4

Fundamentals of digital image processing

Although the distinction between digital image processing and digital image analysis is not immediately obvious, it is an extremely important one. Image processing can be thought of as a transformation which takes an image into an image, i.e. it starts with an image and produces a modified (enhanced) image. On the other hand, digital image analysis is a transformation of an image into something other than an image, i.e. it produces some information representing a description or a decision. However, digital image analysis techniques are usually applied to previously processed (enhanced) images. Since the analysis of enhanced images is accomplished quickly and easily by human observers, it is often erroneously assumed that the analysis phase is easy, if not trivial. Although image processing techniques often provide more pleasing or more visually accessible images when viewed by a human observer, and the human is able to detect salient features without difficulty, it is essential to realize that the interpretation of such emergent features is simple only in the context of the powerful perceptual abilities of the human visual system.

For machine vision systems, the sole purpose of image processing is to produce images which make the subsequent analysis more simple and more reliable. We are not at all interested in how 'well' the image looks. In particular, the image processing phase should facilitate the extraction of information, it should compensate for non-uniform illumination, and, possibly, re-adjust the image to compensate for distortions introduced by the imaging system.

Historically, digital image processing had two main thrusts to its development. One was the natural extension of one-dimensional (temporal) digital signal processing to two (spatial) dimensions. Consequently, two-dimensional signal processing was approached from a mathematical basis, allowing a great deal of rigorous manipulation to be performed, using classical linear system theory. The second, more heuristic, thrust considers digital images as a set of discrete sample points, performing arithmetic operations on the individual points. This contrasts with the signal processing approach which treats images as a discrete representation of a continuous two-dimensional function. We will continue our discussion of

digital image processing from the second viewpoint, addressing some simple but useful techniques, and drawing when appropriate from the signal processing approach.

There are, broadly speaking, three distinct classes of operations: point operations, neighbourhood operations, and geometric operations. We mentioned in the preceding chapter that there is a trend to perform as much image processing as possible in hardware, particularly in advanced frame-grabbers or in their sister boards which have access to the image data via a dedicated video bus. Point and neighbourhood operations are typically the type of operations that are performed by these frame-grabber sister boards, while very few systems offer any extensive operations for geometric image processing.

4.1 Point operations

A point operation is an operation in which each pixel in the output image is a function of the grey-level of the pixel at the corresponding position in the input image, and only of that pixel. Point operations are also referred to as grey-scale manipulation operations. They cannot alter the spatial relationships of the image. Typical uses of point operations include photometric decalibration, to remove the effects of spatial variations in the sensitivity of a camera system; contrast stretching

101	100	103	105	107	105	103	110
110	140	120	122	130	130	121	120
134	134	135	131	137	138	120	121
132	132	132	133	133	150	160	155
134	140	140	135	140	156	160	174
130	138	139	150	169	175	170	165
126	133	138	149	163	169	180	185
130	140	150	169	178	185	190	200

Figure 4.1 Digital image.

(e.g. if a feature or object occupies a relatively small section of the total grey-scale image, these point operations can manipulate the image so that it occupies the entire range); and thresholding, in which all pixels having grey-levels in specified ranges in the input image are assigned a single specific grey-level in the output image. As we shall see, these operations are most effectively accomplished using the hardware input look-up tables (LUTs) which are provided by most frame-grabbers.

4.1.1 Contrast stretching

Consider the contrived digital image shown in Figure 4.1. If we look at the distribution of the grey-levels in this image, we find that there are grey-levels in the ranges from 100 to 200. Obviously the complete range is not being used and the contrast of this image would be quite poor. The contrived grey-level histogram shown in Figure 4.2 illustrates graphically this poor use of the available grey-scale. We wish to enhance the contrast so that all grey-levels of the grey-scale are utilized. This contrast stretching is achieved by first shifting all the values so that the actual pixel grey-level range begins at 0, i.e. add to every pixel the difference between the final low value (0) and the initial low value (100): $0 - 100 = -100$. The effect of this is shown in Figure 4.3.

Next, we scale everything, reassigning values in the range 0–100 to the range 0–255, i.e. we scale by a factor = $(255 - 0)/(100 - 0) = 2.55$; see Figure 4.4.

Thus, in this instance, to stretch the contrast so that all grey-levels in the grey-scale are utilized, one must simply apply the following operation:

```
new pixel value:=(old pixel value-100)*2.55
```

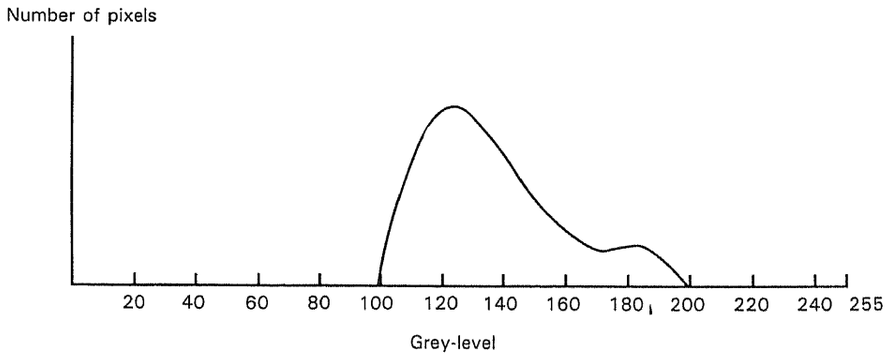

Figure 4.2 Grey-level histogram.

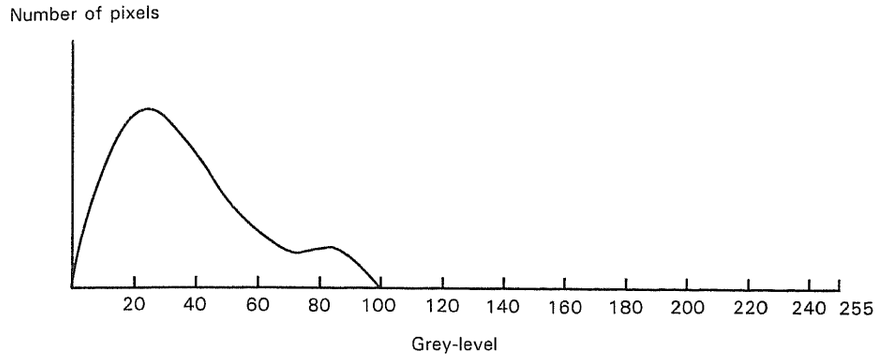

Figure 4.3 Shifted grey-level histogram.

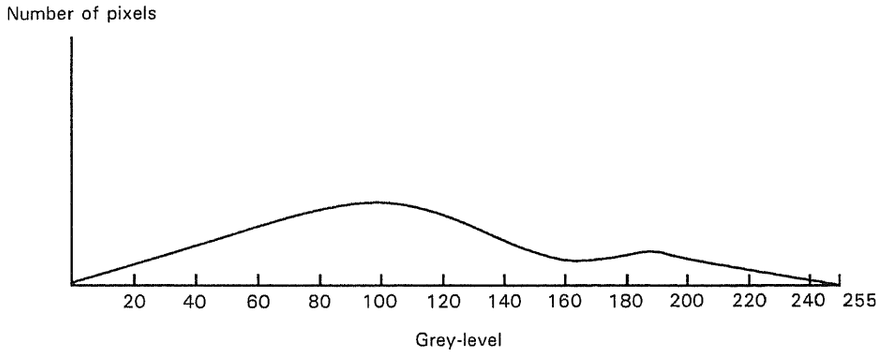

Figure 4.4 Stretched grey-level histogram.

By way of example, let us consider the application of this stretching to any pixel having either the lowest grey-level (100) in the original image or the highest grey-level (200) in the original image:

```
If old pixel value=100
  new pixel value:=(100-100)*2.55
                  =0

If old pixel value=200
  new pixel value:=(200-100)*2.55
                  =255
```

We can express the algorithm a little more formally in pseudo-code as follows:

```
/* contrast stretching in a 512x512 pixel image */

/* HIGH and LOW are assumed to be the highest and lowest
grey-levels, respectively, in the unstretched image */

scale_factor:=255 / (HIGH-LOW)

FOR i:=1 TO 512 DO
 FOR j:=1 to 512 DO

  IF image[i,j] < LOW
   THEN
    image[i,j]:=0
   ELSE
    IF image[i,j] > HIGH
     THEN
      image[i,j]:=255
     ELSE

      /* scale */

      image[i,j]:=(image[i,j] - LOW)
      *scale_factor
```

while the LUT formulation might be written as:

```
/* contrast stretching using LUT */

scale_factor:=255 / (HIGH-LOW)

/* initialise LUT */

FOR i:=0 TO LOW-1 DO
 LUT[i]:=0

FOR I:=LOW TO HIGH DO
 LUT[i]:=(i-LOW)*scale-factor

FOR i:=HIGH+1 TO 255 DO
 LUT[i]:=255

/* stretch using LUT */

FOR i:=1 TO 512 DO
 FOR j:=1 to 512 DO

  image[i,j]:=LUT[ image[i,j] ]
```


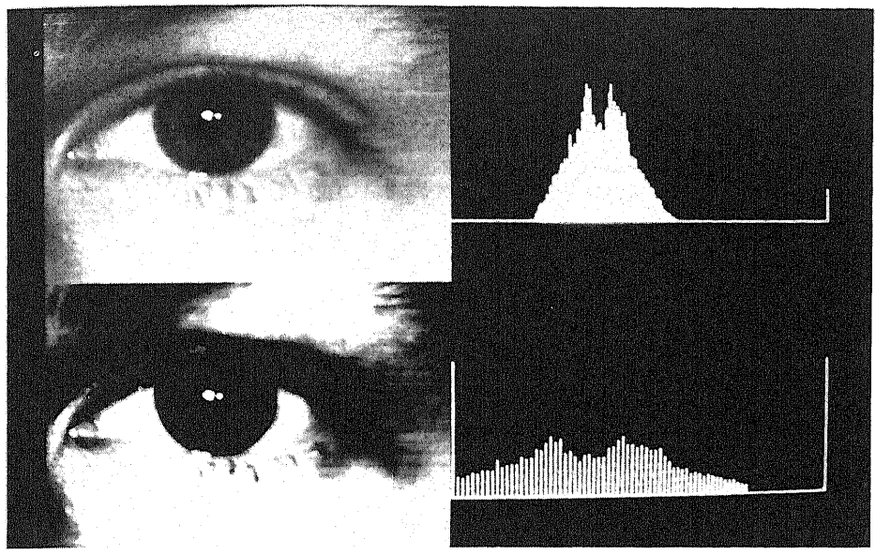

Figure 4.5 Contrast stretching. The grey-level histogram of the original image (top-left) is shown at the top-right; the stretched image with its histogram is shown below.

As an example, Figure 4.5 shows the results of applying this contrast stretching algorithm to a real digital image. The associated grey-level histograms are displayed to the right of the image.

There are many other approaches which can be used for contrast enhancement, e.g. histogram equalization is a technique which computes the histogram of the grey-level distribution in the image and reassigns grey-levels to pixels in an effort to generate a histogram where there are equally many pixels at every grey-level, thereby producing an image with a flat or uniform grey-level histogram.

4.1.2 Thresholding

Some scenes, e.g. those of bare printed circuit boards (PCBs), can be very simple, from a visual point of view, in that they comprise just one or two types of objects: in this case the solder tracks and the circuit board. Suppose we wish to process an image of such a scene so that it exhibits extremely high contrast and so that the solder tracks and circuit board are very easily distinguishable. If we stipulate that there are just two allowable grey-levels, black and white, then this would certainly result in the requisite contrast. The problem is to convert a grey-scale image (typically with 256 grey-levels) into an image with just two levels of grey. Consider again a contrived grey-scale histogram. In Figure 4.6 we can see that all the image

pixels representing the circuit board *probably* have grey-levels in the range 0–160 while the solder tracks *probably* have grey-levels in the range 160–255. If we assign all the pixels in the range 0–160 to be black and all pixels in the range 160–255 to be white we will give effect to this extreme contrast stretching. Happily, we also now have an explicit labelling in our image: the circuit board pixels are labelled (identified) with a grey-level of 0 and the solder is labelled with a grey-level of 255. In effect we have nominated the grey-level 160 as the *threshold* and the reassignment of pixel values is known as *thresholding*. The effect of thresholding on the grey-scale histogram can be seen in Figure 4.7; the pseudo-code that follows summarizes the thresholding algorithm:

```
/* Threshold an image in place (i.e. without an */
/* explicit destination_image) */

FOR i:=1 TO 512 DO
 FOR j:=1 to 512 DO

  IF image[i,j] > threshold
   THEN
    image[i,j]:=255
   ELSE
    image[i,j]:=0
```

The algorithm may also be formulated using a LUT:

```
/* Threshold an image in place (i.e. without an */
/* explicit destination_image) using a LUT */

/* initialise LUT */

FOR i:=0 TO threshold DO
 LUT[i]:=0

FOR i:=threshold+1 TO 255 DO
 LUT[i]:=255

/* threshold using LUT */

FOR i:=1 TO 512 DO
 FOR j:=1 to 512 DO

  image[i,j]:=LUT[ image[i,j] ]
```

Note that the subject of thresholding is an important one to which we will return and will discuss in greater detail in Chapter 5, specifically with regard to the

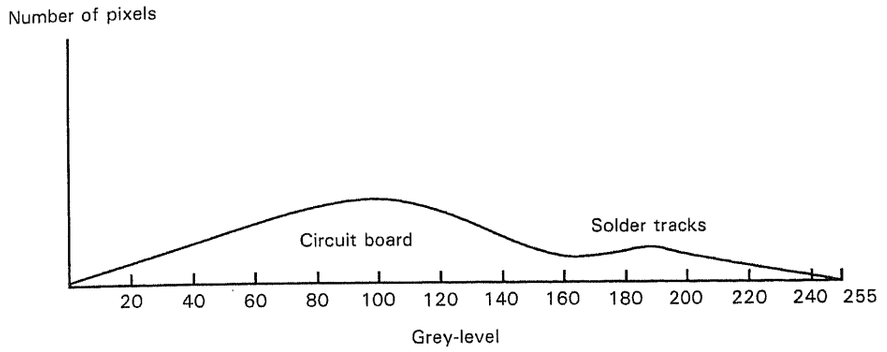

Figure 4.6 Grey-level histogram of a PCB.

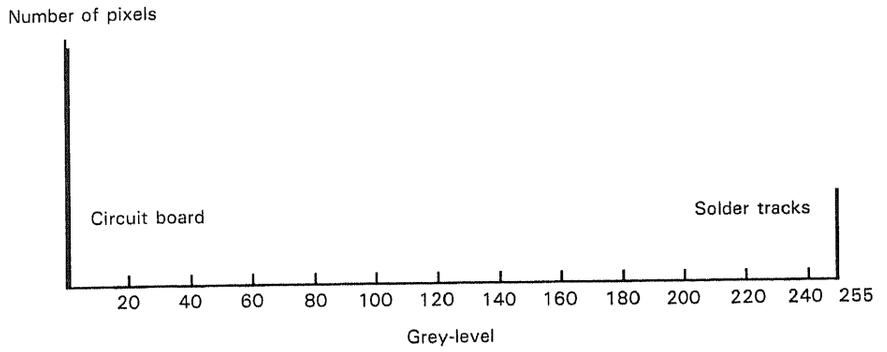

Figure 4.7 Grey-level histogram of a PCB after thresholding.

selection of the threshold point and to the incorporation of contextual information when applying the threshold.

An example of digital image thresholding is shown in Figure 4.8.

4.1.3 Noise suppression by image addition

If it is possible to obtain multiple images of a scene, each taken at a different time, and if the scene contains no moving objects, then the noise in the image can be reduced by averaging these images. The rationale is quite simple: in the averaging process the constant part of the image (that which is due to light reflected from stationary objects) is unchanged while the noise, which will in general change from image to image, will accumulate more slowly. The assumptions inherent in this

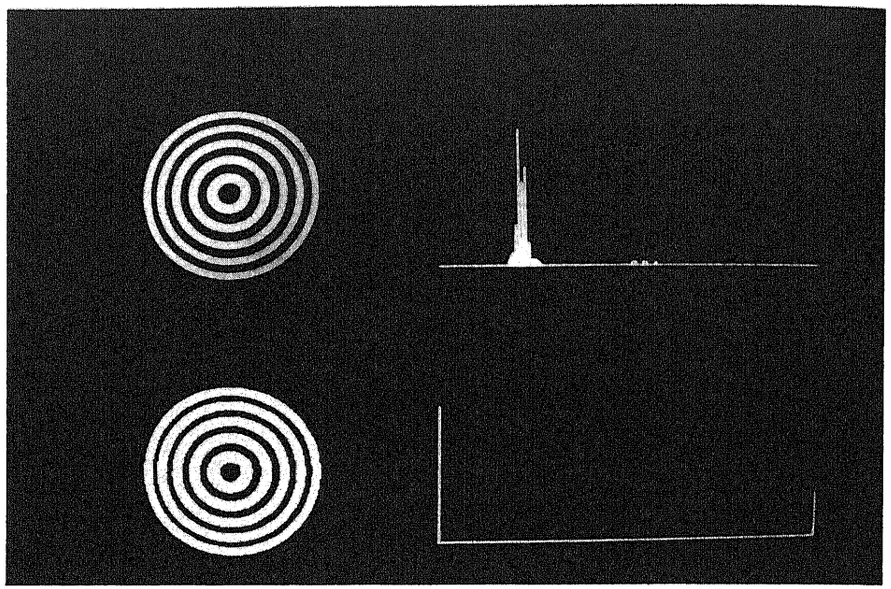

Figure 4.8 Thresholding. A grey-scale image (top-left) with its histogram (top-right) is thresholded at a grey-scale value of 128 (bottom-left); the resultant histogram is shown in the bottom-left quadrant.

approach are as follows:

1. The noise in each image is indeed uncorrelated.
2. The noise has a zero mean value, i.e. it averages out to an image with a grey-level of zero which contributes nothing to the real image.

With these assumptions, it is possible to show that averaging N images increases the signal-to-noise ratio by $\sqrt{N}$.

Many commercial framestores incorporate facilities to accomplish this averaging in real time (i.e. as the image is acquired) and, as such, it is worth bearing this technique in mind as it involves very little computational overhead. However, you should also bear in mind the assumptions upon which this technique is based; not all noise has these desirable properties.

4.1.4 Background subtraction

Digital image subtraction refers to an operation in which the pixel values of two images are subtracted on a point by point basis. It can be useful for the subtraction of a *known* pattern (or image) of superimposed noise or, indeed, for motion detection: stationary objects cancel each other out while moving objects are

highlighted when two images of the same dynamic scene, which have been taken at slightly different times, are subtracted. This process of subtraction of an uninteresting background image from a foreground image containing information of interest is referred to, not surprisingly, as 'background subtraction'.

Photometric decalibration is one of the most important applications of background subtraction. In some circumstances, camera systems exhibit non-uniform response to light across the field of view. Quite often, this is caused by the lens and is manifested as an image centre which is somewhat brighter than the periphery. This can cause severe problems if one is using thresholding techniques to isolate objects. Since the grey-level which is assumed to correspond to the object may change (from point to point) depending on the position in the image, one solution (apart from replacing the lens) is to model this non-uniform response by, e.g. taking an image of a uniformly shaded surface, identifying the minimum grey-level of this image and subtracting this value from each pixel to generate an image which represents the effective response of the camera. Images which are subsequently acquired can then be processed by subtracting this calibration image from them.

4.2 Neighbourhood operations

A neighbourhood operation generates an 'output' pixel on the basis of the pixel at the corresponding position in the input image *and on the basis of its neighbouring pixels*. The size of the neighbourhood may vary: several techniques use 3×3 or 5×5 neighbourhoods centred at the input pixel, but many of the more advanced and useful techniques now use neighbourhoods which may be as large as 63×63 pixels. The neighbourhood operations are often referred to as 'filtering operations'. This is particularly true if they involve the *convolution* of an image with a filter kernel or mask. Such filtering often addresses the removal (or suppression) of noise or the enhancement of edges, and is most effectively accomplished using convolver (or filtering) hardware, available as sister boards for most frame-grabbers.

Other neighbourhood operations are concerned with modifying the image, not by filtering it in the strict sense, but by applying some logical test based on, e.g. the presence or absence of object pixels in the local neighbourhood surrounding the pixel in question. Object thinning, or skeletonizing, is a typical example of this type of operation, as are the related operations of erosion and dilation, which, respectively, seek to contract and enlarge an object in an orderly manner.

Since convolution is such an important part of digital image processing, we will discuss it in detail before proceeding.

4.2.1 Convolution

The convolution operation is much used in digital image processing and, though it can appear very inaccessible when presented in a formal manner for real continuous

functions (signals and images), it is quite a simple operation when considered in a discrete domain.

Before discussing discrete convolution, let us consider why one is interested in the operation in the first place. The two-dimensional convolution integral, which is given by the equation:

$$g(i,j) = f * h = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(i-m, j-n)\ h(m,n)\ \mathrm{d}m\ \mathrm{d}n$$

embodies the fact that the output g of a shift-invariant linear system (i.e. most optical electronic and optoelectronic systems, to a good approximation, and most filtering techniques) is given by the 'convolution' or application of the input signal f with a function h which is characteristic of the system. Thus, the form of g depends on the input f (obviously) and on the form of the system h through which it is being passed. The relationship is given by the convolution integral. The function h is normally referred to as the filter, since it dictates what elements of the input image are allowed to pass through to the output image. By choosing an appropriate filter, we can enhance certain aspects of the output and attenuate

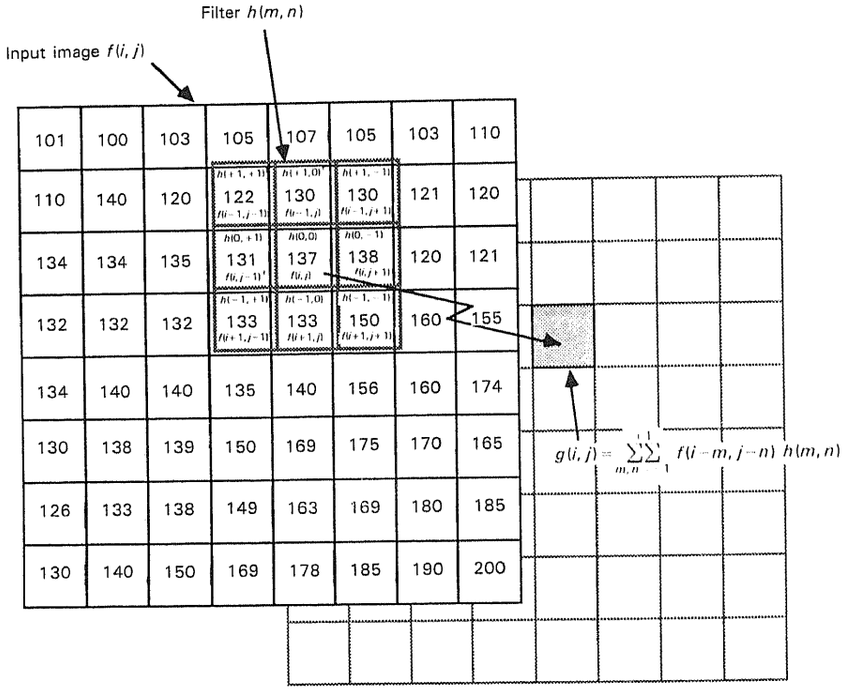

Figure 4.9. Convolution.

others. A particular filter h is often referred to as a 'filter kernel'. Attempts to form a conceptual model of this convolution operation in one's mind are often fruitless. However, it becomes a little easier if we transfer now to the discrete domain of digital images and replace the integral with the sigma (summation) operator. Now the convolution operation is given by:

$$g(i,j) = f * h = \sum_m \sum_n f(i-m, j-n)\ h(m,n)$$

The summation is taken only over the area where f and h overlap. This multiplication and summation is illustrated graphically in Figure 4.9. Here, a filter kernel h is a 3×3 pixel image, with the origin $h(0,0)$ at the centre, representing a mask of nine distinct weights: $h(-1,-1) \ldots h(+1,+1)$; see Figure 4.10. The kernel or mask is superimposed on the input image, the input image pixel values are multiplied by the corresponding weight, the nine results are summed, and the final value of the summation is the value of the output image pixel at a position corresponding to the position of the centre element of the kernel. Note that the convolution formula requires that the mask h be first rotated by 180° since, e.g., $f(i-1, j-1)$ must be multiplied by $h(1,1)$, $f(i-1, j)$ must be multiplied by $h(1,0), \ldots$, and $f(i+1, j+1)$ must be multiplied by $h(-1,-1)$. Quite often, the rotation by 180° is omitted if the mask is symmetric.

The algorithm to generate g is given by the following pseudo-code:

```
FOR i:=(low_limit_of_i + 1) TO (high_limit_of_i - 1) DO
 FOR j:=(low_limit_of_j + 1) TO (high_limit_of_j - 1) DO

  /* for each feasible point in the image
   ... form the convolution */

  temp:=0
  FOR m:= -1 TO +1 DO
   FOR n:=-1 TO +1 DO
    temp:=temp+F[i-m, j-n] * H[m, n]

  G[i,j]:=temp
```

Before returning to the discussion of neighbourhood operators, a short note about the relationship of the convolution operator to classical two-dimensional digital signal processing is appropriate. This can be safely skipped without any loss of continuity.

All image systems introduce some amount of distortion into an image, for example image blur due to camera shake. Since optical and electrical systems are (to an approximation) linear, the imaging system may also be assumed to be linear. This implies that, in the language of linear system theory, it has some transfer function $H(\omega_x, \omega_y)$. This is the frequency domain equivalent of the system impulse response $h(x,y)$ in the spatial domain; $h(x,y)$ typifies how the system would

$h(-1,-1)$	$h(-1,0)$	$h(-1,+1)$
$h(0,-1)$	$h(0,0)$	$h(0,+1)$
$h(+1,-1)$	$h(+1,0)$	$h(+1,+1)$

Figure 4.10 3×32 convolution filter h.

respond if a single unit spike were input to it. Thus, the transfer function describes how some input image is transformed into some output image. Since the output of this system is a distortion of the correct image we would expect that if we could model the system, i.e. define its transfer function, compute the inverse, and apply it to the distorted image, we would arrive at a much better approximation to the original. This is accomplished by convolving the inverse impulse response with the image: this operation is referred to as a 'deconvolution'. Alternatively, the image may be transformed to the frequency domain using the digital Fourier transform, multiplied by the inverse of the transfer function, and transformed back to the spatial domain by use of the inverse Fourier transform.

4.2.2 Noise suppression

If we define noise as any unwanted contamination of an image then the mechanism by which noise is removed depends on the assumption we make regarding the form of this unwanted contamination. One of the most common (and realistic) assumptions made about noise is that it has a high spatial frequency (refer back to Section 3.1.1). In this case, it is often adequate to apply a low-pass spatial filter which will attenuate the higher spatial frequencies and allow the low spatial frequency component to pass through to the resultant destination image. Of course if the image itself exhibits high spatial frequencies then it will be somewhat degraded after filtering.

These low-pass filters can be implemented by convolving the image with some simple mask; the mask values constitute the weighting factors which will be applied to the corresponding image point when the convolution is being performed. For example, each of the mask values might be equally weighted, in which case the operation we are performing is simply the evaluation of the local mean of the image in the vicinity of the mask.

Figure 4.11 shows this local neighbourhood average mask and Figure 4.12 illustrates the application of the mask to part of an image. Referring to Figure 4.12, we can see that the result of this filtering, i.e. the value of the output pixel which would be placed in the output image at the same position as the input pixel corresponding to the centre position of the mask, is:

$$101*1/9 + 100*1/9 + 103*1/9 + 110*1/9 + 140*1/9 + 120*1/9 + 134*1/9 + 134*1/9 + 135*1/9$$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Figure 4.11 Local average mask.

101 * 1/9	100 * 1/9	103 * 1/9	105	107	105	103	110
110 * 1/9	140 * 1/9	120 * 1/9	122	130	130	121	120
134 * 1/9	134 * 1/9	135 * 1/9	131	137	138	120	121
132	132	132	133	133	150	160	155
134	140	140	135	140	156	160	174
130	138	139	150	169	175	170	165
126	133	138	149	163	169	180	185
130	140	150	169	178	185	190	200

Figure 4.12 Image smoothing using local average mask.

which is equivalent to:

$$1/9*[101+100+103 \\ +110+140+120 \\ +134+134+135]$$

which is equal to 121.

Thus, the central point becomes 121 instead of 140 and the image will appear much smoother. This averaging or smoothing is, of course, applied at all points of the image.

Occasionally, it may be more useful to apply this smoothing subject to some condition, e.g. the centre pixel is only assigned if the difference between the average value and the original pixel value is greater than a previously set threshold. This goes some way towards removing noise without smoothing out too much of the detail in the original image.

The algorithm may be expressed in pseudo-code as follows:

```
/* For a 512×512 image, indexed from 0 to 511 */

FOR i:=1 TO 510 DO
 FOR j:=1 TO 510 DO

  /* for each feasible point in the image: compute
   average */

  average:=(source_image[i-1, j-1]    +
            source_image[i-1, j]      +
            source_image[i-1, j+1]    +
            source_image[i, j-1]      +
            source_image[i, j]        +
            source_image[i, j+1]      +
            source_image[i+1, j-1]    +
            source_image[i+1, j]      +
            source_image[i+1, j+1]    ) / 9

  /* destination image assumes the average value */

  destination_image[i,j]:=average
```

There are other noise-suppression techniques, of course. For example, median filtering is a noise-reducing technique whereby a pixel is assigned the value of the median of pixel values in some local neighbourhood. The size of the neighbourhood is arbitrary, but neighbourhoods in excess of 3×3 or 5×5 may be impractical from a computational point of view since the evaluation of the median requires that the image pixel values be first sorted. In general, the median filter is superior to the mean filter in that image blurring is minimized. Unfortunately, it is computationally

complex and is not easily effected in hardware and thus tends not to be used much in machine vision.

Gaussian smoothing, whereby the image is convolved with a Gaussian function, is perhaps one of the most commonly used smoothing techniques in advanced computer vision since it possesses several useful properties. Unfortunately, it is not yet widely used in industrial machine vision because the sizes of the masks are very large and the processing is consequently computationally intensive. It should be noted, however, that some vendors do offer dedicated image processing boards for Gaussian filtering and, if such hardware is available, this type of smoothing should be considered. The Gaussian function $G(x, y)$ is defined by:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp[-(x^2+y^2)/2\sigma^2]$$

where σ defines the effective spread of the function: Gaussian functions with a small value for σ are narrow, whereas those with a large value for σ are broad. Figure 4.13 illustrates the shape of a Gaussian function with $\sigma = 1.5, 3.0$, and 6.0 pixels respectively. The mask weights are included for reference. Note that, since the Gaussian function is defined over an infinite support, i.e. it has non-zero (but very small) values at $x, y = \pm\infty$, we must decide at what point to truncate the function. Normally, one chooses the quantization resolution of the function by deciding on the integer number which will represent the maximum amplitude at the centre point (e.g. 1000), and then one chooses the mask size which includes all non-zero values. Typically, a mask size of 23×23 pixels would be required to represent a two-dimensional Gaussian function with a value of 3.0 pixels for σ. Fortunately, there is no need to use two-dimensional Gaussian functions since the convolution of a two-dimensional Gaussian can be effected by two convolutions with one-dimensional Gaussian functions. Specifically:

$$G(x, y) * I(x, y) = G(x) * \{G(y) * I(x, y)\}$$

Thus, the image is first convolved with a 'vertical' Gaussian and then the resulting image is convolved with a 'horizontal' Gaussian.

Why is the Gaussian such a popular smoothing function? The reason is quite straightforward. While the primary purpose of a smoothing function is to reduce noise, it is often desirable to be able to choose the resolution at which intensity changes are manifested in the image, i.e. to choose the level of detail which is retained in the image. For example, an image which has been smoothed just a little (small σ) will retain a significant amount of detail, while one which has been smoothed a great deal (large σ) will retain only the gross structure (you can accomplish the same thing yourself by squinting). In more formal terms, we wish to sharply delimit the spatial frequencies (see Section 3.1.1) which are present in the image, i.e. to localize the spatial frequency bandwidth of the image. On the other hand, we also need to ensure that the smoothing function does not distort the image excessively by smearing the features. To allow for this, we need to ensure that

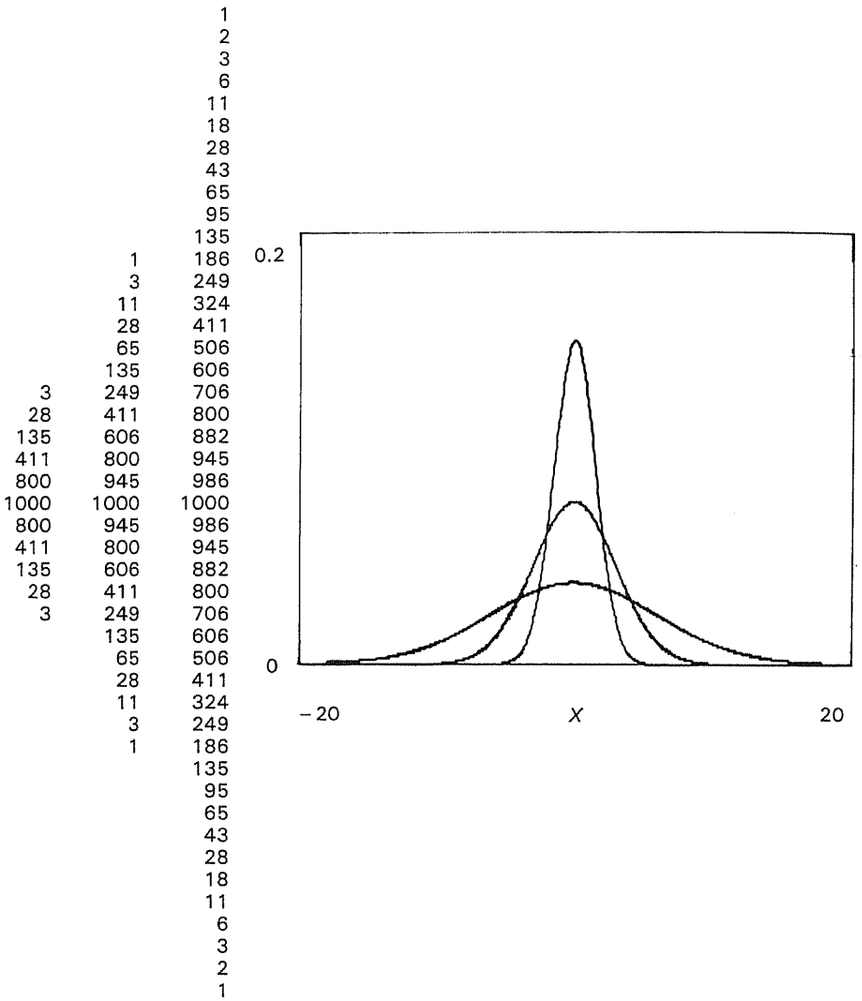

Figure 4.13 The Gaussian function for three values of σ (1.5, 3.0, and 6.0) together with their corresponding discrete one-dimensional masks; note that the result of convolution with these masks should be normalized by dividing by the sum of the mask weights.

the function has a limited support in space. The Gaussian function optimizes the trade-off between these two conflicting requirements.

4.2.3 Thinning, erosion, and dilation

Thinning is an iterative neighbourhood operation which generates a skeletal representation of an object. It assumes, of course, that you know exactly what constitutes the object in the image and what constitutes the background (i.e. everything which is not part of the object). Such an image is said to have been *segmented* into its component parts: the topic of segmentation is extremely important and will be discussed in detail in the next chapter.

The skeleton of an object may be thought of as a generalized axis of symmetry of the object and hence it is a suitable representation for objects which display obvious axial symmetry. The medial axis transform (MAT) proposed by Blum is one of the earliest and most widely studied techniques for generating the skeleton. More recently, Brady has introduced the related (but extended) concept of smoothed local symmetries (SLS) which we will discuss in the last section of Chapter 7.

The skeleton is frequently used as a shape descriptor which exhibits three topological properties: connectedness (one object generates one skeleton); invariance to scaling and rotation; and information preservation in the sense that the object can be reconstructed from the medial axis. The concept of thinning a binary image – an image comprising just two grey-levels: black and white – of an object is related to such medial axis transformations in that it generates a representation of an *approximate* axis of symmetry of a shape by successive deletion of pixels from the boundary of the object. In general, this thinned representation is not formally related to the original object shape and it is not possible to reconstruct the original boundary from the object.

Thinning can be viewed as a logical neighbourhood operation where object pixels are removed from an image. Obviously, the removal must be constrained somewhat so that we have a set of conditions for pixel removal. *The first restriction is that the pixel must lie on the border of the object. This implies that it has at least one 4-connected neighbouring pixel which is a background pixel.* The removal of pixels from all borders simultaneously would cause difficulties: for example, an object two pixels thick will vanish if all border pixels are removed simultaneously. A solution to this is to remove pixels of one border orientation only on each pass of the image by the thinning operator. Opposite border orientations are used alternately to ensure that the resultant skeleton is as close to the medial axis as possible.

The second restriction is that the deletion of a pixel should not destroy the object's connectedness, i.e. the number of skeletons after thinning should be the same as the number of objects in the image before thinning. This problem depends on the manner in which each pixel in the object is connected to every other pixel. A pixel can be considered to be connected to, and a component of, an object if it has

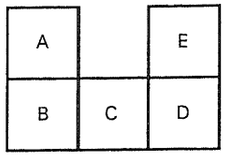

Figure 4.14 A critically connected object.

a grey-level of 255 and at least one adjacent object pixel (note that we are assuming object pixels have a grey-level of 255 – white – and background pixels have a grey-level of 0 – black). Consider now the 5 pixel object shown in Figure 4.14. The pixel C 'connects' the two object segments AB and ED, that is, if C were removed then this would break the object in two; this pixel is 'critically connected'. Obviously, this property may occur in many more cases than this, and critical-connectivity may be characterized as follows:

> Given a pixel, labelled 9 and its eight adjacent neighbours, labelled 0–7 (see Figure 3.6), and assume that writing the pixel number (e.g. 7) indicates presence, i.e. it is an object pixel, whereas writing it with an overbar (e.g. $\bar{7}$) indicates absence, i.e. it is a background pixel. Assume, also, normal Boolean logic sign conventions (+ indicates logical OR, and . indicates logical AND). Then pixel 8 is critically connected if the following expression is true.

$$
\begin{aligned}
8.\{&[(1+2+3).(5+6+7).\bar{4}.\bar{0}] \\
&+ [(1+0+7).(3+4+5).\bar{2}.\bar{6}] \\
&+ [3.(5+6+7+0+1).\bar{2}.\bar{4}] \\
&+ [1.(3+4+5+6+7).\bar{2}.\bar{0}] \\
&+ [7.(1+2+3+4+5).\bar{0}.\bar{6}] \\
&+ [5.(7+0+1+2+3).\bar{4}.\bar{6}]\}
\end{aligned}
$$

Figure 4.15 depicts these six neighbourhood conditions which correspond to the presence of critical connectivity. *Hence, the second restriction implies that if a pixel is critically connected then it should not be deleted.*

A thinning algorithm should also preserve an object's length. To facilitate this, a third restriction must be imposed such that arc-ends, i.e. object pixels which are adjacent to just one other pixel, must not be deleted.

Note that a thinned image should be invariant under the thinning operator, i.e. the application of the thinning algorithm to a fully thinned image should produce no changes. This is also important as it provides us with a condition for stopping the thinning algorithm. Since the pixels of a fully thinned image are either critically connected or are arc-ends, imposing the second and third restrictions allows this property to be fulfilled. The final thinning algorithm, then, is to scan the image in a raster fashion, removing all object pixels according to these three restrictions, varying border from pass to pass. The image is thinned until four successive passes (corresponding to the four border orientations) producing no

changes to the image are made, at which stage thinning ceases. Figure 4.16 illustrates the result of thinning an object in a binary image.

The concepts of *erosion* and *dilation* are related to thinning in the sense that erosion can be considered as a single pass of a thinning operator (stripping object pixels of all four border orientations). However, there is one significant difference in that, for most applications, one does not mind if the object breaks in two and, hence, the complex check for critical connectivity is no longer required. In fact, it

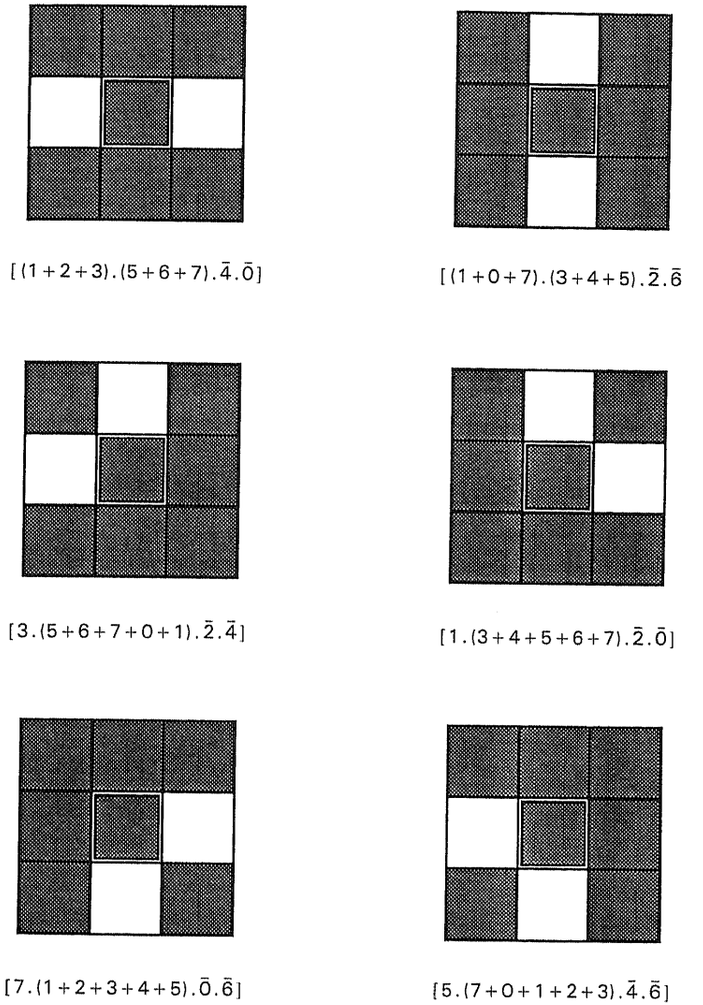

Figure 4.15 Neighbourhoods exhibiting critical connectivity.

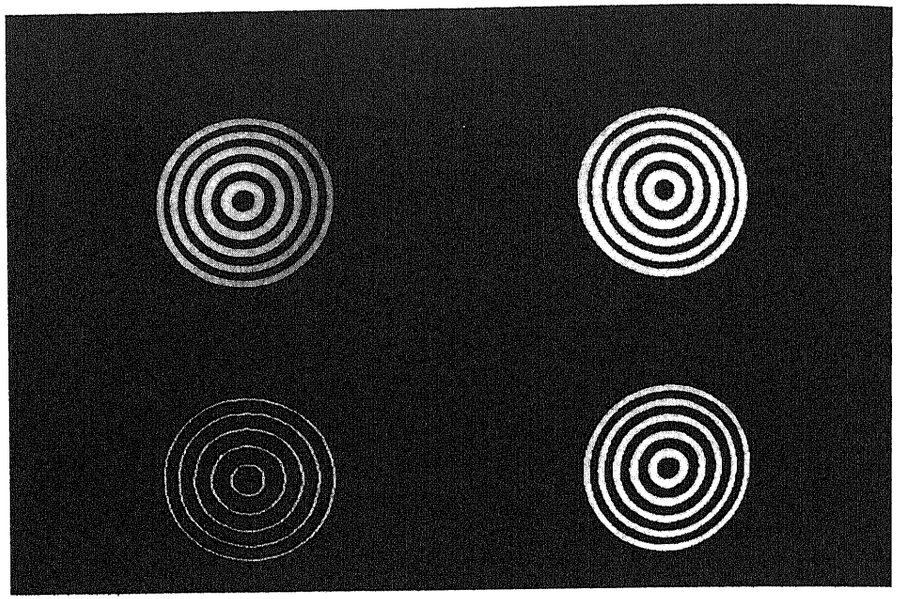

Figure 4.16 A grey-scale image (top-left) is thresholded to produce a binary image (top-right) which is then thinned (bottom-right). The image at the bottom-left is an intermediate partially thinned version.

is this property of object splitting that makes the erosion operation particularly useful. The dilation operation effects the reverse of erosion, i.e. an expansion of the object into all those background pixel cells which border the object. Thus, erosion 'shrinks' an object while dilation 'enlarges' it. This interpretation of erosion and dilation, although common, is quite a loose one. The operations of erosion and dilation do have a much more formal meaning in the context of a branch of image processing referred to as *mathematical morphology*. We will return to mathematical morphology in Section 4.4 but, for the present, we will continue with the informal treatment.

To see the usefulness of erosion and dilation operations, consider the following common printed circuit board inspection problem. Typically, in PCB manufacturing, a film negative depicting the appropriate pattern of electrical contacts (pads) and conductors (tracks) is contact-printed on a copper-clad board which has been covered with a photoresistive solution. The board is then etched, leaving just the copper circuit patterns. Such a process can lead to many problems with the final circuit pattern, e.g. the track may be broken, it may be too wide or too thin, or there may be spurious copper remaining on the PCB. These potential faults (shorts and breaks) are impossible for conventional functional (electrical) testing to detect and it is for this reason that these visual techniques are so useful.

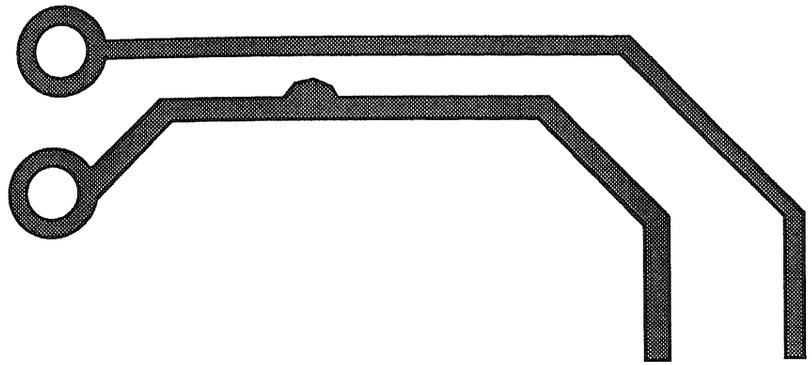

Figure 4.17 PCB track with extraneous copper.

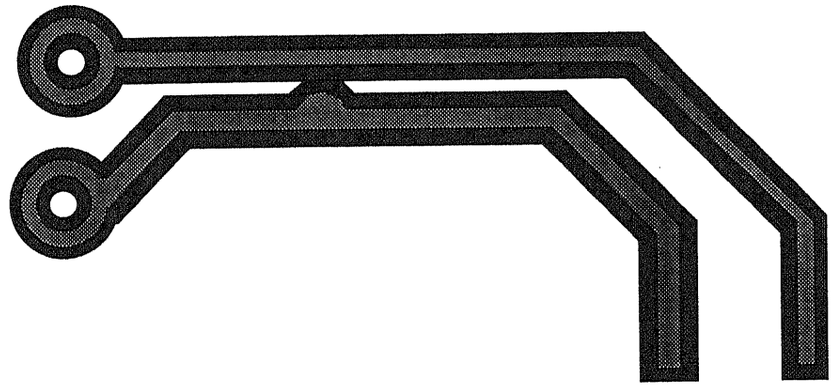

Figure 4.18 Dilated PCB track.

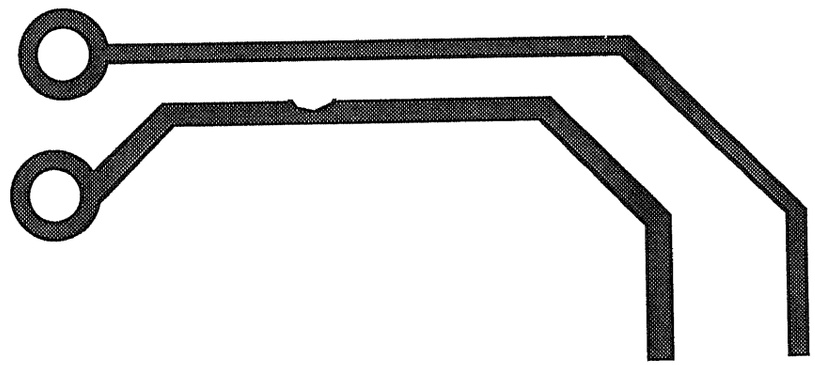

Figure 4.19 PCB track with a neck.

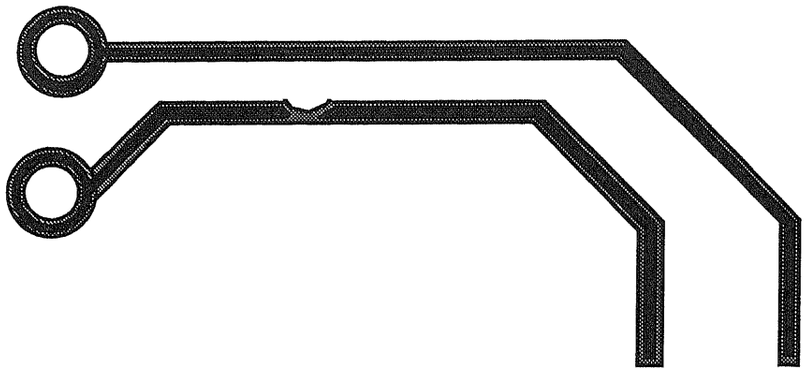

Figure 4.20 Eroded PCB track.

If the tracks (and/or pads) are too large or have extraneous copper attached (see Figure 4.17), then dilating the image a number of times will cause the two tracks to merge (see Figure 4.18) and a subsequent analysis of the track connectivity will identify this potential fault. Conversely, a track which is too thin or has a neck (see Figure 4.19) will break when eroded (Figure 4.20). Similar connectivity analysis will identify this potential circuit break.

A pseudo-coded algorithm for erosion can be formulated as follows:

```
/* erosion */

FOR all pixels in the image
  IF the pixel is an object pixel AND all its
  neighbours are
   object pixels
   copy it to the destination image
```

while a dilation algorithm can be formulated as:

```
/* dilation */

FOR all pixels in the image
  IF the pixel is an object pixel
     make it and its eight neighbours object pixels
     in the destination image
```

Figure 4.21 illustrates the effect of several applications of these erosion and dilation algorithms to an object in a binary image.

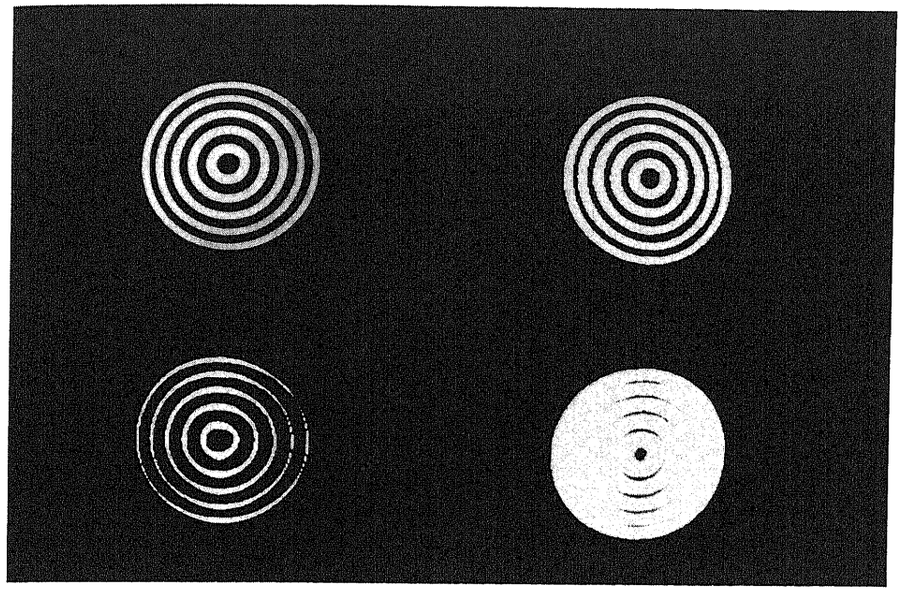

Figure 4.21 A grey-scale image (top-left) is thresholded to produce a binary image (top-right) which is then eroded twice (bottom-left). The application of two passes of the dilation algorithm is shown at bottom-right.

4.3 Geometric operations

Geometric operations change the spatial relationships between objects in an image, i.e. the relative distances between points a, b and c will typically be different after a geometric operation or 'warping'. The applications of such warping include geometric decalibration, i.e. the correction of geometric distortion introduced by the imaging system (most people are familiar with the barrel distortion that arises in photography when using a very short focal length 'fish-eye' lens), and image registration, i.e. the intentional distortion of one image with respect to another so that the objects in each image superimpose on one another. The techniques used in both these applications are identical and will be discussed in detail before describing actual usage.

4.3.1 Spatial warping

The approach to geometric image manipulation described here is called *spatial warping* and involves the computation of a mathematical model for the required

distortion, its application to the image, and the creation of a new corrected (decalibrated or registered) image.

The distortion may be specified by locating control points (also called fiducial points) in the input image (the image to be warped) and identifying their corresponding control points in an ideal (undistorted or registered) image. The distortion model is then computed in terms of the transformation between these control points generating a spatial warping function which will allow one to build the output image pixel by pixel, by identifying the corresponding point in the input image.

Since, in general, the estimates of the coordinates of input pixels yielded by the warping function will not correspond to exact (integer) pixel locations, we also require some method of estimating the grey-level of the output pixel when the 'corresponding' pixel falls 'between the integer coordinates' (see Figure 4.22). The question is: how do the four pixels surrounding the computed point contribute to our estimate of its grey-level, i.e. how do we interpolate between the four? We will return to this question later; suffice it at present to summarize the two requirements of geometric operations:

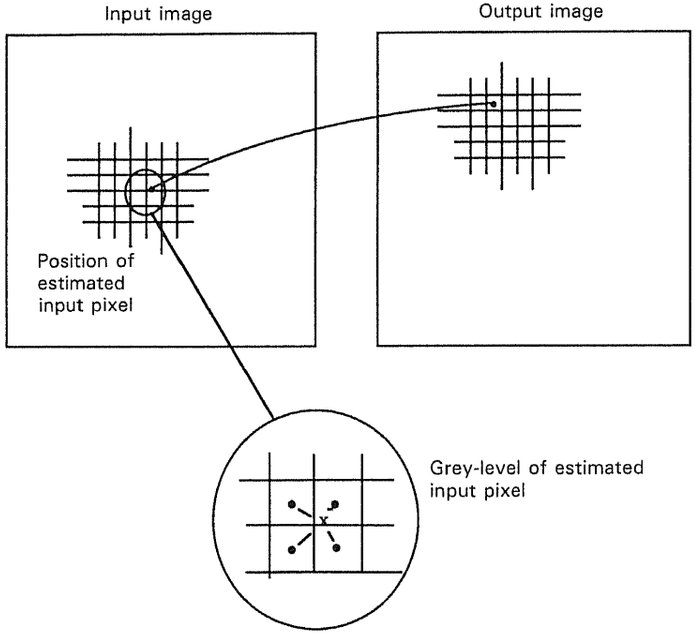

Figure 4.22 Spatial transformation and grey-level interpolation.

(a) a spatial transformation which allows one to derive the position of a pixel in the input which corresponds to the pixel being 'filled' or generated in the output;
(b) an interpolation scheme to estimate the grey-level of this input pixel.

Note that the grey-level interpolation algorithm may be permanently established in the software whereas the spatial transformation will change as the environment changes (e.g. camera, lens, registration requirements).

4.3.1.1 The spatial transformation

The spatial transformation is expressed in general form as a mapping from a point (x, y) in the output image to its corresponding (warped) position (i, j) in the input image:

$$(i, j) = (W_x(x, y), W_y(x, y))$$

That is, the first coordinate, i, of the warped point is a function of the current position in the output; likewise for the second coordinate, j. Thus, given any point (x, y) in the output image, the coordinates of the corresponding point in the input image may be generated using the warping functions W_x and W_y respectively. It would, of course, be ideal if we had some analytic expression for W_x and W_y but this is rarely the case. Instead, we normally model each spatial warping function by a polynomial function. So, we assume that, for example, the warping functions are given by the following equations:

$$W_x(x, y) = \sum_p^n \sum_q^n a_{pq} x^p y^q$$

$$W_y(x, y) = \sum_p^n \sum_q^n b_{pq} x^p y^q$$

For example, if $n = 2$ (which is adequate to correct for most distortions):

$$\begin{aligned} W_x(x, y) = {} & a_{00}x^0y^0 + a_{10}x^1y^0 + a_{20}x^2y^0 \\ & + a_{01}x^0y^1 + a_{11}x^1y^1 + a_{21}x^2y^1 \\ & + a_{02}x^0y^2 + a_{12}x^1y^2 + a_{22}x^2y^2 \end{aligned}$$

$$\begin{aligned} W_y(x, y) = {} & b_{00}x^0y^0 + b_{10}x^1y^0 + b_{20}x^2y^0 \\ & + b_{01}x^0y^1 + b_{11}x^1y^1 + b_{21}x^2y^1 \\ & + b_{02}x^0y^2 + b_{12}x^1y^2 + b_{22}x^2y^2 \end{aligned}$$

Now, the only thing that remains to complete the specification of the spatial warping function is to determine the values of these coefficients, i.e. to compute a_{00}–a_{22} and b_{00}–b_{22}.

To do this, we have to assume that we know the transformation exactly for a number of points (at least as many as the number of coefficients; nine in this case), that is, to assume that we know the values of x and y and their corresponding i

and j values. We then write the relationships explicitly in the form of the two equations above. We then solve these equations simultaneously to determine the value of the coefficients. Remember that we have two sets of simultaneous equations to set up: one for the 'a' coefficients and one for the 'b' coefficients. However, the same values relating x, y to i, j can be used in each case. This is now where the control points come in as we are going to use these to provide us with the (known) relationships between (x, y) and (i, j).

If we have nine unknown coefficients, as in the example above, then in order to obtain a solution we require at least nine such observations, $\{(x_1, y_1), (i_1, j_1)\} \ldots \{(x_9, y_9), (i_9, j_9)\}$, say. Such a system is said to be exactly determined. However, the solution of these exact systems is often ill-conditioned (numerically unstable) and it is usually good practice to overdetermine the system by specifying more control points than you need (and hence generate more simultaneous equations). These equations can then be solved, yielding the coefficients and hence the warping functions, using standard personal computer based maths packages such as MATLAB©.

For the sake of completeness, we include here details of how to solve such an overdetermined system. The reader can safely skip this section if (s)he so wishes.

The first point to note is that an overdetermined system (where the number of equations is greater than the number of unknown values) does not have an exact solution and there are going to be some errors for some points. The idea, then, is to minimize these errors. We will use the common approach of minimizing the sum of the square of each error (i.e. to generate the so-called least-square-error solution).

Consider, again, a single control point and assume we are attempting to compute the a_{pq} coefficients:

$$\begin{aligned} i_1 = {} & a_{00}x_1{}^0y_1{}^0 + a_{10}x_1{}^1y_1{}^0 + a_{20}x^2{}_1y_1{}^0 \\ & + a_{01}x_1{}^0y_1{}^1 + a_{11}x_1{}^1y_1{}^1 + a_{21}x_1{}^2y_1{}^1 \\ & + a_{02}x_1{}^0y_1{}^1 + a_{12}x_1{}^1y_1{}^2 + a_{22}x_1{}^2y_1{}^2 \end{aligned}$$

If we use m control points in total we will have m such equations which (noting that x^0 and y^0 are both equal to 1) we may write in matrix form as:

$$\begin{bmatrix} i_1 \\ i_2 \\ \vdots \\ i_m \end{bmatrix} = \begin{bmatrix} 1 x_1{}^1 x_1{}^2 y_1{}^1 x_1{}^1 y_1{}^1 x_1{}^2 y_1{}^1 y_1{}^2 x_1{}^1 y_1{}^2 x_1{}^2 y_1{}^2 \\ 1 x_2{}^1 x_2{}^2 y_2{}^1 x_2{}^1 y_2{}^1 x_2{}^2 y_2{}^1 y_2{}^2 x_2{}^1 y_2{}^2 x_2{}^2 y_2{}^2 \\ \vdots \\ 1 x_m{}^1 x_m{}^2 y_m{}^1 x_m{}^1 y_m{}^1 x_m{}^2 y_m{}^1 y_m{}^2 x_m{}^1 y_m{}^2 x_m{}^2 y_m{}^2 \end{bmatrix} * \begin{bmatrix} a_{00} \\ a_{10} \\ \vdots \\ a_{22} \end{bmatrix} + \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_m \end{bmatrix}$$

We have to include the errors since there will not be a set of a_{00}–a_{22} which will simultaneously provide us with exactly i_1–i_m, in the overdetermined case.

Let us abbreviate this matrix equation to:

$$i = Xa + e$$

Similar

$$j = Xb + e$$

We req … a, so we might think of multiplying across by X^{-1} to obtain an appropr… expression. Unfortunately, X is non-square (number of equations is greater … the number of coefficients) and one cannot invert a non-square matrix. Howeve… we can indulge in a little algebra and calculus to derive an expression for a in ter… of X and i:

$$\begin{aligned} i &= Xa + e \\ e &= i - Xa \end{aligned}$$

We forr… ıe sum of the square of each error by computing e^Te:

$$e^Te = (i - Xa)^T(i - Xa)$$

Differen… ing e^Te with respect to a, to find out how the errors change as the coefficie… change:

$$\begin{aligned} \frac{\mathrm{d}(e^Te)}{\mathrm{d}(a)} &= (0 - XI)^T(i - Xa) + (i - Xa)^T(0 - XI) \\ &= (-XI)^T(i - Xa) + (i^T - (Xa)^T)(-XI) \\ &= -IX^T(i - Xa) + (i^T - a^TX^T)(-XI) \\ &= -IX^Ti + IX^TXa - i^TXI + a^TX^TXI \end{aligned}$$

But noting that i^TXI and a^TX^TXI are 1×1 matrices and that the transpose of a 1×1 matrix is equal to itself, we transpose these two sub-expressions:

$$\begin{aligned} &= -IX^Ti + IX^TXa - IX^Ti + IX^TXa \\ &= 2(I)(X^TXa - X^Ti) \end{aligned}$$

The sum of the square of each error is minimized when $\mathrm{d}(e^Te)/\mathrm{d}(a)$ is equal to zero, thus:

$$\begin{aligned} 0 &= 2(I)(X^TXa - X^Ti) \\ (X^TX)^{-1}X^TXa &= (X^TX)^{-1}X^Ti \\ a &= (X^TX)^{-1}X^Ti \end{aligned}$$

$(X^TX)^{-1}X^T$ is commonly referred to as the 'pseudo-inverse' of X and is written $X^\dagger$.

When this has been computed, the coefficient matrix a may be computed by simply multiplying $X^\dagger$ by i; b is obtained in a like manner.

4.3.1.2 Grey-level interpolation

Once the spatial mapping function has been found, the output image can be built, pixel by pixel and line by line. The coordinates given by the warping function, denoting the corresponding points in the input image, will not in general be integer values and the grey-level must be interpolated from the grey-levels of the surrounding pixels.

The simplest interpolation function is *nearest-neighbour* interpolation (zero-order interpolation) whereby the grey-level of the output pixel (which is what we are trying to estimate) is given by the grey-level of the input pixel which is nearest to the calculated point in the input image (see Figure 4.23). The computation involved in this interpolation function is quite trivial but the function generally yields quite adequate results. If the image exhibits very fine detail in which adjacent pixel grey-level varies significantly, i.e. the image exhibits high spatial frequencies, some of this detail may be lost. In such cases, *bi-linear* (i.e. first-order) interpolation should be considered since the estimate is made on the basis of four neighbouring input pixels. Consider the case shown in Figure 4.24 where we need to estimate the grey-level of a point somewhere between image pixels (i, j), $(i, j+1)$, $(i+1, j)$, and $(i+1, j+1)$. Let the position of this point relative to pixel (i, j) be given by coordinates (p, q); $0 \leqslant p, q \leqslant 1$. The grey-level at point (p, q) is constrained by the grey-level at the four neighbouring pixels and is a function of its position between these neighbours. To estimate the grey-level, $f(p, q)$, we fit a surface through the four neighbours' grey-levels, the equation of which will in general identify the grey-level at any point between the neighbours. The surface we fit is a hyperbolic paraboloid and is defined by the bilinear equation:

$$f(p, q) = ap + bq + cpq + d$$

There are four coefficients, a, b, c, and d, which we must determine to identify this

101	100	103	105	107	105	103	110
110	140	120	122	130	130	121	120
134	134	135	131	137	138	120	121
132	132	132	133	133	150	160	155
134	140	140	135	140	156	160	174
130	138	139	150	169	175	170	165
126	133	138	149	163	169	180	185
130	140	150	169	178	185	190	200

Nearest neighbour

Computed point

Figure 4.23 Nearest-neighbour interpolation.

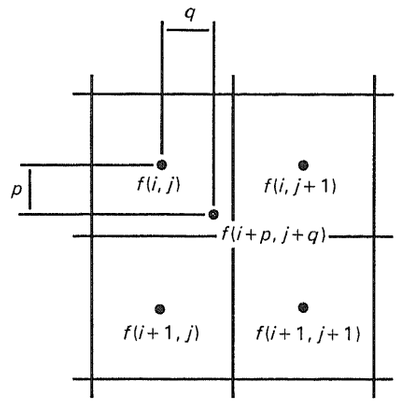

Figure 4.24 Bilinear interpolation.

function for any given 2×2 neighbourhood in which we wish to interpolate. Thus we require four simultaneous equations in a, b, c, and d; these are supplied from our knowledge of the grey-level at the four neighbours given by relative coordinates $(0, 0)$, $(0, 1)$, $(1, 0)$, $(1, 1)$. Specifically, we know that:

$$a \times 0 + b \times 0 + c \times 0 \times 0 + d = f(i, j) \tag{4.1}$$

$$a \times 0 + b \times 1 + c \times 0 \times 1 + d = f(i, j+1) \tag{4.2}$$

$$a \times 1 + b \times 0 + c \times 1 \times 0 + d = f(i+1, j) \tag{4.3}$$

$$a \times 1 + b \times 1 + c \times 1 \times 1 + d = f(i+1, j+1) \tag{4.4}$$

Directly from (4.1), we have:

$$d = f(i, j) \tag{4.5}$$

Rearranging (4.2) and substituting for d, we have:

$$b = f(i, j+1) - f(i, j) \tag{4.6}$$

Rearranging (4.3) and substituting for d, we have:

$$a = f(i+1, j) - f(i, j) \tag{4.7}$$

Rearranging (4.4) and substituting for a, b, and d, we have:

$$c = f(i+1, j+1) + f(i, j) - f(i+1, j) - f(i, j+1) \tag{4.8}$$

Equations (4.5)–(4.8) allow us to compute the coefficients a, b, c, and d, which define the bilinear interpolation for a given 2×2 neighbourhood with known pixel grey-levels.

For example, if the coordinates of the point at which we wish to estimate the grey-level are (60.4, 128.1) and the grey-level at pixels (60, 128), (60, 129), (61, 128),

and (61, 129) are 10, 12, 14, and 15, respectively, then the grey-level at this point, in relative coordinates, is given by:

$$\begin{aligned} f(0.4, 0.1) &= (14 - 10) \times 0.4 \\ &\quad + (12 - 10) \times 0.1 \\ &\quad + (15 + 10 - 14 - 12) \times 0.4 \times 0.1 \\ &\quad + 10 \\ &= 11.76 \end{aligned}$$

4.3.2 Registration and geometric decalibration

This technique for spatial warping can be used directly to effect either registration of two images or to correct the geometric distortion which may have been introduced by the imaging system. In the former case, one merely needs to identify several corresponding points in the two images and use these as the control points when generating the polynomial coefficients. In the latter case, one might use the imaging system to generate an image of a test card (e.g. a square grid) and superimpose an undistorted copy of this pattern on the distorted image. The corresponding control points can then be explicitly identified in each version of the pattern. For example, in the case of the grid pattern, the control points might be the points of intersection of the grid lines.

4.4 Mathematical morphology

4.4.1 Basic set theory

Mathematical morphology is a methodology for image processing and image analysis which is based on set theory and topology. As such, it is a formal and rigorous mathematical technique and we need to establish a basic 'language' before proceeding. For the most part, this is the language of set theory. In the following, points and vectors will be denoted by latin lowercase letters: x, y, z; sets will be denoted by latin uppercase letters: X, Y, Z; and the symbol ϕ denotes the empty set. The more common set operations we will encounter in this brief treatment of morphology include the following:

- *Set inclusion.* This is written: $Y \subset X$, i.e. Y is a subset of (is included in) the set X. This is defined as $y \in Y \Rightarrow y \in X$: if y is an element of Y, then y is also an element of X.
- *Complement.* For any set X, the complement of X is written X^c. This is the set of all elements which are *not* elements of X.
- *Union.* The union of two sets X and Y, written $X \cup Y$, is defined:

$$X \cup Y = \{x \mid x \in X \text{ or } x \in Y\}$$

This should be read: the set X union Y is the set of all x such that x is an element of X or x is an element of Y.

- *Intersection.* The intersection of two sets X and Y, written $X \cap Y$, is defined:

$$X \cap Y = (X^c \cup Y^c)^c$$

In effect, the intersection of two sets is the complement of the union of their respective complements. Thus, the intersection of X and Y is the complement of the set of elements which are not in X or not in Y, i.e. *the set of elements which is common to both sets X and Y.*

Let us now consider two further concepts. The first is translation. The translation of a set X by h is denoted X_h. This is a set where each element (point) is translated by a vector h. Second, we need to introduce a general symbolism for set transformation. A transformation of a set X is denoted by $\Psi(X)$. $\Psi(\ )$ is the set transformation and in an expression such as $Y = \Psi(X)$, Y is the transformed set.

Finally, we require the concept of *duality* of set transformations. Given some transformation Ψ, we define the dual of the transformation Ψ^*:

$$\Psi^*(X) \rightarrow (\Psi(X^c)^c$$

For example, intersection is the dual of union since $X \cap Y = (X^c \cup Y^c)^c$.

4.4.2 Structuring elements and hit or miss transformations

We are now in a position to continue with the discussion of mathematical morphology. Let us begin with the concept of a *structuring element.*

A structuring element B_x, centred at x, is a set of points which is used to 'extract' structure in a set, X, say. For example, the structuring element might be a square or a disk, as shown in Figure 4.25, or any other appropriate shape.

Now we define a *hit or miss transformation* as the 'point by point' transformation of a set X, working as follows. We choose, and fix, a structuring element B. Define $B_x{}^1$ to be that subset of B_x (recall B_x is the translate of B to a position x) whose elements belong to the 'foreground' and $B_x{}^2$ to be the subset of B_x whose elements belong to the 'background' (i.e. $B^1 \cap B^2 = \phi$).

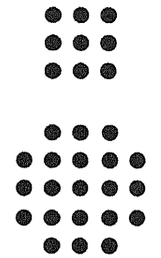

Figure 4.25 Square and circular structuring elements.

A point x belongs to the hit or miss transform, denoted $X \otimes B$, if and only if $B_x{}^1$ is included in X and $B_x{}^2$ is included in X^c the complement of X:

$$X \otimes B = \{x \mid B_x{}^1 \subset X; B_x{}^2 \subset X^c\}$$

Thus, $X \otimes B$ defines the points where the structuring element B exactly matches (hits) the set X, i.e. the image.

For example, let B be the structuring element shown in Figure 4.26(a) and let X be the set shown in Figure 4.26(b). Then $X \otimes B$ is the set shown in Figure 4.26(c).

4.4.3 Erosion and dilation

In Section 4.2.3, we informally introduced the concepts of erosion and dilation. We will now define them formally. Let $\check{B}$ be the transposed set of B, i.e. the

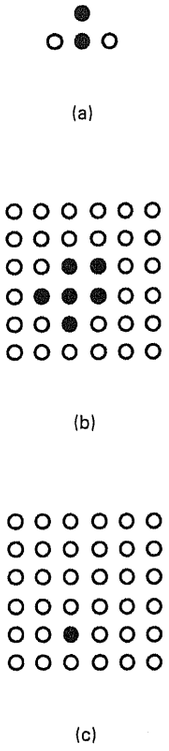

Figure 4.26 (a) Structuring element B; (b) image set X; (c) B 'hit or miss' $X : X \otimes B$.

symmetrical set of B with respect to its origin. Then, the erosion operation is denoted $\ominus$ and the erosion of a set X with $\check{B}$ is defined:

$$X \ominus \check{B} = \{x \mid B_x \subset X\}$$

Note that this is equivalent to a hit or miss transformation of X with B, where $B^2 = \phi$, i.e. where there are *no* background points. Note also that $X \ominus B$ is not an erosion – it is the *Minkowski subtraction* of B from X.

Intuitively, the erosion of a set by a structuring element amounts to the generation of a new (transformed/eroded) set where each element in the transformed set is a point where the structuring element is included in the original set X.

For example, let B and X be the structuring element and set shown in Figure 4.27(a) and (b), respectively; then $X \ominus \check{B}$ is depicted in Figure 4.27(c).

In a more complex case, if B is a circular structuring element, Figure 4.28 schematically illustrates the effect of erosion.

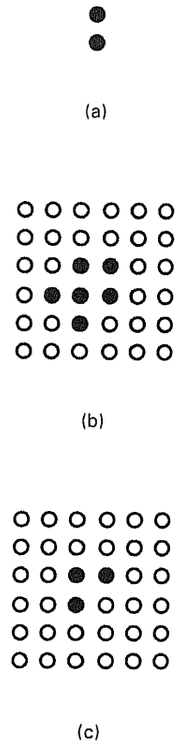

Figure 4.27 (a) Structuring element B; (b) image set X; (c) the erosion of X with $B : X \ominus \check{B}$.

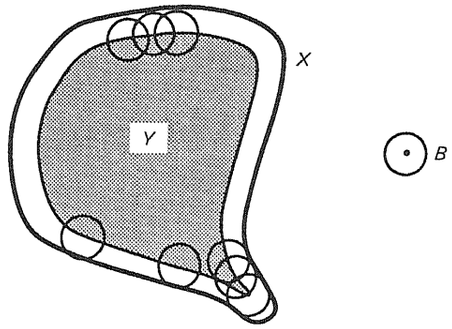

Figure 4.28 Erosion of X by a circular structuring element B.

Dilation is a closely related operation. In fact, dilation is the dual of erosion. The dilation operation is denoted by the symbol $\oplus$. Thus:

$$X \oplus \check{B} = X^c \ominus \check{B}$$

That is, the dilation of X is the erosion of X^c. This amounts to saying that the erosion of a set (e.g. an object) with a given structuring element is equivalent to the dilation of its complement (i.e. its background) with the same structuring element, and *vice versa*.

4.4.4 Opening and closing

After having eroded X by B, it is not possible in general to recover the initial set by dilating the eroded set $X \ominus B$ by the same B. This dilate reconstitutes only a part of X, which is simpler and has fewer details, but may be considered as that part which is most essential to the structure of X. This new set (i.e. the results of erosion followed by dilation) filters out (i.e. generates) a new subset of X which is extremely rich in morphological and size distribution properties. This transformation is called an *opening*.

The opening of a set X with a structuring element B, denoted X_B, is defined:

$$X_B = (X \ominus \check{B}) \oplus B$$

The *closing* of X with respect to B, denoted X^B, is defined:

$$X^B = (X \oplus \check{B}) \ominus B$$

Opening is the dual of closing, i.e.:

$$(X^c)_B = (X^B)^c$$

and

$$(X_B)^c = (X^c)^B$$

The opening is the domain swept out by all the translates of B which are included in X. This effects a smoothing of the contours of X, cuts narrow isthmuses, suppresses small islands and sharp edges in X.

4.4.5 Thinning and the extraction of endpoints

As we have seen in Section 4.2.3, an approximation to the skeleton can be achieved by an iterative transformation known as thinning. From the perspective of mathematical morphology, the thinning of a set X by a sequence of structuring elements L, is denoted

$$X \bigcirc \{L^i\}$$

that is

$$(((...(X \bigcirc L^1) \bigcirc L^2) \bigcirc L^3)... \bigcirc L^i)$$

$X \bigcirc L$ is defined:

$$X \bigcirc L = X / X \otimes L$$

That is, the set X less the set of points in X which hit L. Thus, if $X \otimes L$ identified border points, and L is appropriately structured to maintain connectivity of a set, then repeated application of the thinning process successively removes border points from a set until the skeleton is achieved. At this point, further application of the thinning transform yields no change in the skeletal set. This, of course, should be reminiscent of the thinning algorithm discussed in Section 4.2.3.

Recalling the definition of a hit and miss transformation:

$$X \otimes L = \{x \mid {}^1L_x \subset X; {}^2L_x \subset X^c\}$$

$${}^1L_x \cap {}^2L_x = \phi$$

We can now proceed to develop a thinning algorithm by defining L. The sequence $\{L\}$ which is used for thinning is based on a single structuring element and is generated by rotating the structuring element (through 360° in increments of 45° for a square lattice). This sequence $\{L\}$ is shown in Figure 4.29. The thinning algorithm then amounts to the repeated transformation of a set $X_i \rightarrow X_{i+1}$ defined:

$$X_{i+1} = (((...(X_i \bigcirc L_1) \bigcirc L_2) \bigcirc L_3)... \bigcirc L_8)$$

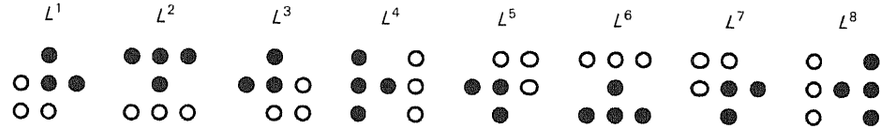

Figure 4.29 Sequence of structuring elements used in the thinning operation.

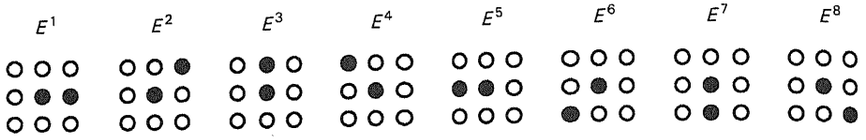

Figure 4.30 Structuring elements used to identify end-points.

The skeleton is achieved when $X_i = X_{i+1}$. Initially $X_0 = X$, i.e. the original (unthinned) image.

Given a skeleton set X, we can identify the endpoints, i.e. points which are connected to just one other point, using the hit or miss transform and an appropriate set of structuring elements $\{E\}$, shown in Figure 4.30. Thus, the endpoints of the skeleton are given by:

$$Y = \bigcup_{i=0}^{8} X \otimes E^i$$

That is, the union of all those points which *hit* with one of these endpoint structuring elements.

4.4.6 Application: identification of endpoints of electrical wires

In Chapter 8, we will be considering a complete vision and robotics case study: automated crimping of electrical wires. Part of this application is concerned with the identification of the positions of the ends of these electrical wires lying on a flat surface. In Chapter 8, we will be employing conventional techniques, but for the sake of illustration we will briefly consider a morphological approach here. Assuming that we are dealing with a grey-scale image of long thin wires (see Figures 8.18 and 8.19), then there are just three simple steps in achieving our objective:

Step 1. Threshold the grey-level image to obtain a binary image. Call this set X_1.
Step 2. Generate the skeleton X_2:

$$X_2 = X_1 \bigcirc \{L\}$$

Step 3. Identify endpoints of skeleton X_3:

$$X_3 = \bigcup_{i=0}^{8} X_2 \otimes E^i$$

X_3 is the set of all endpoints of these electrical wires.

4.4.7 A brief introduction to grey-scale mathematical morphology

So far, all of the mathematical morphology has assumed that we are dealing with

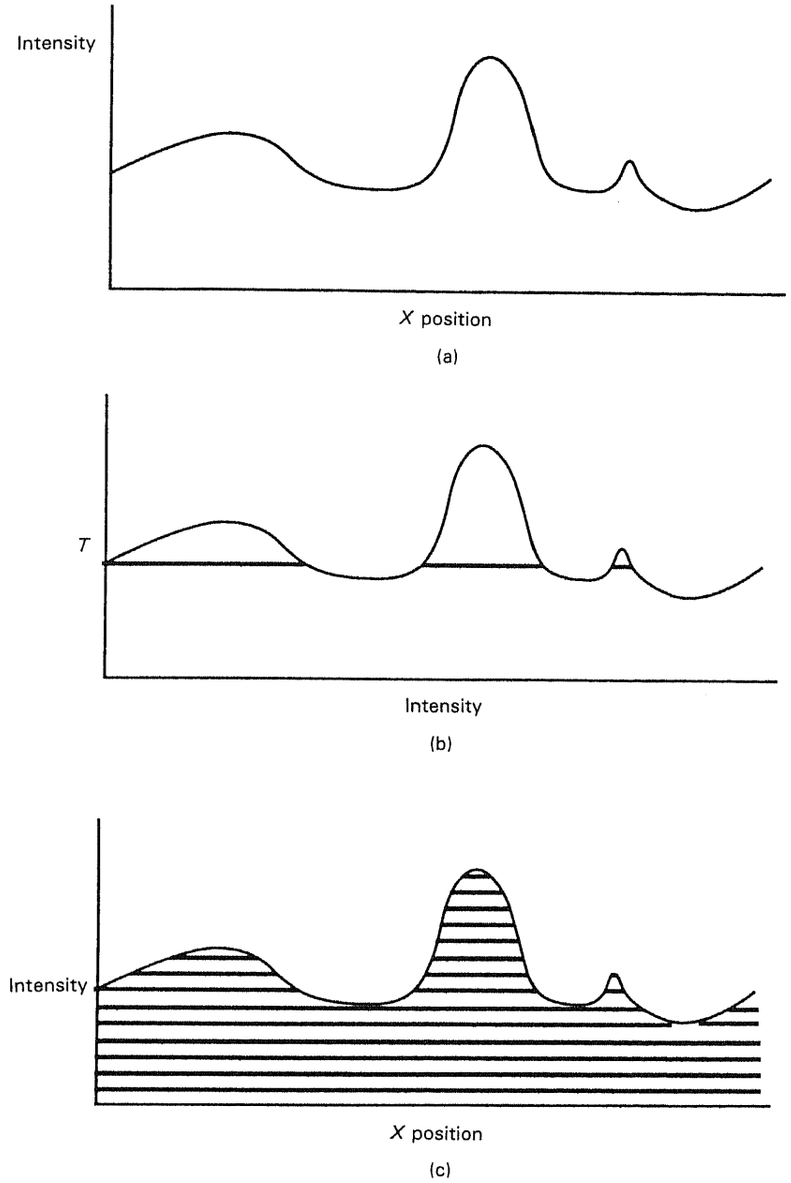

Figure 4.31 (a) A one-dimensional slice of a two-dimensional image function. (b) Thresholding this one-dimensional function at a value T generates a set $x_\lambda = \{x \mid x \geqslant T\}$ depicted by the bold line. (c) Representation of an image by a sequence of such sets.

a set X and its complement X^c, i.e. that we have been dealing with binary images comprising a foreground (the set of object points X) and a background (the set of all other points X^c). Unfortunately, grey-scale mathematical morphology is considerably more difficult to understand than binary morphology and this section is just intended to serve as an introduction to the manner in which we approach grey-level images from a morphological point of view.

For the following, we will consider one-dimensional slices of a two-dimensional image function. This makes it easier to represent in diagrams and it is somewhat easier to follow. Thus, a slice along the X-axis (i.e. a line of) a grey-scale image can be viewed as shown in Figure 4.31(a). If we threshold this function choosing values $\geqslant x_\lambda$, that is, generate a set x_λ:

$$x_\lambda = \{x \mid x \geqslant T\}$$

where T is the threshold value, we generate the set of points shown in a bold horizontal line in Figure 4.31(b). A grey-scale image, then, is considered to be a function f and is a sequence of sets:

$$f \Leftrightarrow \{x_\lambda(f)\}$$

and the grey-scale image is effectively the entire sequence of sets generated by successively decreasing the threshold level, as shown in Figure 4.31(c).

Grey-level erosion is defined:

$$f \rightarrow f \ominus \check{B})$$

and equivalently:

$$\{x_\lambda(f)\} \rightarrow \{x_\lambda(f) \ominus \check{B})\}$$

This grey-level erosion is the function, or sequence of sets, which are individually the erosion of sets generated at successive thresholds. Figure 4.32 illustrates the erosion of a (one-dimensional) function with a flat structuring element.

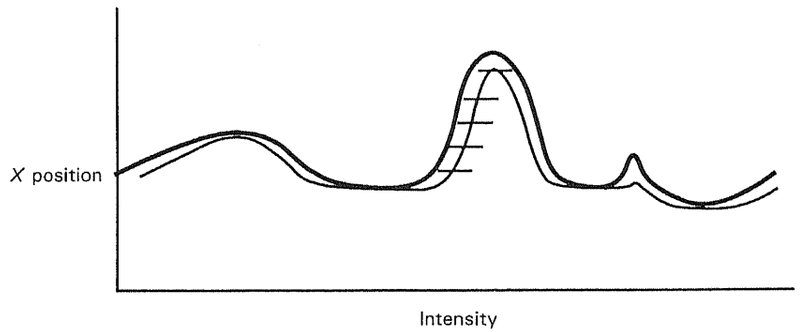

Figure 4.32 Erosion of a one-dimensional function with a flat structuring element; the eroded function is depicted by the thin curve.

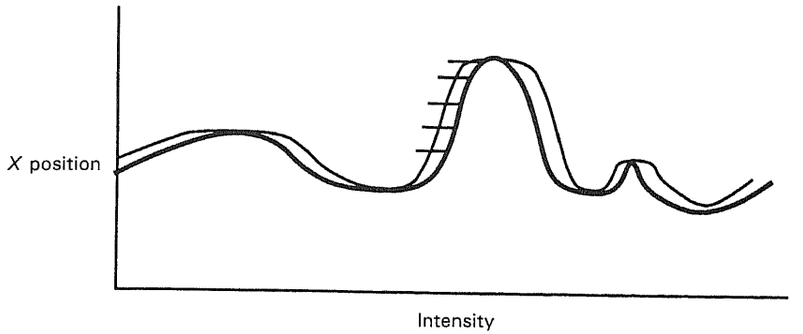

Figure 4.33 Dilation of a one-dimensional function with a flat structuring element; the dilated function is depicted by the thin curve.

Similarly, *grey-level dilation* is defined:

$$f \rightarrow f \oplus \check{B})$$
$$\{x_\lambda(f)\} \rightarrow \{x_\lambda(f) \oplus \check{B})\}$$

and Figure 4.33 illustrates the dilation of a (one-dimensional) function with a flat structuring element.

Exercises

1. Why is convolution a useful operation in image processing? Be specific in your answer by identifying the relationship between convolution and filtering.
2. Identify and annotate two simple techniques for noise removal in digital images; detail any assumptions upon which the techniques are based.
3. What is the essential difference between erosion and thinning? What is the relationship between thinning and the medial axis transform?
4. Why are look-up table (LUT) formulations of algorithms computationally efficient?
5. If two images can be registered by translation and rotation operations, is it necessary to use spatial warping techniques? Will grey-level interpolation be an issue?

References and further reading

Arcelli, C. 1979 'A condition for digital points removal', *Signal Processing*, Vol. 1, pp. 283–5.

Blum, H. 1967 'A transformation for extracting new descriptors of shape', in *Models for the Perception of Speech and Visual Form*, W. Wathen-Dunn (ed.), MIT Press Cambridge, Massachusetts, pp. 153–71.

Brady, M. and Asada, H. 1984 'Smoothed local symmetries and their implementation', *The International Journal of Robotics Research*, Vol. 3, No. 3, pp. 36–61.

Castleman, K.R. 1979 *Digital Image Processing*, Prentice Hall, New York.

Gonzalez, R.C. and Wintz, P. 1977 *Digital Image Processing*, Addison-Wesley, Reading, Massachusetts.

Hall, E.L. 1979 *Computer Image Processing and Recognition*, Academic Press, New York.

Hilditch, C.J. 1983 'Comparison of thinning algorithms on a parallel processor', *Image and Vision Computing*, Vol. 1, No. 3, pp. 115–32.

Kenny, P.A., Dowsett, D.J., Vernon, D. and Ennis J.T. 1990 'The application of spatial warping to produce aerosol ventilation images of the lung immediately after perfusion with the same labelled isotope', *Physics in Medicine and Biology*, Vol. 35, No. 5, 679–85.

Motzkin, Th. 1935 'Sur Quelques Proprietes Caracteristiques des Ensembles Bornes Non Convexes', *Atti. Acad. Naz. Lincei*, 21, pp. 773–9.

Nackman, L.R. and Pizer, S.M. 1985 'Three dimensional shape description using the symmetric axis transform 1: Theory', *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-7, No. 2, pp. 187–202.

Pratt, W.K. 1978 *Digital Image Processing*, Wiley, New York.

Rosenfeld, A. and Kak, A. 1982 *Digital Picture Processing*, Academic Press, New York.

Rosenfeld, A. 1975 'A characterization of parallel thinning algorithms', *Information and Control*, Vol. 29, pp. 286–91.

Serra, J. 1982 *Image Analysis and Mathematical Morphology*, Academic Press, London.

Tamura, H. 1978 'A comparison of line-thinning algorithms from a digital geometry viewpoint', *Proceedings 4th International Joint Conference on Pattern Recognition*, pp. 715–19.

Zhang, T.Y. and Suen, C.Y. 1984 'A fast parallel algorithm for thinning digital patterns', *Communications of the ACM*, Vol. 27, No. 3, pp. 236–9.

5

The segmentation problem

5.1 Introduction: region- and boundary-based approaches

Segmentation is a word used to describe a grouping process in which the components of a group are similar with respect to some feature or set of features. The inference is that this grouping will identify regions in the image which correspond to unique and distinct objects in the visual environment.

There are two complementary approaches to the problem of segmenting images and isolating objects: boundary detection and region growing. Region growing effects the segmentation process by grouping elemental areas (in simple cases, individual image pixels) sharing a common feature into connected two-dimensional areas called regions. Such features might be pixel grey-level or some elementary textural pattern, e.g. the short thin bars present in a herringbone texture.

Boundary-based segmentation is concerned with detecting or enhancing the boundary pixels of objects within the image and subsequently isolating them from the rest of the image. The boundary of the object, once extracted, may easily be used to define the location and shape of the object, effectively completing the isolation.

An image comprising boundaries alone is a much higher level representation of the scene than is the original grey-scale image, in that it represents important information explicitly. In cases where the boundary shape is complicated, the gap between these two representations is wide and it may be necessary to introduce an intermediate representation which is independent of the shape. Since boundaries of objects are often manifested as intensity discontinuities, a natural intermediate representation is composed of local object-independent discontinuities in image intensity, normally referred to as 'edges'. The many definitions of the term *edge* can be summarized by the observation (or premise) that an edge occurs in an image