4

Low-level feature extraction (including edge detection)

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 a form of low-level feature extraction performed as a point operation. 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, although 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 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, as for the aeroplane in Figure 4.1(b) and (e). These are another low-level feature that 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. We shall start with the edge detection techniques, with the first order operators, which accords with the chronology of development. The first order techniques date back more than 30 years.

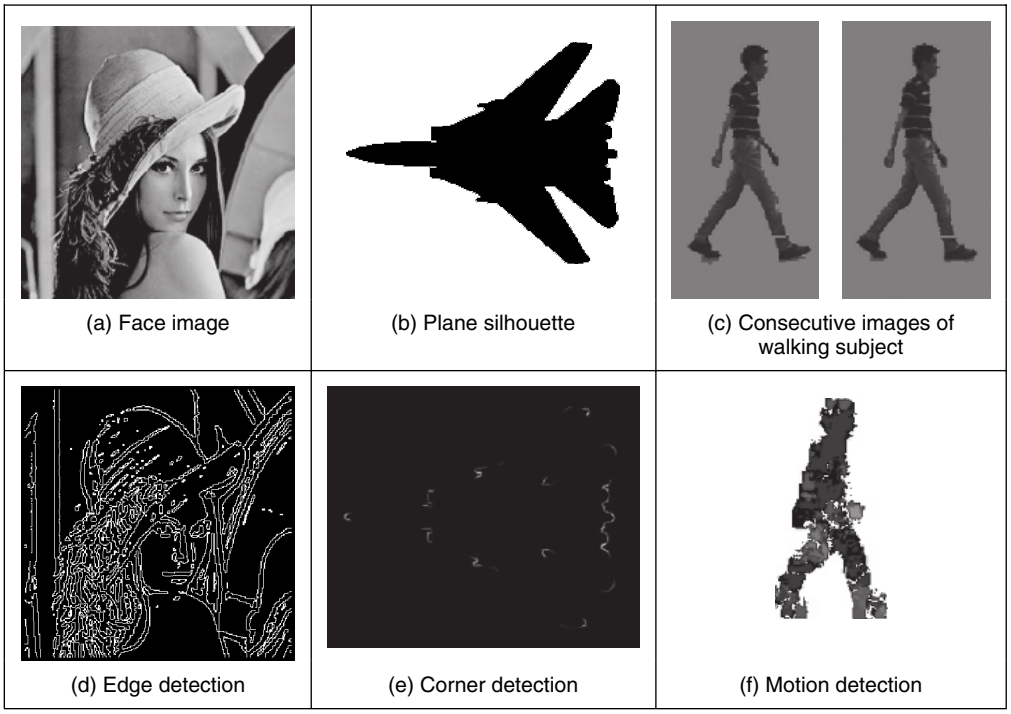


Figure 4.1 Low-level feature detection

Table 4.1 Overview of Chapter 4

Main topic	Sub topics	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.
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.

(*Continued*)

Table 4.1 (Continued)

Main topic	Sub topics	Main points
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: SIFT operator and saliency.
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.

4.2 First order edge detection operators

4.2.1 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 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 $\mathbf{P}$ the action of the horizontal edge-detector forms the difference between two horizontally adjacent points, as such detecting the vertical edges, $\mathbf{Ex}$, as:

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

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, $\mathbf{Ey}$, according to:

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

Figure 4.2(b) and (c) show the application of the vertical and horizontal operators to the synthesized image of the square in Figure 4.2(a). The left-hand vertical edge in Figure 4.2(b)

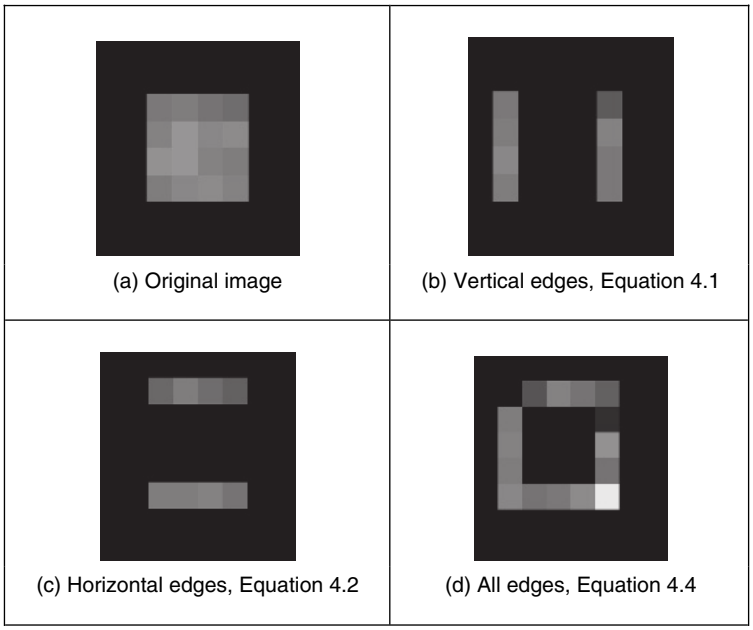


Figure 4.2 First order edge detection

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 **E** that can detect vertical and horizontal edges *together*. That is,

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

which gives:

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

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.

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

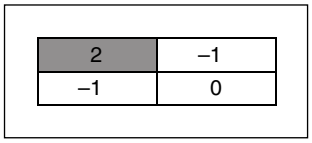


Figure 4.3 Template for first order difference

```
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.

4.2.2 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:

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

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 Equation 4.1 at two adjacent points, as a new horizontal difference **Exx**, where

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

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

$$\mathbf{Exx}_{x,y} = \left|\mathbf{P}_{x+1,y} - \mathbf{P}_{x-1,y}\right| \qquad \forall x \in 2, N-1; y \in 1, N \tag{4.8}$$

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

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

By differencing Equation 4.9 from Equation 4.5, we obtain the first order derivative as:

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

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 Equation 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 centre pixel. A transposed version of the template gives a vertical edge detection operator (Figure 4.4b).

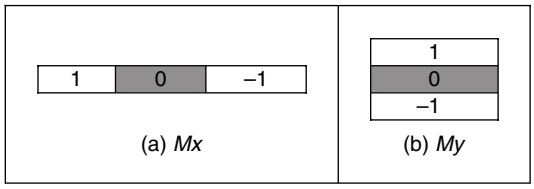


Figure 4.4 Templates for improved first order difference

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 which difference 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.

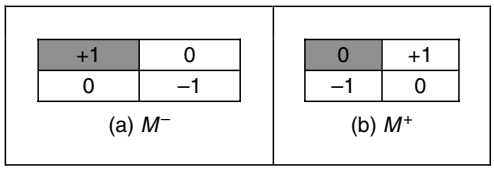


Figure 4.5 Templates for Roberts cross operator

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 $\mathbf{P}_{x,y}$:

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

The application of the Roberts cross operator to the image of the square is shown in Figure 4.6. The two templates provide the results 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).

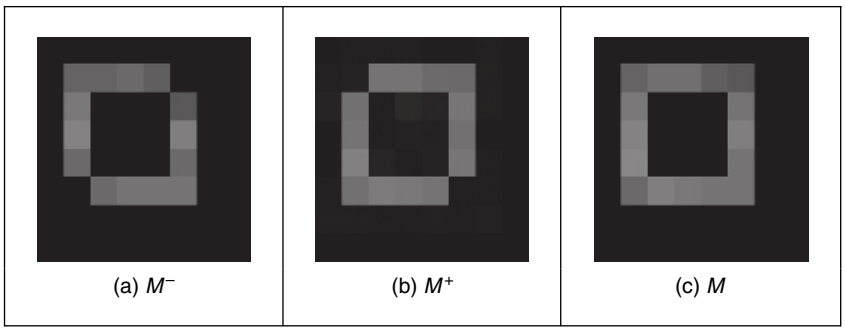


Figure 4.6 Applying the Roberts cross operator

An alternative to taking the maximum is simply to *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 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.

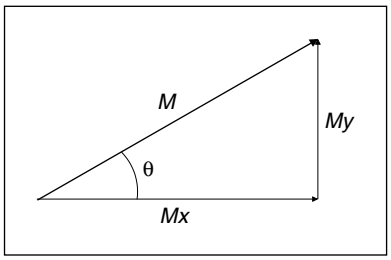


Figure 4.7 Edge detection in vectorial format

4.2.3 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 analogue is high-pass filtering, as illustrated in Figure 2.26c). It is therefore prudent to incorporate *averaging* within the edge detection process. We can then extend the vertical template, *Mx*, along three rows,

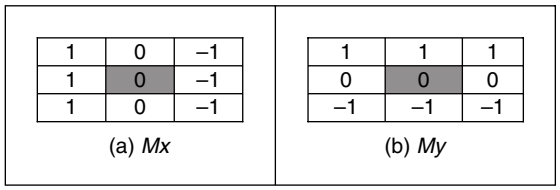


Figure 4.8 Templates for Prewitt operator

and the horizontal template, *My*, along three columns. These give the *Prewitt edge detection operator* (Prewitt and Mendelsohn, 1966), which 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:

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

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

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).

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

(a) *Mx*

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

(b) *My*

Code 4.2 Implementing the Prewitt operator

When applied to the image of the square (Figure 4.9a), we obtain the edge magnitude and direction (Figure 4.9b and d, respectively, where part d does not include the border points, only the edge direction at processed points). The edge direction in Figure 4.9(d) is shown measured in degrees, where 0° and 360° are horizontal, to the right, and 90° is vertical, upwards. Although the regions of edge points are wider owing 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

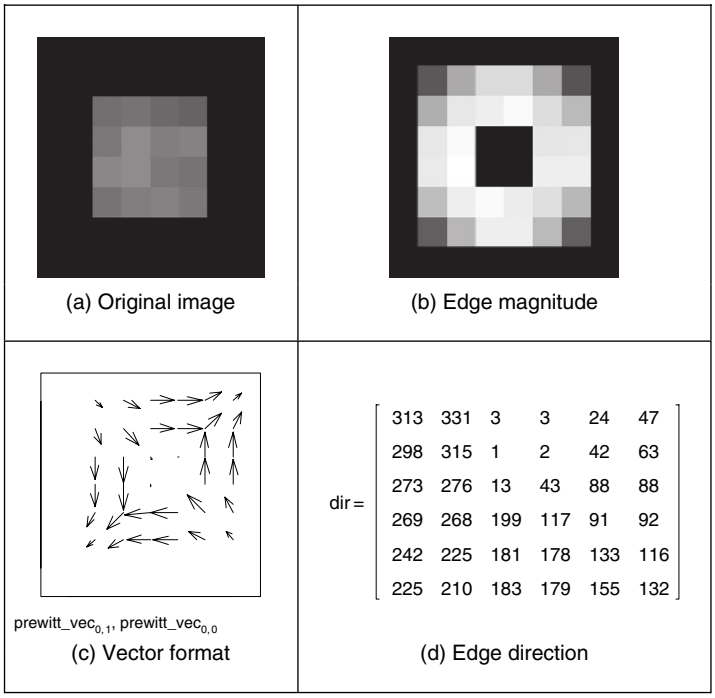


Figure 4.9 Applying the Prewitt operator

less well defined at the corners of the square (as expected, since the first order derivative is discontinuous at these points).

4.2.4 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 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, approach 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	1		4		6		4		1

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 template, or window, size. The coefficients of smoothing within the Sobel operator (Figure 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.

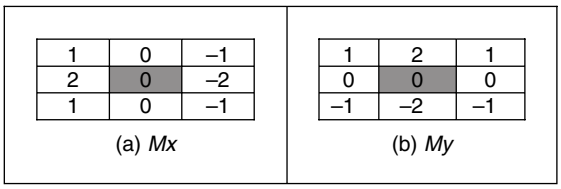


Figure 4.10 Templates for Sobel operator

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

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	1		2		0		−2		−1

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

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

Code 4.4 Pascal's triangle

The differencing template, diff_{x_win}, is then given by the difference between two Pascal expansions, as given in Code 4.5.

$$\text{diff}_{x\_win} := \text{Pascal}(x\_win, winsize-2) - \text{Pascal}(x\_win-1, winsize-2)$$

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 then 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 a 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 4.11a) has many edges in the hair and in the region of the eyes. This is shown in the edge magnitude image (Figure 4.11b). When this is thresholded at a suitable value, many edge points are found (Figure 4.11c). Note that in areas of the image where the brightness

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

Figure 4.11 Applying the Sobel operator

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 axes. If only one of the templates is inverted, the measured edge direction will be inverted around the chosen axis.

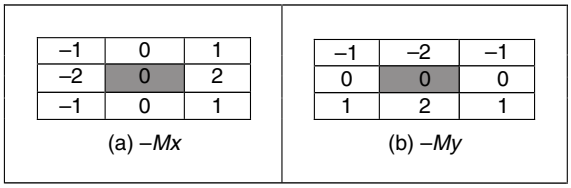


Figure 4.12 Inverted templates for Sobel operator

This gives *four* possible directions for measurement of the edge direction provided by the Sobel operator, two of which (for the templates of Figures 4.10 and 4.12) are illustrated in Figure 4.13(a) and (b), respectively, where inverting the *Mx* template does not highlight

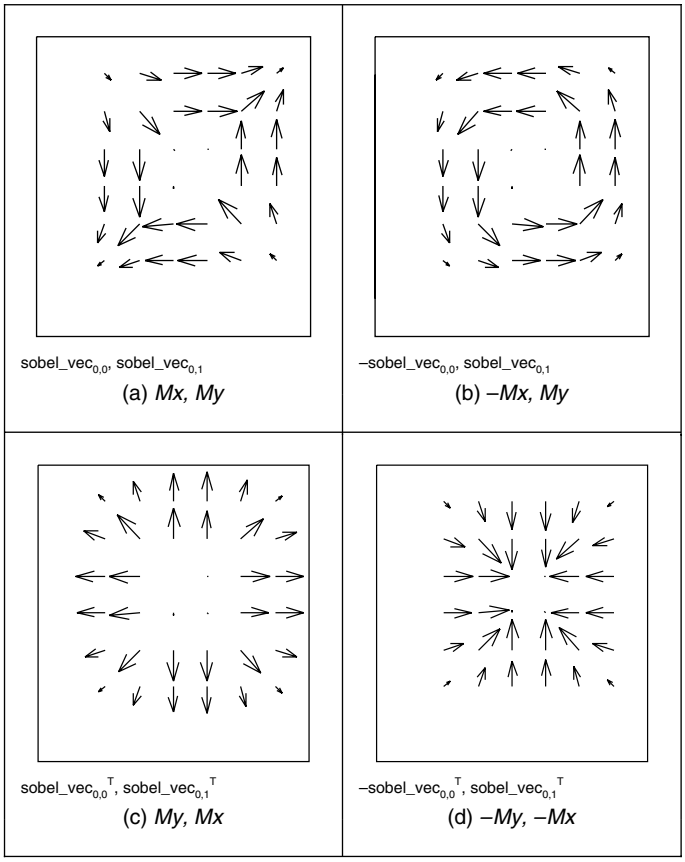


Figure 4.13 Alternative arrangements of edge direction

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 4.9b). 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 that 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 the other. In Figure 4.14, the smoothing is of horizontal spatial frequencies, while the differencing is of vertical spatial frequencies.

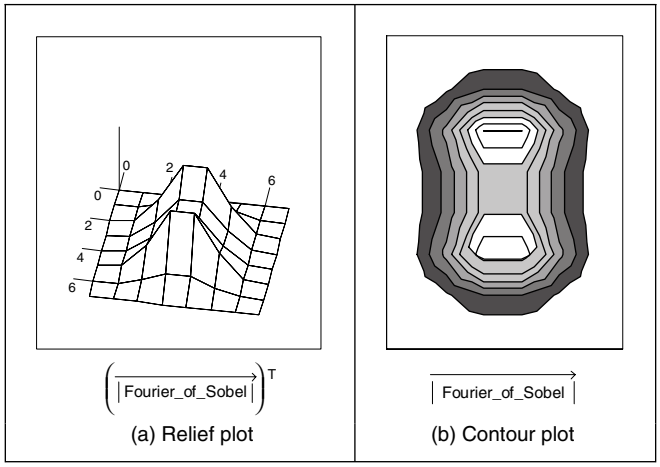


Figure 4.14 Fourier transform of the Sobel operator

An alternative frequency domain analysis of the Sobel can be derived via the z-transform operator. This is more 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 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) $z^{-1} = e^{-j\omega t}$ to transform from the time (z) domain to the frequency domain (ω), we have

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

where the transform *Sobel* is a function of spatial frequency, ω_x, ω_y, along the x and the y axes. This conforms rather well to the separation between smoothing along one axis (the first part of Equation 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 discrete Fourier transform (DFT), and this approach is evident by applying the DFT relationship (Equation 2.15) to the components of the Sobel operator.

4.2.5 Canny edge detection operator

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

- *optimal* detection with no spurious responses
- *good* localization with minimal distance between detected and true edge position
- *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 *non-maximum suppression* (which is equivalent to peak detection). Non-maximum 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 non-maximum 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:

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

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

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

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. 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 then 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 substitution of Equation 4.18 in Equation 4.20,

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

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 non-maximum 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 that 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:

1. Use Gaussian smoothing (as in Section 3.4.4) (Figure 4.15a).
2. Use the Sobel operator (Figure 4.15b).
3. Use non-maximal suppression (Figure 4.15c).
4. Threshold with hysteresis to connect edge points (Figure 4.15d).

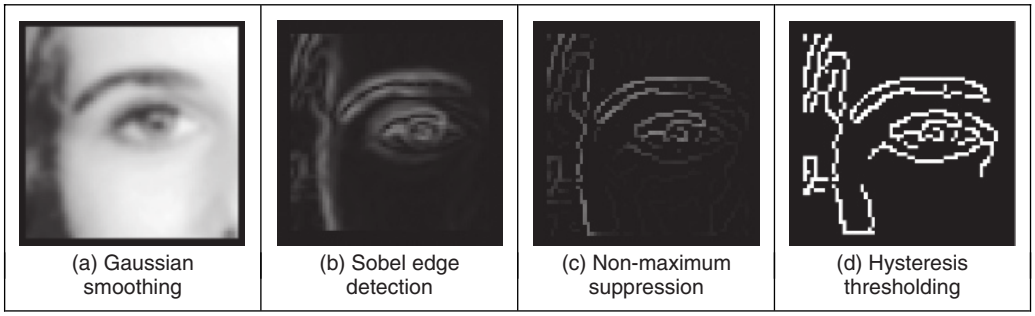


Figure 4.15 Stages in Canny edge detection

Note that the first two stages can be combined using a version of Equation 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 two-dimensional (2D) recursive filters, claiming performance and implementation advantage over the approximation here.

Non-maximum 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 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 that is normal to the edge at a point. This is illustrated in Figure 4.16, which shows the neighbouring 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 neighbourhood, M_1 and

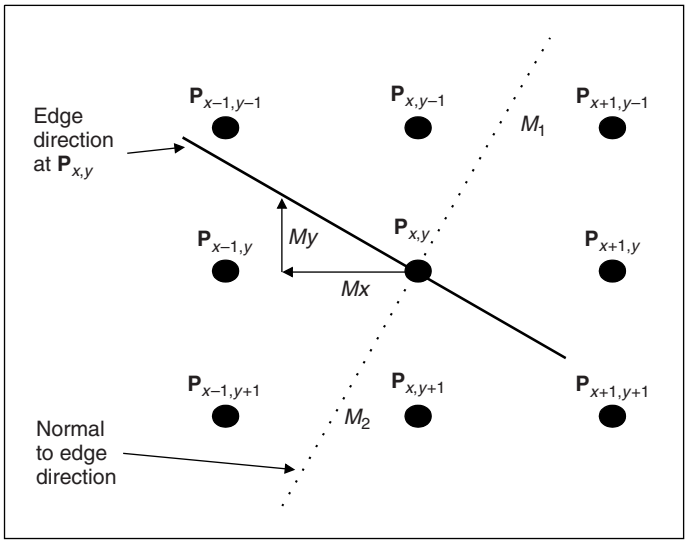


Figure 4.16 Interpolation in non-maximum suppression

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 neighbours 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 non-maximum suppression first requires a function that 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 non-maximum 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 is retained, otherwise it is discarded. Note that the potential singularity in Equations 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 hand-crafted interpretation of Equations 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 Chapter 4.

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.

```
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 Non-maximum suppression

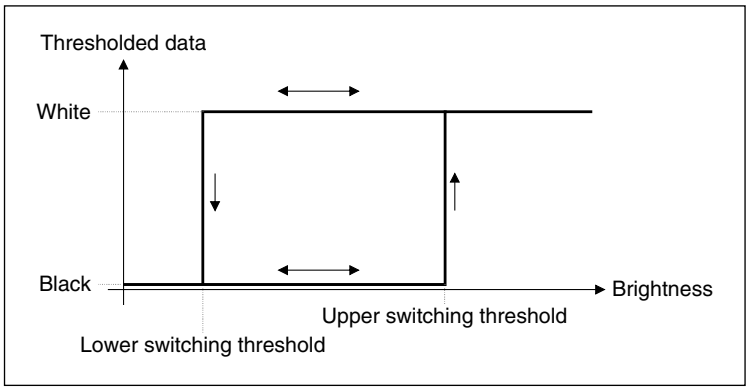


Figure 4.17 Hysteresis thresholding transfer function

The application of non-maximum suppression and hysteresis thresholding is illustrated in Figure 4.18. This contains a ridge of edge data, the edge magnitude. The action of non-maximum 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

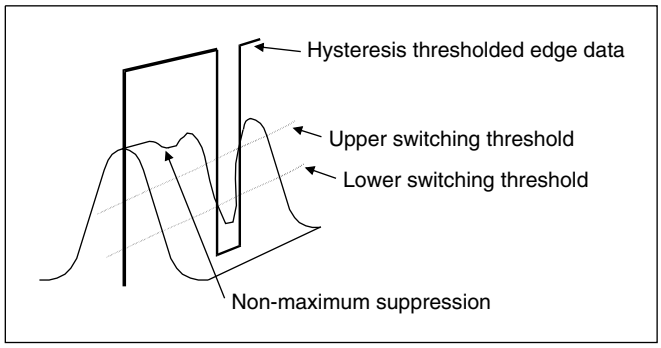


Figure 4.18 Action of non-maximum suppression and hysteresis thresholding

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 non-maximum suppression is found to exceed the upper threshold. This is labelled as an edge point (usually white, with a value of 255) and forms the first point of a line of edge points. The neighbours of the point are then searched to determine whether or not they exceed the lower threshold, as in Figure 4.19. Any neighbour that exceeds the lower threshold is labelled as an edge point and its neighbours 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* point for a search. Its neighbours, in turn, become seed points if they exceed the lower threshold, and so the search extends, along branches arising from neighbours that exceeded the lower threshold. For each branch, the search terminates at points that have no neighbours above the lower threshold.

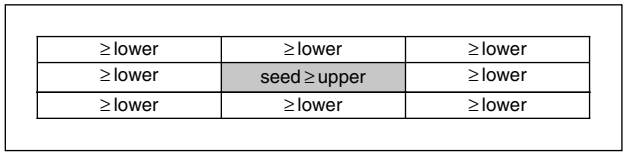


Figure 4.19 Neighbourhood search for hysteresis thresholding

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 neighbours 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 neighbourhood (as in Figure 4.19) is then searched by a function `connect` (Code 4.12) which is fed with the non-maximum suppressed edge image, the coordinates of the seed point whose connectivity is under analysis and the lower switching threshold. Each of the neighbours is searched if its value exceeds the lower threshold, and the point has not already been labelled as white (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.

```
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

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 non-maximum 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 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 pixels, and a lower threshold of 10 pixels. Figure 4.20(b) and (c) show the result of uniform thresholding applied to the image with thresholds of 40 pixels and 10 pixels, respectively. Uniform thresholding can select too few points if the threshold is too high, and too many if it is too low. Hysteresis thresholding selects *all* the points in Figure 4.20(b), and *some* of those in Figure 4.20(c), those connected to the points in (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 (a), whereas it is only partial in (b) and complete (but obscured) in (c). Hysteresis thresholding therefore has an ability to detect major features of interest in the edge image, in an improved manner to uniform thresholding.

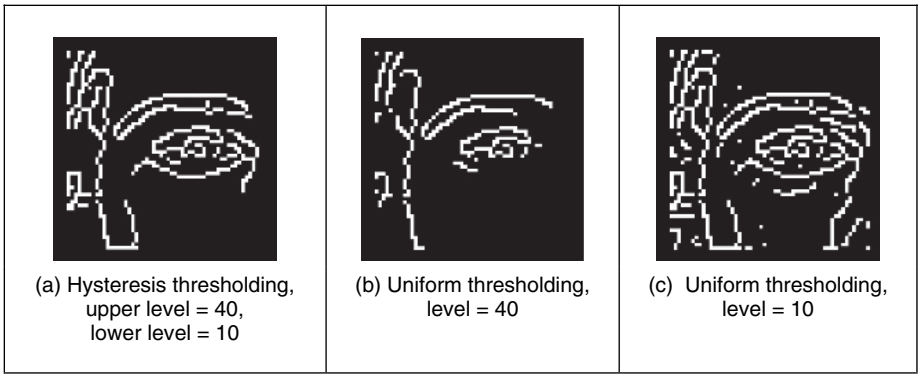


Figure 4.20 Comparing hysteresis thresholding with uniform thresholding

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).

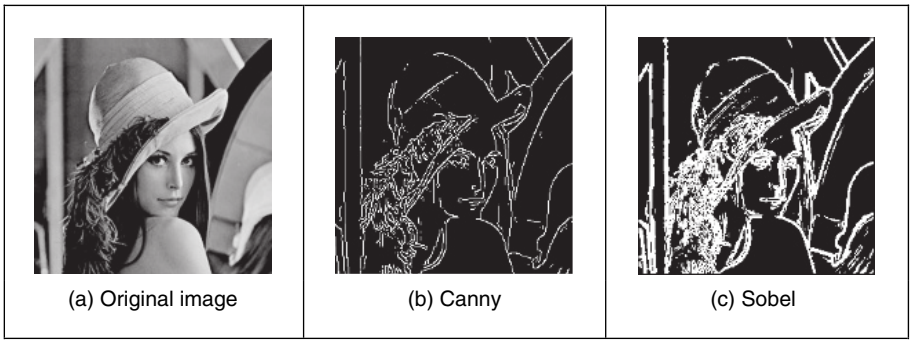


Figure 4.21 Comparing Canny with Sobel

4.3 Second order edge detection operators

4.3.1 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 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 higher order derivatives; applied to the same cross-section of data, the *second order* derivative, $f''(x) = \mathrm{d}^2 f/\mathrm{d}x^2$ in Figure 4.22(c), is 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.

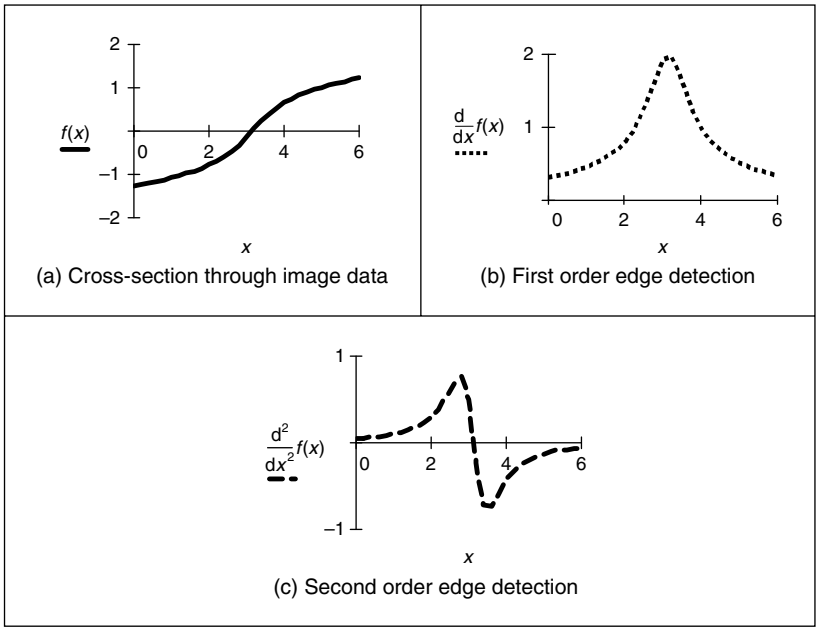


Figure 4.22 First and second order edge detection

4.3.2 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 Equation 4.6, gives

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

This gives a horizontal second order template, as given in Figure 4.23.

Figure 4.23 Horizontal second order template

When the horizontal second order operator is combined with a vertical second order difference we obtain the full Laplacian template, given in Figure 4.24. Essentially, this computes the difference between a point and the average of its four direct neighbours. This was the operator used earlier in anisotropic diffusion (Section 3.5.4), where it is an approximate solution to the heat equation.

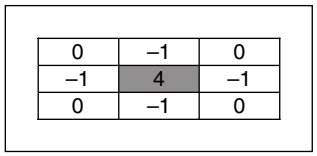


Figure 4.24 Laplacian edge detection operator

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 lie between the edge of the square and its background.

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

(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

An alternative structure to the template in Figure 4.24 is one where the central weighting is 8 and the neighbours are all weighted as -1. 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 neighbouring points, including the corners.) In both structures, the central weighting can be negative and that of the four or the eight neighbours can be positive, without loss of generality. It is important to ensure that the sum of template coefficients is zero, so that 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. Although interest in the Laplacian operator abated with rising interest in the Marr–Hildreth approach, a non-linear Laplacian operator was developed (Vliet and Young, 1989) and shown to have good performance, especially in low-noise situations.

4.3.3 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 $\mathbf{P}$. 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 $\nabla^2 (g(x, y))$ and convolve this with the image. By further differentiation of Equation 4.17, we achieve a LoG operator:

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

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.

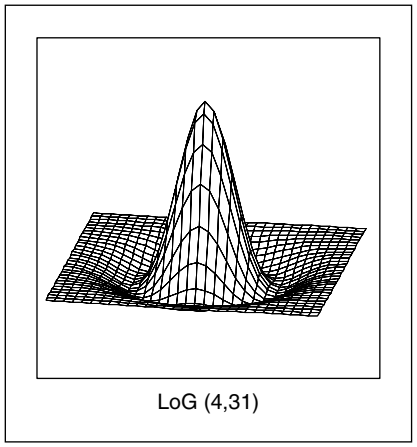


Figure 4.26 Shape of Laplacian of Gaussian operator

The calculation of the Laplacian of Gaussian 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* (Equation 4.17) is a *first* order operator including Gaussian smoothing, $\nabla g(x, y)$. It does seem counter-intuitive that the difference of two smoothing operators should lead to second order edge detection. The approximation is illustrated in Figure 4.27, where in one dimension, two Gaussian distributions of different variance are subtracted to form a one-dimensional (1D) operator whose cross-section is equivalent to the shape of the LoG operator (a cross-section of Figure 4.26).

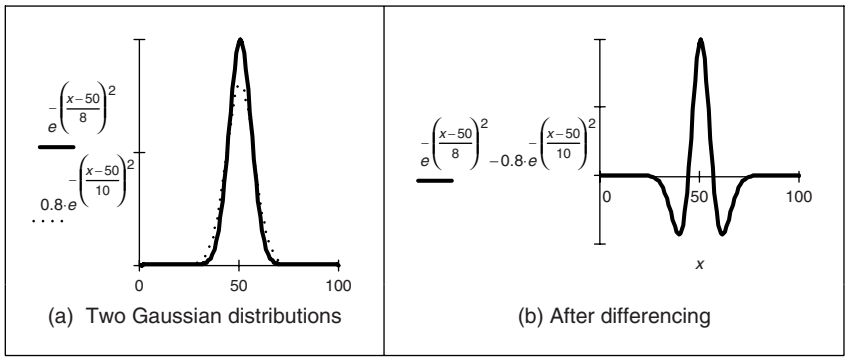


Figure 4.27 Approximating the Laplacian of Gaussian by difference of Gaussian

The implementation of Equation 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 centre of the template, basing differentiation on those points nearer the centre; the standard deviation, σ, is chosen to ensure this action. Again, it is isotropic consistent with Gaussian smoothing.

```
LoG( ,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← (1/2)·((nx^2+ny^2)/2 - 2).e^-((nx^2+ny^2)/(2.^2))
              | template←normalize(template)
              | template
```

Code 4.14 Implementation of the Laplacian of Gaussian operator

Determining the zero-crossing points is a major difficulty with this approach. A variety of techniques 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 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 3).

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 centre 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 quadrant, there must be a zero-crossing at the centre point. In `zerox`, Code 4.15, the average intensity in each quadrant is then evaluated, giving four values and `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 neighbourhood. If one exists, the output image at that point is marked as white, otherwise it is set to black.

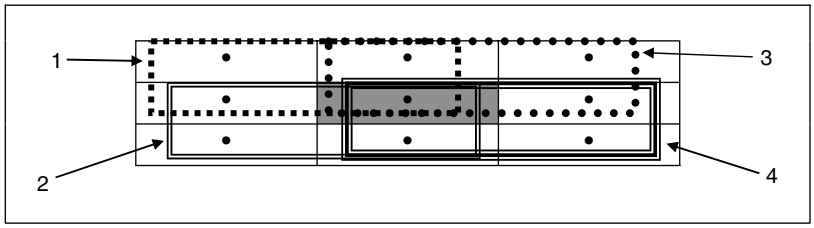


Figure 4.28 Regions for zero crossing detection

```
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 given in Figure 4.29, applied to the face image 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 neighbourhood) are shown in Figure 4.29(b) and (c) for LoG operators of size and variance 11×11 with $\sigma = 0.8$ and 15×15 with $\sigma = 1.8$, 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 in Figure 4.29(b) join to form larger regions 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

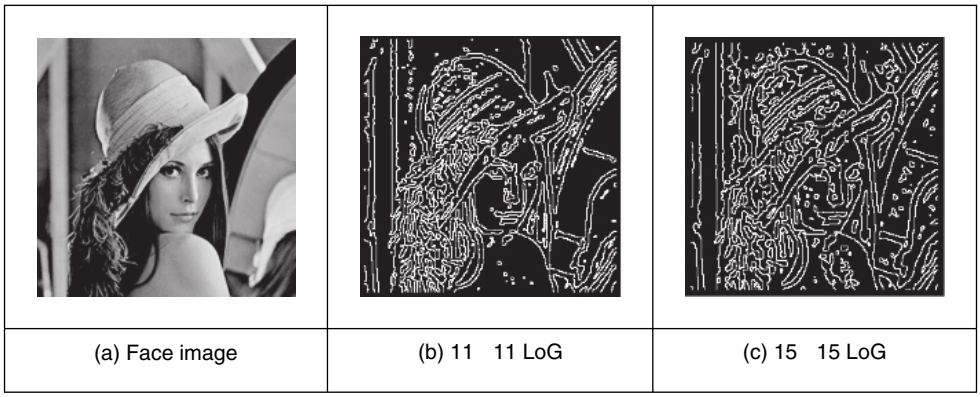


Figure 4.29 Marr–Hildreth edge detection

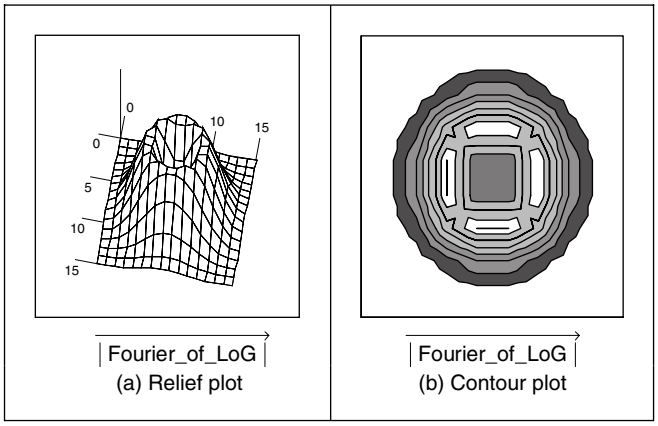


Figure 4.30 Fourier transform of LoG operator

those far away from the origin) it is 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 multiresolution 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 could give good performance. Unfortunately, the implementation appeared to be different from the original LoG operator (and has appeared in some texts in this form), as noted by one of the Marr–Hildreth study's originators (Grimson and Hildreth, 1985). This lead to a somewhat spirited reply (Haralick, 1985) 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). Although 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 that 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 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 centre). The similarity is not coincidental: Mark developed the one here after conversations with Steve Gunn, who he works with!

4.4 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 operators 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. These operators 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 will be reviewed briefly, 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 modelled 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, 1985), as such challenging the optimality of the Gaussian operator for noise smoothing (in step edge detection), although such advantage should be explored in application.

Petrou questioned the validity of the step-edge model for real images (Petrou and Kittler, 1991). 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 much wider 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 grey level change. The *Susan* operator (Smith and Brady, 1997) derives from an approach aimed to find more that just edges, since it can also be used to derive *corners* (where feature boundaries change direction sharply, as in *curvature* detection in Section 4.8) 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 centred on the nucleus. The USAN is essentially the number of pixels within the circular mask that have similar brightness to the nucleus. The edge strength is then derived by subtracting the USAN size from a geometric threshold, which is, say, three-quarters of the maximum USAN size. The method includes a way of calculating edge direction, which is essential if non-maximum suppression is to be 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.

4.5 Comparison of edge detection operators

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. 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 3.5.4), ultrasound images are very *noisy* and require *filtering* before analysis. Figure 4.31(a) is part of the ultrasound image which could have been filtered using the truncated median operator (Section 3.5.3). The image contains a feature called the pitus (the 'splodge' in the middle) and we shall see how different edge operators can be used to detect its perimeter, although without noise filtering. Earlier, in Section 3.5.4, we considered a comparison of statistical operators on an ultrasound image. The median is perhaps the most popular of these processes for general (i.e. non-ultrasound) applications. Accordingly, it is of interest that one study (Bovik et al., 1987) has suggested that the known advantages of median filtering (the removal 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, although 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 4.31b) responds rather nicely to the noise and it is difficult to select a threshold that reveals a major part of the pitus border. Some is present in the Prewitt and Sobel operators' results (Figure 4.31c and d, respectively), but there is still much noise in the processed image, although there is less in the Sobel. The Laplacian operator (Figure 4.31e) 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 4.31f), 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

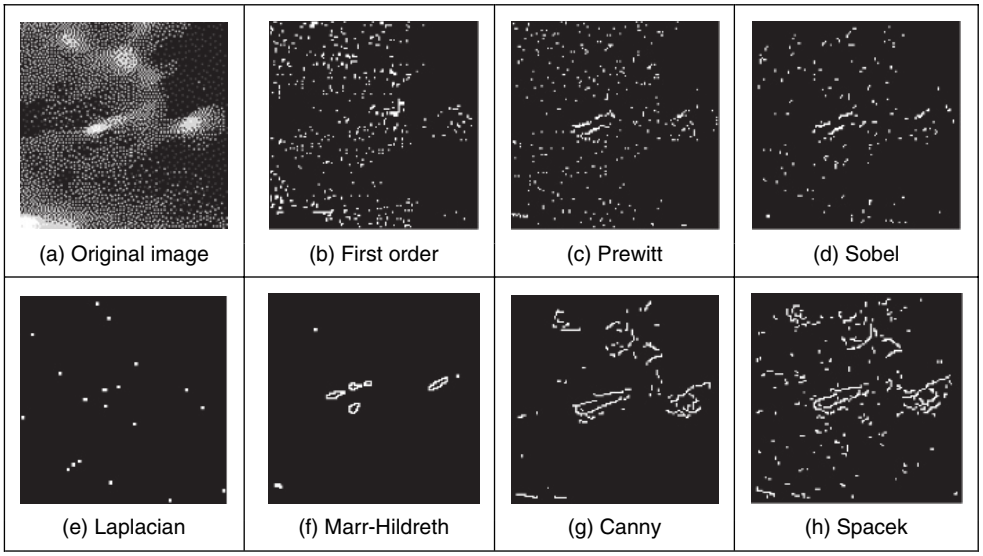


Figure 4.31 Comparison of edge detection operators

pitus itself. In an automated analysis system, for this application, the extra complexity of the more sophisticated operators would clearly be warranted.

4.6 Further reading on edge detection

Few computer vision and image processing texts omit detail concerning edge detection operators, although few give explicit details concerning implementation. Many of the earlier texts omit the more recent techniques. Parker (1994) only includes C code for some of the most basic edge detection techniques. Further information can be found in journal papers; Petrou's excellent study of edge detection (Petrou, 1994) highlights study of the performance factors involved in the optimality of the Canny, Spacek and Petrou operators with extensive tutorial support (although I suspect that Petrou junior may one day be embarrassed by the frequency with which 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. See, for example, Torre and Poggio (1986), which 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) suggest 'a general tool to assist in practical implementations of parametric edge detectors where an automatic process is required' and use 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 conference proceedings to see any new techniques, or perhaps more likely performance comparison or improvement, that may help you to solve a problem.

4.7 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, however, 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 (i.e. time). It is illustrated detecting some 1D features in Figure 4.32, where the features are the solid lines: a (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 4.32a), the constituent frequencies (the dotted lines in Figure 4.32a) 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 to find the feature in which we are interested, 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.

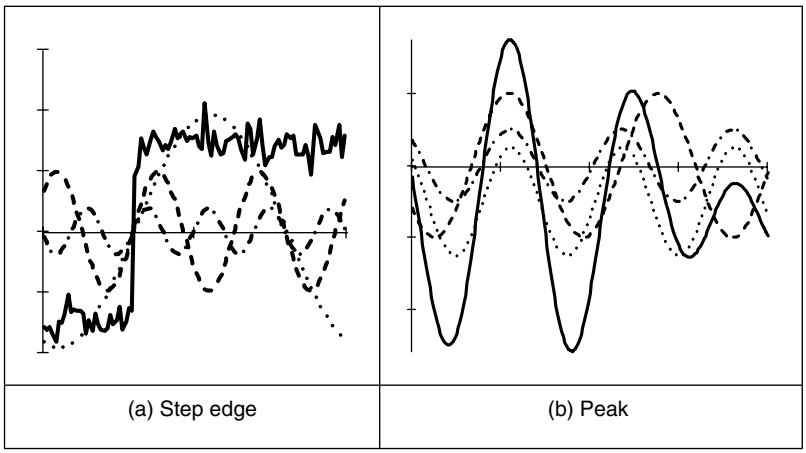


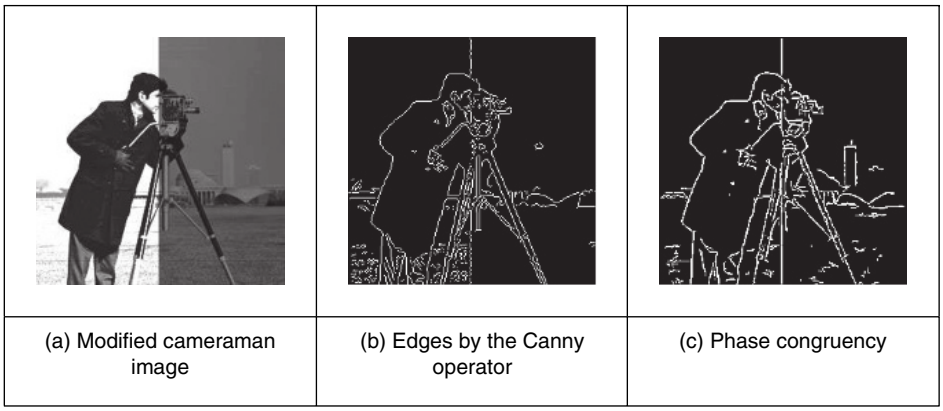
Figure 4.32 Low-level feature extraction by phase congruency

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 Equation 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 (Equation 2.7) and $\phi(\mathbf{Fp}_u) = \langle \mathbf{Fp}_u$ is the argument, the phase in Equation 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.

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 4.33a). Edges detected by Canny are shown in Figure 4.33(b) and by phase congruency in 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 by phase congruency. 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.



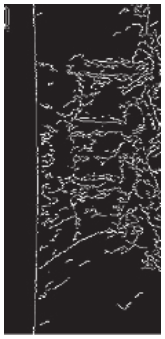
(a) Modified cameraman image | (b) Edges by the Canny operator | (c) Phase congruency

Figure 4.33 Edge detection by Canny and by phase congruency

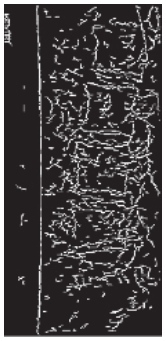
Such a change in brightness might appear unlikely in practical application, but this is not the case with moving objects which interact with illumination or in fixed applications where illumination changes. In studies aimed to extract spinal information from digital videofluoroscopic X-ray images to 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 4.34b), but the detail of a section of the phase congruency result (Figure 4.34c) shows that the vertebrae information is readily available for later high-level feature extraction.

(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)

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 Equation 4.29, Morrone and Owens defined a measure of phase congruency *PC* as

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

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 $\bar{\phi}(x)$ 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 *PC* 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.

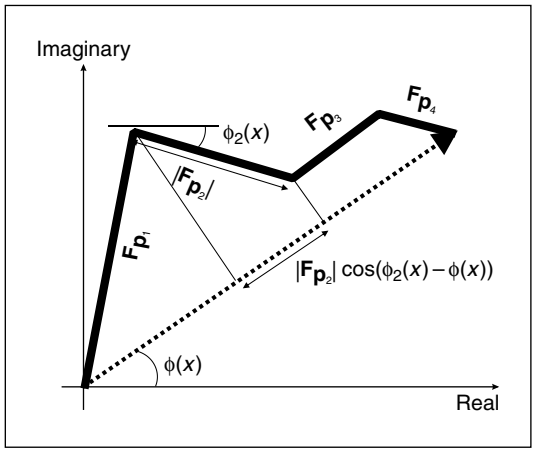


Figure 4.35 Summation in phase congruency

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, although the phase congruency is largest at the right place.

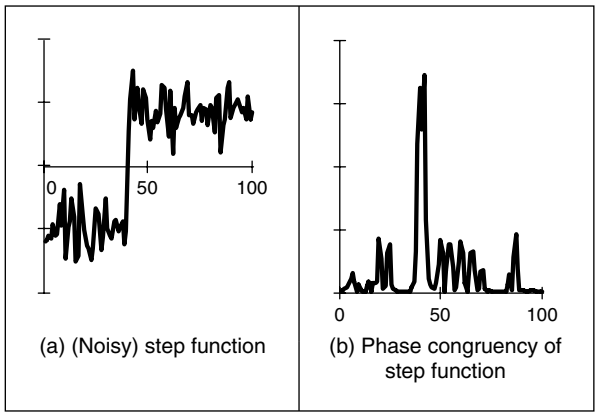


Figure 4.36 One-dimensional phase congruency

One interpretation of the measure is that since for small angles, $\cos\theta = 1 - \theta^2$, then Equation 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, although 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 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 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 which correspond to each scale of the filter. Given that we are only interested 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

$$F(x) = \sum_n e_n(x)$$

$$H(x) = \sum_n o_n(x) \tag{4.33}$$

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

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

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 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 two dimensions (Kovesi, 1999). For 2D (image) analysis, 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 $\omega = 0$. 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

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

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

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

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

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

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

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

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

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

where the addition of a small factor ε numerator 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 keywords recur 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, as well as 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).

4.8 Localized feature extraction

Two main areas are 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 analysing an image by its corners is especially suited to images of artificial objects. The second area includes more modern approaches that improve performance by using region or patch-based analysis. We shall start with the more established curvature-based operators, before moving to the patch or region-based analysis.

4.8.1 Detecting image curvature (corner extraction)

4.8.1.1 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 endpoints 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 $\varphi(t) = \tan^{-1}\left(\dot{y}(t)/\dot{x}(t)\right)$. The curvature at a point $v(t)$ describes the changes in the direction $\varphi(t)$ with respect to changes in arc length. That is,

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

where s is arc length, along the edge itself. Here φ is the angle of the tangent to the curve. That is, $\varphi = \theta \pm 90°$, where θ is the gradient direction defined in Equation 4.13. That is, if we apply an edge detector operator to an image, 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. That is, 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

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

By considering that $\varphi(t) = \tan^{-1}\left(\dot{y}(t)/\dot{x}(t)\right)$, then the curvature at a point $v(t)$ in Equation 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

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

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

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

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

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

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

In general, the curvature of a parametric curve is computed by evaluating Equation 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 $\dot{x}\left(t\right) = r\cos\left(t\right)$ and $\dot{y}\left(t\right) = -r\sin\left(t\right)$. Thus, $\ddot{y}\left(t\right) = -r\cos\left(t\right)$, $\ddot{x}\left(t\right) = -r\sin\left(t\right)$ and $\kappa\left(t\right) = 1 \big/ r$. However, for curves in digital images, the derivatives must be computed from discrete data. This can be done in three 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 changes in image intensity. Finally, a measure of curvature can be obtained by correlation.

4.8.1.2 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 (Bennett 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 Equation 4.41 as the difference between neighbouring pixels. As such, curvature is simply given by

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

where the sequence $\ldots \varphi_{t-1}, \varphi_t, \varphi_{t+1}, \varphi_{t+2} \ldots$ 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 $\varphi_t = (y_{t-1} - y_{t+1}) \big/ (x_{t-1} - x_{t+1})$, 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. That is,

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

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 the filtering is related to the size of the template (as in Section 3.4.3).

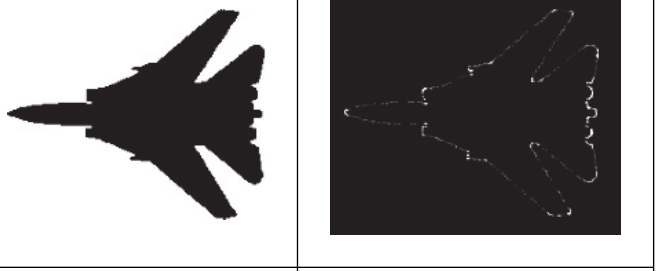
To compute angular differences, we need to determine connected edges. This can easily be implemented with the code already developed for hysteresis thresholding in the Canny edge operator. To compute the difference of points in a curve, the `connect` routine (Code 4.12) only needs to be arranged to store 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 neighbour pixels of each edge. Each edge pixel is connected to one or two neighbours. 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 neighbour. The function `NextPixel` obtains the position of a neighbouring pixel

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

          [rows,columns]=size(inputimage);  %Image size
          outputimage=zeros(rows,columns);  %Result image
          [Mag,Ang]=Edges(inputimage);      %Edge Detection
          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

by taking the position of a pixel and the direction of its neighbour. The curvature is computed as the difference in gradient direction of connected neighbour pixels.

The result of applying this form of curvature detection to an image is shown in Figure 4.37. Figure 4.37(a) contains the silhouette of an object; Figure 4.39(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. 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!



(a) Image (b) Detected corners

Figure 4.37 Curvature detection by difference

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* (Bennett and MacDonald, 1975), a factor which will return to plague us later when we study methods for describing shapes.

4.8.1.3 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 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, we have that the normal tangent direction at a point in a line is given by $\varphi'(x, y) = \tan^{-1}\left(Mx/(-My)\right)$. From this geometry we can observe that

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

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 use an alternative direction to measure of curvature. We can differentiate *backwards* (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,

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

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
         [Myx,Myy]=Gradient(My);           %gradient image


         %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')
                            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 Curvature by measuring changes in intensity

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 appear to offer better performance than that derived by reformulating hysteresis thresholding (Figure 4.37b), although there is little discernible performance advantage between the direction 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

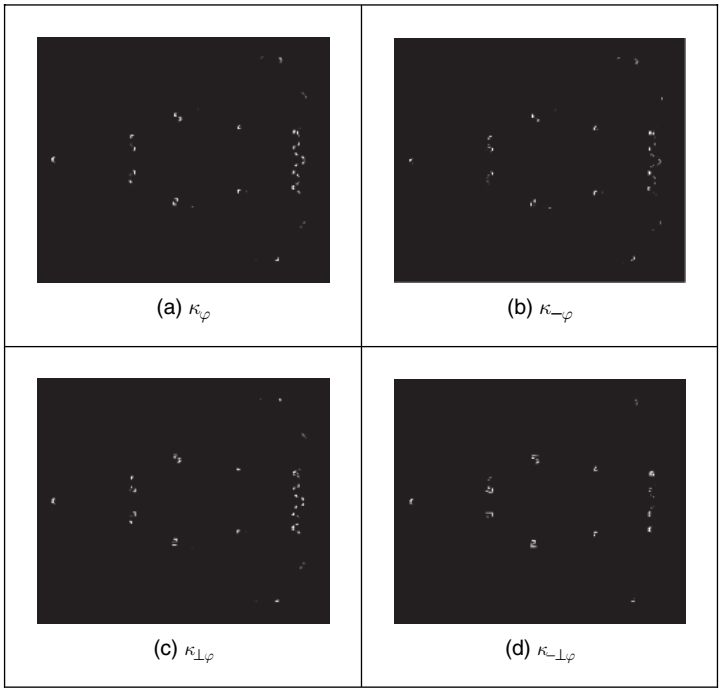


Figure 4.38 Comparing image curvature detection operators

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.)

4.8.1.4 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. That is, for a pixel with coordinates (x, y), and a window size of $2w + 1$ we have:

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

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. That is, 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 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 Equation 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

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

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 Equation 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). 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 Equation 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 were defined in Equation 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 the axes of the coordinate system. That is, we rotate the function $\mathbf{E}_{u,v}(x, y)$ to obtain

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

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 more sensitive the computed curvature will be to changes in the image (and therefore to noise).

In practice, 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 Equation 4.61. This can be derived by considering the matrix forms of Equations 4.61 and 4.64. If we define the vector $D^T = [u, v]$, then Equations 4.60 and 4.63 can be written as

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

where ^{T} denotes the transpose and where

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

To relate Equations 4.60 and 4.63, we consider that $F_{u,v}(x, y)$ is obtained by rotating $E_{u,v}(x, y)$ by a transformation R that rotates the axis defined by D. That is,

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

This can be arranged as

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

By comparison with Equation 4.66, we have:

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

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 Equation 4.70 to obtain the value of $\alpha\beta$, which defines the first term in Equation 4.65 by considering the determinant of the matrices. That is, $\det(\mathbf{Q}) = \det(\mathbf{R}^T)\det(\mathbf{M})\det(\mathbf{R})$. Since $\mathbf{R}$ is a rotation matrix $\det(\mathbf{R}^T)\det(\mathbf{R}) = 1$, thus

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

which defines the first term in Equation 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 Equation 4.70 to obtain the value of $\alpha + \beta$, which defines the first term in Equation 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 Equations 4.64 and 4.73. The equation to be used is selected by the `op` parameter. Curvature is only computed at edge points; that is, 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 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 $\kappa_k(x, y)$

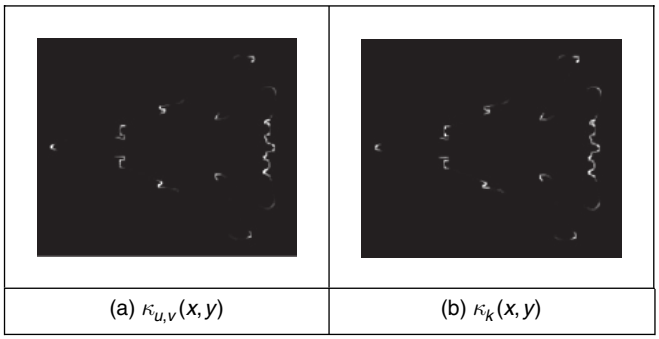


Figure 4.39 Curvature via the Harris operator

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 Equation 4.73. In general, the measure of correlation is not only useful to compute curvature; this technique has much wider application in finding points for matching pairs of images.

4.8.1.5 Further reading on curvature

Many of the arguments earlier advanced on extensions to edge detection in Section 4.4 apply to corner detection as well, so the same advice applies. There is much less attention paid by established textbooks to corner detection through Davis (2005) devotes a chapter to the topic. Van Otterloo's fine book on shape analysis (van Otterloo, 1991) 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 many other corner detection schemes, each offering different attributes, although 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 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 compact way of representing shapes, and at different scales, from coarse (low-level) to fine (detail), and with the ability to handle appearance transformations.

4.8.2 Modern approaches: region/patch analysis

4.8.2.1 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. SIFT 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. It also 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 analysed 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 4.40. 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 image, there are many more of the minor features, concentrated particularly in the textured regions of the image (Figure 4.43). Later, we shall see how this can be used within shape extraction, but the purpose here is the basic low-level features extracted by this new technique.

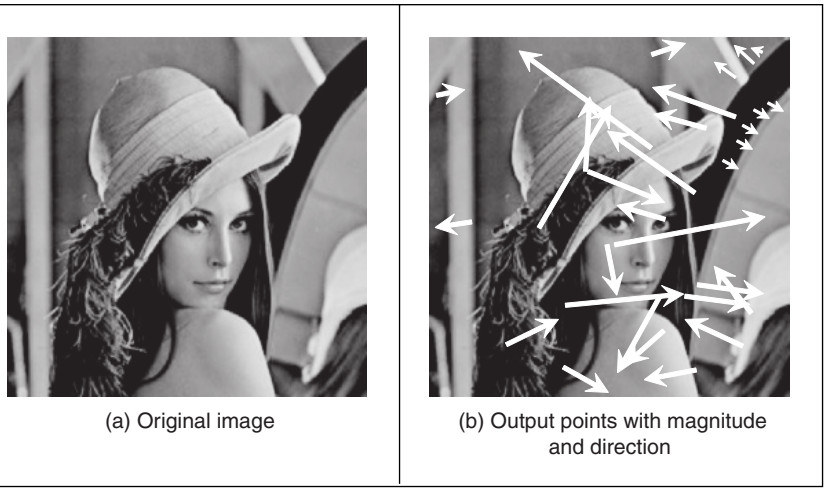


Figure 4.40 Detecting features with the SIFT operator

In the first stage, the difference of Gaussians for an image **P** is computed in the manner of Equation 4.28 as

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

The function L is a scale-space function which can be used to define smoothed images at different scales. Note again the influence of scale-space in the more modern techniques. 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 neighbours. 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 neighbours at that scale and with the nine neighbours in each of the adjacent scales, to determine whether it is a minimum or a maximum, as well as image resampling to ensure comparison between the different scales.

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 4.8.1.4, by thresholding the ratio of Equations 4.71 and 4.72.

To characterize the filtered keypoint features at each scale, the gradient magnitude is calculated in exactly the manner of Equations 4.12 and 4.13 as

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

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

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 non-linear change in image brightness (linear changes are removed by the gradient operations) by analysing 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 4.41, 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 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): a feast for a developer. These images were derived using siftWin32, version 4.

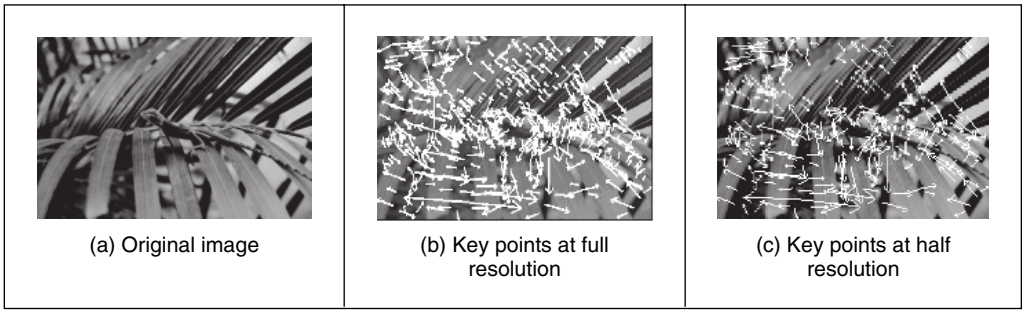


Figure 4.41 SIFT feature detection at different scales

4.8.2.2 Saliency

The new *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–meaning 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 new method has practical capability in that it can be made invariant to rotation, translation, non-uniform scaling and uniform intensity variations and robust to small changes in viewpoint. An example result of processing the image in Fig. 4.42(a) is shown in Figure 4.42(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)

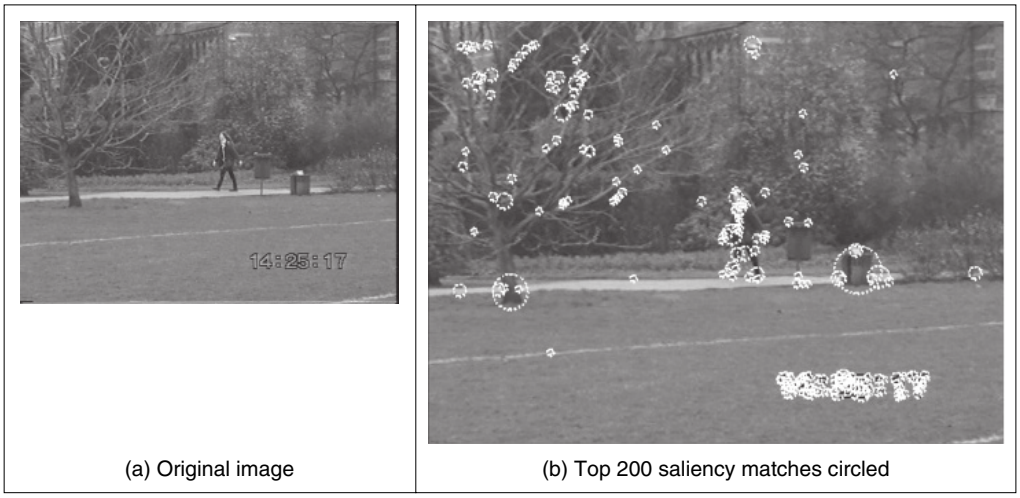


Figure 4.42 Detecting features by saliency

from unlabelled 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 website (http://www.robots.ox.ac.uk/~timork/).

4.8.2.3 Other techniques and performance issues

There has been a recent comprehensive performance review (Mikolajczyk and Schmid, 2005), comparing established and new patch-based operators. The techniques that were compared included 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 this is 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.

4.9 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 *motion* is by image *differencing*. That is, changes or motion can be located by subtracting the intensity values; when there is not motion, the subtraction will give a zero value and when an object in the image moves their pixel's intensity changes, so the subtraction will give a value different of zero.

To denote a sequence of images, we include a time index in our previous notation. That is, $\mathbf{P}(t)_{x,y}$. 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

$$\mathbf{D}(t) = \mathbf{P}(t) - \mathbf{P}(t-1) \tag{4.77}$$

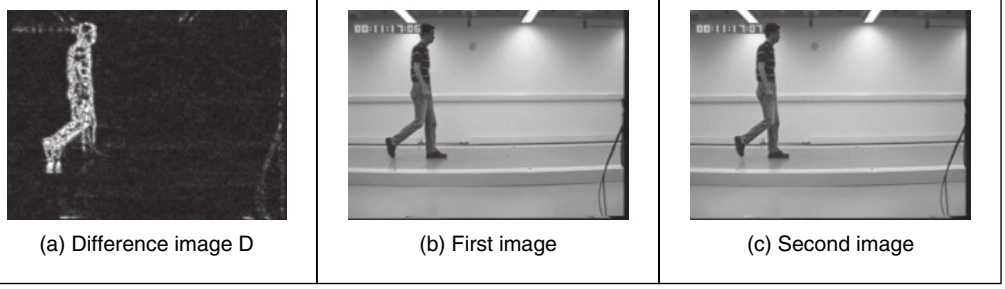


Figure 4.43 Detecting motion by differencing

Figure 4.43 shows an example of this operation. The image in 4.43(a) is the result of subtracting the image in Figure 4.43(b) from the one in Figure 4.43(c). This shows rather more than just the bits that are moving; we have not just highlighted the moving subject, we have also highlighted bits above the subject's head and around his 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. To describe the way in which the points in an image move, we should study how the pixels' position changes in each image frame.

4.9.1 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 project 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/unit of time, more simply pixels/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, the constraints imposed and 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 neighbourhood). 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 non-occluded regions can be related to each other by means of a general transformation of the form

$$\mathbf{P}(t+1)_{x+\delta x, y+\delta y} = \mathbf{P}(t)_{x,y} + \mathbf{H}(t)_{x,y} \tag{4.78}$$

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 Equation 4.78 is generally simplified by assuming that

- the *brightness* of a point in an image is *constant*
- that *neighbouring* points move with *similar* velocity.

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

$$\mathbf{P}(t+1)_{x+\delta x, y+\delta y} = \mathbf{P}(t)_{x,y} \tag{4.79}$$

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

$$e_{x,y} = S\left(\mathbf{P}(t+1)_{x+\delta x, y+\delta y}, \mathbf{P}(t)_{x,y}\right) \tag{4.80}$$

where $S()$ represents a function that measures the similarity between pixels. As such, the optimum is given by the displacements that minimizes the image differences. Alternative measures of similarity 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 Equation 4.80 equivalent to maximum likelihood estimation then we should minimize the squared error. That is,

$$e_{x,y} = \left(\mathbf{P}(t+1)_{x+\delta x, y+\delta y}, \mathbf{P}(t)_{x,y}\right)^2 \tag{4.81}$$

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

$$e_{x,y} = \sum_{(x',y')\in W} \left(\mathbf{P}(t+1)_{x'+\delta x, y'+\delta y}, \mathbf{P}(t)_{x',y'}\right)^2 \tag{4.82}$$

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. 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. Other extensions have considered more elaborate assumptions about the speed of neighbouring 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 4.44. Figure 4.44(a) and (b) show an object moving up into a static background (at least for the two frames we are considering). Figure 4.44(c)–(e) shows the displacements obtained by considering windows of increasing size. Here, we can observe that as the size of the window increases, the result is smoother, but detail has been about

```
%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

the boundary of the object. We can also observe that when the window is small, the are noisy displacements near the object's border. This can be explained by considering that Equation 4.78 supposes that pixels appear in both images, but this is not true near the border since pixels appear and disappear (i.e. occlusion) from and behind the moving object. In addition, there are problems in regions that lack of intensity variations (texture). This is because the minimization

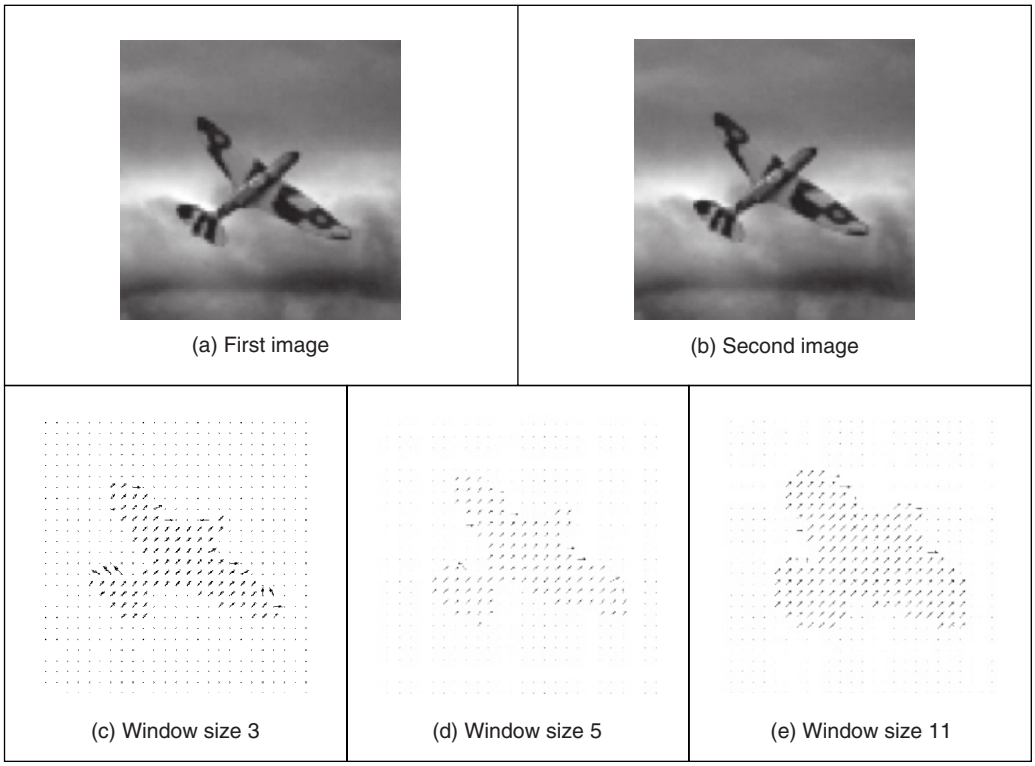


Figure 4.44 Example of area-based motion computation

function in Equation 4.81 is almost flat and there is no clear evidence of the motion. In general, there is not a very effective way of handling these problems since they are caused by the lack of information in the image.

4.9.2 Differential approach

Another popular way to estimate motion focuses on the observation of the differential changes in the pixel values. There are 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 Equation 4.79. According to this, the brightness at the point in the *new* position should be the same as the brightness at the *old* position. Like Equation 4.5, we can expand $\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y}$ by using a Taylor series as

$$\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} + \delta x\frac{\partial \mathbf{P}(t)_{x,y}}{\partial x} + \delta y\frac{\partial \mathbf{P}(t)_{x,y}}{\partial y} + \delta t\frac{\partial \mathbf{P}(t)_{x,y}}{\partial t} + \xi \tag{4.83}$$

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

$$\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y} = \mathbf{P}(t)_{x,y} + \delta x\frac{\partial \mathbf{P}(t)_{x,y}}{\partial x} + \delta y\frac{\partial \mathbf{P}(t)_{x,y}}{\partial y} + \delta t\frac{\partial \mathbf{P}(t)_{x,y}}{\partial t} \tag{4.84}$$

Now by Equation 4.79 we can substitute for $\mathbf{P}(t+\delta t)_{x+\delta x,y+\delta y}$ to give

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

which with some rearrangement gives the motion constraint equation

$$\frac{\delta x}{\delta t}\frac{\partial \mathbf{P}}{\partial x} + \frac{\delta y}{\delta t}\frac{\partial \mathbf{P}}{\partial y} = -\frac{\partial \mathbf{P}}{\partial t} \tag{4.86}$$

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 $u = \delta x/\delta t$ and $v = \delta y/\delta t$. 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

$$u\frac{\partial \mathbf{P}}{\partial x} + v\frac{\partial \mathbf{P}}{\partial y} = -\frac{\partial \mathbf{P}}{\partial t} \tag{4.87}$$

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 be expressed more simply in vector form in terms of the intensity change $\nabla \mathbf{P} = [\nabla x\ \nabla y] = \left[\partial \mathbf{P}/\partial x\ \partial \mathbf{P}/\partial y\right]$ and the optical flow $\mathbf{v} = [u\ v]^T$, as the dot product

$$\nabla \mathbf{P} \cdot \mathbf{v} = -\dot{\mathbf{P}} \tag{4.88}$$

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 Equation 4.77. Unfortunately, we cannot determine the optical flow components from Equation 4.87 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 called the *aperture problem* and makes the problem *ill-posed*. Essentially, we seek estimates of u and v that minimize the error in Equation 4.90 over the entire image. By expressing Equation 4.87 as

$$u\nabla x + v\nabla y + \nabla t = 0 \tag{4.89}$$

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

$$ec = \iint (u\nabla x + v\nabla y + \nabla t)^2 \mathrm{d}x\mathrm{d}y \tag{4.90}$$

We can approach the solution (equations to determine u and v) by considering the second assumption we made earlier, namely that neighbouring points move with similar velocity. This is 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 Equation 4.90. 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, as

$$es = \iint \left(\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 \right) \mathrm{d}x\mathrm{d}y \tag{4.91}$$

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

$$\begin{aligned} e &= \lambda \times ec + es \\ &= \iint \left(\lambda \times \left(u\frac{\partial \mathbf{P}}{\partial x} + v\frac{\partial \mathbf{P}}{\partial y} + \frac{\partial \mathbf{P}}{\partial t} \right)^2 + \left(\left(\frac{\partial u}{\partial x} \right)^2 + \left(\frac{\partial u}{\partial y} \right)^2 + \left(\frac{\partial v}{\partial x} \right)^2 + \left(\frac{\partial v}{\partial y} \right)^2 \right) \right) \mathrm{d}x\mathrm{d}y \end{aligned} \tag{4.92}$$

There are several 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 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 Equation 4.1 at the start of this chapter. Equation 4.90 can be implemented in discrete form as

$$es_{x,y} = \sum_x \sum_y \frac{1}{4} \left(\left(u_{x+1,y} - u_{x,y} \right)^2 + \left(u_{x,y+1} - u_{x,y} \right)^2 + \left(v_{x+1,y} - v_{x,y} \right)^2 + \left(v_{x,y+1} - v_{x,y} \right)^2 \right) \tag{4.93}$$

The discrete form of the smoothness constraint is then that the average rate of change of flow should be minimized. To obtain the discrete form of Equation 4.92 we then add in the discrete of *ec* (the discrete form of Equation 4.90) to give

$$ec_{x,y} = \sum_x \sum_y \left(u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y} \right)^2 \tag{4.94}$$

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

$$\begin{aligned} e_{x,y} &= \sum_x \sum_y \left(\lambda \times ec_{x,y} + es_{x,y} \right) \\ &= \sum_x \sum_y \left(\begin{array}{l} \lambda \times \left(u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y} \right)^2 + \\ \frac{1}{4} \left(\left(u_{x+1,y} - u_{x,y} \right)^2 + \left(u_{x,y+1} - u_{x,y} \right)^2 + \left(v_{x+1,y} - v_{x,y} \right)^2 + \left(v_{x,y+1} - v_{x,y} \right)^2 \right) \end{array} \right) \end{aligned} \tag{4.95}$$

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,

$$\frac{\partial e_{x,y}}{\partial u_{x,y}} = \left(\lambda \times 2 \left(u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y} \right) \nabla x_{x,y} + 2 \left(u_{x,y} - \overline{u}_{x,y} \right) \right) = 0 \tag{4.96}$$

and

$$\frac{\partial e_{x,y}}{\partial v_{x,y}} = \left(\lambda \times 2 \left(u_{x,y}\nabla x_{x,y} + v_{x,y}\nabla y_{x,y} + \nabla t_{x,y} \right) \nabla y_{x,y} + 2 \left(v_{x,y} - \overline{v}_{x,y} \right) \right) = 0 \tag{4.97}$$

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

$$\begin{aligned} \left(1 + \lambda \left(\nabla x_{x,y} \right)^2 \right) u_{x,y} + \lambda \nabla x_{x,y}\nabla y_{x,y} v_{x,y} &= \overline{u}_{x,y} - \lambda \nabla x_{x,y}\nabla t_{x,y} \\ \lambda \nabla x_{x,y}\nabla y_{x,y} u_{x,y} + \left(1 + \lambda \left(\nabla y_{x,y} \right)^2 \right) v_{x,y} &= \overline{v}_{x,y} - \lambda \nabla x_{x,y}\nabla t_{x,y} \end{aligned} \tag{4.98}$$

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

$$\begin{aligned}\left(1+\lambda\left(\left(\nabla x_{x,y}\right)^2+\left(\nabla y_{x,y}\right)^2\right)\right)u_{x,y}&=\left(1+\lambda\left(\nabla y_{x,y}\right)^2\right)\overline{u}_{x,y}-\lambda\nabla x_{x,y}\nabla y_{x,y}\overline{v}_{x,y}-\lambda\nabla x_{x,y}\nabla t_{x,y}\\ \left(1+\lambda\left(\left(\nabla x_{x,y}\right)^2+\left(\nabla y_{x,y}\right)^2\right)\right)v_{x,y}&=-\lambda\nabla x_{x,y}\nabla y_{x,y}\overline{u}_{x,y}+\left(1+\lambda\left(\nabla x_{x,y}\right)^2\right)\overline{v}_{x,y}-\lambda\nabla y_{x,y}\nabla t_{x,y}\end{aligned} \tag{4.99}$$

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

$$\begin{aligned}u_{x,y}^{<n+1>}&=\overline{u}_{x,y}^{<n>}-\lambda\left(\frac{\nabla x_{x,y}\overline{u}_{x,y}+\nabla y_{x,y}\overline{v}_{x,y}+\nabla t_{x,y}}{\left(1+\lambda\left(\nabla x_{x,y}{}^2+\nabla y_{x,y}{}^2\right)\right)}\right)\left(\nabla x_{x,y}\right)\\ v_{x,y}^{<n+1>}&=\overline{v}_{x,y}^{<n>}-\lambda\left(\frac{\nabla x_{x,y}\overline{u}_{x,y}+\nabla y_{x,y}\overline{v}_{x,y}+\nabla t_{x,y}}{\left(1+\lambda\left(\nabla x_{x,y}{}^2+\nabla y_{x,y}{}^2\right)\right)}\right)\left(\nabla y_{x,y}\right)\end{aligned} \tag{4.100}$$

Now we have it, the pair of equations gives iterative means for calculating the images of optical flow based on differentials. To estimate the first order differentials, rather than use our earlier equations, we can consider neighbouring points in quadrants in successive images. This gives approximate estimates of the gradient based on the two frames. That is,

$$\begin{aligned}\nabla x_{x,y}&=\frac{\left(\mathbf{P}(0)_{x+1,y}+\mathbf{P}(1)_{x+1,y}+\mathbf{P}(0)_{x+1,y+1}+\mathbf{P}(1)_{x+1,y+1}\right)-\left(\mathbf{P}(0)_{x,y}+\mathbf{P}(1)_{x,y}+\mathbf{P}(0)_{x,y+1}+\mathbf{P}(1)_{x,y+1}\right)}{8}\\ \nabla y_{x,y}&=\frac{\left(\mathbf{P}(0)_{x,y+1}+\mathbf{P}(1)_{x,y+1}+\mathbf{P}(0)_{x+1,y+1}+\mathbf{P}(1)_{x+1,y+1}\right)-\left(\mathbf{P}(0)_{x,y}+\mathbf{P}(1)_{x,y}+\mathbf{P}(0)_{x+1,y}+\mathbf{P}(1)_{x+1,y}\right)}{8}\end{aligned} \tag{4.101}$$

In fact, in a later reflection (Horn and Schunk, 1993) on the earlier presentation, Horn noted with rancour that some difficulty experienced with the original technique had 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

$$\nabla t_{x,y}=\frac{\left(\mathbf{P}(1)_{x,y}+\mathbf{P}(1)_{x+1,y}+\mathbf{P}(1)_{x,y+1}+\mathbf{P}(1)_{x+1,y+1}\right)-\left(\mathbf{P}(0)_{x,y}+\mathbf{P}(0)_{x+1,y}+\mathbf{P}(0)_{x,y+1}+\mathbf{P}(0)_{x+1,y+1}\right)}{8} \tag{4.102}$$

Note that if the spacing between the images is other than one unit, this will change the denominator in Equations 4.101 and 4.102, but this is a constant scale factor. We also need means to calculate the averages. These can be computed as

$$\begin{aligned}\bar{u}_{x,y}&=\frac{u_{x-1,y}+u_{x,y-1}+u_{x+1,y}+u_{x,y+1}}{2}+\frac{u_{x-1,y-1}+u_{x-1,y+1}+u_{x+1,y-1}+u_{x+1,y+1}}{4}\\ \bar{v}_{x,y}&=\frac{v_{x-1,y}+v_{x,y-1}+v_{x+1,y}+v_{x,y+1}}{2}+\frac{v_{x-1,y-1}+v_{x-1,y+1}+v_{x+1,y-1}+v_{x+1,y+1}}{4}\end{aligned} \tag{4.103}$$

The implementation of the computation of optical flow by the iterative solution in Equation 4.100 is presented in Code 4.20. This function has two parameters that define the smoothing parameter

```
%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

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 Equation 4.100. Derivatives and averages are computed by using simplified forms of Equations 4.101–4.103. 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 reduce noise. In addition, 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.

Figure 4.45 shows some examples of optical flow computation. In these examples, we used the same images as in Figure 4.44. 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 4.45 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, upwards, while some of the smaller vectors are not exactly correct. This is because there are occlusions and some regions have similar textures. We could select the brightest of these points

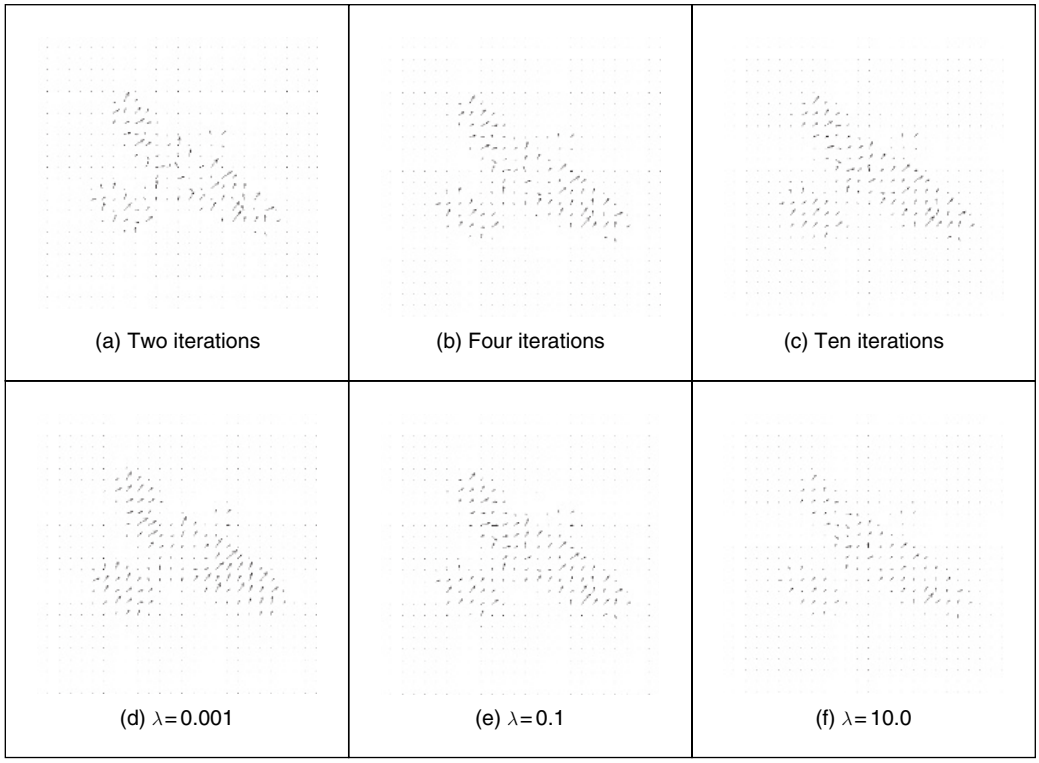


Figure 4.45 Example of differential-based motion computation

by thresholding according to magnitude. That would leave the largest vectors (the ones that 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 the correlation approach suitable (note that fast versions of area-based correspondence are possible; Zabir and Woodfill, 1994). Figure 4.46 shows the result for a walking subject where brightness depicts magnitude (direction is not shown). Figure 4.46(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 4.46(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.

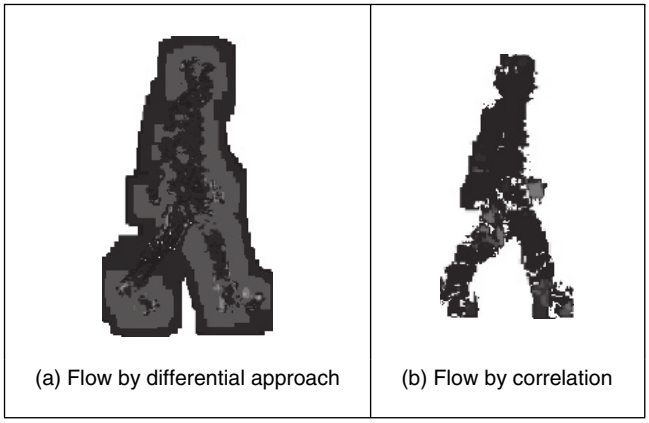


Figure 4.46 Optical flow of walking subject

4.9.3 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 studied here, a Fourier technique and a correlation-based method. Their conclusion was that a local differential method 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). In fact, there have been a number of studies on performance, e.g. of affine flow in Grossmann and Santos-Victor (1997). A thorough analysis of correlation techniques has been developed (Giachetti, 2000) with new algorithms for sub-pixel estimation. One study (Liu et al., 1998) notes how developments have been made for fast or for accurate techniques, without consideration of the trade-off between these two factors. The study compared the techniques mentioned previously with two newer approaches (one fast and one accurate), and also surveys real-time implementations that include implementation via parallel computers and special purpose VLSI chips.

4.10 Conclusions

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, however, 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 and motion extraction have proved sufficiently robust and with requisite performance that they shall endure for some time.

Next, we 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.

4.11 References

Adelson, E. H. and Bergen, J. R., Spatiotemporal Energy Models for the Perception of Motion, *J. Opt. Soc. Am.*, **A2**(2), pp. 284–299, 1985

Apostol, T. M., *Calculus*, 2nd edn, Vol. 1, Xerox College Publishing, Waltham, 1966

Asada, H. and Brady, M., The Curvature Primal Sketch, *IEEE Trans. PAMI*, **8**(1), pp. 2–14, 1986

Barnard, S. T. and Fichler, M. A., Stereo Vision, In: *Encyclopedia of Artificial Intelligence*, John Wiley, New York, pp. 1083–2090, 1987

Barron, J. L., Fleet, D. J. and Beauchemin, S. S., Performance of Optical Flow Techniques, *Int. J. Comput. Vision*, **12**(1), pp. 43–77, 1994

Beauchemin, S. S. and Barron, J. L., The Computation of Optical Flow, *Communs ACM*, pp. 433–467, 1995

Bennet, J. R. and MacDonald, J. S., On the Measurement of Curvature in a Quantized Environment, *IEEE Trans. Comput.*, **C-24**(8), pp. 803–820, 1975

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

Bovik, A. C., Huang, T. S. and Munson, D. C., The Effect of Median Filtering on Edge Estimation and Detection, *IEEE Trans. PAMI*, **9**(2), pp. 181–194, 1987.

Bulthoff, H., Little, J. and Poggio, T., A Parallel Algorithm for Real-Time Computation of Optical Flow, *Nature*, **337**(9), pp. 549–553, 1989

Canny, J., A Computational Approach to Edge Detection, *IEEE Trans. PAMI*, **8**(6), pp. 679–698, 1986

Clark, J. J., Authenticating Edges Produced by Zero-Crossing Algorithms, *IEEE Trans. PAMI*, **11**(1), pp. 43–57, 1989

Davies, E. R., *Machine Vision: Theory, Algorithms and Practicalities*, 3rd edn, Morgan Kaufmann (Elsevier), 2005
Deriche, R., Using Canny's Criteria to Derive a Recursively Implemented Optimal Edge Detector, *Int. J. Comput. Vision*, **1**, pp. 167–187, 1987
Dhond, U. R. and Aggarwal, J. K., Structure From Stereo – A Review, *IEEE Trans. SMC*, **19**(6), pp. 1489–1510, 1989
Fergus, R., Perona, P. and Zisserman, A., Object Class Recognition by Unsupervised Scale-Invariant Learning, *Proc. CVPR 2003*, II, pp. 264–271, 2003
Fleet, D. J. and Jepson, A. D., Computation of Component Image Velocity from Local Phase Information, *Int. J. Comput. Vision*, 5(1), pp. 77–104, 1990
Forshaw, M. R. B., Speeding Up the Marr–Hildreth Edge Operator, *CVGIP*, **41**, pp. 172–185, 1988
Giachetti, A., Matching Techniques to Compute Image Motion, *Image Vision Comput*., **18**(3), pp. 247–260, 2000
Goetz, A., *Introduction to Differential Geometry*, Addison-Wesley, Reading, MA, 1970
Grimson, W. E. L. and Hildreth, E. C., Comments on 'Digital Step Edges from Zero Crossings of Second Directional Derivatives', *IEEE Trans. PAMI*, **7**(1), pp. 121–127, 1985
Groan, F. and Verbeek, P., Freeman-Code Probabilities of Object Boundary Quantized Contours, *CVGIP*, **7**, pp. 391–402, 1978
Grossmann, E. and Santos-Victor, J., Performance Evaluation of Optical Flow: Assessment of a New Affine Flow Method, *Robotics Auton. Syst*., **21**, pp. 69–82, 1997
Gunn, S. R., On the Discrete Representation of the Laplacian of Gaussian, *Pattern Recog*., **32**(8), pp. 1463–1472, 1999
Haddon, J. F., Generalized Threshold Selection for Edge Detection, *Pattern Recog*., **21**(3), pp. 195–203, 1988
Haralick, R. M., Digital Step Edges from Zero-Crossings of Second Directional Derivatives, *IEEE Trans. PAMI*, **6**(1), pp. 58–68, 1984
Haralick, R. M., Author's Reply, *IEEE Trans. PAMI*, **7**(1), pp. 127–129, 1985
Harris, C. and Stephens, M., A Combined Corner and Edge Detector, *Proc. 4th Alvey Vision Conference*, pp. 147–151, 1988
Heath, M. D., Sarkar, S., Sanocki, T. and Bowyer, K. W., A Robust Visual Method of Assessing the Relative Performance of Edge Detection Algorithms, *IEEE Trans. PAMI*, **19**(12), pp. 1338–1359, 1997
Horn, B. K. P., *Robot Vision*, MIT Press, Cambridge, MA, 1986
Horn, B. K. P. and Schunk, B. G., Determining Optical Flow, *Artif. Intell*., **17**, pp. 185–203, 1981
Horn, B. K. P. and Schunk, B. G., Determining Optical Flow: A Retrospective, *Artif. Intell*., **59**, pp. 81–87, 1993
Huang, P. S., Harris, C. J. and Nixon, M. S., Human Gait Recognition in Canonical Space using Temporal Templates, *IEE Proc. Vision Image Signal Process*., **146**(2), pp. 93–100, 1999
Huertas, A. and Medioni, G., Detection of Intensity Changes with Subpixel Accuracy using Laplacian–Gaussian Masks, *IEEE Trans. PAMI*, **8**(1), pp. 651–664, 1986
Jia, X. and Nixon, M. S., Extending the Feature Vector for Automatic Face Recognition, *IEEE Trans. PAMI*, **17**(12), pp. 1167–1176, 1995
Jordan, J. R. III and Bovik, A. C. M. S., Using Chromatic Information in Dense Stereo Correspondence, *Pattern Recog*., **25**, pp. 367–383, 1992
Kadir, T. and Brady, M., Scale, Saliency and Image Description, *Int. J. Comput. Vision*, **45**(2), pp. 83–105, 2001

Kanade, T. and Okutomi, M., A Stereo Matching Algorithm with an Adaptive Window: Theory and Experiment, *IEEE Trans. PAMI*, **16**, pp. 920–932, 1994

Kass, M., Witkin, A. and Terzopoulos, D., Snakes: Active Contour Models, *Int. J. Comput. Vis*., **1**(4), 321–331, 1988

Kitchen, L. and Rosenfeld, A., Gray-Level Corner Detection, *Pattern Recog. Lett*., **1**(2), pp. 95–102, 1982

Korn, A. F., Toward a Symbolic Representation of Intensity Changes in Images, *IEEE Trans. PAMI*, **10**(5), pp. 610–625, 1988

Kovesi, P., Image Features from Phase Congruency. *Videre: J. Comput. Vision Res*., **1**(3), pp. 1–27, 1999

Lawton, D. T., Processing Translational Motion Sequences, *CVGIP*, **22**, pp. 116–144, 1983

Lee, C. K., Haralick, M. and Deguchi, K., Estimation of Curvature from Sampled Noisy Data, *ICVPR'93*, pp. 536–541, 1993

Lindeberg, T., Scale-Space Theory: A Basic Tool for Analysing Structures at Different Scales, *J. Appl. Statist*., **21**(2), pp. 224–270, 1994

Little, J. J. and Boyd, J. E., Recognizing People By Their Gait: The Shape of Motion, *Videre*, **1**(2), pp. 2–32, 1998, online at http://mitpress.mit.edu/e-journals/VIDE/001/v12.html

Little, J. J., Bulthoff, H. H. and Poggio, T., Parallel Optical Flow using Local Voting, *Proc. Int. Conf. Comput. Vision*, pp. 454–457, 1988

Liu, H., Hong, T.-S., Herman, M., Camus, T. and Chellappa, R., Accuracy vs Efficiency Trade-offs in Optical Flow Algorithms, *Comput. Vision Image Understand*., **72**(3), pp. 271–286, 1998

Lowe, D. G., Object Recognition from Local Scale-Invariant Features, *Proc. Int. Conf. Comput. Vision*, pp. 1150–1157, 1999

Lowe, D. G., Distinctive Image Features from Scale-Invariant Key Points, *Int. J. Comput. Vision*, **60**(2), pp. 91–110, 2004

Lucas, B. and Kanade, T., An Iterative Image Registration Technique with an Application to Stereo Vision, *Proc DARPA Image Understanding Workshop*, pp. 121–130, 1981

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

Marr, D. C. and Hildreth, E., Theory of Edge Detection, *Proc. R. Soc. Lond*., **B207**, pp. 187–217, 1980

Mikolajczyk, K. and Schmid, C., A Performance Evaluation of Local Descriptors, *IEEE Trans. PAMI*, **27**(10), pp. 1615–1630, 2005

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

Mokhtarian, F. and Mackworth, A. K., Scale-Space Description and Recognition of Planar Curves and Two-Dimensional Shapes, *IEEE Trans. PAMI*, **8**(1), pp. 34–43, 1986

Morrone, M. C. and Burr, D. C., Feature Detection in Human Vision: A Phase-Dependent Energy Model, *Proc. R. Soc. Lond. B, Biol. Sci*., **235**(1280), pp. 221–245, 1988

Morrone, M. C. and Owens, R. A., Feature Detection from Local Energy, *Pattern Recog. Lett*., **6**, pp. 303–313, 1987

Mulet-Parada, M. and Noble, J. A., 2D+T Acoustic Boundary Detection in Echocardiography, *Med. Image Analysis*, **4**, 21–30, 2000

Myerscough, P.J. and Nixon, M. S., Temporal Phase Congruency, *Proc. IEEE Southwest Symposium on Image Analysis and Interpretation SSIAI '04*, pp. 76–79, 2004

Nagel, H. H., On the Estimation of Optical Flow: Relations between Different Approaches and Some New Results, *Artif. Intell*., **33**, pp. 299–324, 1987

van Otterloo, P. J., *A Contour-Oriented Approach to Shape Analysis*, Prentice Hall International (UK), Hemel Hempstead, 1991
Parker, J. R., *Practical Computer Vision using C*, Wiley & Sons, New York, 1994
Petrou, M., The Differentiating Filter Approach to Edge Detection, *Adv. Electron. Electron Phys*., **88**, pp. 297–345, 1994
Petrou, M. and Kittler, J., Optimal Edge Detectors for Ramp Edges, *IEEE Trans. PAMI*, **13**(5), pp. 483–491, 1991
Prewitt, J. M. S. and Mendelsohn, M. L., The Analysis of Cell Images, *Ann. N. Y. Acad. Sci*., **128**, pp. 1035–1053, 1966
Roberts, L. G., Machine Perception of Three-Dimensional Solids, In: *Optical and Electro-Optical Information Processing*, MIT Press, Cambridge, MA, pp. 159–197, 1965
Rosin, P. L., Augmenting Corner Descriptors, *Graphical Models Image Process*., **58**(3), pp. 286–294, 1996
Smith, S. M. and Brady, J. M., SUSAN – A New Approach to Low Level Image Processing. *Int. J. Comput. Vision*, **23**(1), pp. 45–78, May 1997
Sobel, I. E., *Camera Models and Machine Perception*, PhD Thesis, Stanford University, 1970
Spacek, L. A., Edge Detection and Motion Detection, *Image Vision Comput*., **4**(1), pp. 43–56, 1986
Torre, V. and Poggio, T. A., On Edge Detection, *IEEE Trans. PAMI*, **8**(2), pp. 147–163, 1986
Ulupinar, F. and Medioni, G., Refining Edges Detected by a LoG Operator, *CVGIP*, **51**, pp. 275–298, 1990
Venkatesh, S. and Owens, R. A., An Energy Feature Detection Scheme. *Proc. Int. Conf. Image Process*., Singapore, pp. 553–557, 1989
Venkatesh, S. and Rosin, P. L., Dynamic Threshold Determination by Local and Global Edge Evaluation, *Graphical Models Image Process*., **57**(2), pp. 146–160, 1995
Vliet, L. J. and Young, I. T., A Nonlinear Laplacian Operator as Edge Detector in Noisy Images, *CVGIP*, **45**, pp. 167–195, 1989
Yitzhaky, Y. and Peli, E., A Method for Objective Edge Detection Evaluation and Detector Parameter Selection, *IEEE Trans. PAMI*, **25**(8), pp. 1027–1033, 2003
Zabir, R. and Woodfill, J., Non-Parametric Local Transforms for Computing Visual Correspondence, *Proc. Eur. Conf.Comput. Vision*, pp. 151–158, 1994
Zheng, Y., Nixon, M. S. and Allen, R., Automatic Segmentation of Lumbar Vertebrae in Digital Videofluoroscopic Imaging, *IEEE Trans. Med. Imaging*, **23**(1), pp. 45–52. 2004

This page intentionally left blank

5 Feature extraction by shape matching

5.1 Overview

High-level *feature extraction* concerns finding shapes in computer images. To be able to recognize faces automatically, for example, one approach is to extract the component features. This requires extraction of, say, the eyes, the ears and the nose, which are the major facial features. To find them, we can use their shape: the white part of the eyes is ellipsoidal; the mouth can appear as two lines, as do the eyebrows. Shape extraction implies finding their position, their orientation and their size. This feature extraction process can be viewed as similar to the way in which we perceive the world: many books for babies describe basic geometric shapes such as triangles, circles and squares. More complex pictures can be decomposed into a structure of simple shapes. In many applications, analysis can be guided by the way in which the shapes are arranged. For the example of face image analysis, we expect to find the eyes above (and either side of) the nose, and we expect to find the mouth below the nose.

In feature extraction, we generally seek *invariance properties* so that the extraction process does not vary according to chosen (or specified) conditions. That is, techniques should find shapes reliably and robustly whatever the value of any parameter that can control the appearance of a shape. As a basic *invariant*, we seek immunity to changes in the *illumination* level: we seek to find a shape whether it is light or dark. In principle, as long as there is contrast between a shape and its background, the shape can be said to exist, and can then be detected. (Clearly, any computer vision technique will fail in extreme lighting conditions; you cannot see anything when it is completely dark.) Following illumination, the next most important parameter is *position*: we seek to find a shape wherever it appears. This is usually called *position*, *location* or *translation invariance*. Then, we often seek to find a shape irrespective of its *rotation* (assuming that the object or the camera has an unknown orientation); this is usually called *rotation* or *orientation invariance*. Then, we might seek to determine the object at whatever *size* it appears, which might be due to physical change, or to how close the object has been placed to the camera. This requires *size* or *scale invariance*. These are the main invariance properties we shall seek from our shape extraction techniques. However, nature (as usual) tends to roll balls under our feet: there is always *noise* in images. In addition, since we are concerned with shapes, there may be more than one in the image. If one is on top of the other it will *occlude*, or hide, the other, so not all of the shape of one object will be visible.

But before we can develop image analysis techniques, we need techniques to extract the shapes. Extraction is more complex than *detection*, since extraction implies that we have a

description of a shape, such as its position and size, whereas detection of a shape merely implies knowledge of its existence within an image.

The techniques presented in this chapter are outlined in Table 5.1. To extract a shape from an image, it is necessary to identify it from the background elements. This can be done by considering the intensity information or by comparing the pixels against a given template. In the first approach, if the brightness of the shape is known, then the pixels that form the shape can be extracted by classifying the pixels according to a fixed intensity threshold. Alternatively, if the background image is known, this can be subtracted to obtain the pixels that define the shape of an object superimposed on the background. Template matching is a model-based approach in which the shape is extracted by searching for the best correlation between a known model and the pixels in an image. There are alternative ways in which to compute the correlation between the template and the image. Correlation can be implemented by considering the image or frequency domains. In addition, the template can be defined by considering intensity values or a binary shape. The *Hough transform* defines an efficient implementation of template matching for binary templates. This technique is capable of extracting simple shapes such as lines and quadratic forms, as well as arbitrary shapes. In any case, the complexity of the implementation can be reduced by considering invariant features of the shapes.

Table 5.1 Overview of Chapter 5

Main topic	Sub topics	Main points
Pixel operations	How we detect features at a *pixel* level. Moving object detection. *Limitations* and *advantages* of this approach. Need for *shape* information.	Thresholding. Background subtraction.
Template matching	Shape extraction by *matching*. Advantages and disadvantages. Need for *efficient* implementation.	Template matching. Direct and Fourier implementations. Noise and occlusion.
Hough transform	Feature extraction by *matching*. Hough transforms for *conic sections*. Hough transform for *arbitrary shapes*. *Invariant* formulations. Advantages in *speed* and *efficacy*.	Feature extraction by evidence gathering. Hough transforms for lines, circles and ellipses. Generalized and Invariant Hough transforms.

5.2 Thresholding and subtraction

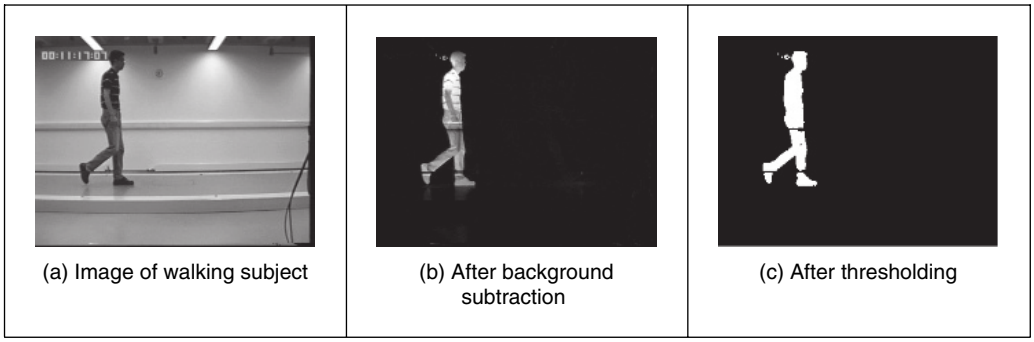
Thresholding is a simple shape extraction technique, as shown in Section 3.3.4, where the images could be viewed as the result of trying to separate the eye from the background. If it can be assumed that the shape to be extracted is defined by its brightness, then thresholding an image at that brightness level should find the shape. Thresholding is clearly sensitive to change in illumination: if the image illumination changes then so will the perceived brightness of the target shape. Unless the threshold level can be arranged to adapt to the change in brightness level, any thresholding technique will fail. Its attraction is *simplicity*: thresholding does not require much computational effort. If the illumination level changes in a linear fashion, using histogram equalization will result in an image that does not vary. Unfortunately, the result of histogram

equalization is sensitive to noise, shadows and variant illumination; noise can affect the resulting image quite dramatically and this will again render a thresholding technique useless.

Thresholding after *intensity normalization* (Section 3.3.2) is less sensitive to noise, since the noise is stretched with the original image and cannot affect the stretching process by much. It is, however, still sensitive to shadows and variant illumination. Again, it can only find application where the illumination can be carefully controlled. This requirement is germane to any application that uses basic thresholding. If the overall illumination level cannot be controlled, it is possible to threshold edge magnitude data since this is insensitive to overall brightness level, by virtue of the implicit differencing process. However, edge data is rarely continuous and there can be gaps in the detected perimeter of a shape. Another major difficulty, which applies to thresholding the brightness data as well, is that there are often more shapes than one. If the shapes are on top of each other, one occludes the other and the shapes need to be separated.

An alternative approach is to *subtract* an image from a known background before thresholding. (We saw how we can estimate the background in Section 3.4.2.) This assumes that the background is known precisely, otherwise many more details than just the target feature will appear in the resulting image; clearly, the subtraction will be unfeasible if there is *noise* on either image, and especially on both. In this approach, there is no implicit shape description, but if the thresholding process is sufficient, it is simple to estimate basic shape parameters, such as position.

The subtraction approach is illustrated in Figure 5.1. Here, we seek to separate or extract the walking subject from the background. We saw earlier, in Figure 3.22, how the median filter can be used to provide an estimate of the background to the sequence of images that Figure 5.1(a) comes from. When we subtract the background of Figure 3.22(i) from the image of Figure 5.1(a), we obtain most of the subject with some extra background just behind the subject's head. This is due to the effect of the moving subject on *lighting*. Also, removing the background removes some of the subject: the horizontal bars in the background have been removed from the subject by the subtraction process. These aspects are highlighted in the thresholded image (Figure 5.1c). It is not a particularly poor way of separating the subject from the background (we have the subject, but we have chopped out his midriff), but it is not especially good either.



(a) Image of walking subject

(b) After background subtraction

(c) After thresholding

Figure 5.1 Shape extraction by subtraction and thresholding

Even though thresholding and subtraction are attractive (because of simplicity and hence their speed), the performance of both techniques is sensitive to partial shape data, noise, variation in illumination and occlusion of the target shape by other objects. Accordingly, many approaches to image interpretation use higher level information in shape extraction, namely how the pixels are connected within the shape. This can resolve these factors.

5.3 Template matching

5.3.1 Definition

Template matching is conceptually a simple process. We need to match a *template* to an image, where the template is a subimage that contains the shape we are trying to find. Accordingly, we centre the template on an image point and count up how many points in the template *matched* those in the image. The procedure is repeated for the entire image, and the point that led to the best match, the maximum count, is deemed to be the point where the shape (given by the template) lies within the image.

Consider that we want to find the template of Figure 5.2(b) in the image of Figure 5.2(a). The template is first positioned at the origin and then matched with the image to give a count which reflects how well the template matched that part of the image at that position. The count of matching pixels is increased by one for each point where the brightness of the template matches the brightness of the image. This is similar to the process of template convolution, illustrated earlier in Figure 3.11. The difference here is that points in the image are matched with those in the template, and the sum is of the number of matching points as opposed to the weighted sum of image data. The best match is when the template is placed at the position where the rectangle is matched to itself. This process can be generalized to find, for example, templates of different *size* or *orientation*. In these cases, we have to try all the templates (at expected rotation and size) to determine the best match.

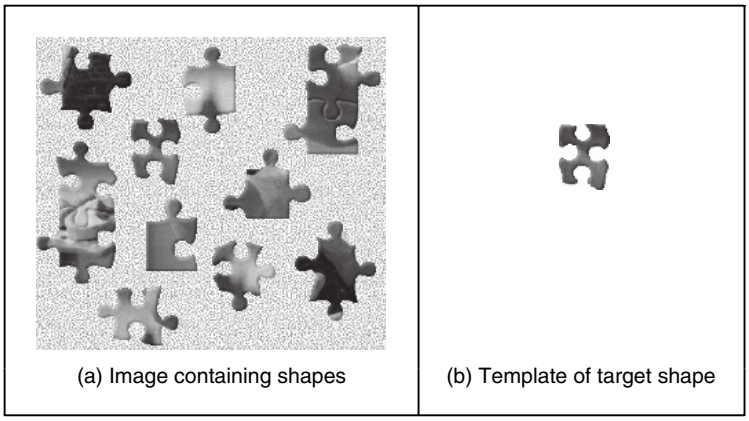


Figure 5.2 Illustrating template matching

Formally, template matching can be defined as a method of parameter estimation. The parameters define the position (and pose) of the template. We can define a template as a discrete function $\mathbf{T}x, y$. This function takes values in a window. That is, the coordinates of the points $(x, y) \in \mathbf{W}$. For example, for a 2×2 template we have that the set of points $\mathbf{W} = \{(0,0), (0,1), (1,0), (1,1)\}$.

Let us consider that each pixel in the image $\mathbf{I}x, y$ is corrupted by additive Gaussian noise. The noise has a mean value of zero and the (unknown) standard deviation is σ. Thus, the

probability that a point in the template placed at coordinates (i, j) matches the corresponding pixel at position $(x, y) \in \mathbf{W}$ is given by the normal distribution

$$p_{i,j}(x, y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2} \tag{5.1}$$

Since the noise affecting each pixel is independent, the probability that the template is at position (i, j) is the combined probability of each pixel that the template covers. That is,

$$L_{i,j} = \prod_{(x,y)\in\mathbf{W}} p_{i,j}(x, y) \tag{5.2}$$

By substitution of Equation 5.1, we have:

$$L_{i,j} = \left(\frac{1}{\sqrt{2\pi}\sigma}\right)^n e^{-\frac{1}{2}\sum_{(x,y)\in\mathbf{W}}\left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2} \tag{5.3}$$

where n is the number of pixels in the template. This function is called the *likelihood* function. Generally, it is expressed in logarithmic form to simplify the analysis. Notice that the logarithm scales the function, but it does not change the position of the maximum. Thus, by taking the logarithm, the likelihood function is redefined as

$$\ln(L_{i,j}) = n \ln\left(\frac{1}{\sqrt{2\pi}\sigma}\right) - \frac{1}{2} \sum_{(x,y)\in\mathbf{W}} \left(\frac{\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}}{\sigma}\right)^2 \tag{5.4}$$

In *maximum likelihood estimation*, we have to choose the parameter that maximizes the likelihood function. That is, the positions that minimize the rate of change of the objective function

$$\frac{\partial \ln(L_{i,j})}{\partial i} = 0 \quad \text{and} \quad \frac{\partial \ln(L_{i,j})}{\partial j} = 0 \tag{5.5}$$

That is,

$$\sum_{(x,y)\in\mathbf{W}} (\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}) \frac{\partial \mathbf{I}_{x+i,y+j}}{\partial i} = 0$$

$$\sum_{(x,y)\in\mathbf{W}} (\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}) \frac{\partial \mathbf{I}_{x+i,y+j}}{\partial j} = 0 \tag{5.6}$$

We can observe that these equations are also the solution of the minimization problem given by

$$\min\ e = \sum_{(x,y)\in\mathbf{W}} \left(\mathbf{I}_{x+i,y+j} - \mathbf{T}_{x,y}\right)^2 \tag{5.7}$$

That is, maximum likelihood estimation is equivalent to choosing the template position that minimizes the squared error (the squared values of the differences between the template points and the corresponding image points). The position where the template best matches the image is the estimated position of the template within the image. Thus, if you measure the match using the squared error criterion, then you will be choosing the *maximum likelihood* solution. This implies that the result achieved by template matching is optimal for images corrupted by Gaussian noise. A more detailed examination of the method of least squares is given in Appendix 3, Section 11.2. (Note that the *central limit theorem* suggests that practically experienced noise can be assumed to be Gaussian distributed, although many images appear to contradict this assumption.) You can use other error criteria, such as the absolute difference, rather than the squared difference or, if you feel more adventurous, you might consider robust measures such as M-estimators.

We can derive alternative forms of the squared error criterion by considering that Equation 5.7 can be written as

$$\min\ e = \sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}{}^2 - 2\mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y} + \mathbf{T}_{x,y}{}^2 \tag{5.8}$$

The last term does not depend on the template position (i, j). As such, it is constant and cannot be minimized. Thus, the optimum in this equation can be obtained by minimizing

$$\min\ e = \sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}{}^2 - 2\sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y} \tag{5.9}$$

If the first term

$$\sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}{}^2 \tag{5.10}$$

is approximately constant, then the remaining term gives a measure of the similarity between the image and the template. That is, we can maximize the *cross-correlation* between the template and the image. Thus, the best position can be computed by

$$\max\ e = \sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y} \tag{5.11}$$

However, the squared term in Equation 5.10 can vary with position, so the match defined by Equation 5.11 can be poor. In addition, the range of the cross-correlation is dependent on the size of the template and it is non-invariant to changes in image lighting conditions. Thus, in an implementation it is more convenient to use either Equation 5.7 or Equation 5.9 (in spite of being computationally more demanding than the cross-correlation in Equation 5.11).

Alternatively, cross-correlation can be *normalized* as follows. We can rewrite Equation 5.8 as

$$\min\ e = 1 - 2\frac{\sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y}}{\sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}{}^2} \tag{5.12}$$

Here, the first term is constant and, thus, the optimum value can be obtained by

$$\max\ e = \frac{\sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}\mathbf{T}_{x,y}}{\sum_{(x,y)\in\mathbf{W}} \mathbf{I}_{x+i,y+j}{}^2} \tag{5.13}$$

In general, it is convenient to normalize the grey level of each image window under the template. That is,

$$\max\ e = \frac{\sum_{(x,y)\in\mathbf{W}} \left(\mathbf{I}_{x+i,y+j} - \bar{\mathbf{I}}_{i,j}\right)\left(\mathbf{T}_{x,y} - \overline{\mathbf{T}}\right)}{\sum_{(x,y)\in\mathbf{W}} \left(\mathbf{I}_{x+i,y+j} - \bar{\mathbf{I}}_{i,j}\right)^2} \tag{5.14}$$

where $\bar{\mathbf{I}}_{i,j}$ is the mean of the pixels $\mathbf{I}_{x+i,y+j}$ for points within the window (i.e. $(x, y) \in \mathbf{W}$) and $\overline{\mathbf{T}}$ is the mean of the pixels of the template. An alternative form of Equation 5.14 is given by *normalizing* the cross-correlation. This does not change the position of the optimum and gives an

interpretation as the normalization of the cross-correlation vector. That is, the cross-correlation is divided by its modulus. Thus,

$$\max\ e = \frac{\sum_{(x,y)\in \mathbf{W}} \left(\mathbf{I}_{x+i,y+j} - \bar{\mathbf{I}}_{i,j}\right)\left(\mathbf{T}_{x,y} - \overline{\mathbf{T}}\right)}{\sqrt{\sum_{(x,y)\in \mathbf{W}} \left(\mathbf{I}_{x+i,y+j} - \bar{\mathbf{I}}_{i,j}\right)^2 \left(\mathbf{T}_{x,y} - \overline{\mathbf{T}}\right)^2}} \tag{5.15}$$

However, this equation has a similar computational complexity to the original formulation in Equation 5.7.

A particular implementation of template matching is when the image and the template are binary. In this case, the binary image can represent regions in the image or it can contain the edges. These two cases are illustrated in the example in Figure 5.3. The advantage of using binary images is that the amount of *computation* can be *reduced*. That is, each term in Equation 5.7 will take only two values: it will be one when $\mathbf{I}_{x+i,y+j} = \mathbf{T}_{x,y}$, and zero otherwise. Thus, Equation 5.7 can be implemented as

$$\max\ e = \sum_{(x,y)\in \mathbf{W}} \overline{\mathbf{I}_{x+i,y+j} \oplus \mathbf{T}_{x,y}} \tag{5.16}$$

where the symbol $\overline{\oplus}$ denotes the exclusive NOR operator. This equation can be easily implemented and requires significantly less resource than the original matching function.

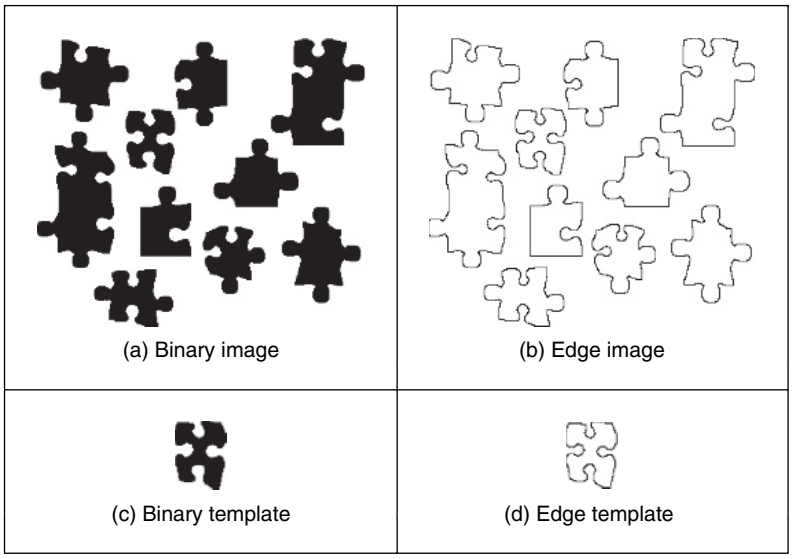


Figure 5.3 Example of binary and edge template matching

Template matching develops an *accumulator space* that stores the match of the template to the image at different locations; this corresponds to an implementation of Equation 5.7. It is called an accumulator, since the match is *accumulated* during application. Essentially, the accumulator is a two-dimensional (2D) array that holds the difference between the template and the image at different positions. The position in the image gives the same position of match in the

accumulator. Alternatively, Equation 5.11 suggests that the peaks in the accumulator resulting from template correlation give the location of the template in an image: the coordinates of the point of best match. Accordingly, template correlation and template matching can be viewed as similar processes. The location of a template can be determined by either process. The binary implementation of template matching (Equation 5.16) is usually concerned with thresholded edge data. This equation will be reconsidered in the definition of the Hough transform, the topic of the following section.

The Matlab code to implement template matching is the function `TMatching` given in Code 5.1. This function first clears an accumulator array, `accum`, then searches the whole picture, using pointers `i` and `j`, and then searches the whole template for matches, using pointers `x` and `y`. Notice that the position of the template is given by its centre. The accumulator elements are incremented according to Equation 5.7. The accumulator array is delivered as the result. The match for each position is stored in the array. After computing all the matches, the minimum element in the array defines the position where most pixels in the template matched those in the image. As such, the minimum is deemed to be the coordinates of the point where the template's shape is most likely to lie within the original image. It is possible to implement a version of template matching without the accumulator array, by storing the location of the minimum alone. This will give the same result and it requires little storage. However, this implementation will provide a result that cannot support later image interpretation that may require knowledge of more than just the best match.

```
%Template Matching Implementation

function accum=TMatching(inputimage,template)

%Image size & template size
  [rows,columns]=size(inputimage);
  [rowsT,columnsT]=size(template);

  %Centre of the template
cx=floor(columnsT/2)+1;   cy=floor(rowsT/2)+1;

%Accumulator
accum=zeros(rows,columns);
%Template Position
for i=cx:columns-cx
  for j=cy:rows-cy
     %Template elements
       for x=1-cx:cx-1
         for y=1-cy:cy-1
             err=(double(inputimage(j+y,i+x))
             -double(template(y+cy,x+cx)))^2;
         accum(j,i)=accum(j,i)+err;
       end
     end
  end
end
```

Code 5.1 Implementation of template matching

The results of applying the template matching procedure are illustrated in Figure 5.4. This example shows the accumulator arrays for matching the images shown in Figures 5.2(a), 5.3(a) and 5.3(b) with their respective templates. The dark points in each image are at the coordinates of the origin of the position where the template best matched the image (the minimum). Note that there is a border where the template has not been matched to the image data. At these border points, the template extended beyond the image data, so no matching has been performed. This is the same border as experienced with template convolution (Section 3.4.1). We can observe that a clearer minimum is obtained (Figure 5.4c) from the edge images of Figure 5.3. This is because for grey-level and binary images, there is some match when the template is not exactly in the best position. In the case of edges, the count of matching pixels is less.

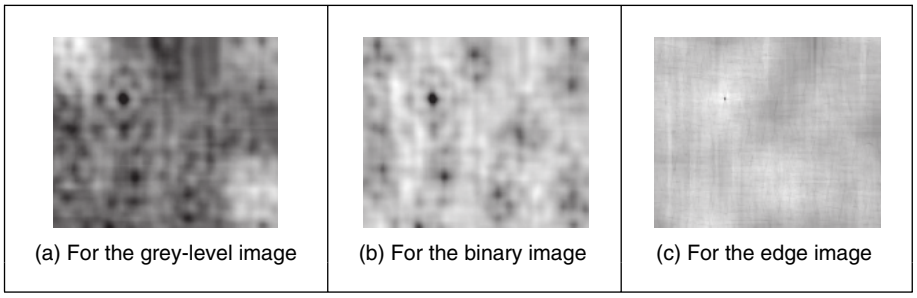


Figure 5.4 Accumulator arrays from template matching

Most applications require further degrees of freedom such as rotation (orientation), scale (size) or perspective deformations. Rotation can be handled by rotating the template or by using polar coordinates; scale invariance can be achieved using templates of differing size. Having more parameters of interest implies that the accumulator space becomes larger; its dimensions increase by one for each extra parameter of interest. *Position*-invariant template matching, as considered here, implies a 2D parameter space, whereas the extension to *scale*- and *position*-invariant template matching requires a 3D parameter space.

The computational cost of template matching is *large*. If the template is square and of size $m \times m$ and is matched to an image of size $N \times N$, since the m^2 pixels are matched at all image points (except for the border) the computational cost is $O(N^2m^2)$. This is the cost for position-invariant template matching. Any further parameters of interest *increase* the computational cost in proportion to the number of values of the extra parameters. This is clearly a large penalty and so a direct digital implementation of template matching is slow. Accordingly, this guarantees interest in techniques that can deliver the same result, but faster, such as using a Fourier implementation based on fast transform calculus.

The main *advantages* of template matching are its *insensitivity* to *noise* and *occlusion*. Noise can occur in any image, on any signal, just like on a telephone line. In digital photographs, the noise might appear low, but in computer vision it is made worse by edge detection by virtue of the differencing (differentiation) processes. Likewise, shapes can easily be occluded or *hidden*: a person can walk behind a lamppost, or illumination can cause occlusion. The *averaging* inherent in template matching reduces the susceptibility to noise; the *maximization* process reduces susceptibility to occlusion.

These advantages are illustrated in Figure 5.5, which shows detection in the presence of increasing noise. Here, we will use template matching to locate the vertical rectangle near

the top of the image (so we are matching a binary template of a black template on a white background to the binary image). The lowest noise level is in Figure 5.5(a) and the highest is in Figure 5.5(c); the position of the origin of the detected rectangle is shown as a black cross in a white square. The position of the origin is detected correctly in Figure 5.5(a) and (b) but incorrectly in the noisiest image, Figure 5.5(c). Clearly, template matching can handle quite high noise corruption. (Admittedly this is somewhat artificial: the noise would usually be filtered out by one of the techniques described in Chapter 3, but we are illustrating basic properties here.) The ability to handle noise is shown by correct determination of the position of the target shape, until the noise becomes too much and there are more points due to noise than there are due to the shape itself. When this occurs, the votes resulting from the noise exceed those occurring from the shape, and so the maximum is not found where the shape exists.

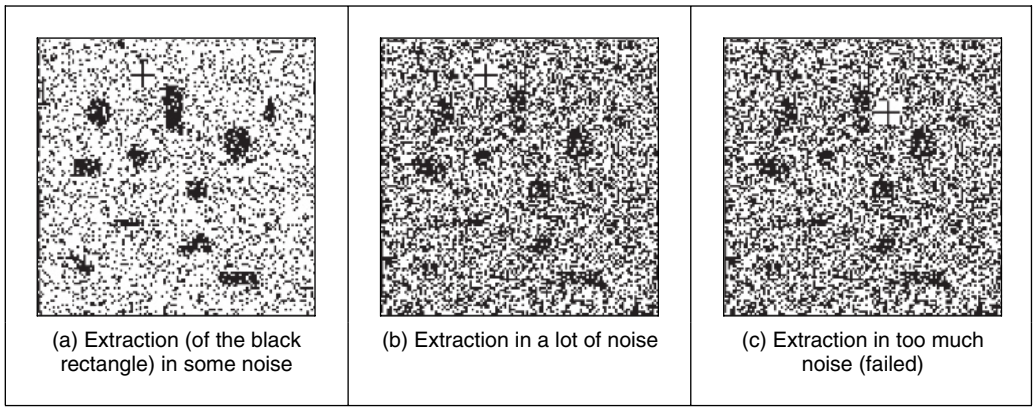


Figure 5.5 Template matching in noisy images

Occlusion is shown by placing a grey bar across the image; in Figure 5.6(a) the bar does not occlude the target rectangle, whereas in Figure 5.6(c) the rectangle is completely obscured. As with performance in the presence of noise, detection of the shape fails when the votes occurring from the shape exceed those from the rest of the image, and the cross indicating the

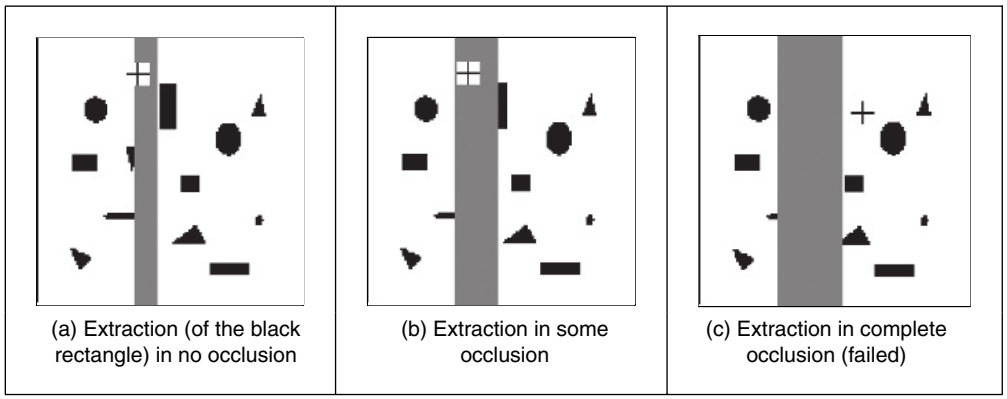


Figure 5.6 Template matching in occluded images

position of the origin of the rectangle is drawn in completely the wrong place. This is what happens when the rectangle is completely obscured in Figure 5.6(c).

So it can operate well, with practical advantage. We can include edge detection to concentrate on a shape's borders. Its main problem is *speed*: a direct implementation is slow, especially when handling shapes that are rotated or scaled (and there are other implementation difficulties too). Recalling that from Section 3.4.2 that template matching can be speeded up by using the Fourier transform, let us see whether that can be used here too.

5.3.2 Fourier transform implementation

We can implement template matching via the Fourier transform by using the *duality* between convolution and multiplication. This duality establishes that a multiplication in the space domain corresponds to a convolution in the frequency domain and vice versa. This can be exploited for faster computation by using the frequency domain, given the fast Fourier transform (FFT) algorithm. Thus, to find a shape we can compute the cross-correlation as a multiplication in the frequency domain. However, the matching process in Equation 5.11 is *correlation* (Section 2.3), *not* convolution. Thus, we need to express the correlation in terms of a convolution. This can be done as follows. First, we can rewrite the *correlation* in Equation 5.11 as

$$\mathbf{I} \otimes \mathbf{T} = \sum_{(x,y)\in W} \mathbf{I}_{x',y'} \mathbf{T}_{x'-i,y'-j} \tag{5.17}$$

where $x' = x + i$ and $y' = y + j$. *Convolution* is defined as

$$\mathbf{I} * \mathbf{T} = \sum_{(x,y)\in W} \mathbf{I}_{x',y'} \mathbf{T}_{i-x',j-y'} \tag{5.18}$$

Thus, to implement template matching in the frequency domain, we need to express Equation 5.17 in terms of Equation 5.18. This can be achieved by considering that

$$\mathbf{I} \otimes \mathbf{T} = \mathbf{I} * \mathbf{T}' = \sum_{(x,y)\in W} \mathbf{I}_{x',y'} \mathbf{T}'_{i-x',j-y'} \tag{5.19}$$

where

$$\mathbf{T}' = \mathbf{T}_{-x,-y} \tag{5.20}$$

That is, correlation is equivalent to convolution when the template is changed according to Equation 5.20. This equation reverses the coordinate axes and it corresponds to a horizontal and a vertical flip.

In the frequency domain, convolution corresponds to *multiplication*. As such, Equation 5.19 can be implemented by

$$\mathbf{I} * \mathbf{T}' = \Im^{-1}\left(\Im(\mathbf{I})\Im(\mathbf{T}')\right) \tag{5.21}$$

where $\Im$ denotes Fourier transformation as in Chapter 2 (and calculated by the FFT) and $\Im^{-1}$ denotes the inverse FFT. This is computationally faster than its direct implementation, given the speed advantage of the FFT. There are two ways of implementing this equation. In the first approach, we can compute $\mathbf{T}'$ by flipping the template and then computing its Fourier transform, $\Im(\mathbf{T}')$. In the second approach, we compute the transform of $\Im(\mathbf{T})$ and then we compute the complex *conjugate*. That is,

$$\Im(\mathbf{T}') = [\Im(\mathbf{T})]^* \tag{5.22}$$

where $[\ ]^*$ denotes the complex conjugate of the transform data (yes, it is an unfortunate symbol clash with convolution, but both are standard symbols). So conjugation of the transform of the template implies that the product of the two transforms leads to correlation. That is,

$$\mathbf{I} * \mathbf{T}' = \Im^{-1}\left(\Im(\mathbf{I})\left[\Im(\mathbf{T})\right]^*\right) \tag{5.23}$$

For both implementations, Equations 5.21 and 5.23 will evaluate the match, and more quickly for large templates than by direct implementation of template matching. Note that one assumption is that the transforms are of the same size, even though the template's shape is usually much smaller than the image. There is a selection of approaches; a simple solution is to include extra zero values (*zero-padding*) to make the image of the template the same size as the image.

The code to implement template matching by Fourier, `FTConv`, is given in Code 5.2. The implementation takes the image and the flipped template. The template is zero-padded and then transforms are evaluated. The required convolution is obtained by multiplying the transforms and then applying the inverse. The resulting image is the magnitude of the inverse transform. This could be invoked as a single function, rather than as procedure, but the implementation is less clear. This process can be formulated using brightness or edge data, as appropriate. Should we seek *scale* invariance, to find the position of a template irrespective of its size, then we need to formulate a set of templates that range in size between the maximum expected variation. Each of the templates of differing size is then matched by frequency domain multiplication. The maximum frequency domain value, for all sizes of template, indicates the position of the template and gives a value for its size. This can be a rather lengthy procedure when the template ranges considerably in size.

```
%Fourier Transform Convolution

function FTConv(inputimage,template)

%image size
[rows,columns]=size(inputimage);

%FT
Fimage=fft2(inputimage,rows,columns);
Ftemplate=fft2(template,rows,columns);

%Convolution
G=Fimage.*Ftemplate;

%Modulus
Z=log(abs(fftshift(G)));

%Inverse
R=real(ifft2(G));
```

Code 5.2 Implementation of convolution by the frequency domain

Figure 5.7 illustrates the results of template matching in the Fourier domain. This example uses the image and template shown in Figure 5.2. Figure 5.7(a) shows the flipped and padded template. The Fourier transforms of the image and of the flipped template are given in Figure 5.7(b)

and (c), respectively. These transforms are multiplied, point by point, to achieve the image in Figure 5.7(d). When this is inverse Fourier transformed, the result (Figure 5.7e) shows where the template best matched the image (the coordinates of the template's top left-hand corner). The resulting image contains several local maxima (in white). This can be explained by the fact that this implementation does not consider the term in Equation 5.10. In addition, the shape can partially match several patterns in the image. Figure 5.7(f) shows a zoom of the region where the peak is located. We can see that this peak is well defined. In contrast to template matching, the implementation in the frequency domain does not have any border. This is due to the fact that Fourier theory assumes picture replication to infinity. Note that in application, the Fourier transforms do not need to be rearranged (`fftshif`) so that the d.c. is at the centre, since this has been done here for display purposes only.

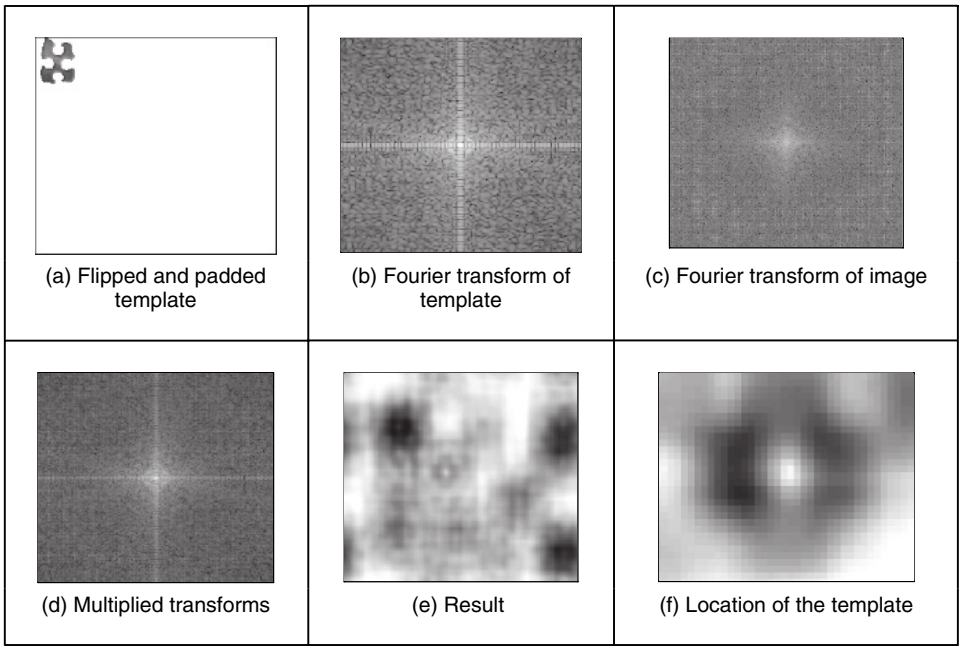


Figure 5.7 Template matching by Fourier transformation

There are several further difficulties in using the transform domain for template matching in discrete images. If we seek rotation invariance, then an image can be expressed in terms of its polar coordinates. Discretization gives further difficulty since the points in a rotated discrete shape can map imperfectly to the original shape. This problem is better manifest when an image is scaled in size to become larger. In such a case, the spacing between points will increase in the enlarged image. The difficulty is how to allocate values for pixels in the enlarged image which are not defined in the enlargement process. There are several interpolation approaches, but it can often appear prudent to reformulate the original approach. Further difficulties can include the influence of the image borders: Fourier theory assumes that an image replicates spatially to infinity. Such difficulty can be reduced by using window operators, such as the Hamming or the Hanning windows. These difficulties do not arise for optical Fourier transforms and so using the Fourier transform for position-invariant template matching is often confined to optical implementations.

5.3.3 Discussion of template matching

The advantages associated with template matching are mainly theoretical since it can be very difficult to develop a template matching technique that operates satisfactorily. The results presented here have been for *position* invariance only. This can cause difficulty if invariance to *rotation* and *scale* is also required. This is because the template is stored as a discrete set of points. When these are rotated, *gaps* can appear owing to the discrete nature of the coordinate system. If the template is increased in size then there will be missing points in the scaled-up version. Again, there is a frequency domain version that can handle variation in size, since scale-invariant template matching can be achieved using the *Mellin transform* (Bracewell, 1986). This avoids using many templates to accommodate the variation in size by evaluating the scale-invariant match in a single pass. The Mellin transform essentially scales the spatial coordinates of the image using an exponential function. A point is then moved to a position given by a logarithmic function of its original coordinates. The transform of the scaled image is then multiplied by the transform of the template. The maximum again indicates the best match between the transform and the image. This can be considered to be equivalent to a change of variable. The logarithmic mapping ensures that scaling (multiplication) becomes addition. By the logarithmic mapping, the problem of scale invariance becomes a problem of finding the position of a match.

The Mellin transform only provides scale-invariant matching. For scale and position invariance, the Mellin transform is combined with the Fourier transform, to give the *Fourier–Mellin* transform. The Fourier–Mellin transform has many disadvantages in a digital implementation, owing to the problems in spatial resolution, although there are approaches to reduce these problems (Altmann and Reitbock, 1984), as well as the difficulties with discrete images experienced in Fourier transform approaches.

Again, the Mellin transform appears to be much better suited to an *optical* implementation (Casasent and Psaltis, 1977), where *continuous* functions are available, rather than to discrete image analysis. A further difficulty with the Mellin transform is that its result is independent of the *form factor* of the template. Accordingly, a rectangle and a square appear to be the same to this transform. This implies a loss of information since the form factor can indicate that an object has been imaged from an oblique angle. There is resurgent interest in *log-polar* mappings for image analysis (e.g. Traver and Pla, 2003; Zokai and Wollberg, 2005).

So, there are innate difficulties with template matching, whether it is implemented directly or by transform operations. For these reasons, and because many shape extraction techniques require more than just edge or brightness data, direct digital implementations of feature extraction are usually preferred. This is perhaps also influenced by the speed advantage that one popular technique can confer over template matching. This is the Hough transform, which is covered next.

5.4 Hough transform

5.4.1 Overview

The *Hough transform* (HT) (Hough, 1962) is a technique that locates shapes in images. In particular, it has been used to extract *lines*, *circles* and *ellipses* (or conic sections). In the case of lines, its mathematical definition is equivalent to the Radon transform (Deans, 1981). The HT was introduced by Hough (1962) and then used to find bubble tracks rather than shapes in images. However, Rosenfeld (1969) noted its potential advantages as an image processing

algorithm. The HT was thus implemented to find lines in images (Duda and Hart, 1972) and it has been extended greatly, since it has many advantages and many potential routes for improvement. Its prime advantage is that it can deliver the *same* result as that for template matching, but *faster* (Stockman and Agrawala, 1977; Sklansky, 1978; Princen et al., 1992b). This is achieved by a reformulation of the template matching process, based on an *evidence-gathering* approach, where the evidence is the *votes* cast in an accumulator array. The HT implementation defines a *mapping* from the image points into an accumulator space (Hough space). The mapping is achieved in a computationally efficient manner, based on the function that describes the target shape. This mapping requires much fewer computational resources than template matching. However, it still requires significant storage and has high computational requirements. These problems are addressed later, since they give focus for the continuing development of the HT. However, the fact that the HT is equivalent to template matching has given sufficient impetus for the technique to be among the most popular of all existing shape extraction techniques.

5.4.2 Lines

We will first consider finding lines in an image. In a Cartesian parameterization, collinear points in an image with coordinates (x, y) are related by their slope m and an intercept c according to:

$$y = mx + c \tag{5.24}$$

This equation can be written in homogeneous form as

$$Ay + Bx + 1 = 0 \tag{5.25}$$

where $A = -1/c$ and $B = m/c$. Thus, a line is defined by giving a pair of values (A, B). However, we can observe a symmetry in the definition in Equation 5.25. This equation is symmetric since a pair of coordinates (x, y) also defines a line in the space with parameters (A, B). That is, Equation 5.25 can be seen as the equation of a line for fixed coordinates (x, y) or as the equation of a line for fixed parameters (A, B). Thus, pairs can be used to define points and lines simultaneously (Aguado et al., 2000a). The HT gathers evidence of the point (A, B) by considering that all the points (x, y) define the same line in the space (A, B). That is, if the set of collinear points $\{(x_i, y_i)\}$ defines the line (A, B), then

$$Ay_i + Bx_i + 1 = 0 \tag{5.26}$$

This equation can be seen as a system of equations and it can simply be rewritten in terms of the Cartesian parameterization as

$$c = -x_i m + y_i \tag{5.27}$$

Thus, to determine the line we must find the values of the parameters (m, c) [or (A, B) in homogeneous form] that satisfy Equation 5.27 (or 5.26, respectively). However, we must notice that the system is generally overdetermined. That is, we have more equations than unknowns. Thus, we must find the solution that comes close to satisfying all the equations simultaneously. This kind of problem can be solved, for example, using linear least squares techniques. The HT uses an evidence-gathering approach to provide the solution.

The relationship between a point (x_i, y_i) in an image and the line given in Equation 5.27 is illustrated in Figure 5.8. The points (x_i, y_i) and (x_j, y_j) in Figure 5.8(a) define the lines U_i and U_j in Figure 5.8(b), respectively. All the collinear elements in an image will define dual lines with the same concurrent point (A, B). This is independent of the line parameterization used. The HT solves it in an efficient way by simply counting the potential solutions in an accumulator

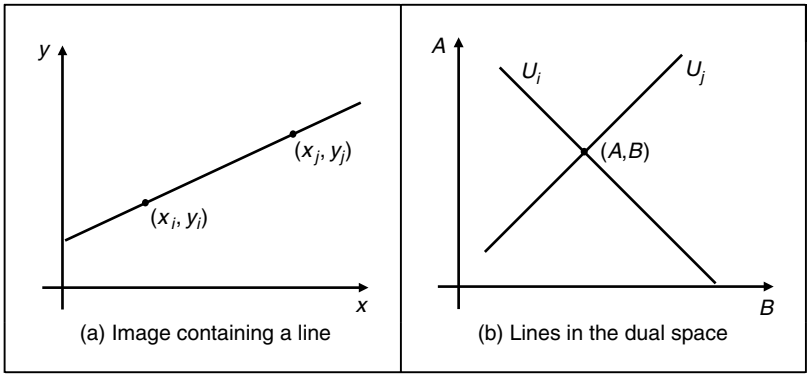


Figure 5.8 Illustrating the Hough transform for lines

array that stores the evidence, or votes. The count is made by tracing all the dual lines for each point (x_i, y_i). Each point in the trace increments an element in the array; thus, the problem of line extraction is transformed into the problem of locating a maximum in the accumulator space. This strategy is robust and has been demonstrated to be able to handle noise and occlusion.

The axes in the dual space represent the parameters of the line. In the case of the Cartesian parameterization m can take an *infinite* range of values, since lines can vary from horizontal to vertical. Since votes are gathered in a discrete array, this will produce *bias* errors. It is possible to consider a range of votes in the accumulator space that cover all possible values. This corresponds to techniques of antialiasing and can improve the gathering strategy (Brown, 1983; Kiryati and Bruckstein, 1991).

The implementation of the HT for lines, `HTLine`, is given in Code 5.3. It is important to observe that Equation 5.27 is not suitable for implementation since the parameters can take an infinite range of values. To handle the infinite range for c, we use two arrays in the implementation in Code 5.3. When the slope m is between $-45°$ and $45°$, c does not take a large value. For other values of m the intercept c can take a very large value. Thus, we consider an accumulator for each case. In the second case, we use an array that stores the intercept with the x-axis. This only solves the problem partially, since we cannot guarantee that the value of c will be small when the slope m is between $-45°$ and $45°$.

Figure 5.9 shows three examples of locating lines using the HT implemented in Code 5.3. In Figure 5.9(a) there is a single line which generates the peak seen in Figure 5.9(d). The magnitude of the peak is proportional to the number of pixels in the line from which it was generated. The edges of the wrench in Figure 5.9(b) and (c) define two main lines. Image 5.9(c) contains much more noise. This image was obtained by using a lower threshold value in the edge detector operator which gave rise to more noise. The accumulator results of the HT for the images in Figure 5.9(b) and (c) are shown in Figure 5.9(e) and (f), respectively. The two accumulator arrays are broadly similar in shape, and the peak in each is at the same place. The coordinates of the peaks are at combinations of parameters of the lines that best fit the image. The extra number of edge points in the noisy image of the wrench gives rise to more votes in the accumulator space, as can be seen by the increased number of votes in Figure 5.9(f) compared with Figure 5.9(e). Since the peak is in the same place, this shows that the HT can indeed tolerate noise. The results of extraction, when superimposed on the edge image, are shown in Figure 5.9(g)–(i). Only the two lines corresponding to significant peaks have been drawn for the image of the wrench. Here, we can see that the parameters describing the lines

```
%Hough Transform for Lines
function HTLine(inputimage)
%image size
[rows,columns]=size(inputimage);
%accumulator
acc1=zeros(rows,91);
acc2=zeros(columns,91);
%image
for x=1:columns
  for y=1:rows
     if(inputimage(y,x)==0)
      for m=-45:45
        b=round(y-tan((m*pi)/180)*x);
        if(b<rows & b>0)
           acc1(b,m+45+1)=acc1(b,m+45+1)+1;
         end
       end
       for m=45:135
         b=round(x-y/tan((m*pi)/180));
         if(b<columns & b>0)
           acc2(b,m-45+1)=acc2(b,m-45+1)+1;
         end
       end
    end
  end
end
```

Code 5.3 Implementation of the Hough transform for lines

have been extracted well. Note that the endpoints of the lines are not delivered by the HT, only the parameters that describe them. You have to go back to the image to obtain line length.

The HT delivers a correct response; that is, correct estimates of the parameters used to specify the line, so long as the number of collinear points along that line exceeds the number of collinear points on any other line in the image. As such, the HT has the same properties in respect of noise and occlusion as template matching. However, the non-linearity of the parameters and the discretization produce noisy accumulators. A major problem in implementing the basic HT for lines is the definition of an appropriate accumulator space. In application, Bresenham's line drawing algorithm (Bresenham, 1965) can be used to draw the lines of votes in the accumulator space. This ensures that lines of connected votes are drawn, as opposed to the use of Equation 5.27, which can lead to gaps in the drawn line. *Backmapping* (Gerig and Klein, 1986) can be used to determine exactly which edge points contributed to a particular peak. Backmapping is an *inverse* mapping from the accumulator space to the edge data and can allow for shape analysis of the image by removal of the edge points that contributed to particular peaks, and then by reaccumulation using the HT. Note that the computational cost of the HT depends on the number of edge points (n_e) and the length of the lines formed in the parameter space (l), giving a computational cost of $O(n_e l)$. This is considerably less than that for template matching, given earlier as $O(N^2 m^2)$.

One way to avoid the problems of the Cartesian parameterization in the HT is to base the mapping function on an alternative parameterization. One of the most proven techniques is

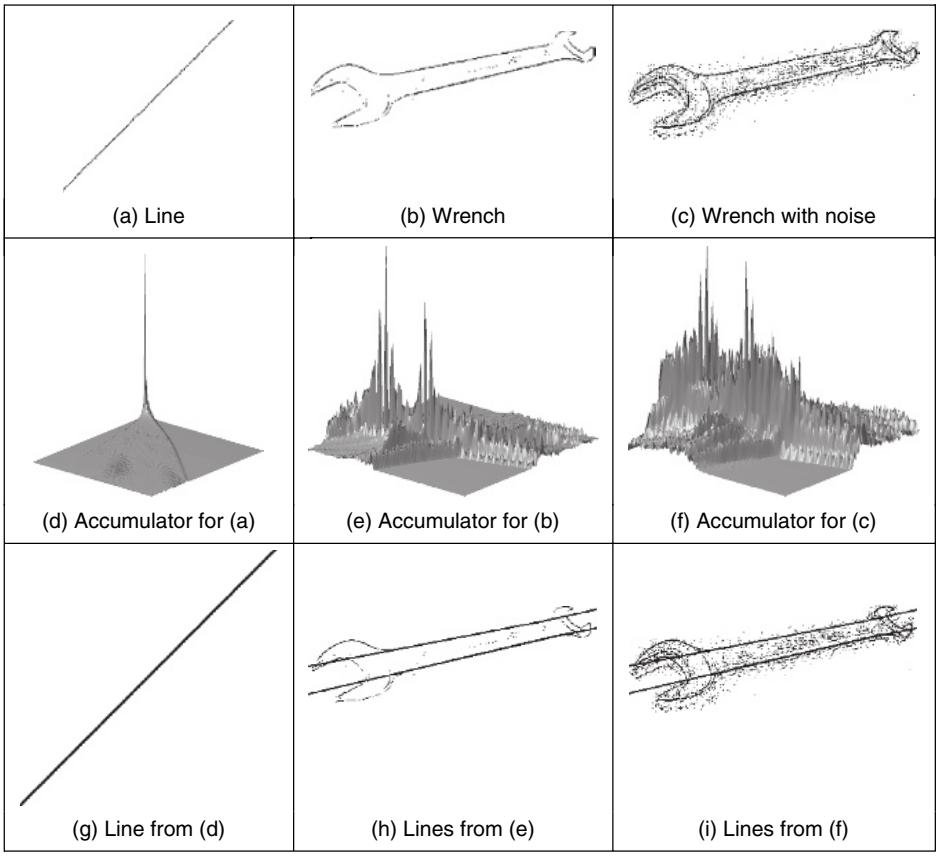


Figure 5.9 Applying the Hough transform for lines

called the *foot-of-normal* parameterization. This parameterizes a line by considering a point (x, y) as a function of an angle normal to the line, passing through the origin of the image. This gives a form of the HT for lines known as the *polar HT for lines* (Duda and Hart, 1972). The point where this line intersects the line in the image is given by

$$\rho = x\cos(\theta) + y\sin(\theta) \tag{5.28}$$

where θ is the angle of the line normal to the line in an image and ρ is the length between the origin and the point where the lines intersect, as illustrated in Figure 5.10.

By recalling that two lines are perpendicular if the product of their slopes is -1, and by considering the geometry of the arrangement in Figure 5.10, we obtain

$$c = \frac{\rho}{\sin(\theta)} \quad m = -\frac{1}{\tan(\theta)} \tag{5.29}$$

By substitution in Equation 5.24 we obtain the polar form (Equation 5.28). This provides a different mapping function: votes are now cast in a sinusoidal manner, in a 2D accumulator array in terms of θ and ρ, the parameters of interest. The advantage of this alternative mapping is that the values of the parameters θ and ρ are now bounded to lie within a specific range. The range for θ is within 180°; the possible values of ρ are given by the image size, since

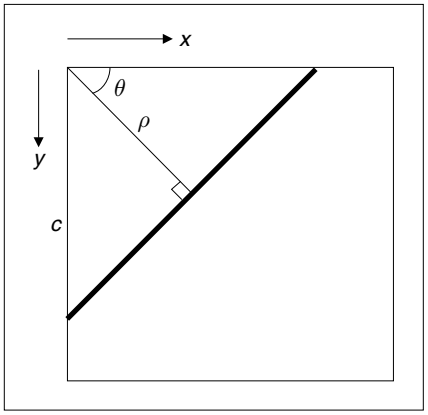


Figure 5.10 Polar consideration of a line

the maximum length of the line is $\sqrt{2} \times N$, where N is the (square) image size. The range of possible values is now fixed, so the technique is practicable.

As the voting function has now changed, we shall draw different loci in the accumulator space. In the conventional HT for lines, a straight line mapped to a straight line as in Figure 5.8. In the polar HT for lines, points map to curves in the accumulator space. This is illustrated in Figure 5.11(a)–(c), which shows the polar HT accumulator spaces for one, two and three points, respectively. For a single point in the upper row of Figure 5.11(a) we obtain a single curve shown in the lower row of Figure 5.11(a). For two points we obtain two curves, which intersect at a position which describes the parameters of the line joining them (Figure 5.11b). An additional curve is obtained for the third point and there is now a peak in the accumulator array containing three votes (Figure 5.11c).

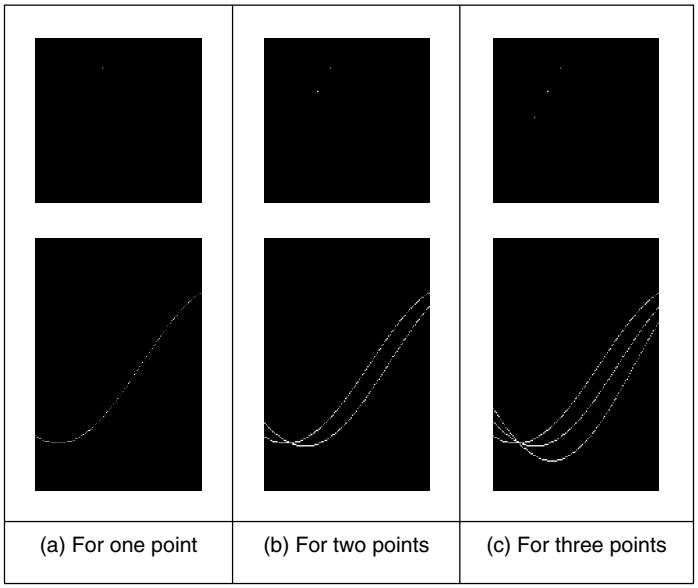
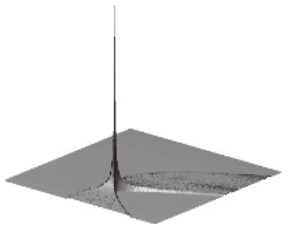


Figure 5.11 Images and the accumulator space of the polar Hough transform

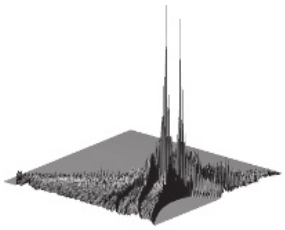
The implementation of the *polar HT for lines* is the function `HTPLine` in Code 5.4. The accumulator array is a set of 180 bins for value of θ in the range 0–180°, and for values of ρ in the range 0 to $\sqrt{N^2 + M^2}$, where $N \times M$ is the picture size. Then, for image (edge) points greater than a chosen threshold, the angle relating to the bin size is evaluated (as radians in the range 0 to π) and then the value of ρ is evaluated from Equation 5.28 and the appropriate accumulator cell is incremented so long as the parameters are within range. The accumulator arrays obtained by applying this implementation to the images in Figure 5.9 are shown in Figure 5.12. Figure 5.12(a) shows that a single line defines a well-delineated peak. Figure 5.12(b)

```
%Polar Hough Transform for Lines

function HTPLine(inputimage)

%image size
[rows,columns]=size(inputimage);

%accumulator
rmax=round(sqrt(rows^2+columns^2));
acc=zeros(rmax,180);

%image
for x=1:columns
  for y=1:rows
     if(inputimage(y,x)==0)
        for m=1:180
           r=round(x*cos((m*pi)/180)
             +y*sin((m*pi)/180));
           if(r<rmax & r>0)
              acc(r,m)= acc(r,m)+1; end
        end
     end
  end
end
```

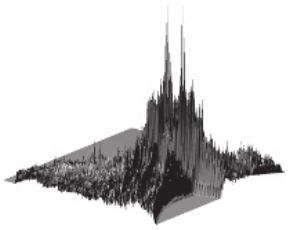
Code 5.4 Implementation of the polar Hough transform for lines



(a) Accumulator for Figure 5.9(a)



(b) Accumulator for Figure 5.9(b)



(c) Accumulator for Figure 5.9(c)

Figure 5.12 Applying the polar Hough transform for lines

and (c) show a clearer peak compared with the implementation of the Cartesian parameterization. This is because discretization effects are reduced in the polar parameterization. This feature makes the polar implementation far more practicable than the earlier, Cartesian, version.

5.4.3 Hough transform for circles

The HT can be extended by replacing the equation of the curve in the detection process. The equation of the curve can be given in *explicit* or *parametric* form. In explicit form, the HT can be defined by considering the equation for a circle given by

$$(x - x_0)^2 + (y - y_0)^2 = r^2 \tag{5.30}$$

This equation defines a locus of points (x, y) centred on an origin (x_0, y_0) and with radius r. This equation can again be visualized in two ways: as a locus of points (x, y) in an image, or as a locus of points (x_0, y_0) centred on (x, y) with radius r.

Figure 5.13 illustrates this dual definition. Each edge point in Figure 5.13(a) defines a set of circles in the accumulator space. These circles are defined by all possible values of the radius and they are centred on the coordinates of the edge point. Figure 5.13(b) shows three circles

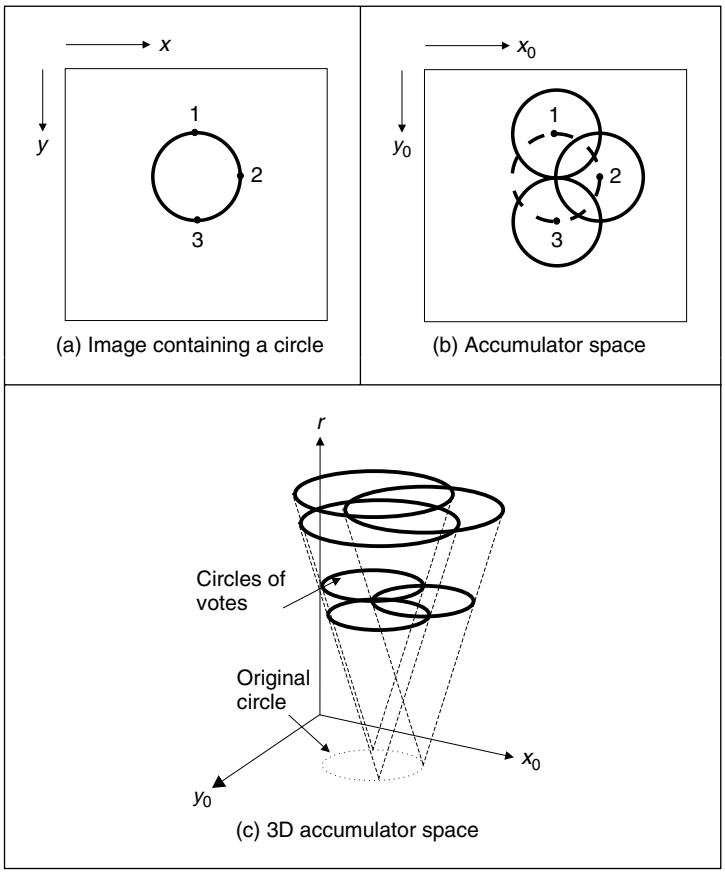


Figure 5.13 Illustrating the Hough transform for circles

defined by three edge points. These circles are defined for a given radius value. Each edge point defines circles for the other values of the radius. This implies that the accumulator space is three dimensional (for the three parameters of interest) and that edge points map to a *cone* of votes in the accumulator space. Figure 5.13(c) illustrates this accumulator. After gathering evidence of all the edge points, the maximum in the accumulator space again corresponds to the parameters of the circle in the original image. The procedure of evidence gathering is the same as that for the HT for lines, but votes are generated in cones, according to Equation 5.30.

Equation 5.30 can be defined in *parametric* form as

$$x = x_0 + r\cos(\theta) \quad y = y_0 + r\sin(\theta) \tag{5.31}$$

The advantage of this representation is that it allows us to solve for the parameters. Thus, the HT mapping is defined by

$$x_0 = x - r\cos(\theta) \quad y_0 = y - r\sin(\theta) \tag{5.32}$$

These equations define the points in the accumulator space (Figure 5.13b) dependent on the radius r. Note that θ is not a free parameter, but defines the trace of the curve. The trace of the curve (or surface) is commonly referred to as the *point spread function*.

The implementation of the HT for circles, `HTCircle`, is shown in Code 5.5. This is similar to the HT for lines, except that the voting function corresponds to that in Equation 5.32 and the accumulator space is for circle data. The accumulator in the implementation is two dimensions,

```
%Hough Transform for Circles

function HTCircle(inputimage,r)

%image size
[rows,columns]=size(inputimage);

%accumulator
acc=zeros(rows,columns);

%image
for x=1:columns
  for y=1:rows
    if(inputimage(y,x)==0)
      for ang=0:360
        t=(ang*pi)/180;
        x0=round(x-r*cos(t));
        y0=round(y-r*sin(t));
        if(x0<columns & x0>0 & y0<rows & y0>0)
          acc(y0,x0)=acc(y0,x0)+1;
        end
      end
    end
  end
end
```

Code 5.5 Implementation of the Hough transform for circles

in terms of the centre parameters for a fixed value of the radius given as an argument to the function. This function should be called for all potential radii. A circle of votes is generated by varying t (i.e. θ, but Matlab does not allow Greek symbols!) from 0° to 360°. The discretization of t controls the granularity of voting; too small an increment gives very fine coverage of the parameter space, too large a value results in very sparse coverage. The accumulator space, `acc` (initially zero), is incremented only for points whose coordinates lie within the specified range (in this case the centre cannot lie outside the original image).

The application of the HT for circles is illustrated in Figure 5.14. Figure 5.14(a) shows an image with a synthetic circle. In this figure, the edges are complete and well defined. The result of the HT process is shown in Figure 5.14(d). The peak of the accumulator space is at the centre of the circle. Note that votes exist away from the circle's centre, and rise towards

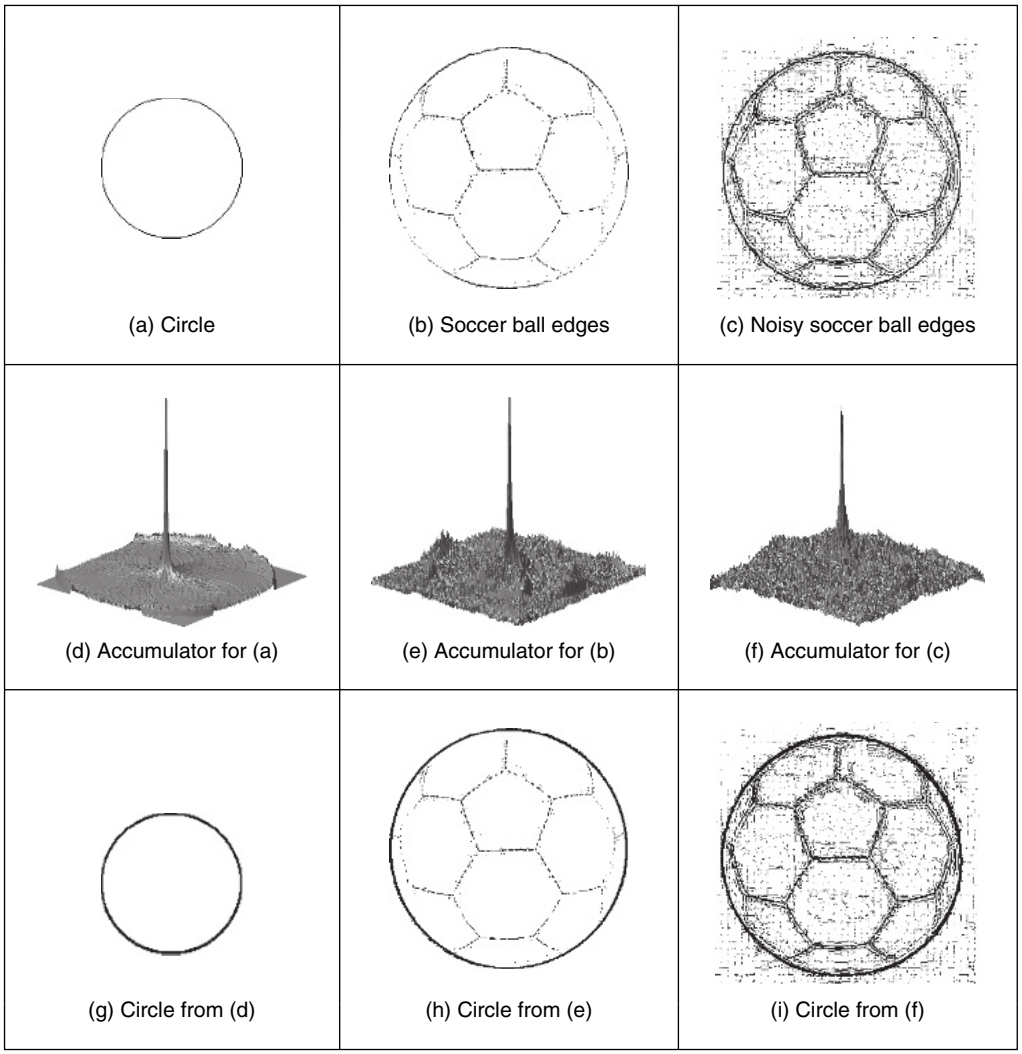
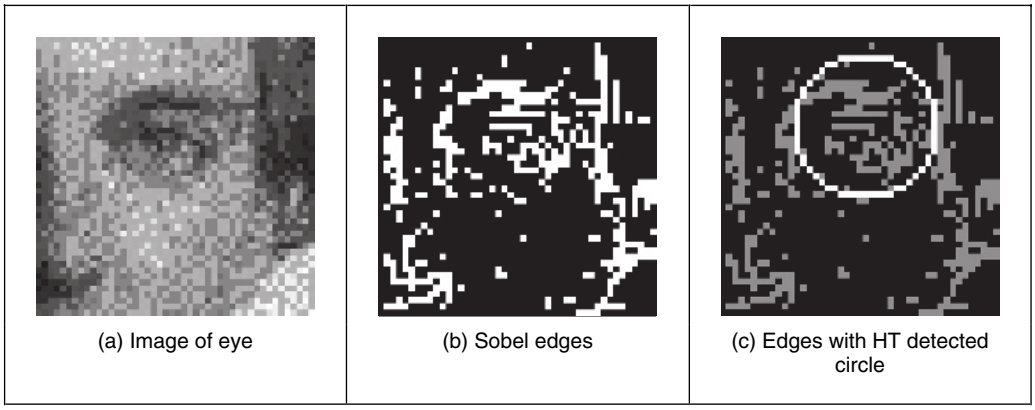


Figure 5.14 Applying the Hough transform for circles

the locus of the actual circle, although these background votes are much less than the actual peak. Figure 5.14(b) shows an example of data containing occlusion and noise. The image in Figure 5.14(c) corresponds to the same scene, but the noise level has been increased by changing the threshold value in the edge detection process. The accumulators for these two images are shown in Figure 5.14(e) and (f) and the circles related to the parameter space peaks are superimposed (in black) on the edge images in Figure 5.14(g)–(i). We can see that the HT has the ability to tolerate occlusion and noise. Note that we do not have the earlier problem with the start and the end of the lines since the circle is a *closed* shape. In Figure 5.14(c), there are many edge points, which implies that the amount of processing time increases. The HT will detect the circle (provide the right result) as long as more points are in a circular locus described by the parameters of the target circle than there are on any other circle. This is exactly the same performance as for the HT for lines, as expected, and is consistent with the result of template matching.

In application code, *Bresenham's algorithm* for discrete circles (Bresenham, 1977) can be used to draw the circle of votes, rather than use the polar implementation of Equation 5.32. This ensures that the complete locus of points is drawn and avoids the need to choose a value for increase in the angle used to trace the circle. Bresenham's algorithm can be used to generate the points in one octant, since the remaining points can be obtained by reflection. Backmapping can be used to determine which points contributed to the extracted circle.

An additional example of the circle HT extraction is shown in Figure 5.15. Figure 5.15(a) is a real image (albeit one with low resolution) which was processed by Sobel edge detection and thresholded to give the points in Figure 5.15(b). The circle detected by application of `HTCircle` with radius 5 pixels is shown in Figure 5.15(c) superimposed on the edge data. The extracted circle can be seen to *match* the edge data well. This highlights the two major advantages of the HT (and of template matching): its ability to handle *noise* and *occlusion*. Note that the HT merely finds the circle with the maximum number of points; it is possible to include other constraints to control the circle selection process, such as gradient direction for objects with known illumination profile. In the case of the human eye, the (circular) iris is usually darker than its white surroundings.



(a) Image of eye

(b) Sobel edges

(c) Edges with HT detected circle

Figure 5.15 Using the Hough transform for circles

Figure 5.15 also shows some of the difficulties with the HT, namely that it is essentially an implementation of template matching, and does not use some of the *richer* stock of information

available in an image. For example, we might know constraints on *size*; the largest size and iris would be in an image like Figure 5.15. We also know some of the *topology*: the eye region contains two ellipsoidal structures with a circle in the middle. We may also might know *brightness* information: the pupil is darker than the surrounding iris. These factors can be formulated as *constraints* on whether edge points can vote within the accumulator array. A simple modification is to make the votes proportional to edge magnitude; in this manner, points with high contrast will generate more votes and hence have more significance in the voting process. In this way, the feature extracted by the HT can be arranged to suit a particular application.

5.4.4 Hough transform for ellipses

Circles are very important in shape detection since many objects have a circular shape. However, because of the camera's viewpoint, circles do not always look like circles in images. Images are formed by mapping a shape in 3D space into a plane (the image plane). This mapping performs a perspective transformation. In this process, a circle is deformed to look like an ellipse. We can define the mapping between the circle and an ellipse by a similarity transformation. That is,

$$\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos(\rho) & \sin(\rho) \\ -\sin(\rho) & \cos(\rho) \end{bmatrix} \begin{bmatrix} S_x \\ S_y \end{bmatrix} \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} \tag{5.33}$$

where (x', y') define the coordinates of the circle in Equation 5.31, ρ represents the orientation, (S_x, S_y) a scale factor and (t_x, t_y) a translation. If we define

$$\begin{aligned} a_0 &= t_x \quad a_x = S_x \cos(\rho) \quad b_x = S_y \sin(\rho) \\ b_0 &= t_y \quad a_y = -S_x \sin(\rho) \quad b_y = S_y \cos(\rho) \end{aligned} \tag{5.34}$$

then we have that the circle is deformed into

$$\begin{aligned} x &= a_0 + a_x \cos(\theta) + b_x \sin(\theta) \\ y &= b_0 + a_y \cos(\theta) + b_y \sin(\theta) \end{aligned} \tag{5.35}$$

This equation corresponds to the polar representation of an ellipse. This polar form contains six parameters $\left(a_0, b_0, a_x, b_x, a_y, b_y\right)$ that characterize the shape of the ellipse. θ is not a free parameter and it only addresses a particular point in the locus of the ellipse (just as it was used to trace the circle in Equation 5.32). However, one parameter is redundant since it can be computed by considering the orthogonality (independence) of the axes of the ellipse (the product $a_x b_x + a_y b_y = 0$, which is one of the known properties of an ellipse). Thus, an ellipse is defined by its centre (a_0, b_0) and three of the axis parameters $\left(a_x, b_x, a_y, b_y\right)$. This gives five parameters, which is intuitively correct since an ellipse is defined by its centre (two parameters), it size along both axes (two more parameters) and its rotation (one parameter). In total, this states that five parameters describe an ellipse, so our three axis parameters must jointly describe size and rotation. In fact, the axis parameters can be related to the orientation and the length along the axes by

$$\tan(\rho) = \frac{a_y}{a_x} \quad a = \sqrt{a_x^2 + a_y^2} \quad b = \sqrt{b_x^2 + b_y^2} \tag{5.36}$$

where $(a,\ b)$ are the axes of the ellipse, as illustrated in Figure 5.16.

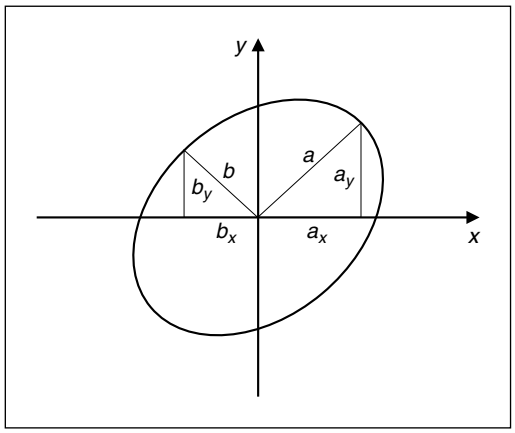


Figure 5.16 Definition of ellipse axes

In a similar way to Equation 5.31, Equation 5.35 can be used to generate the mapping function in the HT. In this case, the location of the centre of the ellipse is given by

$$\begin{aligned} a_0 &= x - a_x \cos(\theta) + b_x \sin(\theta) \\ b_0 &= y - a_y \cos(\theta) + b_y \sin(\theta) \end{aligned} \tag{5.37}$$

The location is dependent on three parameters, thus the mapping defines the trace of a hypersurface in a five-dimensional (5D) space. This space can be very large. For example, if there are 100 possible values for each of the five parameters, the 5D accumulator space contains 10^{10} values. This is 10 GB of storage, which is tiny nowadays (at least, when someone else pays!). Accordingly, there has been much interest in ellipse detection techniques which use much less space and operate much more quickly than direct implementation of Equation 5.37.

Code 5.6 shows the implementation of the HT mapping for ellipses. The function `HTEllipse` computes the centre parameters for an ellipse without rotation and with fixed axis length given as arguments. Thus, the implementation uses a 2D accumulator. In practice, to locate an ellipse, it is necessary to try all potential values of axis length. This is computationally impossible unless we limit the computation to a few values.

Figure 5.17 shows three examples of the application of the ellipse extraction process described in the Code 5.6. The first example (Figure 5.17a) illustrates the case of a perfect ellipse in a synthetic image. The array in Figure 5.17(d) shows a prominent peak whose position corresponds to the centre of the ellipse. The examples in Figure 5.17(b) and (c) illustrate the use of the HT to locate a circular form when the image has an oblique view. Each example was obtained by using a different threshold in the edge detection process. Figure 5.17(c) contains more noise data, which in turn gives rise to more noise in the accumulator. We can observe that there is more than one ellipse to be located in these two figures. This gives rise to the other high values in the accumulator space. As with the earlier examples for line and circle extraction, there is scope for interpreting the accumulator space, to discover which structures produced particular parameter combinations.

```
%Hough Transform for Ellipses

function HTEllipse(inputimage,a,b)

%image size
[rows,columns]=size(inputimage);

%accumulator
acc=zeros(rows,columns);

%image
for x=1:columns
  for y=1:rows
    if(inputimage(y,x)==0)
      for ang=0:360
        t=(ang*pi)/180;
        x0=round(x-a*cos(t));
        y0=round(y-b*sin(t));
        if(x0<columns & x0>0 & y0<rows & y0>0)
          acc(y0,x0)=acc(y0,x0)+1;
        end
      end
    end
  end
end
```

Code 5.6 Implementation of the Hough transform for ellipses

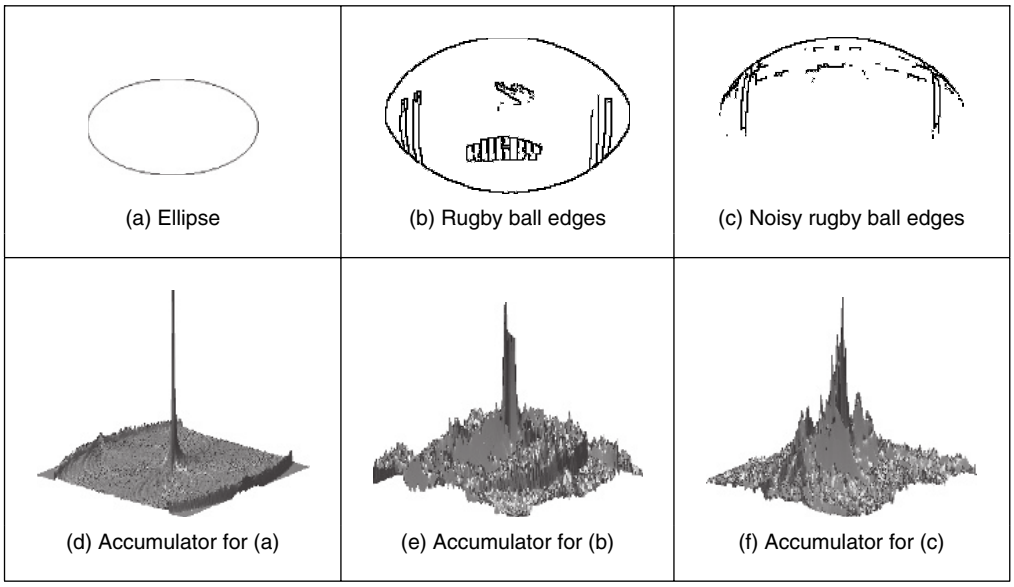


Figure 5.17 Applying the Hough transform for ellipses

5.4.5 Parameter space decomposition

The HT gives the same (optimal) result as template matching and even though it is faster, it still requires significant computational resources. The previous sections showed that as we increase the complexity of the curve under detection, the computational requirements increase in an exponential way. Thus, the HT becomes less practical. For this reason, most of the research in the HT has focused on the development of techniques aimed to reduce its computational complexity (Illingworth and Kittler, 1988; Leavers, 1993). One important way to reduce the computation has been the use of geometric properties of shapes to decompose the parameter space. Several techniques have used different geometric properties. These geometric properties are generally defined by the relationship between points and derivatives.

5.4.5.1 Parameter space reduction for lines

For a line, the accumulator space can be reduced from two dimensions to one dimension by considering that we can compute the slope from the information of the image. The slope can be computed either by using the *gradient direction* at a point or by considering a pair of points. That is,

$$m = \varphi \quad \text{or} \quad m = \frac{y_2 - y_1}{x_2 - x_1} \tag{5.38}$$

where φ is the gradient direction at the point. In the case of two points, by considering Equation 5.24 we have:

$$c = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1} \tag{5.39}$$

Thus, according to Equation 5.29, one of the parameters of the polar representation for lines, θ, is now given by

$$\theta = -\tan^{-1}\left[\frac{1}{\varphi}\right] \quad \text{or} \quad \theta = \tan^{-1}\left[\frac{x_1 - x_2}{y_2 - y_1}\right] \tag{5.40}$$

These equations do not depend on the other parameter ρ and they provide alternative mappings to gather evidence. That is, they decompose the parametric space, such that the two parameters θ and ρ are now *independent*. The use of edge direction information constitutes the base of the line extraction method presented by O'Gorman and Clowes (1976). The use of pairs of points can be related to the definition of the randomized Hough transform (Xu et al., 1990). The number of feature points considered corresponds to all the combinations of points that form pairs. By using statistical techniques, it is possible to reduce the space of points to consider a representative sample of the elements; that is, a subset that provides enough information to obtain the parameters with predefined and small estimation errors.

Code 5.7 shows the implementation of the parameter space decomposition for the HT for lines. The slope of the line is computed by considering a pair of points. Pairs of points are restricted to a neighbourhood of 5×5 pixels. The implementation of Equation 5.40 gives values between $-90°$ and $90°$. Since the accumulators can only store positive values, we add $90°$ to all values. To compute ρ we use Equation 5.28, given the value of ρ computed by Equation 5.40.

Figure 5.18 shows the accumulators for the two parameters θ and ρ as obtained by the implementation of Code 5.7 for the images in Figure 5.9(a) and (b). The accumulators are now one-dimensional, as in Figure 5.18(a), and show a clear peak. The peak in the first accumulator is close to $135°$. Thus, by subtracting the $90°$ introduced to make all values positive, we find that the slope of the line $\theta = -45°$. The peaks in the accumulators in Figure 5.18(b) define

```
%Parameter Decomposition for the Hough Transform for Lines

function HTDLine(inputimage)

%image size
[rows,columns]=size(inputimage);

%accumulator
rmax=round(sqrt(rows^2+columns^2));
accro=zeros(rmax,1);
acct=zeros(180,1);

%image
for x=1:columns
  for y=1:rows
    if(inputimage(y,x)==0)
       for Nx=x-2:x+2
         for Ny=y-2:y+2
           if(x~=Nx | y~=Ny)
              if(Nx>0 & Ny>0 & Nx<columns & Ny<rows)
                 if(inputimage(Ny,Nx)==0)
                    if(Ny-y~=0)
                        t=atan((x-Nx)/(Ny-y)); %Equation (5.40)
                     else t=pi/2;
                    end
                    r=round(x*cos(t)+y*sin(t)); %Equation (5.28)

                    t=round((t+pi/2)*180/pi);
                    acct(t)=acct(t)+1;

                    if(r<rmax & r>0)
                      accro(r)=accro(r)+1;
                    end
                 end
              end
           end
         end
       end
    end
  end
end
```

Code 5.7 Implementation of the parameter space reduction for the Hough transform for lines

two lines with similar slopes. The peak in the first accumulator represents the value of θ, while the two peaks in the second accumulator represent the location of the two lines. In general, when implementing parameter space decomposition it is necessary to follow a two-step process. First, it is necessary to gather data in one accumulator and search for the maximum. Secondly, the location of the maximum value is used as parameter value to gather data on the remaining accumulator.

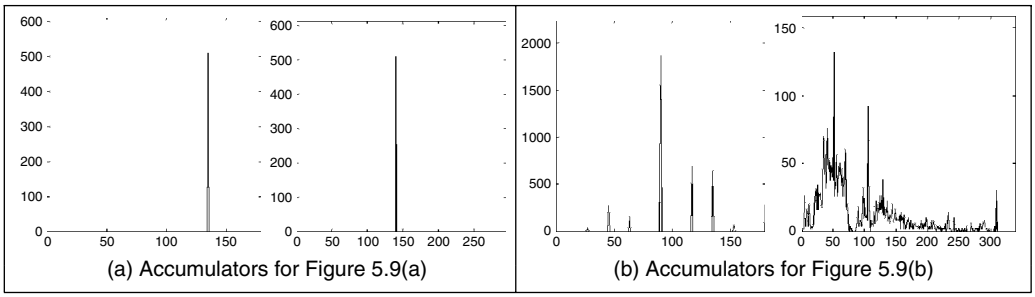


Figure 5.18 Parameter space reduction for the Hough transform for lines

5.4.5.2 Parameter space reduction for circles

In the case of lines, the relationship between local information computed from an image and the inclusion of a group of points (pairs) is in an alternative analytical description which can readily be established. For more complex primitives, it is possible to include several geometric relationships. These relationships are not defined for an arbitrary set of points, but include angular constraints that define relative positions between them. In general, we can consider different geometric properties of the circle to decompose the parameter space. This has motivated the development of many methods of parameter space decomposition (Aguado et al., 1996). An important geometric relationship is given by the geometry of the second directional derivatives. This relationship can be obtained by considering that Equation 5.31 defines a position vector function. That is,

$$\omega(\theta) = x(\theta)\begin{bmatrix}1\\0\end{bmatrix} + y(\theta)\begin{bmatrix}0\\1\end{bmatrix} \tag{5.41}$$

where

$$x(\theta) = x_0 + r\cos(\theta) \quad y(\theta) = y_0 + r\sin(\theta) \tag{5.42}$$

In this definition, we have included the parameter of the curve as an argument to highlight the fact that the function defines a vector for each value of θ. The endpoints of all the vectors trace a circle. The derivatives of Equation 5.41 with respect to θ define the first and second directional derivatives. That is,

$$\upsilon'(\theta) = x'(\theta)\begin{bmatrix}1\\0\end{bmatrix} + y'(\theta)\begin{bmatrix}0\\1\end{bmatrix}$$

$$\upsilon''(\theta) = x''(\theta)\begin{bmatrix}1\\0\end{bmatrix} + y''(\theta)\begin{bmatrix}0\\1\end{bmatrix} \tag{5.43}$$

where

$$x'(\theta) = -r\sin(\theta) \quad y'(\theta) = r\cos(\theta)$$

$$x''(\theta) = -r\cos(\theta) \quad y''(\theta) = -r\sin(\theta) \tag{5.44}$$

Figure 5.19 illustrates the definition of the first and second directional derivatives. The first derivative defines a tangential vector while the second one is similar to the vector function, but it has reverse direction. The fact that the edge direction measured for circles can be arranged

so as to point towards the centre was the basis of one of the early approaches to reducing the computational load of the HT for circles (Kimme et al., 1975).

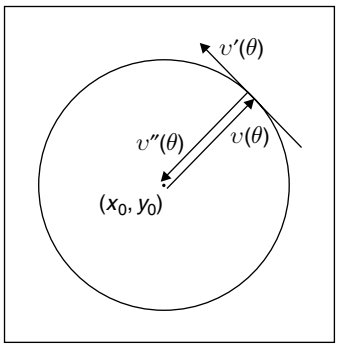


Figure 5.19 Definition of the first and second directional derivatives for a circle

According to Equations 5.42 and 5.44, we observe that the tangent of the angle of the first directional derivative denoted as $\phi'(\theta)$ is given by

$$\phi'(\theta) = \frac{y'(\theta)}{x'(\theta)} = -\frac{1}{\tan(\theta)} \tag{5.45}$$

Angles will be denoted by using the symbol ^. That is,

$$\hat{\phi}'(\theta) = \tan^{-1}(\phi'(\theta)) \tag{5.46}$$

Similarly, for the tangent of the second directional derivative we have:

$$\phi''(\theta) = \frac{y''(\theta)}{x''(\theta)} = \tan(\theta) \quad \text{and} \quad \hat{\phi}''(\theta) = \tan^{-1}(\phi''(\theta)) \tag{5.47}$$

By observing the definition of $\phi''(\theta)$, we have:

$$\phi''(\theta) = \frac{y''(\theta)}{x''(\theta)} = \frac{y(\theta) - y_0}{x(\theta) - x_0} \tag{5.48}$$

This equation defines a straight line passing through the points $(x(\theta), y(\theta))$ and (x_0, y_0) and is perhaps the most important relation in parameter space decomposition. The definition of the line is more evident by rearranging terms. That is,

$$y(\theta) = \phi''(\theta)(x(\theta) - x_0) + y_0 \tag{5.49}$$

This equation is independent of the radius parameter. Thus, it can be used to gather evidence of the location of the shape in a 2D accumulator. The HT mapping is defined by the dual form given by

$$y_0 = \phi''(\theta)(x_0 - x(\theta)) + y(\theta) \tag{5.50}$$

That is, given an image point $(x(\theta), y(\theta))$ and the value of $\phi''(\theta)$ we can generate a line of votes in the 2D accumulator (x_0, y_0). Once the centre of the circle is known, a 1D accumulator can be used to locate the radius. The key aspect of the parameter space decomposition is the method used to obtain the value of $\phi''(\theta)$ from image data. We will consider two alternative

ways. First, we will show that $\phi''(\theta)$ can be obtained by edge direction information. Secondly, we will show how it can be obtained from the information of a pair of points.

To obtain $\phi''(\theta)$, we can use the definition in Equations 5.46 and 5.47. According to these equations, the tangents $\phi''(\theta)$ and $\phi'(\theta)$ are perpendicular. Thus,

$$\phi''(\theta) = -\frac{1}{\phi'(\theta)} \tag{5.51}$$

Thus, the HT mapping in Equation 5.50 can be written in terms of gradient direction $\phi'(\theta)$ as

$$y_0 = y(\theta) + \frac{x(\theta) - x_0}{\phi'(\theta)} \tag{5.52}$$

This equation has a simple geometric interpretation, illustrated in Figure 5.20(a). We can see that the line of votes passes through the points $(x(\theta), y(\theta))$ and (x_0, y_0). The slope of the line is perpendicular to the direction of gradient direction.

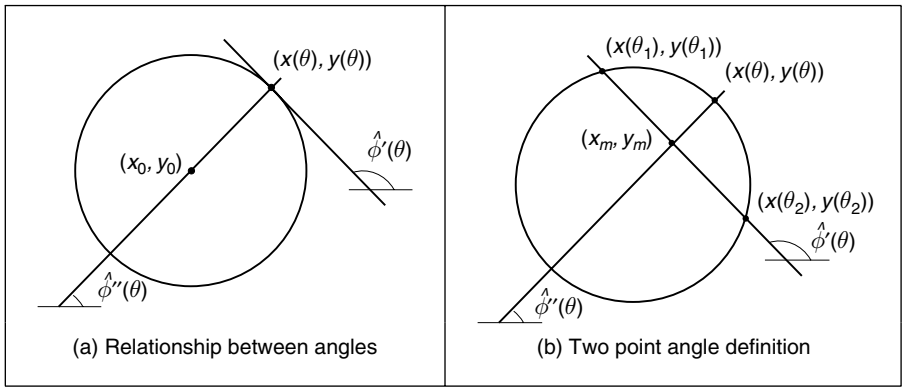


Figure 5.20 Geometry of the angle of the first and second directional derivatives

An alternative decomposition can be obtained by considering the geometry shown in Figure 5.20(b). In the figure we can see that if we take a pair of points (x_1, y_1) and (x_2, y_2), where $x_i = x(\theta_i)$, then the line that passes through the points has the same slope as the line at a point $(x(\theta), y(\theta))$. Accordingly,

$$\phi'(\theta) = \frac{y_2 - y_1}{x_2 - x_1} \tag{5.53}$$

where

$$\theta = \frac{1}{2}(\theta_1 + \theta_2) \tag{5.54}$$

Based on Equation 5.53 we have that

$$\phi''(\theta) = -\frac{x_2 - x_1}{y_2 - y_1} \tag{5.55}$$

The problem with using a pair of points is that by Equation 5.54 we cannot know the location of the point $(x(\theta), y(\theta))$. Fortunately, the voting line also passes through the midpoint of the line between the two selected points. Let us define this point as

$$x_m = \frac{1}{2}(x_1 + x_2) \quad y_m = \frac{1}{2}(y_1 + y_2) \tag{5.56}$$

Thus, by substitution of Equation 5.53 in Equation 5.52 and by replacing the point $(x(\theta), y(\theta))$ by (x_m, y_m), the HT mapping can be expressed as

$$y_0 = y_m + \frac{(x_m - x_0)(x_2 - x_1)}{(y_2 - y_1)} \tag{5.57}$$

This equation does not use gradient direction information, but it is based on pairs of points. This is analogous to the parameter space decomposition of the line presented in Equation 5.40. In that case, the slope can be computed by using gradient direction or, alternatively, by taking a pair of points. In the case of the circle, the tangent (and therefore the angle of the second directional derivative) can be computed by the gradient direction (i.e. Equation 5.51) or by a pair of points (i.e. Equation 5.55). However, it is important to notice that there are some other combinations of parameter space decomposition (Aguado, 1996).

Code 5.8 shows the implementation of the parameter space decomposition for the HT for circles. The implementation only detects the position of the circle and it gathers evidence by using the mapping in Equation 5.57. Pairs of points are restricted to a neighbourhood between 10×10 pixels and 12×12 pixels. We avoid using pixels that are close to each other since they do not produce accurate votes. We also avoid using pixels that are far away from each other, since by distance it is probable that they do not belong to the same circle and would only increase the noise in the accumulator. To trace the line, we use two equations that are selected according to the slope.

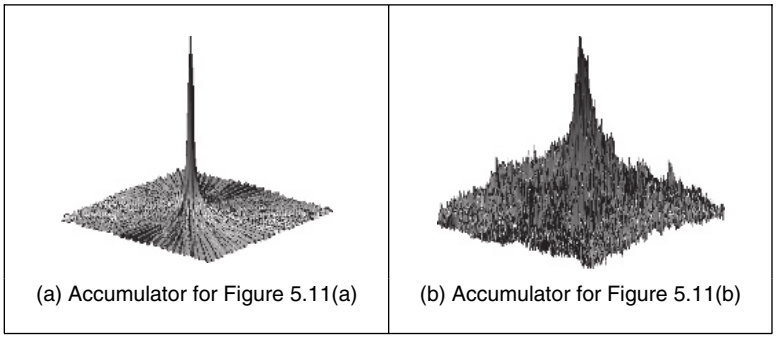
Figure 5.21 shows the accumulators obtained by the implementation of Code 5.8 for the images in Figure 5.11(a) and (b). Both accumulators show a clear peak that represents the location

```
%Parameter Decomposition for the Hough Transform for Circles

function HTDCircle(inputimage)

%image size
[rows,columns]=size(inputimage);

%accumulator
acc=zeros(rows,columns);

%gather evidence
for x1=1:columns
  for y1=1:rows
    if(inputimage(y1,x1)==0)
       for x2=x1-12:x1+12
         for y2=y1-12:y1+12
           if(abs(x2-x1)>10 | abs(y2-y1)>10)
             if(x2>0 & y2>0 & x2<columns & y2<rows)
                if(inputimage(y2,x2)==0)
                   xm=(x1+x2)/2;   ym=(y1+y2)/2;
                    if(y2-y1~=0)   m=((x2-x1)/(y2-y1));
                       else   m=99999999;
                    end
```

```
                    if(m>-1 & m<1)
                       for x0=1:columns
                           y0=round(ym+m*(xm-x0));
                           if(y0>0 & y0<rows)
                              acc(y0,x0)=acc(y0,x0)+1;
                           end
                       end
                    else
                       for y0=1:rows
                           x0= round(xm+(ym-y0)/m);
                           if(x0>0 & x0<columns)
                              acc(y0,x0)=acc(y0,x0)+1;
                           end
                       end
                    end
                 end
              end
           end
        end
     end
   end
end
```

Code 5.8 Parameter space reduction for the Hough transform for circles

of the circle. Small peaks in the background of the accumulator in Figure 5.11(b) correspond to circles with only a few points. In general, there is a compromise between the *width* of the peak and the *noise* in the accumulator. The peak can be made narrower by considering pairs of points that are more widely spaced. However, this can also increase the level of background noise. Background noise can be reduced by taking points that are closer together, but this makes the peak wider.



(a) Accumulator for Figure 5.11(a) (b) Accumulator for Figure 5.11(b)

Figure 5.21 Parameter space reduction for the Hough transform for circles

5.4.5.3 Parameter space reduction for ellipses

Part of the simplicity in the parameter decomposition for circles comes from the fact that circles are isotropic. Ellipses have more free parameters and are geometrically more complex. Thus, geometrical properties involve more complex relationships between points, tangents and angles. However, they maintain the geometric relationship defined by the angle of the second derivative. According to Equations 5.41 and 5.43, the vector position and directional derivatives of an ellipse in Equation 5.35 have the components

$$\begin{aligned} x'(\theta) &= -a_x \sin(\theta) + b_x \cos(\theta) \quad y'(\theta) = -a_y \sin(\theta) + b_y \cos(\theta) \\ x''(\theta) &= -a_x \cos(\theta) - b_x \sin(\theta) \quad y''(\theta) = -a_y \cos(\theta) - b_y \sin(\theta) \end{aligned} \tag{5.58}$$

The tangent of angle of the first and second directional derivatives are given by

$$\begin{aligned} \phi'(\theta) &= \frac{y'(\theta)}{x'(\theta)} = \frac{-a_y \cos(\theta) + b_y \sin(\theta)}{-a_x \cos(\theta) + b_x \sin(\theta)} \\ \phi''(\theta) &= \frac{y''(\theta)}{x''(\theta)} = \frac{-a_y \cos(\theta) - b_y \sin(\theta)}{-a_x \cos(\theta) - b_x \sin(\theta)} \end{aligned} \tag{5.59}$$

By considering Equation 5.58, Equation 5.48 is also valid for an ellipse. That is,

$$\frac{y(\theta) - y_0}{x(\theta) - x_0} = \phi''(\theta) \tag{5.60}$$

The geometry of the definition in this equation is illustrated in Figure 5.22(a). As in the case of circles, this equation defines a line that passes through the points $(x(\theta), y(\theta))$ and (x_0, y_0). However, in the case of the ellipse the angles $\hat{\phi}'(\theta)$ and $\hat{\phi}''(\theta)$ are not orthogonal. This makes the computation of $\phi''(\theta)$ more complex. To obtain $\phi''(\theta)$ we can extend the geometry presented in Figure 5.20(b). That is, we take a pair of points to define a line whose slope defines the value of $\phi''(\theta)$ at another point. This is illustrated in Figure 5.22(b). The line in Equation 5.60 passes through the middle point (x_m, y_m). However, it is not orthogonal to the tangent line. To obtain an expression of the HT mapping, we will first show that the relationship in Equation 5.54 is also valid for ellipses. Then we will use this equation to obtain $\phi''(\theta)$.

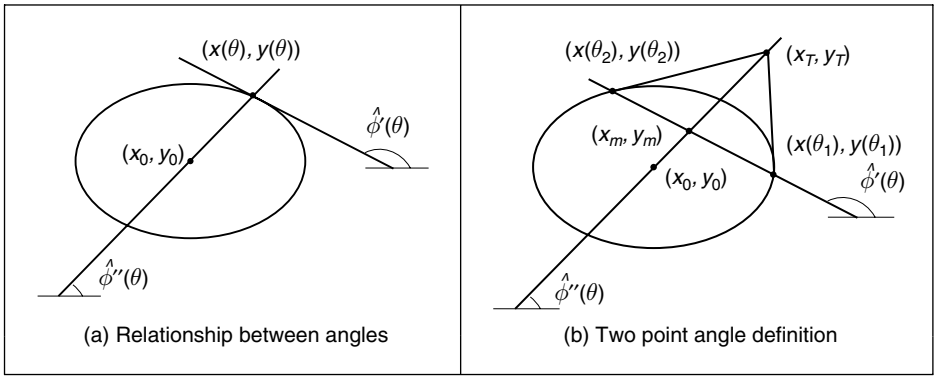


Figure 5.22 Geometry of the angle of the first and second directional derivatives

The relationships in Figure 5.22(b) do not depend on the orientation or position of the ellipse. Thus, three points can be defined by

$$\begin{aligned} x_1 &= a_x \cos(\theta_1) \quad x_2 = a_x \cos(\theta_2) \quad x(\theta) = a_x \cos(\theta) \\ y_1 &= b_x \sin(\theta_1) \quad y_2 = b_x \sin(\theta_2) \quad y(\theta) = b_x \sin(\theta) \end{aligned} \tag{5.61}$$

The point $(x(\theta), y(\theta))$ is given by the intersection of the line in Equation 5.60 with the ellipse. That is,

$$\frac{y(\theta) - y_0}{x(\theta) - x_0} = \frac{a_x}{b_y} \frac{y_m}{x_m} \tag{5.62}$$

By substitution of the values of (x_m, y_m) defined as the average of the coordinates of the points (x_1, y_1) and (x_2, y_2) in Equation 5.56, we have:

$$\tan(\theta) = \frac{a_x}{b_y} \frac{b_y \sin(\theta_1) + b_y \sin(\theta_2)}{a_x \cos(\theta_1) + a_x \cos(\theta_2)} \tag{5.63}$$

Thus,

$$\tan(\theta) = \tan\left(\frac{1}{2}(\theta_1 + \theta_2)\right) \tag{5.64}$$

From this equation is evident that the relationship in Equation 5.54 is also valid for ellipses. Based on this result, the tangent angle of the second directional derivative can be defined as

$$\phi''(\theta) = \frac{b_y}{a_x} \tan(\theta) \tag{5.65}$$

By substitution in Equation 5.62, we have:

$$\phi''(\theta) = \frac{y_m}{x_m} \tag{5.66}$$

This equation is valid when the ellipse is not translated. If the ellipse is translated then the tangent of the angle can be written in terms of the points (x_m, y_m) and (x_T, y_T) as

$$\phi''(\theta) = \frac{y_T - y_m}{x_T - x_m} \tag{5.67}$$

By considering that the point (x_T, y_T) is the intersection point of the tangent lines at (x_1, y_1) and (x_2, y_2), we obtain

$$\phi''(\theta) = \frac{AC + 2BD}{2A + BC} \tag{5.68}$$

where

$$\begin{aligned} A &= y_1 - y_2 \quad B = x_1 - x_2 \\ C &= \phi_1 + \phi_2 \quad D = \phi_1 \cdot \phi_2 \end{aligned} \tag{5.69}$$

and ϕ_1, ϕ_2 are the slope of the tangent line to the points. Finally, by considering Equation 5.60, the HT mapping for the centre parameter is defined as

$$y_0 = y_m + \frac{AC + 2BD}{2A + BC}(x_0 - x_m) \tag{5.70}$$

This equation can be used to gather evidence that is independent of rotation or scale. Once the location is known, a 3D parameter space is needed to obtain the remaining parameters. However, these parameters can also be computed independently using two 2D parameter spaces (Aguado

et al., 1996). You can avoid using the gradient direction in Equation 5.68 by including more points. In fact, the tangent $\phi''(\theta)$ can be computed by taking four points (Aguado, 1996). However, the inclusion of more points generally leads to more background noise in the accumulator.

Code 5.9 shows the implementation of the ellipse location mapping in Equation 5.57. As in the case of the circle, pairs of points need to be restricted to a neighbourhood. In the implementation, we consider pairs at a fixed distance given by the variable *i*. Since we are including gradient direction information, the resulting peak is generally quite wide. Again, the selection of the distance between points is a compromise between the level of background noise and the width of the peak.

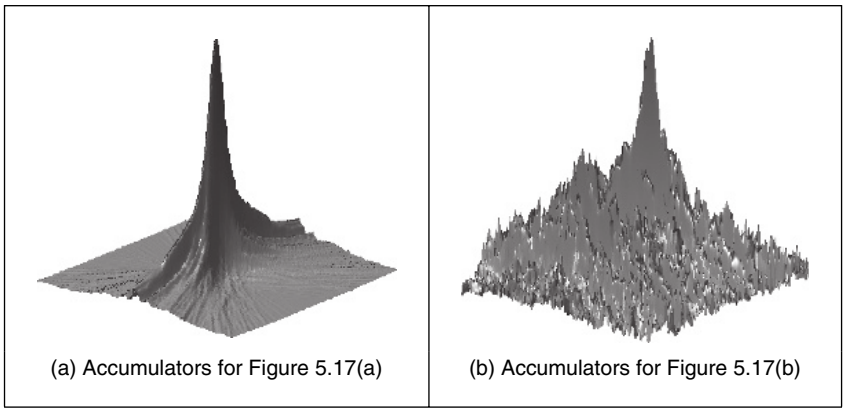
```
%Parameter Decomposition for Ellipses
function HTDEllipse(inputimage)

%image size
[rows,columns]=size(inputimage);

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

%accumulator
acc=zeros(rows,columns);

%gather evidence
for x1=1:columns
  for y1=1:1:rows
    if(M(y1,x1)~=0)
      for i=60:60
        x2=x1-i; y2=y1-I;
        incx=1; incy=0;
        for k=0: 8*i-1
          if(x2>0 & y2>0 & x2<columns & y2<rows)
          if M(y2,x2)~=0

            m1=Ang(y1,x1); m2=Ang(y2,x2);

            if(abs(m1-m2)>.2)

              xm=(x1+x2)/2; ym=(y1+y2)/2;
              m1=tan(m1); m2=tan(m2);

              A=y1-y2; B=x2-x1;
              C=m1+m2; D=m1*m2;
              N=(2*A+B*C);
              if N~=0
                m=(A*C+2*B*D)/N;
              else
                m=99999999;
              end;
```

```
            if(m>-1 & m<1)
              for x0=1:columns
                y0=round(ym+m*(x0-xm));
                if(y0>0 & y0<rows)
                   acc(y0,x0)=acc(y0,x0)+1;
                 end
              end
            else
              for y0=1:rows
                x0= round(xm+(y0-ym)/m);
                if(x0>0 & x0<columns)
                  acc(y0,x0)=acc(y0,x0)+1;
                end
              end
            end % if abs
          end  % if M
          end

          x2=x2+incx; y2=y2+incy;

          if x2>x1+I
            x2=x1+i;
            incx=0;     incy=1;
            y2=y2+incy;
          end

          if y2>y1+i
            y2=y1+i;
            incx=-1;    incy=0;
            x2=x2+incx;
          end

          if x2<x1-i
            x2=x1-i;
            incx=0;     incy=-1;
            y2=y2+incy;
          end
        end % for k
      end % for I
    end % if (x1,y1)
  end % y1
end %x1
```

Code 5.9 Implementation of the parameter space reduction for the Hough transform for ellipses

Figure 5.23 shows the accumulators obtained by the implementation of Code 5.9 for the images in Figure 5.17(a) and (b). The peak represents the location of the ellipses. In general, there is noise and the accumulator is wide, for two main reasons. First, when the gradient direction is not accurate, then the line of votes does not pass exactly over the centre of the ellipse. This forces the peak to become wider with less height. Secondly, to avoid numerical instabilities

we need to select points that are well separated. However, this increases the probability that the points do not belong to the same ellipse, thus generating background noise in the accumulator.



(a) Accumulators for Figure 5.17(a)

(b) Accumulators for Figure 5.17(b)

Figure 5.23 Parameter space reduction for the Hough transform for ellipses

5.5 Generalized Hough transform

Many shapes are far more complex than lines, circles or ellipses. It is often possible to partition a complex shape into several geometric primitives, but this can lead to a highly complex data structure. In general, is more convenient to extract the whole shape. This has motivated the development of techniques that can find *arbitrary* shapes using the evidence-gathering procedure of the HT. These techniques give results equivalent to those delivered by matched template filtering, but with the computational advantage of the evidence-gathering approach. An early approach offered only limited capability only for arbitrary shapes (Merlin and Faber 1975). The full mapping is called the *generalized Hough transform* (GHT) (Ballard, 1981) and can be used to locate arbitrary shapes with unknown *position*, *size* and *orientation*. The GHT can be formally defined by considering the duality of a curve. One possible implementation can be based on the discrete representation given by tabular functions. These two aspects are explained in the following two sections.

5.5.1 Formal definition of the GHT

The formal analysis of the HT provides the route for generalizing it to arbitrary shapes. We can start by generalizing the definitions in Equation 5.41. In this way a model shape can be defined by a curve

$$\boldsymbol{v}(\theta) = x(\theta)\begin{bmatrix}1\\0\end{bmatrix} + y(\theta)\begin{bmatrix}0\\1\end{bmatrix} \tag{5.71}$$

For a circle, for example, we have $x(\theta) = r\cos(\theta)$ and $y(\theta) = r\sin(\theta)$. Any shape can be represented by following a more complex definition of $x(\theta)$ and $y(\theta)$.

In general, we are interested in matching the model shape against a shape in an image. However, the shape in the image has a different location, orientation and scale. Originally, the

GHT defined a scale parameter in the x and y directions, but owing to computational complexity and practical relevance the use of a single scale has become much more popular. Analogous to Equation (5.33), we can define the image shape by considering translation, rotation and change of scale. Thus, the shape in the image can be defined as

$$\omega(\theta, b, \lambda, \rho) = b + \lambda \mathbf{R}(\rho)\upsilon(\theta) \tag{5.72}$$

where $b = (x_0, y_0)$ is the translation vector, λ is a scale factor and $\mathbf{R}(\rho)$ is a rotation matrix (as in Equation 5.31). Here, we have included explicitly the parameters of the transformation as arguments, but to simplify the notation they will be omitted later. The shape of $\omega(\theta, b, \lambda, \rho)$ depends on four parameters. Two parameters define the location b, plus the rotation and scale. It is important to notice that θ does not define a free parameter, it only traces the curve.

To define a mapping for the HT we can follow the approach used to obtain Equation 5.35. Thus, the location of the shape is given by

$$b = \omega(\theta) - \lambda \mathbf{R}(\rho)\upsilon(\theta) \tag{5.73}$$

Given a shape $\omega(\theta)$ and a set of parameters b, λ and ρ, this equation defines the location of the shape. However, we do not know the shape $\omega(\theta)$ (since it depends on the parameters that we are looking for), but we only have a point in the curve. If we call $\omega_i = (\omega_{xi}, \omega_{yi})$ the point in the image, then

$$b = \omega_i - \lambda \mathbf{R}(\rho)\upsilon(\theta) \tag{5.74}$$

defines a system with four unknowns and with as many equations as points in the image. To find the solution we can gather evidence by using a four-dimensional (4D) accumulator space. For each potential value of b, λ and ρ, we trace a point spread function by considering all the values of θ. That is, all the points in the curve $\upsilon(\theta)$.

In the GHT the gathering process is performed by adding an extra constraint to the system that allows us to match points in the image with points in the model shape. This constraint is based on gradient direction information and can be explained as follows. We said that ideally, we would like to use Equation 5.73 to gather evidence. For that we need to know the shape $\omega(\theta)$ and the model $\upsilon(\theta)$, but we only know the discrete points ω_i and we have supposed that these are the same as the shape, i.e. that $\omega(\theta) = \omega_i$. Based on this assumption, we then consider all the potential points in the model shape, $\upsilon(\theta)$. However, this is not necessary since we only need the point in the model, $\upsilon(\theta)$, that corresponds to the point in the shape, $\omega(\theta)$. We cannot know the point in the shape, $\upsilon(\theta)$, but we can compute some properties from the model and from the image. Then, we can check whether these properties are similar at the point in the model and at a point in the image. If they are indeed *similar*, the points might *correspond*: if they do we can gather evidence on the parameters of the shape. The GHT considers as feature the gradient direction at the point. We can generalize Equations 5.45 and 5.46 to define the gradient direction at a point in the arbitrary model. Thus,

$$\phi'(\theta) = \frac{y'(\theta)}{x'(\theta)} \quad \text{and} \quad \hat{\phi}'(\theta) = \tan^{-1}(\phi'(\theta)) \tag{5.75}$$

Thus, Equation 5.73 is true only if the gradient direction at a point in the image matches the rotated gradient direction at a point in the (rotated) model, that is

$$\phi_i' = \hat{\phi}'(\theta) - \rho \tag{5.76}$$

where $\hat{\varphi}_i'$ is the angle at the point ω_i. Note that according to this equation, gradient direction is independent of scale (in theory at least) and it changes in the same ratio as rotation. We can constrain Equation 5.74 to consider only the points $v(\theta)$ for which

$$\phi_i' - \hat{\phi}'(\theta) + \rho = 0 \tag{5.77}$$

That is, a point spread function for a given edge point ω_i is obtained by selecting a subset of points in $v(\theta)$ such that the edge direction at the image point rotated by ρ equals the gradient direction at the model point. For each point ω_i and selected point in $v(\theta)$ the point spread function is defined by the HT mapping in Equation 5.74.

5.5.2 Polar definition

Equation 5.74 defines the mapping of the HT in Cartesian form. That is, it defines the votes in the parameter space as a pair of coordinates (x, y). There is an alternative definition in polar form. The polar implementation is more common than the Cartesian form (Hecker and Bolle, 1994; Sonka et al., 1994). The advantage of the polar form is that it is easy to implement since changes in rotation and scale correspond to addition in the angle-magnitude representation. However, ensuring that the polar vector has the correct direction incurs more complexity.

Equation 5.74 can be written in a form that combines rotation and scale as

$$b = \omega(\theta) - \gamma(\lambda, \rho) \tag{5.78}$$

where $\gamma^{\mathrm{T}}(\lambda, \rho) = [\gamma_x(\lambda, \rho)\gamma_y(\lambda, \rho)]$ and where the combined rotation and scale is

$$\begin{aligned} \gamma_x(\lambda, \rho) &= \lambda(x(\theta)\cos(\rho) - y(\theta)\sin(\rho)) \\ \gamma_y(\lambda, \rho) &= \lambda(x(\theta)\sin(\rho) + y(\theta)\cos(\rho)) \end{aligned} \tag{5.79}$$

This combination of rotation and scale defines a vector, $\gamma(\lambda, \rho)$, whose tangent angle and magnitude are given by

$$\tan(\alpha) = \frac{\gamma_y(\lambda, \rho)}{\gamma_x(\lambda, \rho)} \quad r = \sqrt{\gamma_x^2(\lambda, \rho) + \gamma_y^2(\lambda, \rho)} \tag{5.80}$$

The main idea here is that if we know the values for α and r, then we can gather evidence by considering Equation 5.78 in polar form. That is,

$$b = \omega(\theta) - re^{\alpha} \tag{5.81}$$

Thus, we should focus on computing values for α and r. After some algebraic manipulation, we have:

$$\alpha = \phi(\theta) + \rho \quad r = \lambda\Gamma(\theta) \tag{5.82}$$

where

$$\phi(\theta) = \tan^{-1}\left(\frac{y(\theta)}{x(\theta)}\right) \quad \Gamma(\theta) = \sqrt{x^2(\theta) + y^2(\theta)} \tag{5.83}$$

In this definition, we must include the constraint defined in Equation 5.77. That is, we gather evidence only when the gradient direction is the *same*. Notice that the square root in the definition of the magnitude in Equation 5.83 can have positive and negative values. The sign must be selected in a way that the vector has the correct direction.

5.5.3 The GHT technique

Equations 5.74 and 5.81 define an HT mapping function for arbitrary shapes. The geometry of these equations is shown in Figure 5.24. Given an image point ω_i we have to find a displacement vector $\gamma(\lambda, \rho)$. When the vector is placed at ω_i, its end is at the point b. In the GHT jargon, this point called the reference point. The vector $\gamma(\lambda, \rho)$ can be easily obtained as $\lambda R(\rho)\, \upsilon(\theta)$ or alternatively as re^{α}. However, to evaluate these equations, we need to know the point $\upsilon(\theta)$. This is the crucial step in the evidence gathering process. Notice the remarkable similarity between Figures 5.20(a), 5.22(a) and 5.24(a). This is not a coincidence, but Equation 5.60 is a particular case of Equation 5.73.

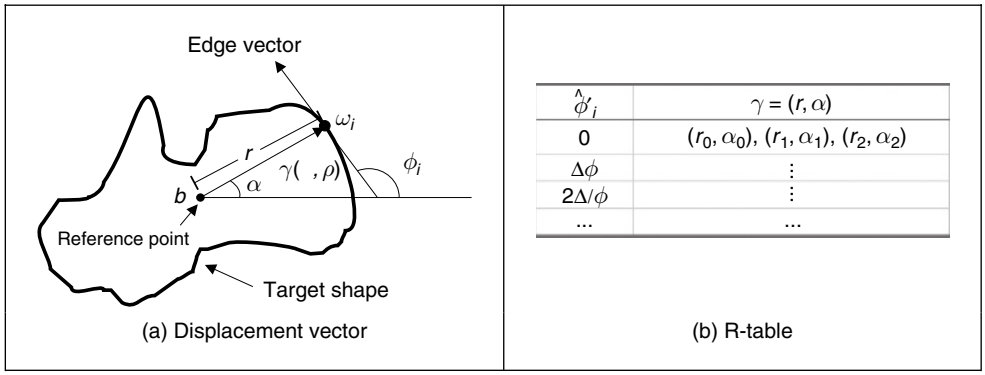


Figure 5.24 Geometry of the GHT

The process of determining $\upsilon(\theta)$ centres on solving Equation 5.76. According to this equation, since we know $\hat{\phi}'_i$, then we need to find the point $\upsilon(\theta)$ whose gradient direction is $\hat{\phi}'_i + \rho = 0$. Then we must use $\upsilon(\theta)$ to obtain the displacement vector $\gamma(\lambda, \rho)$. The GHT precomputes the solution of this problem and stores it an array called the *R-table*. The R-table stores for each value of $\hat{\phi}'_i$ the vector $\gamma(\lambda, \rho)$ for $\rho = 1$ and $\lambda = 1$. In polar form, the vectors are stored as a magnitude direction pair and in Cartesian form as a coordinate pair.

The possible range for $\hat{\phi}'_i$ is between $-\pi/2$ and $\pi/2$ radians. This range is split into N equispaced slots, or bins. These slots become rows of data in the R-table. The edge direction at each border point determines the appropriate row in the R-table. The length, r, and direction, α, from the reference point are entered into a new column element, at that row, for each border point in the shape. In this manner, the N rows of the R-table have elements related to the border information; elements for which there is no information contain null vectors. The length of each row is given by the number of edge points that have the edge direction corresponding to that row; the total number of elements in the R-table equals the number of edge points above a chosen threshold. The *structure* of the R-table for N edge direction bins and m template border points is illustrated in Figure 5.23 (b).

The process of *building* the R-table is illustrated in Code 5.10. In this code, we implement the Cartesian definition given in Equation 5.74. According to this equation the displacement vector is given by

$$\gamma(1, 0) = \omega(\theta) - b \tag{5.84}$$

```
%R-Table

function T=RTable(entries,inputimage)

%image size
[rows,columns]=size(inputimage);

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

%compute reference point
xr=0; yr=0; p=0;
for x=1:columns
  for y=1:rows
     if(M(y,x)~=0)
        xr=xr+x;
        yr=yr+y;
        p=p+1;
     end
  end
end
xr=round(xr/p);
yr=round(yr/p);

%accumulator
D=pi/entries;

s=0;            % number of entries in the table
t=[];
F=zeros(entries,1); % number of entries in the row

% for each edge point
for x=1:columns
   for y=1:rows
      if(M(y,x)~=0)

         phi=Ang(y,x);
         i=round((phi+(pi/2))/D);
         if(i==0) i=1; end;

         V=F(i)+1;

         if(V>s)
            s=s+1;
            T(:,:,s)=zeros(entries,2);
         end;

         T(i,1,V)=x-xr;
         T(i,2,V)=y-yr;
         F(i)=F(I)+1;

      end %if
   end % y
end% x
```

Code 5.10 Implementation of the construction of the R-table

The matrix **T** stores the coordinates of γ (1, 0). This matrix is expanded to accommodate all of the computed entries.

Code 5.11 shows the implementation of the gathering process of the GHT. In this case we use the Cartesian definition in Equation 5.74. The coordinates of points given by evaluation of all R-table points for the particular row indexed by the gradient magnitude are used to increment cells in the accumulator array. The maximum number of votes occurs at the location of the original reference point. After all edge points have been inspected, the location of the shape is given by the maximum of an accumulator array.

```
%Generalized Hough Transform

function GHT(inputimage,RTable)

%image size
[rows,columns]=size(inputimage);

%table size
[rowsT,h,columnsT]=size(RTable);
D=pi/rowsT;

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

%accumulator
acc=zeros(rows,columns);

%for each edge point
for x=1:columns
   for y=1:rows
      if(M(y,x)~=0)

         phi=Ang(y,x);
         i=round((phi+(pi/2))/D);
         if(i==0) i=1; end;

         for j=1:columnsT
           if(RTable(i,1,j)==0 & RTable(i,2,j)==0)
             j=columnsT; %no more entries
           else
             a0=x-RTable(i,1,j); b0=y-RTable(i,2,j);
             if(a0>0 & a0<columns & b0>0 & b0<rows)
               acc(b0,a0)=acc(b0,a0)+1;
             end
           end
         end
      end %if
   end % y
end% x
```

Code 5.11 Implementation of the GHT

Note that if we want to try other values for rotation and scale, then it is necessary to compute a table $\gamma(\lambda, \rho)$ for all potential values. However, this can be avoided by considering that $\gamma(\lambda, \rho)$ can be computed from $\gamma(1, 0)$. That is, if we want to accumulate evidence for $\gamma(\lambda, \rho)$, then we use the entry indexed by $\hat{\phi}'_i + \rho$ and we rotate and scale the vector $\gamma(1, 0)$. That is,

$$\begin{aligned} \gamma_x(\lambda, \rho) &= \lambda(\gamma_x(1, 0)\cos(\rho) - \gamma_y(1, 0)\sin(\rho)) \\ \gamma_y(\lambda, \rho) &= \lambda(\gamma_x(1, 0)\sin(\rho) + \gamma_y(1, 0)\cos(\rho)) \end{aligned} \tag{5.85}$$

In the case of the polar form, the angle and magnitude need to be defined according to Equation 5.82.

The application of the GHT to detect an arbitrary shape with unknown translation is illustrated in Figure 5.25. We constructed an R-table from the template shown in Figure 5.2(a). The table contains 30 rows. The accumulator in Figure 5.25(c) was obtained by applying the GHT to the image in Figure 5.25(b). Since the table was obtained from a shape with the same scale and rotation as the primitive in the image, the GHT produces an accumulator with a clear peak at the centre of mass of the shape.

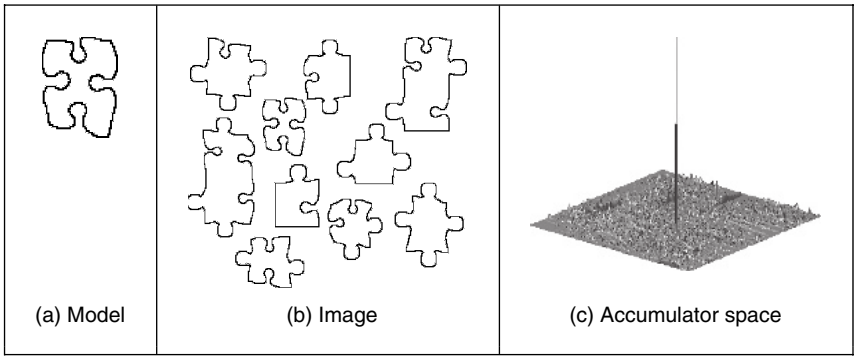


Figure 5.25 Example of the GHT

Although the example in Figure 5.25 shows that the GHT is an effective method for shape extraction, there are several inherent difficulties in its formulation (Grimson and Huttenglocher, 1990; Aguado et al., 2000b). The most evident problem is that the table does not provide an accurate representation when objects are scaled and translated. This is because the table implicitly assumes that the curve is represented in discrete form. Thus, the GHT maps a discrete form into a discrete parameter space. In addition, the transformation of scale and rotation can induce other discretization errors. This is because when discrete images are mapped to be larger, or when they are rotated, loci which are unbroken sets of points rarely map to unbroken sets in the new image. Another important problem is the excessive computations required by the 4D parameter space. This makes the technique impractical. In addition, the GHT is clearly dependent on the accuracy of directional information. By these factors, the results provided by the GHT can become less reliable. A solution is to use of an analytic form instead of a table (Aguado et al., 1998). This avoids discretization errors and makes the technique more reliable. This also allows the extension to affine or other transformations. However, this technique requires solving for the point $v(\theta)$ in an analytic way, increasing the computational load. A solution is

to reduce the number of points by considering characteristics points defined as points of high curvature. However, this still requires the use of a 4D accumulator. An alternative to reduce this computational load is to include the concept of invariance in the GHT mapping.

5.5.4 Invariant GHT

The problem with the GHT (and other extensions of the HT) is that they are very general. That is, the HT gathers evidence for a single point in the image. However, a point on its own provides little information. Thus, it is necessary to consider a large parameter space to cover all the potential shapes defined by a given image point. The GHT improves evidence gathering by considering a point and its gradient direction. However, since gradient direction changes with rotation, the evidence gathering is improved in terms of noise handling, but little is done about computational complexity.

To reduce the computational complexity of the GHT, we can consider replacing the gradient direction by another feature; that is, by a feature that is not affected by *rotation*. Let us explain this idea in more detail. The main aim of the constraint in Equation 5.77 is to include gradient direction to reduce the number of votes in the accumulator by identifying a point $v(\theta)$. Once this point is known, we obtain the displacement vector $\gamma(\lambda, \rho)$. However, for each value of rotation, we have a different point in $v(\theta)$. Now let us replace that constraint in Equation 5.76 by a constraint of the form

$$Q(\omega_i) = Q(v(\theta)) \tag{5.86}$$

The function Q is said to be invariant and it computes a feature at the point. This feature can be, for example, the colour of the point, or any other property that does not change in the model and in the image. By considering Equation 5.86, Equation 5.77 is redefined as

$$Q(\omega_i) = Q(v(\theta)) = 0 \tag{5.87}$$

That is, instead of searching for a point with the same gradient direction, we will search for the point with the same invariant feature. The advantage is that this feature will not change with rotation or scale, so we only require a 2D space to locate the shape. The definition of Q depends on the application and the type of transformation. The most general invariant properties can be obtained by considering geometric definitions. In the case of rotation and scale changes (i.e. similarity transformations) the fundamental invariant property is given by the concept of angle.

An angle is defined by three points and its value remains unchanged when it is rotated and scaled. Thus, if we associate to each edge point ω_i a set of other two points $\{\omega_j, \omega_T\}$, we can compute a geometric feature that is invariant to similarity transformations. That is,

$$Q(\omega_i) = \frac{X_j Y_i - X_i Y_j}{X_i X_j + Y_i Y_j}, \tag{5.88}$$

where $X_k = \omega_k - \omega_T$, $Y_k = \omega_k - \omega_T$. Equation 5.88 defines the tangent of the angle at the point ω_T. In general, we can define the points $\{\omega_j, \omega_T\}$ in different ways. An alternative geometric arrangement is shown in Figure 5.26(a). Given the points ω_i and a fixed angle ϑ, then we determine the point ω_j such that the angle between the tangent line at ω_i and the line that joins the points is ϑ. The third point is defined by the intersection of the tangent lines at ω_i and ω_j. The tangent of the angle β is defined by Equation 5.88. This can be expressed in terms of the points and its gradient directions as

$$Q(\omega_i) = \frac{\phi'_i - \phi'_j}{1 + \phi'_i \phi'_j} \tag{5.89}$$

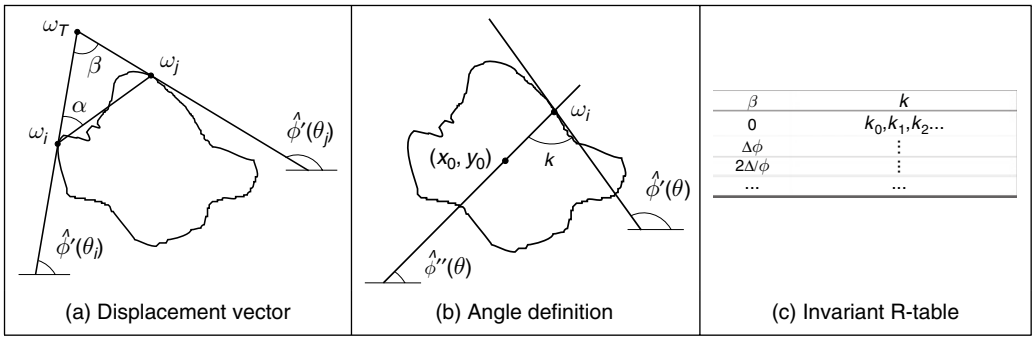


Figure 5.26 Geometry of the invariant GHT

We can replace the gradient angle in the R-table, by the angle β. The form of the new invariant table is shown in Figure 5.26(c). Since the angle β does not change with rotation or change of scale, then we do not need to change the index for each potential rotation and scale. However, the displacement vectors changes according to rotation and scale (i.e. Equation 5.85) that. Thus, if we want an invariant formulation, we must also change the definition of the position vector.

To locate the point b we can generalize the ideas presented in Figures 5.20(a) and 5.22(a). Figure 5.26(b) shows this generalization. As in the case of the circle and ellipse, we can locate the shape by considering a line of votes that passes through the point b. This line is determined by the value of ϕ_i''. We will do two things. First, we will find an invariant definition of this value. Secondly, we will include it on the GHT table.

We can develop Equation 5.73 as

$$\begin{bmatrix} x_0 \\ y_0 \end{bmatrix} = \begin{bmatrix} \omega_{xi} \\ \omega_{yi} \end{bmatrix} + \lambda \begin{bmatrix} \cos(\rho) & \sin(\rho) \\ -\sin(\rho) & \cos(\rho) \end{bmatrix} \begin{bmatrix} x(\theta) \\ y(\theta) \end{bmatrix} \tag{5.90}$$

Thus, Equation 5.60 generalizes to

$$\phi_i'' = \frac{\omega_{yi} - y_0}{\omega_{xi} - x_0} = \frac{[-\sin(\rho)\ \cos(\rho)]y(\theta)}{[\cos(\rho)\ \sin(\rho)]x(\theta)} \tag{5.91}$$

By some algebraic manipulation, we have:

$$\phi_i'' = \tan(\xi - \rho) \tag{5.92}$$

where

$$\xi = \frac{y(\theta)}{x(\theta)} \tag{5.93}$$

To define ϕ_i'', we can consider the tangent angle at the point ω_i. By considering the derivative of Equation 5.72, we have:

$$\phi_i' = \frac{[-\sin(\rho)\ \cos(\rho)]y'(\theta)}{[\cos(\rho)\ \sin(\rho)]x'(\theta)} \tag{5.94}$$

Thus,

$$\phi_i' = \tan(\phi - \rho) \tag{5.95}$$

where

$$\phi = \frac{y'(\theta)}{x'(\theta)} \tag{5.96}$$

By considering Equations 5.92 and 5.95, we define

$$\hat{\phi}_i'' = k + \hat{\phi}_i' \tag{5.97}$$

The important point in this definition is that the value of k is invariant to rotation. Thus, if we use this value in combination with the tangent at a point we can have an invariant characterization. To see that k is invariant, we solve it for Equation 5.97. That is,

$$k = \hat{\phi}_i' - \hat{\phi}_i'' \tag{5.98}$$

Thus,

$$k = \xi - \rho - (\phi - \rho) \tag{5.99}$$

That is,

$$k = \xi - \phi \tag{5.100}$$

This is independent of rotation. The definition of k has a simple geometric interpretation illustrated in Figure 5.26(b).

To obtain an invariant GHT, it is necessary to know for each point ω_i, the corresponding point $v(\theta$ and then compute the value of ϕ_i''. Then evidence can be gathered by the line in Equation 5.91. That is,

$$y_0 = \phi_i''(x_0 - \omega_{xi}) + \omega_{yi} \tag{5.101}$$

To compute ϕ_i'' we can obtain k and then use Equation 5.100. In the standard tabular form the value of k can be precomputed and stored as function of the angle β.

Code 5.12 illustrates the implementation to obtain the invariant R-table. This code is based on Code 5.10. The value of α is set to $\pi/4$ and each element of the table stores a single value computed according to Equation 5.98. The more cumbersome part of the code is to search for the point ω_j. We search in two directions from ω_i and we stop once an edge point has been located. This search is performed by tracing a line. The trace is dependent on the slope. When the slope is between -1 and $+1$ we determine a value of y for each value of x, otherwise we determine a value of x for each value of y.

Code 5.13 illustrates the evidence-gathering process according to Equation 5.101. This code is based in the implementation presented in Code 5.11. We use the value of β defined in Equation 5.89 to index the table passed as parameter to the function `GHTInv`. The data k recovered from the table is used to compute the slope of the angle defined in Equation 5.97. This is the slope of the line of votes traced in the accumulators.

Figure 5.27 shows the accumulator obtained by the implementation of Code 5.13. Figure 5.27(a) shows the template used in this example. This template was used to construct the R-Table in Code 5.12. The R-table was used to accumulate evidence when searching for the piece of the puzzle in the image in Figure 5.27(b). Figure 5.27(c) shows the result of the evidence-gathering process. We can observe a peak in the location of the object. However, this accumulator contains significant noise. The noise is produced since rotation and scale change the value of the computed gradient. Thus, the line of votes is only approximated. Another problem

```
%Invariant R-Table

function T=RTableInv(entries,inputimage)

%image size
[rows,columns]=size(inputimage);

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

alfa=pi/4;
D=pi/entries;
s=0;           %number of entries in the table
t=0;
F=zeros(entries,1);  %number of entries in the row

%compute reference point
xr=0; yr=0; p=0;
for x=1:columns
   for y=1:rows
     if(M(y,x)~=0)
        xr=xr+x;
        yr=yr+y;
        p=p+1;
     end
   end
end
xr=round(xr/p);
yr=round(yr/p);

%for each edge point
for x=1:columns
  for y=1:rows
    if(M(y,x)~=0)
       %search for the second point
       x1=-1; y1=-1;
       phi=Ang(y,x);
       m=tan(phi-alfa);

       if(m>-1 & m<1)
           for i=3:columns
             c=x+i;
             j=round(m*(c-x)+y);
             if(j>0 & j<rows & c>0 & c<columns & M(j,c)~=0)
               x1=c ; y1=j;
               i= columns;
             end

             c=x-i;
             j=round(m*(c-x)+y);
             if(j>0 & j<rows & c>0 & c<columns & M(j,c)~=0)
               x1=i ; y1=j;
               i=columns;
             end
           end
```

```
    else
        for j=3:rows
          c=y+j;
          i=round(x+(c-y)/m);
          if(c>0 & c<rows & i>0 & i< columns & M(c,i)~=0)
            x1=i ; y1=c;
            i=rows;
          end
          c=y-j;
          i=round(x+(c-y)/m);
          if(c>0 & c<rows & i>0 & i< columns & M(c,i)~=0)
            x1=i ; y1=c;
            i= rows;
          end
        end
    end

    if( x1~=-1)
        %compute beta
          phi=tan(Ang(y,x));
          phj= tan(Ang(y1,x1));
          if((1+phi*phj)~=0)
            beta=atan((phi-phj)/(1+phi*phj));
          else
            beta=1.57;
          end

          %compute k
          if((x-xr)~=0)
            ph=atan((y-yr)/(x-xr));
          else
            ph=1.57;
          end
          k=ph-Ang(y,x);

          %insert in the table
          i=round((beta+(pi/2))/D);
          if(i==0) i=1; end;

          V=F(i)+1;

    if(V>s)
        s=s+1;
        T(:,s)=zeros(entries,1);
          end;

          T(i,V)=k;
          F(i)=F(i)+1;
        end

    end %if
  end % y
end % x
```

Code 5.12 Construction of the invariant R-table

```
%Invariant Generalized Hough Transform

function GHTInv(inputimage,RTable)

%image size
[rows,columns]=size(inputimage);

%table size
[rowsT,h,columnsT]=size(RTable);
D=pi/rowsT;

%edges
[M,Ang]=Edges(inputimage);
M=MaxSupr(M,Ang);

alfa=pi/4;

%accumulator
acc=zeros(rows,columns);

% for each edge point
for x=1:columns
  for y=1:rows
    if(M(y,x)~=0)
      % search for the second point
      x1=-1; y1=-1;
      phi=Ang(y,x);
      m=tan(phi-alfa);

      if(m>-1 & m<1)
        for i=3:columns
          c=x+i;
          j=round(m*(c-x)+y);
          if(j>0 & j<rows & c>0 & c<columns & M(j,c)~=0)
            x1=c ;y1=j;
            i= columns;
          end
          c=x-i;
          j=round(m*(c-x)+y);
          if(j>0 & j<rows & c>0 & c<columns & M(j,c)~=0)
            x1=c ;y1=j;
            i=columns;
          end
        end
      else
        for j=3:rows
          c=y+j;
          i=round(x+(c-y)/m);
          if(c>0 & c<rows & i>0 & i< columns & M(c,i)~=0)
            x1=i ;y1=c;
            i=rows;
          end
```

```
          c=y-j;
          i=round(x+(c-y)/m);
          if(c>0 & c<rows & i>0 & i< columns & M(c,i)~=0)
            x1=i ;y1=c;
            i=rows;
          end
        end
      end

   if( x1~=-1)
     %compute beta
          phi=tan(Ang(y,x));
          phj=tan(Ang(y1,x1));
          if((1+phi*phj)~=0)
            beta=atan((phi-phj)/(1+phi*phj));
          else
            beta=1.57;
          end

          i=round((beta+(pi/2))/D);
          if(i==0) i=1; end;

          %search for k
          for j=1:columnsT
            if(RTable(i,j)==0)
               j=columnsT; % no more entries
            else
               k=RTable(i,j);
               %lines of votes
               m=tan(k+Ang(y,x));
            if(m>-1 & m<1)
               for x0=1:columns
                 y0=round(y+m*(x0-x));
                 if(y0>0 & y0<rows)
                    acc(y0,x0)=acc(y0,x0)+1;
                 end
               end
             else
               for y0=1:rows
                 x0= round(x+(y0-y)/m);
                 if(x0>0 & x0<columns)
                   acc(y0,x0)=acc(y0,x0)+1;
                 end
               end
             end
           end
         end
       end
     end %if
   end % y
end % x
```

Code 5.13 Implementation of the invariant GHT

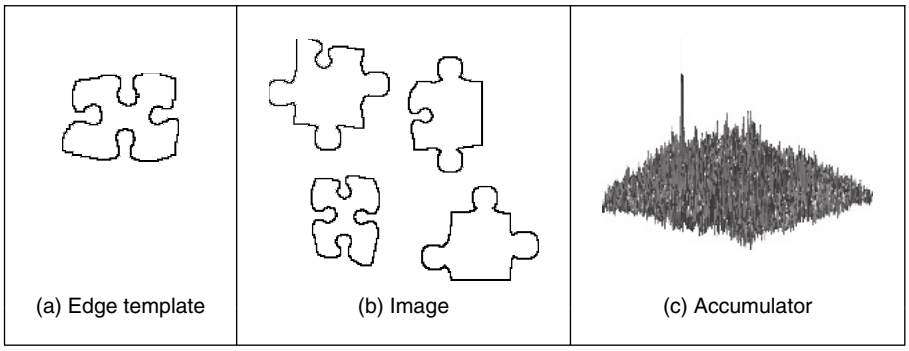


Figure 5.27 Applying the invariant GHT

is that pairs of points ω_i and ω_j might not be found in an image, thus the technique is more sensitive to occlusion and noise than the GHT.

5.6 Other extensions to the Hough transform

The motivation for extending the HT is clear: *keep* the *performance*, but *improve* the *speed*. There are other approaches to reduce the computational load of the HT. These approaches aim to improve speed and reduce memory, focusing on smaller regions of the accumulator space. These approaches have included the *fast HT* (Li and Lavin, 1986), which uses successively splits the accumulator space into quadrants and continues to study the quadrant with most evidence; the *adaptive HT* (Illingworth and Kittler, 1987), which uses a fixed accumulator size to focus iteratively on potential maxima in the accumulator space; the *randomized HT* (Xu et al., 1990) and the *probabilistic HT* (Kälviäinen et al., 1995), which use a random search of the accumulator space; and other pyramidal techniques. One main problem with techniques that do not search the full accumulator space, but a reduced version to save speed, is that the wrong shape can be extracted (Princen et al., 1992a), a problem known as *phantom shape location*. These approaches can also be used (with some variation) to improve speed of performance in template matching. There have been many approaches aimed to improve performance of the HT and the GHT.

There has been a comparative study on the GHT (including efficiency) (Kassim et al., 1999) and alternative approaches to the GHT include two *fuzzy* HTs: Philip (1991) and Sonka et al. (1994) include uncertainty of the perimeter points within a GHT structure, and Han et al. (1994) approximately fits a shape but requires application-specific specification of a fuzzy membership function. There have been two major reviews of the state of research in the HT (Illingworth and Kittler, 1988; Leavers, 1993), but they are rather dated now, and a textbook (Leavers, 1992) which cover many of these topics. The analytic approaches to improving the HT's performance use mathematical analysis to reduce size, and more importantly dimensionality, of the accumulator space. This concurrently improves speed. A review of HT-based techniques for circle extraction (Yuen et al., 1990) covered some of the most popular techniques available at the time.

As techniques move to analysing moving objects, there is the *velocity Hough transform* for detecting moving shapes (Nash et al., 1997). As in any HT, in the velocity HT a moving shape

needs a parameterization which includes the motion. For a circle moving with (linear) velocity we have points which are a function of time t as

$$\begin{aligned} x(t) &= c_x + v_x t + r\cos\theta \\ y(t) &= c_y + v_y t + r\sin\theta \end{aligned} \tag{5.102}$$

where c_x, c_y are the coordinates of the circle's centre, v_x, v_y describe the velocity along the x- and the y-axes, respectively, r is the circle's radius, and θ allows us to draw the locus of the circle at time t. We then construct a 5D accumulator array in terms of the unknown parameters c_x, c_y, v_x, v_y, r and vote for each image of the sequence (after edge detection and thresholding) in this accumulator array. By grouping the information across a sequence the technique was shown to be more reliable in occlusion than extracting a single circle for each frame and determining the track as the locus of centres of the extracted circles. This was extended to a technique for finding *moving lines*, as in the motion of the thigh in a model-based approach to recognizing people by the way they walk (gait biometrics) (Cunado et al., 2003). This is illustrated in Figure 5.28, which shows a walking subject on whom is superimposed a line showing the extracted position and orientation of the (moving) human thigh. It was also used in a generalized HT for *moving shapes* (Grant et al., 2002), which imposed a motion trajectory on the GHT extraction, and in a GHT which includes *deforming* moving shapes (Mowbray and Nixon, 2004).

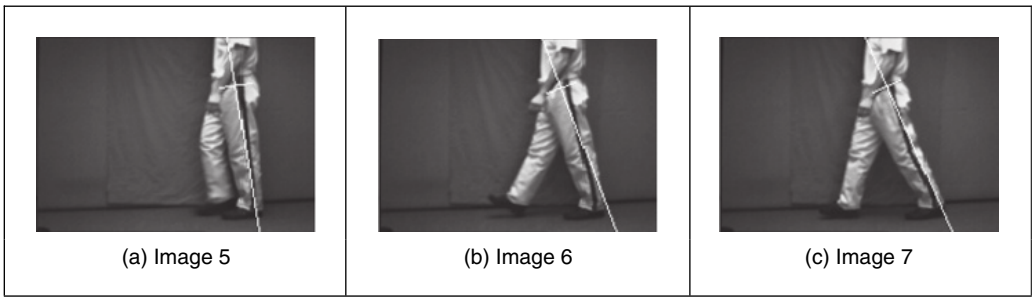


Figure 5.28 Detecting moving lines

5.7 Further reading

The majority of further reading in finding shapes concerns papers, many of which have already been referenced. An excellent survey of the techniques used for feature extraction (including template matching, deformable templates, etc.) can be found in Trier et al. (1996). Few of the textbooks devote much space to shape extraction, sometimes dismissing it in a couple of pages. This contrasts with the volume of research there has been in this area, and the HT finds increasing application as computational power continues to increase (and storage cost reduces). One text alone is dedicated to shape analysis (van Otterloo, 1991) and contains many discussions on symmetry. For implementation, Parker (1994) includes C code for template matching and for the HT for lines, but no more (the more recent version, Parker, 1996, omits it entirely). Other techniques use a similar evidence-gathering process to the HT. These techniques are referred to as geometric hashing and clustering techniques (Stockman, 1987; Lamdan et al., 1988).

In contrast to the HT, these techniques do not define an analytic mapping, but they gather evidence by grouping a set of features computed from the image and from the model.

5.8 References

Aguado, A. S., *Primitive Extraction via Gathering Evidence of Global Parameterised Models*, PhD Thesis, University of Southampton, 1996

Aguado, A. S., Montiel, E. and Nixon, M. S., On Using Directional Information for Parameter Space Decomposition in Ellipse Detection, *Pattern Recog*. **28**(3), pp. 369–381, 1996

Aguado, A. S., Nixon, M. S. and Montiel, M. E., Parameterizing Arbitrary Shapes via Fourier Descriptors for Evidence-Gathering Extraction, *Comput. Vision Image Understand*., **69**(2), pp. 202–221, 1998

Aguado, A. S., Montiel, E. and Nixon, M. S., On the Intimate Relationship Between the Principle of Duality and the Hough Transform, *Proc. R. Soc. A*, **456**, pp. 503–526, 2000

Aguado, A. S., Montiel, E. and Nixon, M. S., Bias Error Analysis of the Generalized Hough Transform, *J. Math. Imaging Vision*, **12**, pp. 25–42, 2000

Altman, J. and Reitbock, H. J. P., A Fast Correlation Method for Scale- and Translation-Invariant Pattern Recognition, *IEEE Trans. PAMI*, **6**(1), pp. 46–57

Ballard, D. H., Generalizing the Hough Transform to Find Arbitrary Shapes, *CVGIP*, **13**, pp. 111–122, 1981

Bracewell, R. N., *The Fourier Transform and its Applications*, 2nd edn, McGraw-Hill Book Co., Singapore, 1986

Bresenham, J. E., Algorithm for Computer Control of a Digital Plotter, *IBM Syst. J*., **4**(1), pp. 25–30, 1965

Bresenham, J. E., A Linear Algorithm for Incremental Digital Display of Circular Arcs, *Communs ACM*, **20**(2), pp. 750–752, 1977

Brown, C. M., Inherent Bias and Noise in the Hough Transform, *IEEE Trans. PAMI*, **5**, pp. 493–505, 1983

Casasent, D. and Psaltis, D., New Optical Transforms for Pattern Recognition, *Proc. IEEE*, **65**(1), pp. 77–83, 1977

Cunado, D., Nixon, M. S. and Carter, J. N., Automatic Extraction and Description of Human Gait Models for Recognition Purposes, *Comput. Vision Image Understand*., **90**(1), pp. 1–41, 2003

Deans, S. R., Hough Transform from the Radon Transform, *IEEE Trans. PAMI*, **13**, pp. 185–188, 1981

Duda, R. O. and Hart, P. E., Use of the Hough Transform to Detect Lines and Curves in Pictures, *Communs. ACM*, **15**, pp. 11–15, 1972

Gerig, G. and Klein, F., Fast Contour Identification Through Efficient Hough Transform and Simplified Interpretation Strategy, *Proc. 8th Int. Conf. Pattern Recog*, pp. 498–500, 1986

Grant, M. G., Nixon., M. S. and Lewis, P. H., Extracting Moving Shapes by Evidence Gathering, *Pattern Recog*., **35**, pp. 1099–1114, 2002

Grimson, W. E. L. and Huttenglocher, D. P., On the Sensitivity of the Hough Transform for Object Recognition, *IEEE Trans. PAMI*, **12**, pp. 255–275, 1990

Han, J. H., Koczy, L. T. and Poston, T., Fuzzy Hough Transform, *Pattern Recog. Lett*., **15**, pp. 649–659, 1994

Hecker, Y. C. and Bolle, R. M., On Geometric Hashing and the Generalized Hough Transform, *IEEE Trans. SMC*, **24**, pp. 1328–1338, 1994

Hough, P. V. C., Method and Means for Recognizing Complex Patterns, *US Patent 3969654*, 1962
Illingworth, J. and Kittler, J., The Adaptive Hough Transform, *IEEE Trans. PAMI*, **9**(5), pp. 690–697, 1987
Illingworth, J. and Kittler, J., A Survey of the Hough Transform, *CVGIP*, **48**, pp. 87–116, 1988
Kälviäinen, H., Hirvonen, P., Xu, L. and Oja, E., Probabilistic and Non-Probabilistic Hough Transforms: Overview and Comparisons, *Image Vision Comput*., **13**(4), May, 239–252, 1995
Kassim, A. A., Tan, T. and Tan K. H., A Comparative Study of Efficient Generalized Hough Transform Techniques, *Image Vision Comput*., **17**(10), pp. 737–748, 1999
Kimme, C., Ballard, D. and Sklansky, J., Finding Circles by an Array of Accumulators, *Communs ACM*, **18**(2), pp. 120–1222, 1975
Kiryati, N. and Bruckstein, A. M., Antialiasing the Hough Transform, *CVGIP: Graphical Models Image Process*., **53**, pp. 213–222, 1991
Lamdan, Y., Schawatz, J. and Wolfon, H., Object Recognition by Affine Invariant Matching, *Proc. IEEE Conf. Comput. Vision Pattern Recog*., pp. 335–344, 1988
Leavers, V., *Shape Detection in Computer Vision using the Hough Transform*, Springer, London, 1992
Leavers, V., Which Hough Transform? *CVGIP: Image Understand*., **58**, pp. 250–264, 1993
Li, H. and Lavin, M. A., Fast Hough Transform: A Hierarchical Approach, *CVGIP*, **36**, pp. 139–161, 1986
Merlin, P. M. and Farber, D. J., A Parallel Mechanism for Detecting Curves in Pictures, *IEEE Trans. Comput*., **24**, pp. 96–98, 1975
Mowbray, S. D. and Nixon, M. S., Extraction and Recognition of Periodically Deforming Objects by Continuous, Spatio-temporal Shape Description, *Proc. CVPR 2004*, **2**, pp. 895–901, 2004
Nash, J. M., Carter, J. N. and Nixon, M. S., Dynamic Feature Extraction via the Velocity Hough Transform, *Pattern Recog. Lett*., **18**(10), pp. 1035–1047, 1997
O'Gorman, F. and Clowes, M. B., Finding Picture Edges Through Collinearity of Feature Points, *IEEE Trans. Comput*., **25**(4), pp. 449–456, 1976
Parker, J. R., *Practical Computer Vision Using C*, Wiley & Sons, New York, 1994
Parker, J. R., *Algorithms for Image Processing and Computer Vision*, Wiley & Sons, New York, 1996
Philip, K. P., *Automatic Detection of Myocardial Contours in Cine Computed Tomographic Images*, PhD Thesis, University of Iowa, 1991
Princen, J., Yuen, H. K., Illingworth, J. and Kittler, J., Properties of the Adaptive Hough Transform, *Proc. 6th Scandinavian Conf. Image Analysis*, Oulu, Finland, June 1992a
Princen, J., Illingworth, J. and Kittler, J., A Formal Definition of the Hough Transform: Properties and Relationships, *J. Math. Imaging Vision*, **1**, pp. 153–168, 1992b
Rosenfeld, A., *Picture Processing by Computer*, Academic Press, London, 1969
Sklansky, J., On the Hough Technique for Curve Detection, *IEEE Trans. Comput*., **27**, pp. 923–926, 1978
Sonka, M., Hllavac, V. and Boyle, R, *Image Processing, Analysis and Computer Vision*, Chapman Hall, London, 1994
Stockman, G. C. and Agrawala, A. K., Equivalence of Hough Curve Detection to Template Matching, *Communs ACM*, **20**, pp. 820–822, 1977
Stockman, G., Object Recognition and Localization via Pose Clustering, *CVGIP*, **40**, pp. 361–387, 1987
Traver V. J. and Pla, F., The Log-Polar Image Representation in Pattern Recognition Tasks, *Lecture Notes Comput. Sci*., **2652**, pp. 1032–1040, 2003

Trier, O. D., Jain, A. K. and Taxt, T., Feature Extraction Methods for Character Recognition – A Survey, *Pattern Recog*., **29**(4), pp. 641–662, 1996

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

Yuen, H. K., Princen, J., Illingworth, J. and Kittler, J., Comparative Study of Hough Transform Methods for Circle Finding, *Image Vision Comput*., **8**(1), pp 71–77, 1990

Xu, L., Oja, E. and Kultanen, P., A New Curve Detection Method: Randomized Hough Transform, *Pattern Recog. Lett*., **11**, pp. 331–338, 1990

Zokai, S. and Wolberg, G., Image Registration using Log-Polar Mappings for Recovery of Large-Scale Similarity and Projective Transformations, *IEEE Trans. Image Process*., **14**, pp. 1422–1434, 2005