_n(x)^2 + o_n(x)^2} \tag{4.32}$$

At each point x we will have an array of vectors that correspond to each scale of the filter. Given that we are interested only in phase congruency that occurs over a wide range of frequencies (rather than just at a couple of scales), the set of wavelet filters needs to be designed so that adjacent components overlap. By summing the even and odd components we obtain:

$$\begin{aligned} F(x) &= \sum_n e_n(x) \\ H(x) &= \sum_n o_n(x) \end{aligned} \tag{4.33}$$

and a measure of the total energy A as

$$\sum_n A_n(x) \approx \sum_n \sqrt{e_n(x)^2 + o_n(x)^2} \tag{4.34}$$

then a measure of phase congruency is

$$\mathrm{PC}(x) = \frac{\sqrt{F(x)^2 + H(x)^2}}{\sum_n A_n(x) + \varepsilon} \tag{4.35}$$

where the addition of a small factor ε in the denominator avoids division by zero and any potential result when values of the numerator are very small. This gives

a measure of phase congruency, which is essentially a measure of the local energy. Kovesi (1999) improved on this, improving on the response to noise, developing a measure which reflects the confidence that the signal is significant relative to the noise. Further, he considers in detail the frequency domain considerations, and its extension to 2D (Kovesi, 1999). For 2D (image) analysis, given that phase congruency can be determined by convolving a set of wavelet filters with an image and calculating the difference between the average filter response and the individual filter responses. The filters are constructed in the frequency domain by using complementary spreading functions; the filters must be constructed in the Fourier domain because the log-Gabor function has a singularity at zero frequency. In order to construct a filter with appropriate properties, a filter is constructed in a manner similar to the Gabor wavelet, but here in the frequency domain and using different functions. Following Kovesi's implementation, the first filter is a low-pass filter, here a Gaussian filter g with L different orientations

$$g(\theta, \theta_l) = \frac{1}{\sqrt{2\pi}\sigma_s} \mathrm{e}^{-\frac{(\theta-\theta_l)^2}{2\sigma_s^2}} \tag{4.36}$$

where θ is the orientation, σ_s controls the spread about that orientation, and θ_l is the angle of local orientation focus. The other spreading function is a band-pass filter, here a log-Gabor filter lg with M different scales.

$$\mathrm{lg}(\omega, \omega_m) = \begin{cases} 0 & \omega = 0 \\ \dfrac{1}{\sqrt{2\pi}\sigma_\beta} \mathrm{e}^{-\frac{\left(\log\left(\omega/\omega_m\right)\right)^2}{2(\log(\beta))^2}} & \omega \neq 0 \end{cases} \tag{4.37}$$

where ω is the scale, β controls bandwidth at that scale, and ω is the center frequency at that scale. The combination of these functions provides a 2D filter l2Dg which can act at different scales and orientations.

$$\mathrm{l2Dg}(\omega, \omega_m, \theta, \theta_l) = g(\theta, \theta_l) \times \mathrm{lg}(\omega, \omega_m) \tag{4.38}$$

One measure of phase congruency based on the convolution of this filter with the image $\mathbf{P}$ is derived by inverse Fourier transformation $\Im^{-1}$ of the filter l2Dg (to yield a spatial domain operator) which is convolved as

$$S(m)_{x,y} = \Im^{-1}(\mathrm{l2Dg}(\omega, \omega_m, \theta, \theta_l))_{x,y} * \mathbf{P}_{x,y} \tag{4.39}$$

to deliver the convolution result S at the mth scale. The measure of phase congruency over the M scales is then

$$\mathrm{PC}_{x,y} = \frac{\left|\sum_{m=1}^{M} S(m)_{x,y}\right|}{\sum_{m=1}^{M} |S(m)_{x,y}| + \varepsilon} \tag{4.40}$$

where the addition of a small factor ε again avoids division by zero and any potential result when values of S are very small. This gives a measure of phase congruency, but is certainly a bit of an ouch, especially as it still needs refinement.

Note that key words reoccur within phase congruency: frequency domain, wavelets, and convolution. By its nature, we are operating in the frequency domain and there is not enough room in this text, and it is inappropriate to the scope here, to expand further. Despite this, the performance of phase congruency certainly encourages its consideration, especially if local illumination is likely to vary and if a range of features is to be considered. It is derived by an alternative conceptual basis, and this gives different insight, let alone performance. Even better, there is a Matlab implementation available, for application to images—allowing you to replicate its excellent results. There has been further research, noting especially its extension in ultrasound image analysis (Mulet-Parada and Noble, 2000) and its extension to spatiotemporal form (Myerscough and Nixon, 2004).

4.4 Localized feature extraction

There are two main areas covered here. The traditional approaches aim to derive local features by measuring specific image properties. The main target has been to estimate curvature: peaks of local curvature are corners and analyzing an image by its corners is especially suited to image of man-made objects. The second area includes more modern approaches that improve performance by employing region or patch-based analysis. We shall start with the more established curvature-based operators, before moving to the patch or region-based analysis.

4.4.1 Detecting image curvature (corner extraction)

4.4.1.1 Definition of curvature

Edges are perhaps the low-level image features that are most obvious to human vision. They preserve significant features, so we can usually recognize what an image contains from its edge-detected version. However, there are **other** low-level features that can be used in computer vision. One important feature is *curvature*. Intuitively, we can consider curvature as the rate of change in edge direction. This rate of change characterizes the points in a curve; points where the edge direction changes rapidly are *corners*, whereas points where there is little change in edge direction correspond to straight lines. Such extreme points are very useful for shape description and matching, since they represent significant information with reduced data.

Curvature is normally defined by considering a parametric form of a planar curve. The parametric contour $v(t) = x(t)U_x + y(t)U_y$ describes the points in a continuous curve as the end points of the position vector. Here, the values of t define an arbitrary parameterization, the unit vectors are again $U_x = [1,0]$ and $U_y = [0,1]$. Changes in the position vector are given by the tangent vector function of the

curve $v(t)$. That is, $\dot{v}(t) = \dot{x}(t)U_x + \dot{y}(t)U_y$. This vectorial expression has a simple intuitive meaning. If we think of the trace of the curve as the motion of a point and t is related to time, the tangent vector defines the instantaneous motion. At any moment, the point moves with a speed given by $|\dot{v}(t)| = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)}$ in the direction $\varphi(t) = \tan^{-1}(\dot{y}(t)/\dot{x}(t))$. The curvature at a point $v(t)$ describes the changes in the direction $\varphi(t)$ with respect to changes in arc length, i.e.,

$$\kappa(t) = \frac{\mathrm{d}\varphi(t)}{\mathrm{d}s} \tag{4.41}$$

where s is arc length, along the edge itself. Here φ is the angle of the tangent to the curve. That is, $\varphi = \theta \pm 90^\circ$, where θ is the gradient direction defined in Eq. (4.13). That is, if we apply an edge detector operator to an image, then we have for each pixel a gradient direction value that represents the normal direction to each point in a curve. The tangent to a curve is given by an orthogonal vector. Curvature is given with respect to arc length because a curve parameterized by arc length maintains a constant speed of motion. Thus, curvature represents changes in direction for constant displacements along the curve. By considering the chain rule, we have

$$\kappa(t) = \frac{\mathrm{d}\varphi(t)}{\mathrm{d}t}\frac{\mathrm{d}t}{\mathrm{d}s} \tag{4.42}$$

The differential $\mathrm{d}s/\mathrm{d}t$ defines the change in arc length with respect to the parameter t. If we again consider the curve as the motion of a point, this differential defines the instantaneous change in distance with respect to time, i.e., the instantaneous speed. Thus,

$$\mathrm{d}s/\mathrm{d}t = |\dot{v}(t)| = \sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \tag{4.43}$$

and

$$\mathrm{d}t/\mathrm{d}s = 1 \Big/ \sqrt{\dot{x}^2(t) + \dot{y}^2(t)} \tag{4.44}$$

By considering that $\varphi(t) = \tan^{-1}(\dot{y}(t)/\dot{x}(t))$, then the curvature at a point $v(t)$ in Eq. (4.42) is given by

$$\kappa(t) = \frac{\dot{x}(t)\ddot{y}(t) - \dot{y}(t)\ddot{x}(t)}{[\dot{x}^2(t) + \dot{y}^2(t)]^{3/2}} \tag{4.45}$$

This relationship is called the *curvature function* and it is the standard measure of curvature for **planar** curves (Apostol, 1966). An important feature of curvature is that it relates the derivative of a tangential vector to a normal vector. This can be explained by the simplified Serret–Frenet equations (Goetz, 1970) as follows. We can express the tangential vector in polar form as

$$\dot{v}(t) = |\dot{v}(t)|(\cos(\varphi(t)) + \mathrm{j}\,\sin(\varphi(t))) \tag{4.46}$$

If the curve is parameterized by arc length, then $|\dot{v}(t)|$ is constant. Thus, the derivative of a tangential vector is simply given by

$$\ddot{v}(t) = |\dot{v}(t)|(-\sin(\varphi(t)) + \mathrm{j}\cos(\varphi(t)))(\mathrm{d}\varphi(t)/\mathrm{d}t) \tag{4.47}$$

Since we are using a normal parameterization, $\mathrm{d}\varphi(t)/\mathrm{d}t = \mathrm{d}\varphi(t)/\mathrm{d}s$. Thus, the tangential vector can be written as follows:

$$\ddot{v}(t) = \kappa(t)\mathbf{n}(t) \tag{4.48}$$

where $\mathbf{n}(t) = |v(t)|(-\sin(\varphi(t)) + \mathrm{j}\cos(\varphi(t)))$ defines the direction of $\ddot{v}(t)$ while the curvature $\kappa(t)$ defines its modulus. The derivative of the normal vector is given by $\dot{\mathbf{n}}(t) = |\dot{v}(t)|(-\cos(\varphi(t)) - \mathrm{i}\sin(\varphi(t)))(\mathrm{d}\varphi(t)/\mathrm{d}s)$ that can be written as

$$\dot{\mathbf{n}}(t) = -\kappa(t)\dot{v}(t) \tag{4.49}$$

Clearly $\mathbf{n}(t)$ is normal to $\dot{v}(t)$. Therefore, for each point in the curve, there is a pair of orthogonal vectors $\dot{v}(t)$ and $\mathbf{n}(t)$ whose moduli are proportionally related by the curvature.

Generally, the curvature of a parametric curve is computed by evaluating Eq. (4.45). For a straight **line**, for example, the second derivatives $\ddot{x}(t)$ and $\ddot{y}(t)$ are **zero**, so the curvature function is **nil**. For a **circle** of radius r, we have that $\dot{x}(t) = r\cos(t)$ and $\dot{y}(t) = -r\sin(t)$. Thus, $\ddot{y}(t) = -r\cos(t)$, $\ddot{x}(t) = -r\sin(t)$, and $\kappa(t) = 1/r$. However, for curves in digital images, the derivatives must be computed from discrete data. This can be done in four main ways. The most obvious approach is to calculate curvature by directly computing the difference between angular direction of successive edge pixels in a curve. A second approach is to derive a measure of curvature from changes in image intensity. Finally, a measure of curvature can be obtained by correlation.

4.4.1.2 Computing differences in edge direction

Perhaps the easier way to compute curvature in digital images is to measure the **angular change** along the curve's path. This approach was considered in early corner detection techniques (Bennet and MacDonald, 1975; Groan and Verbeek, 1978; Kitchen and Rosenfeld, 1982) and it merely computes the **difference** in edge **direction** between connected pixels forming a discrete curve. That is, it approximates the derivative in Eq. (4.41) as the difference between neighboring pixels. As such, curvature is simply given by

$$k(t) = \varphi_{t+1} - \varphi_{t-1} \tag{4.50}$$

where the sequence ..., φ_{t-1}, φ_t, φ_{t+1}, φ_{t+2}, ... represents the gradient direction of a sequence of pixels defining a curve segment. Gradient direction can be obtained as the angle given by an edge detector operator. Alternatively, it can be computed by considering the position of pixels in the sequence. That is, by defining $\varphi_t = (y_{t-1} - y_{t+1})/(x_{t-1} - x_{t+1})$ where (x_t, y_t) denotes pixel t in the sequence. Since edge points are only defined at discrete points, this angle can only take eight values, so the computed curvature is very ragged. This can be smoothed out

by considering the difference in mean angular direction of n pixels on the leading and trailing curve segment, i.e.,

$$k_n(t) = \frac{1}{n}\sum_{i=1}^{n} \varphi_{t+i} - \frac{1}{n}\sum_{i=-n}^{-1} \varphi_{t+i} \tag{4.51}$$

The average also gives some immunity to noise and it can be replaced by a weighted average if Gaussian smoothing is required. The number of pixels considered, the value of n, defines a compromise between accuracy and noise sensitivity. Notice that filtering techniques may also be used to reduce the quantization effect when angles are obtained by an edge-detection operator. As we have already discussed, the level of filtering is related to the size of the template (as in Section 3.4.3).

In order to compute angular differences, we need to determine connected edges. This can easily be implemented with the code already developed for hysteresis thresholding in the Canny edge operator. To compute the difference of points in a curve, the `connect` routine (Code 4.12) only needs to be arranged to store the difference in edge direction between connected points. Code 4.16 shows an implementation for curvature detection. First, edges and magnitudes are determined. Curvature is only detected at edge points. As such, we apply maximal suppression. The function `Cont` returns a matrix containing the connected neighbor

```
%Curvature detection
function outputimage=CurvConnect(inputimage)

   [rows,columns]=size(inputimage);  %Image size
   outputimage=zeros(rows,columns);  %Result image
   [Mag,Ang]=Edges(inputimage);%Edge Detection Magnitude and Angle
   Mag=MaxSupr(Mag,Ang);             %Maximal Suppression
   Next=Cont(Mag,Ang);               %Next connected pixels


   %Compute curvature in each pixel
   for x=1:columns-1
     for y=1:rows-1
       if Mag(y,x)~=0
         n=Next(y,x,1); m=Next(y,x,2);
          if(n~=-1 & m~=-1)
           [px,py]=NextPixel(x,y,n);
           [qx,qy]=NextPixel(x,y,m);
         outputimage(y,x)=abs(Ang(py,px)-Ang(qy,qx));
       end
     end
    end
   end
```

CODE 4.16

Curvature by differences.

pixels of each edge. Each edge pixel is connected to one or two neighbors. The matrix `Next` stores only the direction of consecutive pixels in an edge. We use a value of -1 to indicate that there is no connected neighbor. The function `NextPixel` obtains the position of a neighboring pixel by taking the position of a pixel and the direction of its neighbor. The curvature is computed as the difference in gradient direction of connected neighbor pixels.

The result of applying this form of curvature detection to an image is shown in Figure 4.37. Here Figure 4.37(a) contains the silhouette of an object; Figure 4.37(b) is the curvature obtained by computing the rate of change of edge direction. In this figure, curvature is defined only at the edge points. Here, by its formulation the measurement of curvature κ gives just a thin line of differences in edge direction which can be seen to track the perimeter points of the shapes (at points where there is measured curvature). The brightest points are those with greatest curvature. In order to show the results, we have scaled the curvature values to use 256 intensity values. The estimates of corner points could be obtained by a uniformly thresholded version of Figure 4.37(b), well in theory anyway!

Unfortunately, as can be seen, this approach does not provide reliable results. It is essentially a reformulation of a first-order edge-detection process and presupposes that the corner information lies within the threshold data (and uses no corner structure in detection). One of the major difficulties with this approach is that measurements of angle can be severely affected by **quantization error** and accuracy is **limited** (Bennet and MacDonald, 1975), a factor which will return to plague us later when we study the methods for describing shapes.

4.4.1.3 Measuring curvature by changes in intensity (differentiation)

As an alternative way of measuring curvature, we can derive the curvature as a function of **changes** in **image intensity**. This derivation can be based on the

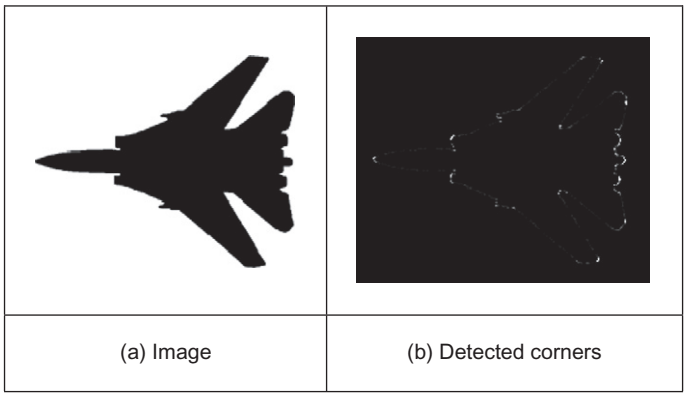


FIGURE 4.37

Curvature detection by difference.

measure of angular changes in the discrete image. We can represent the direction at each image point as the function $\varphi'(x,y)$. Thus, according to the definition of curvature, we should compute the change in these direction values normal to the image edge (i.e., along the curves in an image). The curve at an edge can be locally approximated by the points given by the parametric line defined by $x(t) = x + t\cos(\varphi'(x,y))$ and $y(t) = y + t\sin(\varphi'(x,y))$. Thus, the curvature is given by the change in the function $\varphi'(x,y)$ with respect to t, that is,

$$\kappa_{\varphi'}(x,y) = \frac{\partial \varphi'(x,y)}{\partial t} = \frac{\partial \varphi'(x,y)}{\partial x}\frac{\partial x(t)}{\partial t} + \frac{\partial \varphi'(x,y)}{\partial y}\frac{\partial y(t)}{\partial t} \tag{4.52}$$

where $\partial x(t)/\partial t = \cos(\varphi')$ and $\partial y(t)/\partial t = \sin(\varphi')$. By considering the definition of the gradient angle, the normal tangent direction at a point in a line is given by $\varphi'(x,y) = \tan^{-1}(Mx/(-My))$. From this geometry we can observe that

$$\cos(\varphi') = -My/\sqrt{Mx^2 + My^2} \quad \text{and} \quad \sin(\varphi') = Mx/\sqrt{Mx^2 + My^2} \tag{4.53}$$

By differentiation of $\varphi'(x,y)$ and by considering these definitions, we obtain:

$$\kappa_{\varphi'}(x,y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}}\left\{My^2\frac{\partial Mx}{\partial x} - MxMy\frac{\partial My}{\partial x} + Mx^2\frac{\partial My}{\partial y} - MxMy\frac{\partial Mx}{\partial y}\right\} \tag{4.54}$$

This defines a **forward** measure of curvature along the edge direction. We can actually use an alternative direction to measure of curvature. We can differentiate **backward** (in the direction of $-\varphi'(x,y)$) giving $\kappa_{-\varphi'}(x,y)$. In this case we consider that the curve is given by $x(t) = x + t\cos(-\varphi'(x,y))$ and $y(t) = y + t\sin(-\varphi'(x,y))$. Thus,

$$\kappa_{-\varphi'}(x,y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}}\left\{My^2\frac{\partial Mx}{\partial x} - MxMy\frac{\partial My}{\partial x} - Mx^2\frac{\partial My}{\partial y} + MxMy\frac{\partial Mx}{\partial y}\right\} \tag{4.55}$$

Two **further** measures can be obtained by considering the forward and a backward differential along the **normal**. These differentials cannot be related to the actual definition of curvature but can be explained intuitively. If we consider that curves are more than one pixel wide, differentiation along the edge will measure the difference between the gradient angle between interior and exterior borders of a wide curve. In theory, the tangent angle should be the same. However, in discrete images there is a change due to the measures in a window. If the curve is a straight line, then the interior and exterior borders are the same. Thus, gradient direction normal to the edge does not change locally. As we bend a straight line, we increase the difference between the curves defining the interior and exterior borders. Thus, we expect the measure of gradient direction to change. That is, if we differentiate along the normal direction, we maximize detection of

gross curvature. The value $\kappa_{\perp\varphi'}(x,y)$ is obtained when $x(t) = x + t\sin(\varphi'(x,y))$ and $y(t) = y + t\cos(\varphi'(x,y))$. In this case,

$$\kappa_{\perp\varphi'}(x,y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}}\left\{Mx^2\frac{\partial My}{\partial x} - MxMy\frac{\partial My}{\partial x} - MxMy\frac{\partial My}{\partial y} + My^2\frac{\partial Mx}{\partial y}\right\} \tag{4.56}$$

In a **backward** formulation along a **normal** direction to the edge, we obtain:

$$\kappa_{-\perp\varphi'}(x,y) = \frac{1}{(Mx^2 + My^2)^{\frac{3}{2}}}\left\{-Mx^2\frac{\partial My}{\partial x} + MxMy\frac{\partial Mx}{\partial x} - MxMy\frac{\partial My}{\partial y} + My^2\frac{\partial Mx}{\partial y}\right\} \tag{4.57}$$

This was originally used by Kass et al. (1988) as a means to detect *line terminations*, as part of a feature extraction scheme called snakes (active contours) which are covered in Chapter 6. Code 4.17 shows an implementation of the four measures of curvature. The function `Gradient` is used to obtain the gradient of the image and to obtain its derivatives. The output image is obtained by applying the function according to the selection of parameter `op`.

```
%Gradient Corner Detector
%op=T  tangent direction
%op=TI tangent inverse
%op=N  normal direction
%op=NI normal  inverse

function outputimage=GradCorner(inputimage,op)
         [rows,columns]=size(inputimage);  %Image size
         outputimage=zeros(rows,columns);  %Result image
         [Mx,My]=Gradient(inputimage);     %Gradient images
         [M,A]=Edges(inputimage);          %Edge Suppression
         M=MaxSupr(M,A);
         [Mxx,Mxy]=Gradient(Mx);           %Derivatives of the gradient image
         [Myx,Myy]=Gradient(My);


         %compute curvature
         for x=1:columns
           for y=1:rows
              if(M(y,x)~=0)
                    My2=My(y,x)^2; Mx2=Mx(y,x)^2; MxMy=Mx(y,x)*My(y,x);
                    if((Mx2+My2)~=0)
                        if(op=='TI')
                            outputimage(y,x)=(1/(Mx2+My2)^1.5)*(My2*Mxx(y,x)
                                             -MxMy*Myx(y,x)-Mx2*Myy(y,x)
                                             +MxMy*Mxy(y,x));
                        elseif (op=='N')
                            outputimage(y,x)=(1/(Mx2+My2)^1.5)*(Mx2*Myx(y,x)
                                             -MxMy*Mxx(y,x)-MxMy*Myy(y,x)
                                             +My2*Mxy(y,x));
                        elseif (op=='NI')
```

CODE 4.17

Curvature by measuring changes in intensity.

```
                    outputimage(y,x)=(1/(Mx2+My2)^1.5)*(-Mx2*Myx(y,x)
                                     +MxMy*Mxx(y,x)-MxMy*Myy(y,x)
                                     +My2*Mxy(y,x));
                else %tangential as default
                    outputimage(y,x)=(1/(Mx2+My2)^1.5)*(My2*Mxx(y,x)
                                     -MxMy*Myx(y,x)+Mx2*Myy(y,x)
                                     -MxMy*Mxy(y,x));
                end
            end
        end
    end
end
```

CODE 4.17

(Continued)

Let us see how the four functions for estimating curvature from image intensity perform for the image given in Figure 4.37(a). In general, points where the curvature is large are highlighted by each function. Different measures of curvature (Figure 4.38) highlight differing points on the feature boundary. All measures

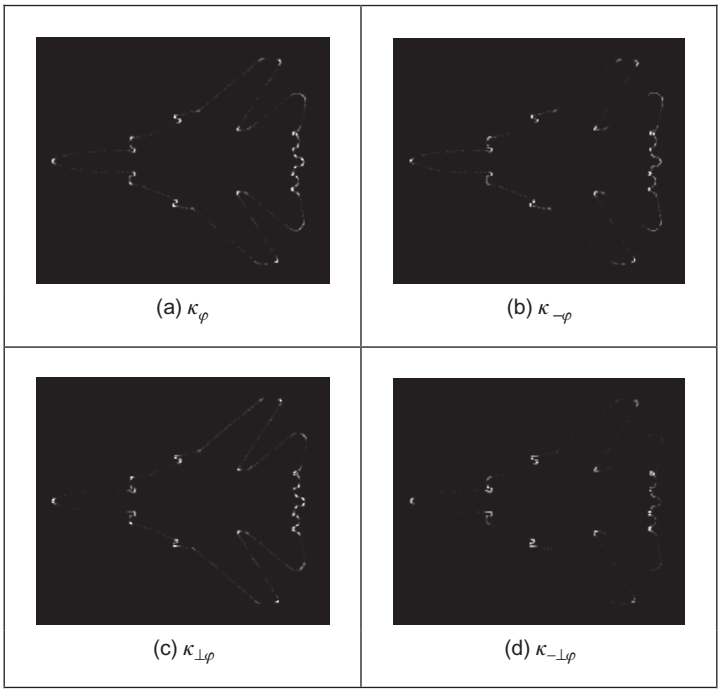


FIGURE 4.38

Comparing image curvature detection operators.

appear to offer better performance than that derived by reformulating hysteresis thresholding (Figure 4.37(b)) though there is little discernible performance advantage between the directions of differentiation. As the results in Figure 4.38 suggest, detecting curvature directly from an image is not a totally reliable way of determining curvature, and hence corner information. This is in part due to the higher order of the differentiation process. (Also, scale has not been included within the analysis.)

4.4.1.4 Moravec and Harris detectors

In the previous section, we measured curvature as the derivative of the function $\varphi(x,y)$ along a particular direction. Alternatively, a measure of curvature can be obtained by considering changes along a particular direction in the image $\mathbf{P}$ itself. This is the basic idea of *Moravec's corner* detection operator. This operator computes the average change in image intensity when a window is shifted in several directions, i.e., for a pixel with coordinates (x,y), and a window size of $2w + 1$ we have

$$\mathbf{E}_{u,v}(x,y) = \sum_{i=-w}^{w} \sum_{j=-w}^{w} [\mathbf{P}_{x+i,y+j} - \mathbf{P}_{x+i+u,y+j+v}]^2 \tag{4.58}$$

This equation approximates the **autocorrelation** function in the direction (u,v). A measure of curvature is given by the minimum value of $\mathbf{E}_{u,v}(x,y)$ obtained by considering the shifts (u,v) in the four main directions, i.e., by $(1,0)$, $(0,-1)$, $(0,1)$, and $(-1,0)$. The minimum is chosen because it agrees with the following two observations. First, if the pixel is in an edge, then defining a straight line, $\mathbf{E}_{u,v}(x,y)$, is small for a shift along the edge and large for a shift perpendicular to the edge. In this case, we should choose the small value since the curvature of the edge is small. Secondly, if the edge defines a corner, then all the shifts produce a large value. Thus, if we also chose the minimum, this value indicates high curvature. The main problem with this approach is that it considers only a small set of possible shifts. This problem is solved in the *Harris corner detector* (Harris and Stephens, 1988) by defining an analytic expression for the autocorrelation. This expression can be obtained by considering the local approximation of intensity changes.

We can consider that the points $\mathbf{P}_{x+i,y+j}$ and $\mathbf{P}_{x+i+u,y+j+v}$ define a vector (u,v) in the image. Thus, in a similar fashion to the development given in Eq. (4.58), the increment in the image function between the points can be approximated by the directional derivative $u\,\partial\mathbf{P}_{x+i,y+j}/\partial x + v\,\partial\mathbf{P}_{x+i,y+j}/\partial y$. Thus, the intensity at $\mathbf{P}_{x+i+u,y+j+v}$ can be approximated as follows:

$$\mathbf{P}_{x+i+u,y+j+v} = \mathbf{P}_{x+i,y+j} + \frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x} u + \frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y} v \tag{4.59}$$

This expression corresponds to the three first terms of the Taylor expansion around $\mathbf{P}_{x+i,y+j}$ (an expansion to first order). If we consider the approximation in Eq. (4.58), we have

$$\mathbf{E}_{u,v}(x,y) = \sum_{i=-w}^{w} \sum_{j=-w}^{w} \left[\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x} u + \frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y} v \right]^2 \tag{4.60}$$

By expansion of the squared term (and since u and v are independent of the summations), we obtain

$$\mathbf{E}_{u,v}(x,y) = A(x,y)u^2 + 2C(x,y)uv + B(x,y)v^2 \tag{4.61}$$

where

$$\begin{aligned} A(x,y) &= \sum_{i=-w}^{w} \sum_{j=-w}^{w} \left(\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x} \right)^2 \quad B(x,y) = \sum_{i=-w}^{w} \sum_{j=-w}^{w} \left(\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y} \right)^2 \\ C(x,y) &= \sum_{i=-w}^{w} \sum_{j=-w}^{w} \left(\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial x} \right) \left(\frac{\partial \mathbf{P}_{x+i,y+j}}{\partial y} \right) \end{aligned} \tag{4.62}$$

that is, the summation of the squared components of the gradient direction for all the pixels in the window. In practice, this average can be weighted by a Gaussian function to make the measure less sensitive to noise (i.e., by filtering the image data). In order to measure the curvature at a point (x,y), it is necessary to find the vector (u,v) that minimizes $\mathbf{E}_{u,v}(x,y)$ given in Eq. (4.61). In a basic approach, we can recall that the minimum is obtained when the window is displaced in the direction of the edge. Thus, we can consider that $u = \cos(\varphi(x,y))$ and $v = \sin(\varphi(x,y))$. These values are defined in Eq. (4.53). Accordingly, the minima values that define curvature are given by

$$\kappa_{u,v}(x,y) = \min \mathbf{E}_{u,v}(x,y) = \frac{A(x,y)M_y^2 + 2C(x,y)M_xM_y + B(x,y)M_x^2}{M_x^2 + M_y^2} \tag{4.63}$$

In a more sophisticated approach, we can consider the form of the function $\mathbf{E}_{u,v}(x,y)$. We can observe that this is a quadratic function, so it has two principal axes. We can rotate the function such that its axes have the same direction as that of the axes of the coordinate system. That is, we rotate the function $\mathbf{E}_{u,v}(x,y)$ to obtain

$$\mathbf{F}_{u,v}(x,y) = \alpha(x,y)^2u^2 + \beta(x,y)^2v^2 \tag{4.64}$$

The values of α and β are proportional to the **autocorrelation** function along the principal axes. Accordingly, if the point (x,y) is in a region of constant intensity, both values are small. If the point defines a straight border in the image, then one value is large and the other is small. If the point defines an edge with

high curvature, both values are large. Based on these observations, a measure of curvature is defined as

$$\kappa_k(x,y) = \alpha\beta - k(\alpha+\beta)^2 \tag{4.65}$$

The first term in this equation makes the measure large when the values of α and β increase. The second term is included to decrease the values in flat borders. The parameter k must be selected to control the sensitivity of the detector. The higher the value, the computed curvature will be more sensitive to changes in the image (and therefore to noise).

In practice, in order to compute $\kappa_k(x,y)$, it is not necessary to compute explicitly the values of α and β, but the curvature can be measured from the coefficient of the quadratic expression in Eq. (4.61). This can be derived by considering the matrix forms of Eqs (4.61) and (4.64). If we define the vector $\mathbf{D}^{\mathrm{T}} = [u,v]$, then Eqs (4.61) and (4.64) can be written as

$$\mathbf{E}_{u,v}(x,y) = \mathbf{D}^{\mathrm{T}}\mathbf{M}\mathbf{D} \quad \text{and} \quad \mathbf{F}_{u,v}(x,y) = \mathbf{D}^{\mathrm{T}}\mathbf{Q}\mathbf{D} \tag{4.66}$$

where $^{\mathrm{T}}$ denotes transpose and where

$$\mathbf{M} = \begin{bmatrix} A(x,y) & C(x,y) \\ C(x,y) & B(x,y) \end{bmatrix} \quad \text{and} \quad \mathbf{Q} = \begin{bmatrix} \alpha & 0 \\ 0 & \beta \end{bmatrix} \tag{4.67}$$

In order to relate Eqs (4.61) and (4.64), we consider that $\mathbf{F}_{u,v}(x,y)$ is obtained by rotating $\mathbf{E}_{u,v}(x,y)$ by a transformation R that rotates the axis defined by $\mathbf{D}$, i.e.,

$$\mathbf{F}_{u,v}(x,y) = (\mathbf{R}\mathbf{D})^{\mathrm{T}}\mathbf{M}\mathbf{R}\mathbf{D} \tag{4.68}$$

This can be arranged as

$$\mathbf{F}_{u,v}(x,y) = \mathbf{D}^{\mathrm{T}}\mathbf{R}^{\mathrm{T}}\mathbf{M}\mathbf{R}\mathbf{D} \tag{4.69}$$

By comparison with Eq. (4.66), we have

$$\mathbf{Q} = \mathbf{R}^{\mathrm{T}}\mathbf{M}\mathbf{R} \tag{4.70}$$

This defines a well-known equation of linear algebra and it means that $\mathbf{Q}$ is an orthogonal decomposition of $\mathbf{M}$. The diagonal elements of $\mathbf{Q}$ are called the eigenvalues. We can use Eq. (4.70) to obtain the value of $\alpha\beta$ which defines the first term in Eq. (4.65) by considering the determinant of the matrices, i.e., $\det(\mathbf{Q}) = \det(\mathbf{R}^{\mathrm{T}})\det(\mathbf{M})\det(\mathbf{R})$. Since $\mathbf{R}$ is a rotation matrix $\det(\mathbf{R}^{\mathrm{T}})\det(\mathbf{R}) = 1$, thus

$$\alpha\beta = A(x,y)B(x,y) - C(x,y)^2 \tag{4.71}$$

which defines the first term in Eq. (4.65). The second term can be obtained by taking the trace of the matrices on each side of this equation. Thus, we have

$$\alpha + \beta = A(x,y) + B(x,y) \tag{4.72}$$

We can also use Eq. (4.70) to obtain the value of $\alpha + \beta$ which defines the first term in Eq. (4.65). By taking the trace of the matrices in each side of this equation, we have

$$\kappa_k(x,y) = A(x,y)B(x,y) - C(x,y)^2 - k(A(x,y) + B(x,y))^2 \tag{4.73}$$

Code 4.18 shows an implementation for Eqs (4.64) and (4.73). The equation to be used is selected by the `op` parameter. Curvature is only computed at edge points, i.e., at pixels whose edge magnitude is different of zero after applying maximal suppression. The first part of the code computes the coefficients of the matrix **M**. Then, these values are used in the curvature computation.

```
%Harris Corner Detector
%op=H  Harris
%op=M  Minimum direction
function outputimage=Harris(inputimage,op)

   w=4;                                        %Window size=2w+1
   k=0.1;                                      %Second term constant
   [rows,columns]=size(inputimage);            %Image size
   outputimage=zeros(rows,columns);            %Result image
   [difx,dify]=Gradient(inputimage);           %Differential
   [M,A]=Edges(inputimage);                    %Edge Suppression
   M=MaxSupr(M,A);

   %compute correlation
   for x=w+1:columns-w                  %pixel (x,y)
     for y=w+1:rows-w
       if M(y,x)~=0
         %compute window average
         A=0;B=0;C=0;
       for i=-w:w
         for j=-w:w
           A=A+difx(y+i,x+j)^2;
           B=B+dify(y+i,x+j)^2;
           C=C+difx(y+i,x+j)*dify(y+i,x+j);
         end
       end

       if(op=='H')
           outputimage(y,x)=A*B-C^2-k*((A+B)^2);;
       else
           dx=difx(y,x);
           dy=dify(y,x);

           if dx*dx+dy*dy~=0
             outputimage(y,x)=((A*dy*dy-
             2*C*dx*dy+B*dx*dx)/(dx*dx+dy*dy));
           end
         end
       end
     end
   end
```

CODE 4.18

Harris corner detector.

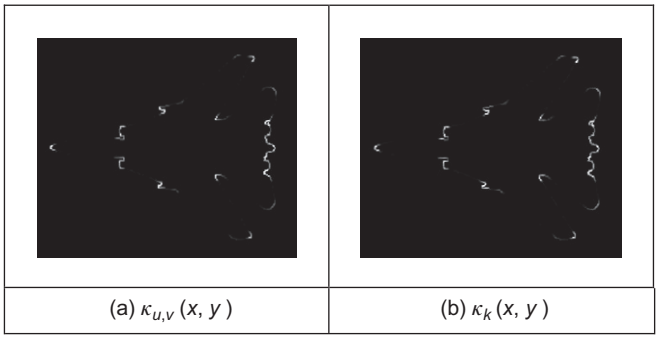


FIGURE 4.39

Curvature via the Harris operator.

Figure 4.39 shows the results of computing curvature using this implementation. The results are capable of showing the different curvature in the border. We can observe that $\kappa_k(x,y)$ produces more contrast between lines with low and high curvature than $\kappa_{u,v}(x,y)$. The reason is the inclusion of the second term in Eq. (4.73). In general, not only the measure of the correlation is useful to compute curvature but also this technique has much wider application in finding points for matching pairs of images.

4.4.1.5 Further reading on curvature

Many of the arguments earlier advanced on extensions to edge detection in Section 4.2 apply to corner detection as well, so the same advice applies. There is much less attention paid by established textbooks to corner detection though Davies (2005) devotes a chapter to the topic. van Otterloo's (1991) fine book on shape analysis contains a detailed analysis of measurement of (planar) curvature.

There are other important issues in corner detection. It has been suggested that corner extraction can be augmented by local knowledge to improve performance (Rosin, 1996). There are actually many other corner detection schemes, each offering different attributes though with differing penalties. Important work has focused on characterizing shapes using corners. In a scheme analogous to the **primal sketch** introduced earlier, there is a *curvature primal sketch* (Asada and Brady, 1986), which includes a set of primitive parameterized curvature discontinuities (such as termination and joining points). There are many other approaches: one (natural) suggestion is to define a corner as the intersection between two lines, this requires a process to find the lines; other techniques use methods that describe shape variation to find corners. We commented that filtering techniques can be included to improve the detection process; however, filtering can also be used to obtain a multiple detail representation. This representation is very useful to shape characterization. A *curvature scale space* has been developed (Mokhtarian and Mackworth, 1986; Mokhtarian and Bober, 2003) to give a

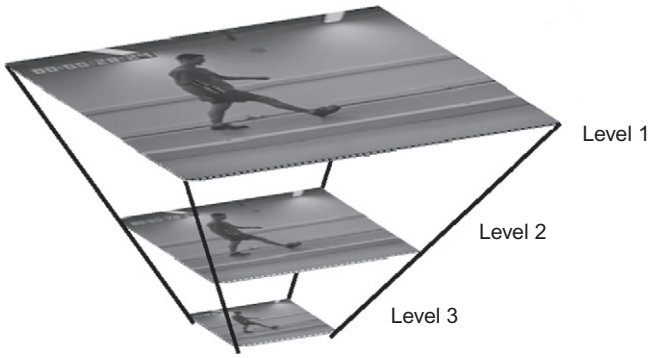


FIGURE 4.40

Illustrating scale space.

compact way of representing shapes, and at different scales, from coarse (low level) to fine (detail) and with the ability to handle appearance transformations.

4.4.2 Modern approaches: region/patch analysis

The modern approaches to local feature extraction aim to relieve some of the constraints on the earlier methods of localized feature extraction. This allows for the inclusion of scale: an object can be recognized irrespective of its apparent size. The object might also be characterized by a collection of points, and this allows for recognition where there has been change in the viewing arrangement (in a planar image an object viewed from a different angle will appear different, but points which represent it still appear in a similar arrangement). Using arrangements of points also allows for recognition where some of the image points have been obscured (because the image contains clutter or noise). In this way, we can achieve a description which allows for object or scene recognition direct from the image itself, by exploiting local neighborhood properties.

The newer techniques depend on the notion of scale space: features of interest are those which persist over selected scales. The scale space is defined by images which are successively smoothed by the Gaussian filter, as in Eq. (3.38), and then subsampled to form an *image pyramid* at different scales, as illustrated in Figure 4.40 for three levels of resolution. There are approaches which exploit structure within the scale space to improve speed, as we shall find.

4.4.2.1 Scale invariant feature transform

The *Scale invariant feature transform* (*SIFT*) (Lowe, 1999, 2004) aims to resolve many of the practical problems in low-level feature extraction and their use in matching images. The earlier Harris operator is sensitive to changes in image

scale and as such is unsuited to matching images of differing size. The SIFT transform actually involves two stages: feature extraction and description. The description stage concerns use of the low-level features in object matching, and this will be considered later. Low-level feature extraction within the SIFT approach selects salient features in a manner invariant to image scale (feature size) and rotation and with partial invariance to change in illumination. Further, the formulation reduces the probability of poor extraction due to occlusion clutter and noise. Further, it shows how many of the techniques considered previously can be combined and capitalized on, to good effect.

First, the difference of Gaussians operator is applied to an image to identify features of potential interest. The formulation aims to ensure that feature selection does not depend on feature size (scale) or orientation. The features are then analyzed to determine location and scale before the orientation is determined by local gradient direction. Finally the features are transformed into a representation that can handle variation in illumination and local shape distortion. Essentially, the operator uses local information to refine the information delivered by standard operators. The detail of the operations is best left to the source material (Lowe 1999, 2004) for it is beyond the level or purpose here. As such we shall concentrate on principle only.

The features detected for the Lena image are illustrated in Figure 4.41. Here, the major features detected are shown by white lines where the length reflects magnitude, and the direction reflects the feature's orientation. These are the major features which include the rim of the hat, face features, and the boa. The minor features are the smaller white lines: the ones shown here are concentrated around a background feature. In the full set of features detected at all scales in this

(a) Original image

(b) Output points with magnitude and direction

FIGURE 4.41

Detecting features with the SIFT operator.

image, there are many more of the minor features, concentrated particularly in the textured regions of the image (Figure 4.42). Later, we shall see how this can be used within shape extraction, but our purpose here is the basic low-level features.

In the first stage, the **difference of Gaussians** for an image **P** is computed in the manner of Eq. (4.28) as

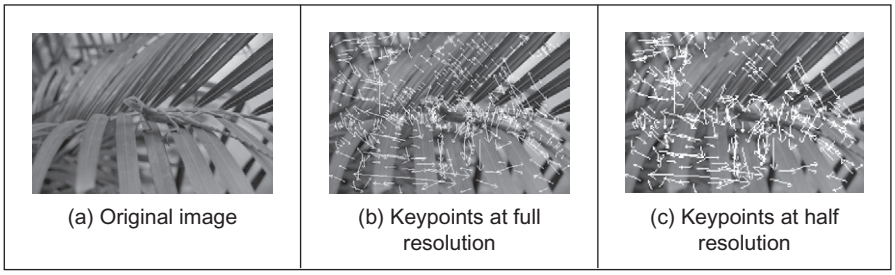
$$\begin{aligned} D(x, y, \sigma) &= (g(x, y, k\sigma) - g(x, y, \sigma)) * \mathbf{P} \\ &= L(x, y, k\sigma) - L(x, y, k) \end{aligned} \tag{4.74}$$

The function L is actually a scale-space function which can be used to define smoothed images at different scales. Rather than any difficulty in locating zero-crossing points, the features are the maxima and minima of the function. Candidate keypoints are then determined by comparing each point in the function with its immediate neighbors. The process then proceeds to analysis between the levels of scale, given appropriate sampling of the scale space. This then implies comparing a point with its eight neighbors at that scale and with the nine neighbors in each of the adjacent scales, to determine whether it is a minimum or maximum, as well as image resampling to ensure comparison between the different scales.

In order to filter the candidate points to reject those which are the result of low local contrast (low-edge strength) or which are poorly localized along an edge, a function is derived by local curve fitting which indicates local edge strength and stability as well as location. Uniform thresholding then removes the keypoints with low contrast. Those that have poor localization, i.e., their position is likely to be influenced by noise, can be filtered by considering the ratio of curvature along an edge to that perpendicular to it, in a manner following the Harris operator in Section 4.4.1.4, by thresholding the ratio of Eqs (4.71) and (4.72).

In order to characterize the filtered keypoint features at each scale, the gradient magnitude is calculated in exactly the manner of Eqs (4.12) and (4.13) as

$$M_{\text{SIFT}}(x, y) = \sqrt{(L(x+1, y) - L(x-1, y))^2 + (L(x, y+1) - L(x, y-1))^2} \tag{4.75}$$



(a) Original image

(b) Keypoints at full resolution

(c) Keypoints at half resolution

FIGURE 4.42

SIFT feature detection at different scales.

$$\theta_{\mathrm{SIFT}}(x, y) = \tan^{-1}\left(\frac{L(x, y+1) - L(x, y-1)}{(L(x+1, y) - L(x-1, y))}\right) \tag{4.76}$$

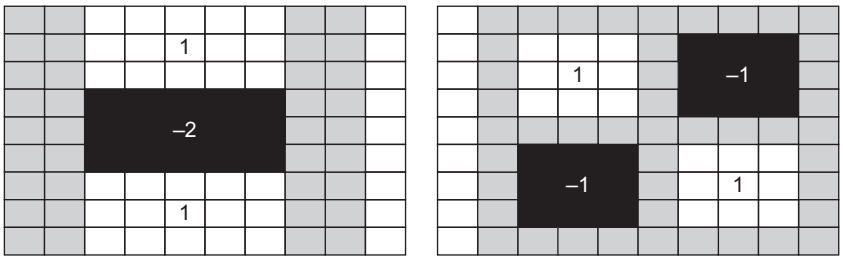
The peak of the histogram of the orientations around a keypoint is then selected as the local direction of the feature. This can be used to derive a canonical orientation, so that the resulting descriptors are invariant with rotation. As such, this contributes to the process which aims to reduce sensitivity to camera viewpoint and to nonlinear change in image brightness (linear changes are removed by the gradient operations) by analyzing regions in the locality of the selected viewpoint. The main description (Lowe, 2004) considers the technique's basis in much greater detail and outlines factors important to its performance such as the need for sampling and performance in noise.

As shown in Figure 4.42, the technique can certainly operate well, and scale is illustrated by applying the operator to the original image and to one at half the resolution. In all, 601 keypoints are determined in the original resolution image and 320 keypoints at half the resolution. By inspection, the major features are retained across scales (a lot of minor regions in the leaves disappear at lower resolution), as expected. Alternatively, the features can of course be filtered further by magnitude, or even direction (if appropriate). If you want more than results to convince you, implementations are available for Windows and Linux (http://www.cs.ubc.ca/spider/lowe/research.html and some of the software sites noted in Table 1.2)—a feast for any developer. These images were derived by using `siftWin32`, version 4.

Note that description is inherent in the process—the standard SIFT keypoint descriptor is created by sampling the magnitudes and orientations of the image gradient in the region of the keypoint. An array of histograms, each with orientation bins, captures the rough spatial structure of the patch. This results in a vector which was later compressed by using principal component analysis (PCA) (Ke and Sukthankar, 2004) to determine the most salient features. Clearly this allows for faster matching than the original SIFT formulation, but the improvement in performance was later doubted (Mikolajczyk and Schmid, 2005).

4.4.2.2 Speeded up robust features

The central property exploited within SIFT is the use of difference of Gaussians to determine local features. In a relationship similar to the one between first- and second-order edge detection, the *speeded up robust features (SURF)* approach (Bay et al., 2006, 2008) employs approximations to second-order edge detection at different scales. The basis of the SURF operator is to use the integral image approach of Section 2.7.3.2 to provide an efficient means to compute approximations of second-order differencing, as shown in Figure 4.43. These are the approximations for a LoG operator with $\sigma = 1.2$ and represent the finest scale in the SURF operator. Other approximations can be derived for larger scales, since the operator—like SIFT—considers features which persist over scale space.



(a) Vertical second-order approximation (b) Diagonal second-order approximation

FIGURE 4.43

Basis of SURF feature detection.

The scale space can be derived by upscaling the approximations (wavelets) using larger templates which gives for faster execution than the use of smoothing and resampling in an image to form a pyramidal structure of different scales, which is more usual in scale-space approaches.

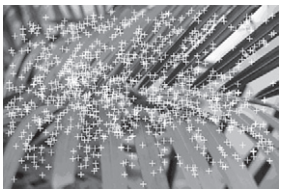
By the Taylor expansion of the image brightness (Eq. (4.59)), we can form a (Hessian) matrix $\mathbf{M}$ (Eq. (4.67)) from which the maxima are used to derive the features. This is

$$\det(\mathbf{M}) = \begin{bmatrix} L_{xx} & L_{xy} \\ L_{xy} & L_{yy} \end{bmatrix} = L_{xx}L_{yy} - w \cdot L_{xy}^2 \tag{4.77}$$

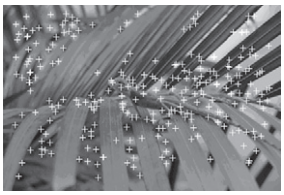
where the terms in $\mathbf{M}$ arise from the convolution of a second-order derivative of Gaussian with the image information as

$$L_{xx} = \frac{\partial^2(g(x,y,\sigma))}{\partial x^2} * \mathbf{P}_{x,y} \quad L_{xy} = \frac{\partial^2(g(x,y,\sigma))}{\partial x \partial y} * \mathbf{P}_{x,y} \quad L_{yy} = \frac{\partial^2(g(x,y,\sigma))}{\partial y^2} * \mathbf{P}_{x,y} \tag{4.78}$$

and where w is carefully chosen to balance the components of the equation. To localize interest points in the image and over scales, nonmaximum suppression is applied in a $3 \times 3 \times 3$ neighborhood. The maxima of the determinant of the Hessian matrix are then interpolated in scale space and in image space and described by orientations derived using the vertical and horizontal Haar wavelets described earlier (Section 2.7.3.2). Note that there is an emphasis on the speed of execution, as well as on performance attributes, and so the generation of the templates to achieve scale space, the factor w, the interpolation operation to derive features, and their description are achieved using optimized processes. The developers have provided downloads for evaluation of SURF from http://www.vision.ee.ethz.ch/~surf/. The performance of the operator is illustrated in Figure 4.44 showing the positions of the detected points for SIFT and for SURF. This shows that SURF can deliver fewer features, and which persist (and hence can be faster), whereas SIFT can provide more features (and be slower). As ever, choice depends



(a) SIFT



(b) SURF

FIGURE 4.44

Comparing features detected by SIFT and SURF.

on application—both techniques are available for evaluation and there are public domain implementations.

4.4.2.3 Saliency

The *saliency* operator (Kadir and Brady, 2001) was also motivated by the need to extract robust and relevant features. In the approach, regions are considered salient if they are simultaneously unpredictable both in some feature and scale space. Unpredictability (rarity) is determined in a statistical sense, generating a space of saliency values over position and scale, as a basis for later understanding. The technique aims to be a generic approach to scale and saliency compared to conventional methods, because both are defined independent of a particular basis morphology, which means that it is not based on a particular geometric feature like a blob, edge, or corner. The technique operates by determining the **entropy** (a measure of rarity) within patches at scales of interest and the saliency is a weighted summation of where the entropy peaks. The method has practical capability in that it can be made invariant to rotation, translation, nonuniform scaling, and uniform intensity variations and robust to small changes in viewpoint. An example result of processing the image in Figure 4.45(a) is shown in Figure 4.45(b) where the 200 most salient points are shown circled, and the radius of the circle is indicative of the scale. Many of the points are around the walking subject and others highlight significant features in the background, such as the waste bins, the tree, or the time index. An example use of saliency was within an approach to learn and recognize object class models (such as faces, cars, or animals) from unlabeled and unsegmented cluttered scenes, irrespective of their overall size (Fergus et al., 2003). For further study and application, descriptions and Matlab binaries are available from Kadir's web site (http://www.robots.ox.ac.uk/~timork/).

4.4.2.4 Other techniques and performance issues

There has been a recent comprehensive performance review (Mikolajczyk and Schmid, 2005) comparing based operators. The techniques which were compared

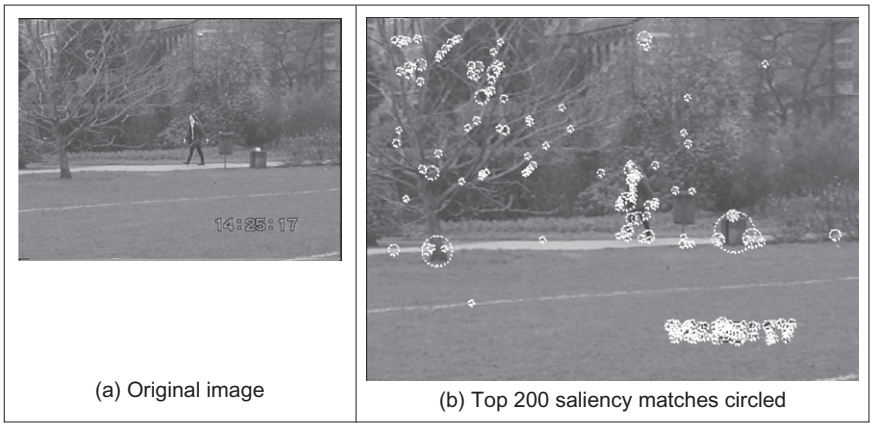


FIGURE 4.45

Detecting features by saliency.

include SIFT, differential derivatives by differentiation, cross correlation for matching, and a gradient location and orientation-based histogram (an extension to SIFT, which performed well)—the saliency approach was not included. The criterion used for evaluation concerned the number of correct matches, and the number of false matches, between feature points selected by the techniques. The matching process was between an original image and one of the same scene when subject to one of six image transformations. The image transformations covered practical effects that can change image appearance and were rotation, scale change, viewpoint change, image blur, JPEG compression, and illumination. For some of these there were two scene types available, which allowed for separation of understanding of scene type and transformation. The study observed that, within its analysis, "the SIFT-based descriptors perform best," but it is of course a complex topic and selection of technique is often application dependent. Note that there is further interest in performance evaluation and in invariance to higher order changes in viewing geometry, such as invariance to affine and projective transformation. There are other comparisons available, either with new operators or in new applications and there (inevitably) are faster implementations too. One survey covers the field in more detail (Tuytelaars and Mikolajczyk, 2007), concerning principle and performance analysis.

4.5 Describing image motion

We have looked at the main low-level features that we can extract from a single image. In the case of motion, we must consider more than one image. If we have two images obtained at different times, the simplest way in which we can detect


(a) Difference image D


(b) First image


(c) Second image

FIGURE 4.46

Detecting motion by differencing.

motion is by image *differencing*. That is, changes or motion can be located by subtracting the intensity values; when there is no motion, the subtraction will give a zero value, and when an object in the image moves, their pixel's intensity changes, and so the subtraction will give a value different of zero. There are links in this section, which determines detection of movement, to later material in Chapter 9 which concerns **detecting** the moving object and *tracking* its movement.

In order to denote a sequence of images, we include a time index in our previous notation, i.e., $\mathbf{P}(t)_{x,y}$. Thus, the image at the origin of our time is $\mathbf{P}(0)_{x,y}$ and the next image is $\mathbf{P}(1)_{x,y}$. As such the image differencing operation which delivered the difference image $\mathbf{D}$ is given by

$$\mathbf{D}(t) = \mathbf{P}(t) - \mathbf{P}(t-1) \tag{4.79}$$

Figure 4.46 shows an example of this operation. The image in Figure 4.46(a) is the result of subtracting the image in Figure 4.46(b) from the one in Figure 4.46(c). Naturally, this shows rather more than just the bits which are moving; we have not just highlighted the moving subject but we have also highlighted bits above the subject's head and around feet. This is due mainly to change in the lighting (the shadows around the feet are to do with the subject's interaction with the lighting). However, perceived change can also be due to motion of the camera and to the motion of other objects in the field of view. In addition to these inaccuracies, perhaps the most important limitation of differencing is the lack of information about the movement itself. That is, we cannot see exactly **how** image points have moved. In order to describe the way the points in an image actually move, we should study how the pixels' position changes in each image frame.

4.5.1 Area-based approach

When a scene is captured at different times, 3D elements are mapped into corresponding pixels in the images. Thus, if image features are not occluded, they can be related to each other and motion can be characterized as a collection of displacements in the image plane. The displacement corresponds to the projection of

movement of the objects in the scene and it is referred to as the *optical flow*. If you were to take an image, and its optical flow, you should be able to construct the **next** frame in the image sequence. So optical flow is like a measurement of velocity, the movement in pixels per unit of time, more simply pixels per frame. Optical flow can be found by looking for corresponding features in images. We can consider alternative features such as points, pixels, curves, or complex descriptions of objects.

The problem of finding correspondences in images has motivated the development of many techniques that can be distinguished by the features, by the constraints imposed, and by the optimization or searching strategy (Dhond and Aggarwal, 1989). When features are pixels, the correspondence can be found by observing the similarities between intensities in image regions (local neighborhood). This approach is known as area-based matching and it is one of the most common techniques used in computer vision (Barnard and Fichler, 1987). In general, pixels in nonoccluded regions can be related to each other by means of a general transformation of the form by

$$\mathbf{P}(t+1)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} + \mathbf{H}(t)_{x,y} \tag{4.80}$$

where the function $\mathbf{H}(t)_{x,y}$ compensates for intensity differences between the images, and $(\delta x,\delta y)$ defines the displacement vector of the pixel at time $t+1$. That is, the intensity of the pixel in the frame at time $t+1$ is equal to the intensity of the pixel in the position (x,y) in the previous frame plus some small change due to physical factors and temporal differences that induce the photometric changes in images. These factors can be due, for example, to shadows, specular reflections, differences in illumination, or changes in observation angles. In a general case, it is extremely difficult to account for the photometric differences; thus the model in Eq. (4.80) is generally simplified by assuming that

1. the **brightness** of a point in an image is **constant** and
2. the **neighboring** points move with **similar** velocity.

According to the first assumption, $\mathbf{H}(x) \approx 0$. Thus,

$$\mathbf{P}(t+1)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} \tag{4.81}$$

Many techniques have used this relationship to express the matching process as an optimization or variational problem (Jordan and Bovik, 1992). The objective is to find the vector $(\delta x,\delta y)$ that minimizes the error given by

$$e_{x,y} = S(\mathbf{P}(t+1)_{x+\delta x,y+\delta y}, \mathbf{P}(t)_{x,y}) \tag{4.82}$$

where $S(\,)$ represents a function that measures the similarity between pixels. As such, the optimum is given by the displacements that minimize the image differences. There are alternative measures of similarity that can be used to define the matching cost (Jordan and Bovik, 1992). For example, we can measure the difference by taking the absolute of the arithmetic difference. Alternatively, we can

consider the correlation or the squared values of the difference or an equivalent normalized form. In practice, it is difficult to try to establish a conclusive advantage of a particular measure, since they will perform differently depending on the kind of image, the kind of noise, and the nature of the motion we are observing. As such, one is free to use any measure as long as it can be justified based on particular practical or theoretical observations. The correlation and the squared difference will be explained in more detail in the next chapter when we consider how a template can be located in an image. We shall see that if we want to make the estimation problem in Eq. (4.82) equivalent to maximum likelihood estimation, then we should minimize the squared error, i.e.,

$$e_{x,y} = (\mathbf{P}(t+1)_{x+\delta x,y+\delta y}, \mathbf{P}(t)_{x,y})^2 \tag{4.83}$$

In practice, the implementation of the minimization is extremely prone to error since the displacement is obtained by comparing intensities of single pixel; it is very likely that the intensity changes or that a pixel can be confused with other pixels. In order to improve the performance, the optimization includes the second assumption presented above. If neighboring points move with similar velocity, we can determine the displacement by considering not just a single pixel, but pixels in a neighborhood. Thus,

$$e_{x,y} = \sum_{(x',y')\in W} (\mathbf{P}(t+1)_{x'+\delta x,y'+\delta y}, \mathbf{P}(t)_{x',y'})^2 \tag{4.84}$$

That is the error in the pixel at position (x,y) is measured by comparing all the pixels (x',y') in a window W. This makes the measure more stable by introducing an implicit smoothing factor. The size of the window is a compromise between noise and accuracy. Naturally, the automatic selection of the window parameter has attracted some interest (Kanade and Okutomi, 1994). Another important problem is the amount of computation involved in the minimization when the displacement between frames is large. This has motivated the development of hierarchical implementations. As you can envisage, other extensions have considered more elaborate assumptions about the speed of neighboring pixels.

A straightforward implementation of the minimization of the square error is presented in Code 4.19. This function has a pair of parameters that define the maximum displacement and the window size. The optimum displacement for each pixel is obtained by comparing the error for all the potential integer displacements. In a more complex implementation, it is possible to obtain displacements with subpixel accuracy (Lawton, 1983). This is normally achieved by a postprocessing step based on subpixel interpolation or by matching surfaces obtained by fitting the data at the integer positions. The effect of the selection of different window parameters can be seen in the example shown in Figure 4.47. Figure 4.47(a) and (b) shows an object moving up into a static background (at least for the two frames we are considering). Figure 4.47(c)–(e) shows the displacements obtained by considering windows of increasing size. Here, we can

```
%Optical flow by correlation
%d: max displacement., w:window size 2w+1
function FlowCorr(inputimage1,inputimage2,d,w)

%Load images
L1=double(imread(inputimage1, 'bmp'));
L2=double(imread(inputimage2,'bmp'));

%image size
[rows,columns]=size(L1); %L2 must have the same size

%result image
u=zeros(rows,columns);
v=zeros(rows,columns);

%correlation for each pixel
for x1=w+d+1:columns-w-d
     for y1=w+d+1:rows-w-d
            min=99999; dx=0; dy=0;
            %displacement position
            for x2=x1-d:x1+d
                 for y2=y1-d:y1+d
                 sum=0;
                 for i=-w:w% window
                     for j=-w:w
                         sum=sum+(double(L1(y1+j,x1+i))-
                         double(L2(y2+j,x2+i)))^2;
                     end
                 end
                 if (sum<min)
                     min=sum;
                     dx=x2-x1; dy=y2-y1;
                 end
            end
     end
     u(y1,x1)=dx;
     v(y1,x1)=dy;
     end
end

%display result
quiver(u,v,.1);
```

CODE 4.19

Implementation of area-based motion computation.

observe that as the size of the window increases, the result is smoother, but we lost detail about the boundary of the object. We can also observe that when the window is small, there are noisy displacements near the object's border. This can be explained by considering that Eq. (4.80) suppose that pixels appear in both

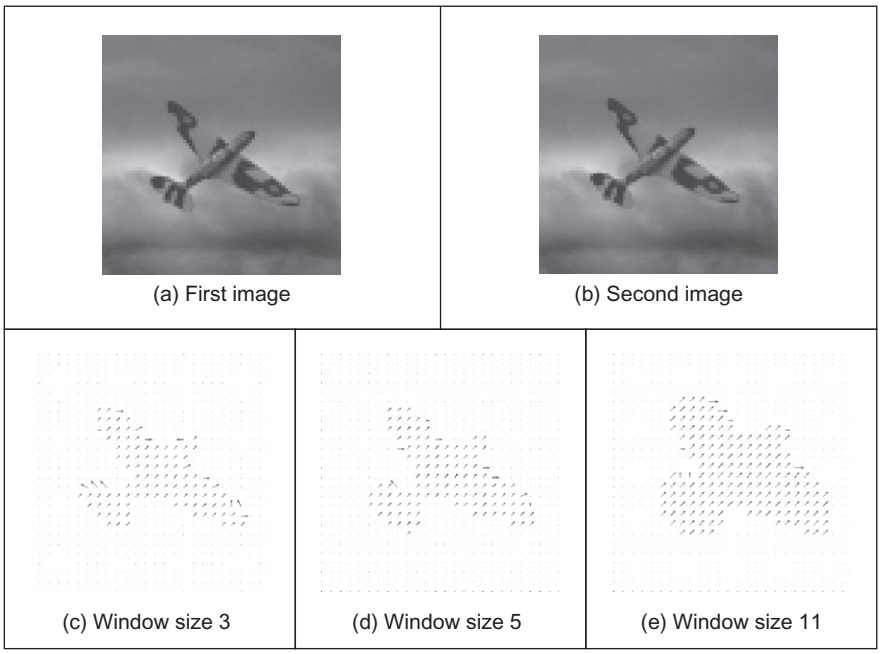


FIGURE 4.47

Example of area-based motion computation.

images, but this is not true near the border since pixels appear and disappear (i.e., occlusion) from and behind the moving object. Additionally, there are problems in regions that lack intensity variations (texture). This is because the minimization function in Eq. (4.83) is almost flat and there is no clear evidence of the motion. In general, there is no effective way of handling these problems since they are due to the lack of information in the image.

4.5.2 Differential approach

Another popular way to estimate motion focuses on the observation of the differential changes in the pixel values. There are actually many ways of calculating the optical flow by this approach (Nagel, 1987; Barron et al., 1994). We shall discuss one of the more popular techniques (Horn and Schunk, 1981). We start by considering the intensity equity in Eq. (4.81). According to this, the brightness at the point in the **new** position should be the same as the brightness at the **old** position. Like Eq. (4.5), we can expand $\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y}$ by using a Taylor series as

$$\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} + \delta x\frac{\partial \mathbf{P}(t)_{x,y}}{\partial x} + \delta y\frac{\partial \mathbf{P}(t)_{x,y}}{\partial y} + \delta t\frac{\partial \mathbf{P}(t)_{x,y}}{\partial t} + \xi \qquad (4.85)$$

where ξ contains higher order terms. If we take the limit as $\delta t \rightarrow 0$, then we can ignore ξ as it also tends to zero and the equation becomes

$$\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} + \delta x \frac{\partial \mathbf{P}(t)_{x,y}}{\partial x} + \delta y \frac{\partial \mathbf{P}(t)_{x,y}}{\partial y} + \delta t \frac{\partial \mathbf{P}(t)_{x,y}}{\partial t} \tag{4.86}$$

Now by substituting Eq. (4.81) for $\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y}$, we get

$$\mathbf{P}(t)_{x,y} = \mathbf{P}(t)_{x,y} + \delta x \frac{\partial \mathbf{P}(t)_{x,y}}{\partial x} + \delta y \frac{\partial \mathbf{P}(t)_{x,y}}{\partial y} + \delta t \frac{\partial \mathbf{P}(t)_{x,y}}{\partial t} \tag{4.87}$$

which with some rearrangement gives the motion constraint equation

$$\frac{\delta x}{\delta t}\frac{\partial \mathbf{P}}{\partial x} + \frac{\delta y}{\delta t}\frac{\partial \mathbf{P}}{\partial y} = -\frac{\partial \mathbf{P}}{\partial t} \tag{4.88}$$

We can recognize some terms in this equation. $\partial\mathbf{P}/\partial x$ and $\partial\mathbf{P}/\partial y$ are the first-order differentials of the image intensity along the two image axes. $\partial\mathbf{P}/\partial t$ is the rate of change of image intensity with time. The other two factors are the ones concerned with optical flow, as they describe movement along the two image axes. Let us call

$$u = \frac{\delta x}{\delta t} \quad \text{and} \quad v = \frac{\delta y}{\delta t}$$

These are the optical flow components: u is the *horizontal optical flow* and v is the *vertical optical flow*. We can write these into our equation to give

$$u\frac{\partial \mathbf{P}}{\partial x} + v\frac{\partial \mathbf{P}}{\partial y} = -\frac{\partial \mathbf{P}}{\partial t} \tag{4.89}$$

This equation suggests that the optical flow and the spatial rate of intensity change together describe how an image changes with time. The equation can actually be expressed more simply in vector form in terms of the intensity change $\nabla\mathbf{P} = [\nabla x\ \nabla y] = [\partial\mathbf{P}/\partial x\ \partial\mathbf{P}/\partial y]$ and the optical flow $\mathbf{v} = [u\ v]^{\mathrm{T}}$, as the dot product

$$\nabla\mathbf{P} \cdot \mathbf{v} = -\dot{\mathbf{P}} \tag{4.90}$$

We already have operators that can estimate the spatial intensity change, $\nabla x = \partial\mathbf{P}/\partial x$ and $\nabla y = \partial\mathbf{P}/\partial y$, by using one of the edge-detection operators described earlier. We also have an operator which can estimate the rate of change of image intensity, $\nabla t = \partial\mathbf{P}/\partial t$, as given by Eq. (4.79). Unfortunately, we cannot determine the optical flow components from Eq. (4.89) since we have one equation in two unknowns (there are many possible pairs of values for u and v that satisfy the equation). This is actually called the *aperture problem* and makes the problem **ill-posed**. Essentially, we seek estimates of u and v that minimize the error in Eq. (4.92) over the entire image. By expressing Eq. (4.89) as

$$u\nabla x + v\nabla y + \nabla t = 0 \tag{4.91}$$

we seek estimates of u and v that minimize the error ec for all the pixels in an image:

$$ec = \iint (u\nabla x + v\nabla y + \nabla t)^2 \mathrm{d}x\,\mathrm{d}y \tag{4.92}$$

We can approach the solution (equations to determine u and v) by considering the second assumption we made earlier, namely that neighboring points move with similar velocity. This is actually called the *smoothness constraint* as it suggests that the velocity field of the brightness varies in a smooth manner without abrupt change (or discontinuity). If we add this in to the formulation, we turn a problem that is ill-posed, without unique solution, to one that is well-posed. Properly, we define the smoothness constraint as an integral over the area of interest, as in Eq. (4.92). Since we want to maximize smoothness, we seek to minimize the rate of change of the optical flow. Accordingly, we seek to minimize an integral of the rate of change of flow along both axes. This is an error es and expressed as

$$es = \iint \left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right) \mathrm{d}x\,\mathrm{d}y \tag{4.93}$$

The total error is the compromise between the importance of the assumption of constant brightness and the assumption of smooth velocity. If this compromise is controlled by a *regularization* parameter λ, then the total error e is

$$\begin{aligned} e &= \lambda \times ec + es \\ &= \iint \left(\lambda \times \left(u\frac{\partial \mathbf{P}}{\partial x} + v\frac{\partial \mathbf{P}}{\partial y} + \frac{\partial \mathbf{P}}{\partial t} \right)^2 + \left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right) \right) \mathrm{d}x\,\mathrm{d}y \end{aligned} \tag{4.94}$$

There is a number of ways to approach the solution (Horn, 1986), but the most appealing is perhaps also the most direct. We are concerned with providing estimates of optical flow at image points. So we are actually interested in computing the values for $u_{x,y}$ and $v_{x,y}$. We can form the error at image points, like $es_{x,y}$. Since we are concerned with image points, we can form $es_{x,y}$ by using first-order differences, just like Eq. (4.1). Equation (4.93) can be implemented in discrete form as

$$es_{x,y} = \sum_x \sum_y \frac{1}{4}((u_{x+1,y} - u_{x,y})^2 + (u_{x,y+1} - u_{x,y})^2 + (v_{x+1,y} - v_{x,y})^2 + (v_{x,y+1} - v_{x,y})^2) \tag{4.95}$$

The discrete form of the smoothness constraint is that the average rate of change of flow should be minimized. To obtain the discrete form of Eq. (4.94), we add in the discrete form of ec (the discrete form of Eq. (4.92)) to give

$$ec_{x,y} = \sum_x \sum_y (u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y})^2 \tag{4.96}$$

where $\nabla x_{x,y} = \partial \mathbf{P}_{x,y}/\partial x$, $\nabla y_{x,y} = \partial \mathbf{P}_{x,y}/\partial y$, and $\nabla t_{x,y} = \partial \mathbf{P}_{x,y}/\partial t$ are local estimates, at the point with coordinates (x,y), of the rate of change of the picture with horizontal direction, vertical direction, and time, respectively. Accordingly, we seek values for $u_{x,y}$ and $v_{x,y}$ that minimize the total error e as given by

$$\begin{aligned} e_{x,y} &= \sum_x \sum_y (\lambda \times ec_{x,y} + es_{x,y}) \\ &= \sum_x \sum_y \left(\begin{array}{l} \lambda \times (u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y})^2 + \\ \frac{1}{4}((u_{x+1,y} - u_{x,y})^2 + (u_{x,y+1} - u_{x,y})^2 + (v_{x+1,y} - v_{x,y})^2 + (v_{x,y+1} - v_{x,y})^2) \end{array} \right) \end{aligned} \tag{4.97}$$

Since we seek to minimize this equation with respect to $u_{x,y}$ and $v_{x,y}$, we differentiate it separately, with respect to the two parameters of interest, and the resulting equations when equated to zero should yield the equations we seek. As such

$$\frac{\partial e_{x,y}}{\partial u_{x,y}} = (\lambda \times 2(u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y})\nabla x_{x,y} + 2(u_{x,y} - \overline{u}_{x,y})) = 0 \tag{4.98}$$

and

$$\frac{\partial e_{x,y}}{\partial v_{x,y}} = (\lambda \times 2(u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y})\nabla y_{x,y} + 2(v_{x,y} - \overline{v}_{x,y})) = 0 \tag{4.99}$$

This gives a pair of equations in $u_{x,y}$ and $v_{x,y}$:

$$\begin{array}{l} (1 + \lambda(\nabla x_{x,y})^2)u_{x,y} + \lambda \nabla x_{x,y}\nabla y_{x,y}v_{x,y} = \overline{u}_{x,y} - \lambda \nabla x_{x,y}\nabla t_{x,y} \\ \lambda \nabla x_{x,y}\nabla y_{x,y}u_{x,y} + (1 + \lambda(\nabla y_{x,y})^2)v_{x,y} = \overline{v}_{x,y} - \lambda \nabla x_{x,y}\nabla t_{x,y} \end{array} \tag{4.100}$$

This is a pair of equations in u and v with solution

$$\begin{array}{l} (1 + \lambda((\nabla x_{x,y})^2 + (\nabla y_{x,y})^2))u_{x,y} = (1 + \lambda(\nabla y_{x,y})^2)\overline{u}_{x,y} - \lambda \nabla x_{x,y}\nabla y_{x,y}\overline{v}_{x,y} - \lambda \nabla x_{x,y}\nabla t_{x,y} \\ (1 + \lambda((\nabla x_{x,y})^2 + (\nabla y_{x,y})^2)v_{x,y} = -\lambda \nabla x_{x,y}\nabla y_{x,y}\overline{u}_{x,y} + (1 + \lambda(\nabla x_{x,y})^2)\overline{v}_{x,y} - \lambda \nabla y_{x,y}\nabla t_{x,y} \end{array} \tag{4.101}$$

The solution to these equations is in iterative form where we shall denote the estimate of u at iteration n as $u^{<n>}$, so each iteration calculates new values for the flow at each point according to

$$\begin{array}{l} u_{x,y}^{<n+1>} = \overline{u}_{x,y}^{<n>} - \lambda \left(\dfrac{\nabla x_{x,y}\overline{u}_{x,y} + \nabla y_{x,y}\overline{v}_{x,y} + \nabla t_{x,y}}{(1 + \lambda(\nabla x_{x,y}^2 + \nabla y_{x,y}^2))} \right)(\nabla x_{x,y}) \\ v_{x,y}^{<n+1>} = \overline{v}_{x,y}^{<n>} - \lambda \left(\dfrac{\nabla x_{x,y}\overline{u}_{x,y} + \nabla y_{x,y}\overline{v}_{x,y} + \nabla t_{x,y}}{(1 + \lambda(\nabla x_{x,y}^2 + \nabla y_{x,y}^2))} \right)(\nabla y_{x,y}) \end{array} \tag{4.102}$$

Now, the pair of equations gives iterative means for calculating the images of optical flow based on differentials. In order to estimate the first-order

differentials, rather than using our earlier equations, we can consider neighboring points in quadrants in successive images. This gives approximate estimates of the gradient based on the two frames, i.e.,

$$\begin{aligned}\nabla x_{x,y} &= \frac{\begin{array}{c}(\mathbf{P}(0)_{x+1,y} + \mathbf{P}(1)_{x+1,y} + \mathbf{P}(0)_{x+1,y+1} + \mathbf{P}(1)_{x+1,y+1}) \\ -(\mathbf{P}(0)_{x,y} + \mathbf{P}(1)_{x,y} + \mathbf{P}(0)_{x,y+1} + \mathbf{P}(1)_{x,y+1})\end{array}}{8} \\ \nabla y_{x,y} &= \frac{\begin{array}{c}(\mathbf{P}(0)_{x,y+1} + \mathbf{P}(1)_{x,y+1} + \mathbf{P}(0)_{x+1,y+1} + \mathbf{P}(1)_{x+1,y+1}) \\ -(\mathbf{P}(0)_{x,y} + \mathbf{P}(1)_{x,y} + \mathbf{P}(0)_{x+1,y} + \mathbf{P}(1)_{x+1,y})\end{array}}{8}\end{aligned} \tag{4.103}$$

In fact, in a later reflection on the earlier presentation, Horn and Schunk (1993) noted with rancor that some difficulty experienced with the original technique had actually been caused by use of simpler methods of edge detection which are not appropriate here, as the simpler versions do not deliver a correctly positioned result between two images. The time differential is given by the difference between the two pixels along the two faces of the cube as

$$\nabla t_{x,y} = \frac{\begin{array}{c}(\mathbf{P}(1)_{x,y} + \mathbf{P}(1)_{x+1,y} + \mathbf{P}(1)_{x,y+1} + \mathbf{P}(1)_{x+1,y+1}) \\ -(\mathbf{P}(0)_{x,y} + \mathbf{P}(0)_{x+1,y} + \mathbf{P}(0)_{x,y+1} + \mathbf{P}(0)_{x+1,y+1})\end{array}}{8} \tag{4.104}$$

Note that if the spacing between the images is other than one unit, this will change the denominator in Eqs (4.103) and (4.104), but this is a constant scale factor. We also need means to calculate the averages. These can be computed as

$$\begin{aligned}\overline{u}_{x,y} &= \frac{u_{x-1,y} + u_{x,y-1} + u_{x+1,y} + u_{x,y+1}}{2} + \frac{u_{x-1,y-1} + u_{x-1,y+1} + u_{x+1,y-1} + u_{x+1,y+1}}{4} \\ \overline{v}_{x,y} &= \frac{v_{x-1,y} + v_{x,y-1} + v_{x+1,y} + v_{x,y+1}}{2} + \frac{v_{x-1,y-1} + v_{x-1,y+1} + v_{x+1,y-1} + v_{x+1,y+1}}{4}\end{aligned} \tag{4.105}$$

The implementation of the computation of optical flow by the iterative solution in Eq. (4.102) is presented in Code 4.20. This function has two parameters that define the smoothing parameter and the number of iterations. In the implementation, we use the matrices `u`, `v`, `tu`, and `tv` to store the old and new estimates in each iteration. The values are updated according to Eq. (4.102). Derivatives and averages are computed by using simplified forms of Eqs (4.103)–(4.105). In a more elaborate implementation, it is convenient to include averages as we discussed in the case of single image feature operators. This will improve the accuracy and will reduce noise. Additionally, since derivatives can only be computed for small displacements, generally, gradient algorithms are implemented with a hierarchical structure. This will enable the computation of displacements larger than one pixel.

```
%Optical flow by gradient method
%s = smoothing parameter
%n = number of iterations
function OpticalFlow(inputimage1,inputimage2,s,n)

%Load images
L1=double(imread(inputimage1, 'bmp'));
L2=double(imread(inputimage2, 'bmp'));

%Image size
[rows,columns]=size(I1); %I2 must have the same size

%Result flow
u=zeros(rows,columns);
v=zeros(rows,columns);

%Temporal flow
tu=zeros(rows,columns);
tv=zeros(rows,columns);

%Flow computation
for k=1:n   %iterations
  for x=2:columns-1
      for y=2:rows-1
      %derivatives
       Ex=(L1(y,x+1)-L1(y,x)+L2(y,x+1)-L2(y,x)+L1(y+1,x+1)
          -L1(y+1,x)+L2(y+1,x+1)-L2(y+1,x))/4;
       Ey=(L1(y+1,x)-L1(y,x)+L2(y+1,x)-L2(y,x)+L1(y+1,x+1)
          -L1(y,x+1)+L2(y+1,x+1)-L2(y,x+1))/4;
       Et=(L2(y,x)-L1(y,x)+L2(y+1,x)-L1(y+1,x)+L2(y,x+1)
          -L1(y,x+1)+L2(y+1,x+1)-L1(y+1,x+1))/4;
      %average
      AU=(u(y,x-1)+u(y,x+1)+u(y-1,x)+u(y+1,x))/4;
      AV=(v(y,x-1)+v(y,x+1)+v(y-1,x)+v(y+1,x))/4;
      %update estimates
      A=(Ex*AU+Ey*AV+Et);
      B=(1+s*(Ex*Ex+Ey*Ey));
      tu(y,x)= AU-(Ex*s*A/B);
      tv(y,x)= AV-(Ey*s*A/B);
      end%for (x,y)
end
%update
for x=2:columns-1
     for y=2:rows-1
       u(y,x)=tu(y,x); v(y,x)=tv(y,x);
     end %for (x,y)
     end
end %iterations

%display result
quiver(u,v,1);
```

CODE 4.20

Implementation of gradient-based motion.

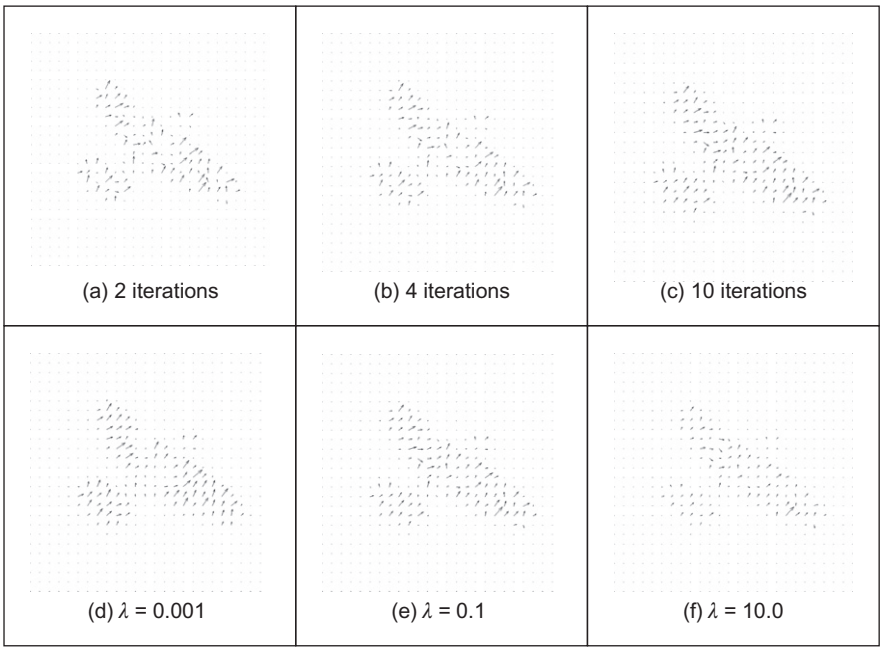


FIGURE 4.48

Example of differential-based motion computation.

Figure 4.48 shows some examples of optical flow computation. In these examples, we used the same images as in Figure 4.47. The first row in the figure shows three results obtained by different number of iterations and fixed smoothing parameter. In this case, the estimates converged quite quickly. Note that at the start, the estimates of flow in are quite noisy, but they quickly improve; as the algorithm progresses, the results are refined and a more smooth and accurate motion is obtained. The second row in Figure 4.48 shows the results for a fixed number of iterations and a variable smoothing parameter. The regularization parameter controls the compromise between the detail and the smoothness. A **large** value of λ will enforce the **smoothness** constraint whereas a **small** value will make the **brightness** constraint dominate the result. In the results, we can observe that the largest vectors point in the expected direction, upward, while some of the smaller vectors are not exactly correct. This is because there are occlusions and some regions have similar textures. Clearly, we could select the brightest of these points by thresholding according to magnitude. That would leave the largest vectors (the ones which point in exactly the right direction).

Optical flow has been used in automatic gait recognition (Little and Boyd, 1998; Huang et al., 1999) among other applications, partly because the displacements can be large between successive images of a walking subject, which makes

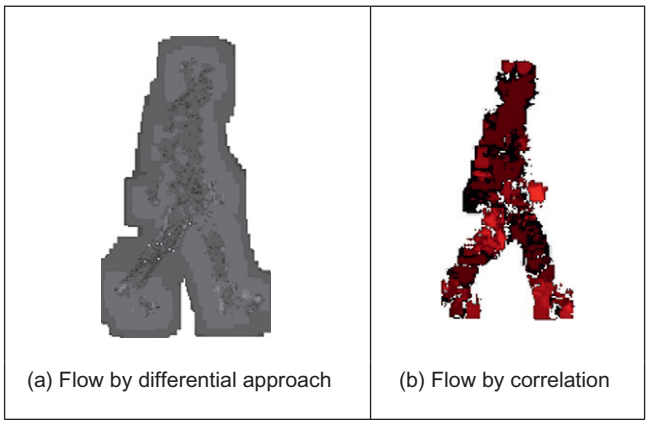


FIGURE 4.49

Optical flow of walking subject.

the correlation approach suitable (note that fast versions of area-based correspondence are possible; Zabir and Woodfill, 1994). Figure 4.49 shows the result for a walking subject where brightness depicts magnitude (direction is not shown). Figure 4.49(a) shows the result for the differential approach, where the flow is clearly more uncertain than that produced by the correlation approach shown in Figure 4.49(b). Another reason for using the correlation approach is that we are not concerned with rotation as people (generally!) walk along flat surfaces. If 360° rotation is to be considered then you have to match regions for every rotation value and this can make the correlation-based techniques computationally very demanding indeed.

4.5.3 Further reading on optical flow

Determining optical flow does not get much of a mention in the established textbooks, even though it is a major low-level feature description. Rather naturally, it is to be found in depth in one of its early proponent's textbooks (Horn, 1986). One approach to motion estimation has considered the **frequency domain** (Adelson and Bergen, 1985) (yes, Fourier transforms get everywhere!). For a further overview of dense optical flow, see Bulthoff et al. (1989) and for implementation, see Little et al. (1988). The major survey (Beauchemin and Barron, 1995) of the approaches to optical flow is rather dated now, as is their performance appraisal (Barron et al., 1994). Such an (accuracy) appraisal is particularly useful in view of the number of ways there are to estimate it. The nine techniques studied included the differential approach we have discussed here, a Fourier technique and a correlation-based method. Their conclusion was that a local differential

method (Lucas and Kanade, 1981) and a phase-based method (Fleet and Jepson, 1990) offered the most consistent performance on the datasets studied. However, there are many variables not only in the data but also in implementation that might lead to preference for a particular technique. Clearly, there are many impediments to the successful calculation of optical flow such as change in illumination or occlusion (and by other moving objects). An updated study (Baker et al., 2007) concentrated on developing the database and the evaluation methodology, comparing five more recent algorithms (though one was derived from Windows Media Player). The study refined and extended the evaluation methodology in terms of performance metrics and widened dissemination. A later version of the work (Baker et al., 2009) has a more extensive analysis than the earlier (conference) paper and there is a web site associated with the work to which developers can submit their work and where its performance is evaluated. One conclusion is that none of the methods was a clear winner on all of the datasets evaluated, though the overall aim of the study was to stimulate further development of technique, as well as performance analysis. Clearly, the web site http://vision.middlebury.edu/flow/ is an important port of call for any developer or user of optical flow algorithms.

4.6 Further reading

This chapter has covered the main ways to extract low-level feature information. In some cases, this can prove sufficient for understanding the image. Often though, the function of low-level feature extraction is to provide information for later higher level analysis. This can be achieved in a variety of ways, with advantages and disadvantages and quickly or at a lower speed (or requiring a faster processor/more memory!). The range of techniques presented here has certainly proved sufficient for the majority of applications. There are other, more minor techniques, but the main approaches to boundary, corner, feature, and motion extraction have proved sufficiently robust and with requisite performance that they shall endure for some time. Given depth and range, the further reading for each low-level operation is to be found at the end of each section.

We now move on to using this information at a higher level. This means collecting the information so as to find shapes and objects, the next stage in understanding the image's content.

4.7 References

Adelson, E.H., Bergen, J.R., 1985. Spatiotemporal energy models for the perception of motion. J. Opt. Soc. Am. A2 (2), 284–299.

Apostol, T.M., 1966. Calculus, second ed. Xerox College Publishing, Waltham, MA, **1**.

Asada, H., Brady, M., 1986. The curvature primal sketch. IEEE Trans. PAMI 8 (1), 2–14.

Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J., Szeliski, R., 2007. A database and evaluation methodology for optical flow. Proceedings of the Eleventh ICCV, 8pp.

Baker, S., Scharstein, D., Lewis, J.P., Roth, S., Black, M.J., Szeliski, R., 2009. A database and evaluation methodology for optical flow. Microsoft Research Technical Report MSR-TR-2009-179. <http://research.microsoft.com/apps/pubs/default.aspx?id=117766\>.

Barnard, S.T., Fichler, M.A., 1987. Stereo vision. Encyclopedia of Artificial Intelligence. Wiley, New York, NY, pp. 1083–2090.

Barron, J.L., Fleet, D.J., Beauchemin, S.S., 1994. Performance of optical flow techniques. Int. J. Comput. Vis. 12 (1), 43–77.

Bay, H., Tuytelaars, T., Van Gool, L., 2006. SURF: Speeded Up Robust Features. Proceedings of the ECCV 2006, pp. 404–417.

Bay, H., Eas, A., Tuytelaars, T., Van Gool, L., 2008. Speeded-Up Robust Features (SURF). Comput. Vis. Image Und. 110 (3), 346–359.

Beauchemin, S.S., Barron, J.L., 1995. The computation of optical flow. Commun. ACM, 433–467.

Bennet, J.R., MacDonald, J.S., 1975. On the measurement of curvature in a quantised environment. IEEE Trans. Comput. C-24 (8), 803–820.

Bergholm, F., 1987. Edge focussing. IEEE Trans. PAMI 9 (6), 726–741.

Bovik, A.C., Huang, T.S., Munson, D.C., 1987. The effect of median filtering on edge estimation and detection. IEEE Trans. PAMI 9 (2), 181–194.

Bulthoff, H., Little, J., Poggio, T., 1989. A parallel algorithm for real-time computation of optical flow. Nature 337 (9), 549–553.

Canny, J., 1986. A computational approach to edge detection. IEEE Trans. PAMI 8 (6), 679–698.

Clark, J.J., 1989. Authenticating edges produced by zero-crossing algorithms. IEEE Trans. PAMI 11 (1), 43–57.

Davies, E.R., 2005. Machine Vision: Theory, Algorithms and Practicalities, Morgan Kaufmann (Elsevier), third ed.

Deriche, R., 1987. Using Canny's criteria to derive a recursively implemented optimal edge detector. Int. J. Comput. Vis. 1, 167–187.

Dhond, U.R., Aggarwal, J.K., 1989. Structure from stereo—a review. IEEE Trans. SMC 19 (6), 1489–1510.

Fergus, R., Perona, P., Zisserman, A., 2003. Object class recognition by unsupervised scale-invariant learning. Proc. CVPR II, 264–271.

Fleet, D.J., Jepson, A.D., 1990. Computation of component image velocity from local phase information. Int. J. Comput. Vis. 5 (1), 77–104.

Forshaw, M.R.B., 1988. Speeding up the Marr–Hildreth edge operator. CVGIP 41, 172–185.

Goetz, A., 1970. Introduction to Differential Geometry. Addison-Wesley, Reading, MA.

Grimson, W.E.L., Hildreth, E.C., 1985. Comments on digital step edges from zero crossings of second directional derivatives. IEEE Trans. PAMI 7 (1), 121–127.

Groan, F., Verbeek, P., 1978. Freeman-code probabilities of object boundary quantized contours. CVGIP 7, 391–402.

Gunn, S.R., 1999. On the discrete representation of the Laplacian of Gaussian. Pattern Recog. 32 (8), 1463–1472.

Haddon, J.F., 1988. Generalised threshold selection for edge detection. Pattern Recog. 21 (3), 195–203.
Haralick, R.M., 1984. Digital step edges from zero-crossings of second directional derivatives. IEEE Trans. PAMI 6 (1), 58–68.
Haralick, R.M., 1985. Author's reply. IEEE Trans. PAMI 7 (1), 127–129.
Harris, C., Stephens, M., 1988. A combined corner and edge detector. Proceedings of the Fourth Alvey Vision Conference, pp. 147–151.
Heath, M.D., Sarkar, S., Sanocki, T., Bowyer, K.W., 1997. A robust visual method of assessing the relative performance of edge detection algorithms. IEEE Trans. PAMI 19 (12), 1338–1359.
Horn, B.K.P., 1986. Robot Vision. MIT Press, Cambridge, MA.
Horn, B.K.P., Schunk, B.G., 1981. Determining optical flow. Artif. Intell. 17, 185–203.
Horn, B.K.P., Schunk, B.G., 1993. Determining optical flow: a retrospective. Artif. Intell. 59, 81–87.
Huang, P.S., Harris, C.J., Nixon, M.S., 1999. Human Gait Recognition in Canonical Space using Temporal Templates. IEE Proc. Vis. Image Signal Process. 146 (2), 93–100.
Huertas, A., Medioni, G., 1986. Detection of intensity changes with subpixel accuracy using Laplacian–Gaussian masks. IEEE Trans. PAMI 8 (1), 651–664.
Jia, X., Nixon, M.S., 1995. Extending the feature vector for automatic face recognition. IEEE Trans. PAMI 17 (12), 1167–1176.
Jordan III, J.R., Bovik, A.C., 1992. Using chromatic information in dense stereo correspondence. Pattern Recog. 25, 367–383.
Kadir, T., Brady, M., 2001. Scale, saliency and image description. Int. J. Comput. Vis. 45 (2), 83–105.
Kanade, T., Okutomi, M., 1994. A stereo matching algorithm with an adaptive window: theory and experiment. IEEE Trans. PAMI 16, 920–932.
Kass, M., Witkin, A., Terzopoulos, D., 1988. Snakes: active contour models. Int. J. Comput. Vis. 1 (4), 321–331.
Ke, Y., Sukthankar, R., 2004. PCA–SIFT: a more distinctive representation for local image descriptors. Proceedings CVPR 2004, II, pp. 506–513.
Kitchen, L., Rosenfeld, A., 1982. Gray-level corner detection. Pattern Recog. Lett. 1 (2), 95–102.
Korn, A.F., 1988. Toward a symbolic representation of intensity changes in images. IEEE Trans. PAMI 10 (5), 610–625.
Kovesi, P., 1999. Image features from phase congruency. Videre: J. Comput. Vis. Res. 1 (3), 1–27.
Lawton, D.T., 1983. Processing translational motion sequences. CVGIP 22, 116–144.
Lindeberg, T., 1994. Scale-space theory: a basic tool for analysing structures at different scales. J. Appl. Statistic. 21 (2), 224–270.
Little, J.J., Boyd, J.E., 1998. Recognizing people by their gait: the shape of motion. Videre 1 (2), 2–32, <http://mitpress.mit.edu/e-journals/VIDE/001/v12.html\>.
Little, J.J., Bulthoff, H.H., Poggio, T., 1988. Parallel optical flow using local voting. Proceedings of the ICCV, pp. 454–457.
Lowe, D.G., 1999. Object Recognition from Local Scale-Invariant Features. Proceedings of the ICCV, pp. 1150–1157.
Lowe, D.G., 2004. Distinctive image features from scale-invariant key points. Int. J. Comput. Vis. 60 (2), 91–110.

Lucas, B., Kanade, T., 1981. An iterative image registration technique with an application to stereo vision. Proceedings of the DARPA Image Understanding Workshop, pp. 121–130.

Marr, D., 1982. Vision. W. H. Freeman and Co., New York, NY.

Marr, D.C., Hildreth, E., 1980. Theory of edge detection. Proc. R. Soc. Lond. B207, 187–217.

Mikolajczyk, K., Schmid, C., 2005. A performance evaluation of local descriptors. IEEE Trans. PAMI 27 (10), 1615–1630.

Mokhtarian, F., Bober, M., 2003. Curvature Scale Space Representation: Theory, Applications and MPEG-7 Standardization. Kluwer Academic Publishers.

Mokhtarian, F., Mackworth, A.K., 1986. Scale-space description and recognition of planar curves and two-dimensional shapes. IEEE Trans. PAMI 8 (1), 34–43.

Morrone, M.C., Burr, D.C., 1988. Feature detection in human vision: a phase-dependent energy model. Proc. R. Soc. Lond. B 235 (1280), 221–245.

Morrone, M.C., Owens, R.A., 1987. Feature detection from local energy. Pattern Recog. Lett. 6, 303–313.

Mulet-Parada, M., Noble, J.A., 2000. 2D + T acoustic boundary detection in echocardiography. Med. Image Anal. 4, 21–30.

Myerscough, P.J., Nixon, M.S., 2004. Temporal phase congruency. Proceedings of the IEEE Southwest Symposium on Image Analysis and Interpretation SSIAI'04, pp. 76–79.

Nagel, H.H., 1987. On the estimation of optical flow: relations between different approaches and some new results. Artif. Intell. 33, 299–324.

van Otterloo, P.J., 1991. A Contour-Oriented Approach to Shape Analysis. Prentice Hall International (UK) Ltd., Hemel Hempstead.

Petrou, M., 1994. The differentiating filter approach to edge detection. Adv. Electron. Electron. Phys. 88, 297–345.

Petrou, M., Kittler, J., 1991. Optimal edge detectors for ramp edges. IEEE Trans. PAMI 13 (5), 483–491.

Prewitt, J.M.S., Mendelsohn, M.L., 1966. The analysis of cell images. Ann. N. Y. Acad. Sci. 128, 1035–1053.

Roberts, L.G., 1965. Machine perception of three-dimensional solids. Optical and Electro-Optical Information Processing. MIT Press, pp. 159–197.

Rosin, P.L., 1996. Augmenting corner descriptors. Graph. Model. Image Process. 58 (3), 286–294.

Smith, S.M., Brady, J.M., 1997. SUSAN—a new approach to low level image processing. Int. J. Comput. Vis. 23 (1), 45–78.

Sobel, I.E., 1970. Camera models and machine perception. PhD Thesis, Stanford University.

Spacek, L.A., 1986. Edge detection and motion detection. Image Vis. Comput. 4 (1), 43–56.

Torre, V., Poggio, T.A., 1986. On edge detection. IEEE Trans. PAMI 8 (2), 147–163.

Tuytelaars, T., Mikolajczyk, K., 2007. Local invariant feature detectors: a survey. Found. Trends Comput. Graph. Vis. 3 (3), 177–280.

Ulupinar, F., Medioni, G., 1990. Refining edges detected by a LoG operator. CVGIP 51, 275–298.

Venkatesh, S., Owens, R.A., 1989. An energy feature detection scheme. Proceedings of an International Conference on Image Processing, Singapore, pp. 553–557.

Venkatesh, S., Rosin, P.L., 1995. Dynamic threshold determination by local and global edge evaluation. Graphical Model. Image Process. 57 (2), 146–160.

Vliet, L.J., Young, I.T., 1989. A nonlinear Laplacian operator as edge detector in noisy images. CVGIP 45, 167–195.

Yitzhaky, Y., Peli, E., 2003. A method for objective edge detection evaluation and detector parameter selection. IEEE Trans. PAMI 25 (8), 1027–1033.

Zabir, R., Woodfill, J., 1994. Nonparametric local transforms for computing visual correspondence. Proceedings of the European Conference on Computer Vision, pp. 151–158.

Zheng, Y., Nixon, M.S., Allen, R., 2004. Automatic segmentation of lumbar vertebrae in digital videofluoroscopic imaging. IEEE Trans. Med. Imaging 23 (1), 45–52.