

Chapter 2

Editorial Introduction

A Machine Vision system is like a chain: it is only as strong as its weakest link. To most people, the ability to handle images inside a computer, or dedicated electronic hardware, is a complete mystery, although the basic concepts are apparently quite straightforward. It is usually far easier to understand *what* an image processing operator does than *how* it does it. When designing/choosing a Machine Vision system, understanding the image processing at this level is usually adequate, although, in Chapter 15, Professor Roy Davies points out the shortcomings of this attitude. One of the major problems facing vision engineers is that everyone thinks that they are an expert on vision. Would-be customers certainly do! It is common for a vision engineer to be given gratuitous and utterly inappropriate advice about how to analyse an image by clients. For some inexplicable reason, many people feel that they must contribute in this way, even though they have no knowledge whatsoever about the details of any of the operators described in this chapter. It must be appreciated that machines do not see as human beings and it is fruitless to try to design them to do so. There are several reasons for this:

- a. We do not know enough about human vision to design algorithms properly on this basis.
- b. The basic computational “atoms” available in electronics and networks of neurons are completely different in nature.
- c. Despite their superficial simplicity, image processing operators possess a variety of subtle nuances and interactions, which together make their use far from straightforward.

The term *image processing* has two distinct uses. On the one hand, it has a specific meaning when it relates to the manipulation of pictures by transforming one image into another. It is also used in a generic sense, to encompass this meaning, as well as feature identification, location and measurement. These are all viewed as *low-level* functions and are discussed in this chapter. These are the principal methods used within most present-day industrial vision systems. *High-level* reasoning about images is discussed in Chapter 3.

Chapter 2

Basic Machine Vision Techniques

B.G. Batchelor and P.F. Whelan


This chapter introduces “low-level” vision processing operators, which lack the ability to perform the quintessential functions of intelligence: logical deduction, searching, classification and learning. Many intermediate-level operations can be implemented by concatenating these basic commands in simple macros. The operators described here are assigned mnemonic names, indicated in square brackets, and are used in Section 3.3.2, in combination with the AI language Prolog to produce smart vision systems.

2.1 Representations of Images

We shall first consider the representation of *Monochrome* (grey-scale) images. Let i and j denote two integers where $1 \leq i \leq m$ and $1 \leq j \leq n$. In addition, let $f(i,j)$ denote an integer function such that $0 \leq f(i,j) \leq W$. (W denotes the white level in a grey-scale image.) An array F will be called a *digital image*.

$$F = \begin{bmatrix} f(1,1), & f(1,2), & * & f(1,n) \\ f(2,1), & f(2,2), & * & f(2,n) \\ 5 & 5 & * & 5 \\ 5 & 5 & * & 5 \\ f(m,1), & f(m,2), & * & f(m,n) \end{bmatrix}$$

An address (i,j) defines a position in F , called a *pixel*, *pel* or *picture element*. The elements of F denote the intensities within a number of small rectangular regions within a real (i.e., optical) image. (Figure 2.1) Strictly speaking, $f(i,j)$ measures the intensity at a single point but if the corresponding rectangular region is small enough, the approximation will be accurate enough for most purposes. The array F contains a total of $m.n$ elements and this product is called the *spatial resolution* of F . We may *arbitrarily* assign intensities according to the following scheme:

$f(i,j) = 0$	black	
$0 < f(i,j) \leq 0.33W$	dark grey	
$0.33W < f(i,j) \leq 0.67W$	mid-grey	
$0.67W < f(i,j) < W$	light grey	
$f(i,j) = W$	white	

Let us consider how much data is required to represent a grey-scale image in this form. Each pixel requires the storage of $\log_2(1 + W)$ bits. This assumes that $(1 + W)$ is an integer power of two. If it is not, then $\log_2(1 + W)$ must be rounded up to the next integer. This can be represented using the ceiling function, $\lceil \log_2(1 + W) \rceil$. Thus, a grey-scale image requires the storage of $\lceil \log_2(1 + W) \rceil$ bits. Since there are $m.n$ pixels, the total data storage for the entire digital image F is equal to $m.n \lceil \log_2(1 + W) \rceil$ bits. If $m = n = 128$, and $W = 64$, we can obtain a good image of a human face. Many of the industrial image processing systems in use nowadays manipulate images in which $m = n = 512$ and $W = 255$. This leads to a storage requirement of 256 Kbytes/image. A *binary image* is one in which only two intensity levels, black (0) and white (1), are permitted. This requires the storage of $m.n$ bits/image.

An impression of colour can be conveyed to the eye by superimposing *four* separate imprints. (Cyan, magenta, yellow and black inks are often used in printing.) Ciné film operates in a similar way, except that when different colours of light, rather than ink, are added together, *three* components (red, green and blue) suffice. Television operates in a similar way to film; the signal from a colour television camera may be represented using three components: $R = \{r(i,j)\}$; $G = \{g(i,j)\}$; $B = \{b(i,j)\}$, where R , G and B are defined in a similar way to F . The vector $\{r(i,j), g(i,j), b(i,j)\}$ defines the intensity and colour at the point (i,j) in the colour image. Colour image analysis is discussed in more detail in Chapter 3. Multispectral images can also be represented using several monochrome images. The total amount of data required to code a colour image with r components is equal to $m.n.r \lceil \log_2(1 + W) \rceil$ bits, where W is simply the maximum signal level on each of the channels.

Ciné film and television will be referred to, in order to explain how moving scenes may be represented in digital form. A ciné film is, in effect, a time-sampled representation of the original moving scene. Each frame in the film is a standard colour, or monochrome image, and can be coded as such. Thus, a monochrome ciné film may be represented digitally as a sequence of two-dimensional arrays $[F_1, F_2, F_3, F_4, \dots]$. Each F_i is an $m.n$ array of integers as we defined above, when discussing the coding of grey-scale images. If the film is in colour, then each of the F_i has three components. In the general case, when we have a sequence of r -component colour images to code, we require $m.n.p.r \lceil \log_2(1 + W) \rceil$ bits/image sequence, where the spatial resolution is $m.n$ pixels, each spectral channel permits $(1 + W)$ intensity levels, there are r spectral channels and p is the total number of “stills” in the image sequence.

We have considered only those image representations, which are relevant to the understanding of simple image processing and analysis functions. Many alternative methods of coding images are possible but these are not relevant to this discussion.

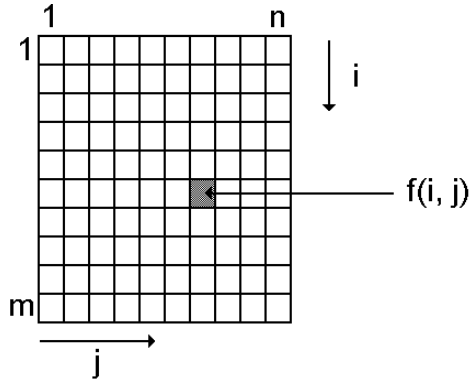


Figure 2.1. A digital image consisting of an array of $m.n$ pixels. The pixel in the i th row and the j th column has an intensity equal to $f(i,j)$.

2.2 Elementary Image Processing Functions

The following notation will be used throughout this section, in which we shall concentrate upon grey-scale images, unless otherwise stated:

- ≠ i and j are row and column address variables and lie within the ranges: $1 \leq i \leq m$ and $1 \leq j \leq n$. (Figure 2.1)
- ≠ $A = \{a(i,j)\}$, $B = \{b(i,j)\}$ and $C = \{c(i,j)\}$.
- ≠ W denotes the white level.
- ≠ $g(X)$ is a function of a single independent variable X .
- ≠ $h(X,Y)$ is a function of two independent variables, X and Y .
- ≠ The assignment operator ' \leftarrow ' will be used to define an operation that is performed upon one data element. In order to indicate that an operation is to be performed upon all pixels within an image, the assignment operator ' \blacktriangleleft ' will be used.
- ≠ k, k_1, k_2, k_3 are constants.
- ≠ $N(i,j)$ is that set of pixels arranged around the pixel (i,j) in the following way:

$(i-1, j-1)$	$(i-1, j)$	$(i-1, j+1)$
$(i, j-1)$	(i, j)	$(i, j+1)$
$(i+1, j-1)$	$(i+1, j)$	$(i+1, j+1)$

Notice that $N(i,j)$ forms a 3×3 set of pixels and is referred to as the 3×3 *neighbourhood* of (i,j) . In order to simplify some of the definitions, we shall refer to the intensities of these pixels using the following notation:

A	B	C
D	E	F
G	H	I

Ambiguities over the dual use of A, B and C should not be troublesome; as the context will make it clear which meaning is intended. The points $\{(i-1, j-1), (i-1, j), (i-1, j+1), (i, j-1), (i, j+1), (i+1, j-1), (i+1, j), (i+1, j+1)\}$ are called the *8-neighbours* of (i, j) and are also said to be *8-connected* to (i, j) . The points $\{(i-1, j), (i, j-1), (i, j+1), (i+1, j)\}$ are called the *4-neighbours* of (i, j) and are said to be *4-connected* to (i, j) .

2.2.1 Monadic, Point-by-point Operators.

These operators have a characteristic equation of the form:

$$c(i,j) \blacktriangleleft g(a(i,j)) \text{ or } E \blacktriangleleft g(E)$$

Such an operation is performed for all (i,j) in the range $[1,m].[1,n]$. (Figure 2.2). Several examples will now be described.

Intensity shift [acn]

$$c(i,j) \blacktriangleleft \begin{cases} 0 & a(i,j) + k < 0 \\ a(i,j) + k & 0 \leq a(i,j) + k \leq W \\ W & W < a(i,j) + k \end{cases}$$

k is a constant, set by the system user. Notice that this definition was carefully designed to maintain $c(i,j)$ within the same range as the input, viz. $[0, W]$. This is an example of a process referred to as *intensity normalisation*. Normalisation is important because it permits iterative processing by this and other operators in a machine having a limited precision for arithmetic (e.g., 8-bits). Normalisation will be used frequently throughout this chapter.

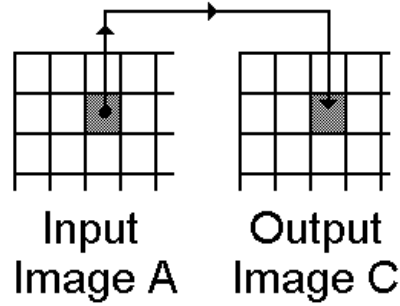


Figure 2.2. Monadic point-by-point operator. The (i,j) th pixel in the input image has intensity $a(i,j)$. This value is used to calculate $c(i,j)$, the intensity of the corresponding pixel in the output image.

Intensity multiply [mcn]

$$c(i,j) \leftarrow \begin{cases} 0 & a(i,j) \cdot k < 0 \\ a(i,j) \cdot k & 0 \leq a(i,j) \cdot k \leq W \\ W & W < a(i,j) \cdot k \end{cases}$$

Logarithm [log]

$$c(i,j) \leftarrow \begin{cases} 0 & a(i,j) = 0 \\ W \cdot \log(a(i,j)) / \log(W) & \text{otherwise} \end{cases}$$

This definition arbitrarily replaces the infinite value of $\log(0)$ by zero, and thereby avoids a difficult rescaling problem.

Antilogarithm (exponential) [exp]

$$c(i,j) \leftarrow W \cdot \exp(a(i,j)) / \exp(W)$$

Negate [neg]

$$c(i,j) \leftarrow W - a(i,j)$$

Threshold [thr]

$$c(i,j) \Leftarrow \begin{cases} W & k1 \leq a(i,j) \leq k2 \\ 0 & \text{otherwise} \end{cases}$$

This is an important function, which converts a grey-scale image to a binary format. Unfortunately, it is often difficult, or even impossible to find satisfactory values for the parameters $k1$ and $k2$.

Highlight [hil]

$$c(i,j) \Leftarrow \begin{cases} k3 & k1 \leq a(i,j) \leq k2 \\ a(i,j) & \text{otherwise} \end{cases}$$

Squaring [sqr]

$$c(i,j) \Leftarrow [a(i,j)]^2/W$$

2.2.2 Dyadic Point-by-point Operators

Dyadic operators have a characteristic equation of the form:

$$c(i,j) \Leftarrow h(a(i,j), b(i,j))$$

There are two input images: $A = \{a(i,j)\}$ and $B = \{b(i,j)\}$ (Figure 2.3), while the output image is $C = \{c(i,j)\}$. It is important to realise that $c(i,j)$ depends upon only $a(i,j)$ and $b(i,j)$. Here are some examples of dyadic operators.

Add [add]

$$c(i,j) \Leftarrow [a(i,j) + b(i,j)]/2.$$

Subtract [sub]

$$c(i,j) \Leftarrow [(a(i,j) - b(i,j)) + W]/2$$

Multiply [mul]

$$c(i,j) \Leftarrow [a(i,j).b(i,j)]/W$$

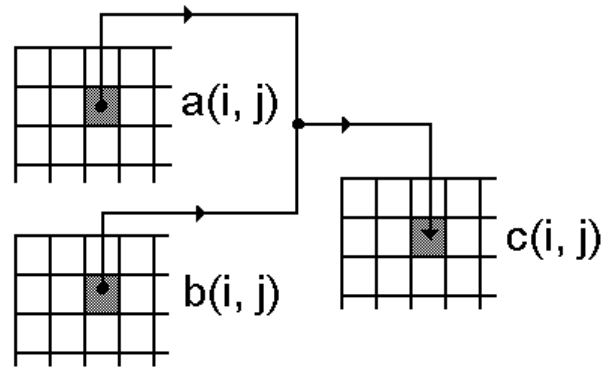


Figure 2.3. Dyadic point-by-point operator. The intensities of the (i, j) th pixels in the two input images (i.e., $a(i, j)$ and $b(i, j)$) are combined to calculate the intensity, $c(i, j)$, at the corresponding address in the output image.

Maximum [max]

$$c(i, j) \Leftarrow \text{MAX}[a(i, j), b(i, j)]$$

When the maximum operator is applied to a pair of binary images, the *union* (OR function) of their white areas is computed. This function may also be used to *superimpose* white writing onto a grey-scale image.

Minimum [min]

$$c(i, j) \Leftarrow \text{MIN}[a(i, j), b(i, j)]$$

When A and B are both binary, the *intersection* (AND function) of their white areas is calculated.

2.2.3 Local Operators

Figure 2.4 illustrates the principle of the operation of local operators. Notice that the intensities of several pixels are combined together, in order to calculate the intensity of just one pixel. Amongst the simplest of the local operators are those which use a set of nine pixels arranged in a 3×3 square. These have a characteristic equation of the following form:

$$c(i, j) \Leftarrow g(a(i-1, j-1), a(i-1, j), a(i-1, j+1), a(i, j-1), a(i, j), a(i, j+1), a(i+1, j-1), a(i+1, j), a(i+1, j+1)))$$

where $g(\cdot)$ is a function of 9 variables. This is an example of a local operator, which uses a 3×3 *processing window*. (That is, it computes the value for one pixel on the basis of the intensities within a region containing 3×3 pixels. Other

local operators employ larger windows and we shall discuss these briefly later.) In the simplified notation which we introduced earlier, the above definition reduces to: $E \Leftarrow g(A, B, C, D, E, F, G, H, I)$.

2.2.4 Linear Local Operators

An important sub-set of the local operators is that group, which performs a linear, weighted sum, and which are therefore known as *linear local operators*. For this group, the characteristic equation is:

$$E \Leftarrow k_1.(A.W_1 + B.W_2 + C.W_3 + D.W_4 + E.W_5 + F.W_6 + G.W_7 + H.W_8 + I.W_9) + k_2$$

where W_1, W_2, \dots, W_9 are weights, which may be positive, negative or zero. Values for the normalisation constants, k_1 and k_2 are given later. The matrix illustrated below is termed the *weight matrix* and is important, because it determines the properties of the linear local operator.

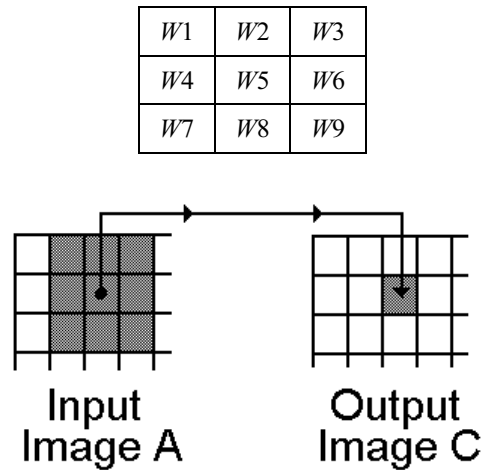


Figure 2.4. Local operator. In this instance, the intensities of nine pixels arranged in a 3×3 window are combined together. Local operators may be defined which uses other, possibly larger windows. The window may, or may not, be square and the calculation may involve linear or non-linear processes.

The following rules summarise the behaviour of this type of operator. (They exclude the case where *all* the weights and normalisation constants are zero, since this would result in a null image.):

- i. If all weights are either positive or zero, the operator will *blur* the input image. Blurring is referred to as *low-pass filtering*. Subtracting a blurred image from the original results in a highlighting of those points where the intensity is changing rapidly and is termed *high-pass filtering*.

- ii. If $W1 = W2 = W3 = W7 = W8 = W9 = 0$, and $W4, W5, W6 > 0$, then the operator blurs along the rows of the image; horizontal features, such as edges and streaks, are not affected.
- iii. If $W1 = W4 = W7 = W3 = W6 = W9 = 0$, and $W2, W5, W8 > 0$, then the operator blurs along the columns of the image; vertical features are not affected.
- iv. If $W2 = W3 = W4 = W6 = W7 = W8 = 0$, and $W1, W5, W9 > 0$, then the operator blurs along the diagonal (top-left to bottom-right). There is no smearing along the orthogonal diagonal.
- v. If the weight matrix can be reduced to a matrix product of the form $P.Q$, where

$$P = \begin{array}{|c|c|c|} \hline 0 & 0 & 0 \\ \hline V4 & V5 & V6 \\ \hline 0 & 0 & 0 \\ \hline \end{array}$$

and

$$Q = \begin{array}{|c|c|c|} \hline 0 & V1 & 0 \\ \hline 0 & V2 & 0 \\ \hline 0 & V3 & 0 \\ \hline \end{array}$$

the operator is said to be of the “separable” type. The importance of this is that it is possible to apply two simpler operators in succession, with weight matrices P and Q , in order to obtain the same effect as that produced by the separable operator.

- vi. The successive application of linear local operators which use windows containing 3×3 pixels produces the same results as linear local operators with larger windows. For example, applying that operator which uses the following weight matrix

1	1	1
1	1	1
1	1	1

twice in succession results in a similar image as that obtained from the 5×5 operator with the following weight matrix. (For the sake of simplicity, normalisation has been ignored here.)

1	2	3	2	1
2	4	6	4	2
3	6	9	6	3
2	4	6	4	2
1	2	3	2	1

Applying the same 3×3 operator thrice is equivalent to using the following 7×7 operator

1	3	6	7	6	3	1
3	9	18	21	18	9	3
6	18	36	42	36	18	6
7	21	42	49	42	21	7
6	18	36	42	36	18	6
3	9	18	21	18	9	3
1	3	6	7	6	3	1

Notice that all of these operators are also separable. Hence it would be possible to replace the last-mentioned 7×7 operator with four simpler operators: 3×1 , 3×1 , 1×3 and 1×3 , applied in any order. It is not always possible to replace a large-window operator with a succession of 3×3 operators. This becomes obvious when one considers, for example, that a 7×7 operator uses 49 weights and that three 3×3 operators provide only 27 degrees of freedom. Separation is often possible, however, when the larger operator has a weight matrix with some redundancy, for example when it is symmetrical.

- vii. In order to perform normalisation, the following values are used for $k1$ and $k2$.

$$k1 \leftarrow 1 / \sum_{p,q} |W_{p,q}|$$

$$k2 \leftarrow \left[1 - \sum_{p,q} W_{p,q} / \sum_{p,q} |W_{p,q}| \right] \cdot W/2$$

- viii. A filter using the following weight matrix performs a *local averaging function* over an 11×11 window $[raf(11,11)]$.

1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1

This produces quite a severe two-directional blurring effect. Subtracting the effects of a blurring operation from the original image generates a picture in which spots, streaks and intensity steps are all emphasised. On the other hand, large areas of constant or slowly changing intensity become uniformly grey. This process is called *high-pass filtering*, and produces an effect similar to unsharp masking, which is familiar to photographers.

2.2.5 Non-linear Local Operators

Largest intensity neighbourhood function [lnb]

$$E \Leftarrow \text{MAX}(A, B, C, D, E, F, G, H, I)$$

This operator has the effect of spreading bright regions and contracting dark ones.

Edge detector [command sequence: lnb, sub]

$$E \Leftarrow \text{MAX}(A, B, C, D, E, F, G, H, I) - E$$

This operator is able to highlight edges (i.e., points where the intensity is changing rapidly).

Median filter [mdf(5)]

$$E \Leftarrow \text{FIFTH_LARGEST}(A, B, C, D, E, F, G, H, I)$$

This filter is particularly useful for reducing the level of noise in an image. (Noise is generated from a range of sources, such as video cameras and x-ray detectors, and can be a nuisance if it is not eliminated by hardware or software filtering.)

Crack detector¹ [lnb, lnb, neg, lnb, lnb, neg]

This operator is equivalent to applying the above sequence of operations and then subtracting the result from the original image. This detector is able to detect thin dark streaks and small dark spots in a grey-scale image; it ignores other features, such as bright spots and streaks, edges (intensity steps) and broad dark streaks.

Roberts edge detector [red]

The Roberts gradient is calculated using a 2×2 mask. This will determine the edge gradient in two diagonal directions (i.e., the cross-differences).

$$E \leftarrow \sqrt{(A - E)^2 + (B - D)^2}$$

The following approximation to the Roberts gradient magnitude is called the Modified Roberts operator. This is simpler and faster to implement and it more precisely defines the operator *red*. It is defined as

$$E \leftarrow \{|A - E| + |B - D|\} / 2$$

Sobel edge detector [sed]

This popular operator highlights the edges in an image; points where the intensity gradient is high are indicated by bright pixels in the output image. The Sobel edge detector uses a 3×3 mask to determine the edge gradient.

$$E \leftarrow \sqrt{[(A + 2.B + C) - (G + 2.H + I)]^2 + [(A + 2.D + G) - (C + 2.F + I)]^2}$$

The following approximation is simpler to implement in software and hardware and more precisely defines the operator *sed*:

$$E \leftarrow \{|(A + 2.B + C) - (G + 2.H + I)| + |(A + 2.D + G) - (C + 2.F + I)|\} / 6$$

Figure 2.5 shows a comparison of the Roberts and Sobel edge detector operators when applied to a sample monochrome image. Note that, while the Roberts operator produces thinner edges, these edges tend to break up in regions of high curvature. The primary disadvantage of the Roberts operator is its high sensitivity to noise, since fewer pixels are used in the calculation of the edge gradient. There is also a slight shift in the image, when the Roberts edge detector is used. The Sobel edge detector does not produce such a shift.

¹ This is an example of an operator that can be described far better using computer notation rather than mathematical notation.

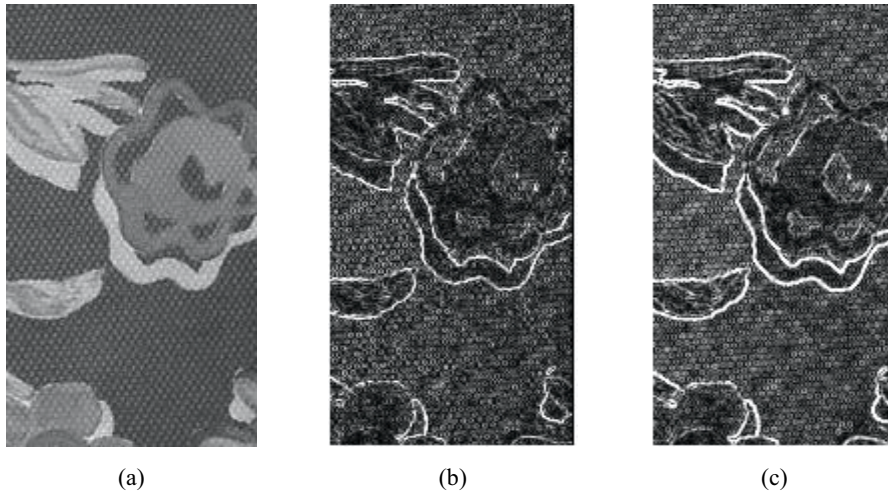


Figure 2.5. Edge detection applied to an image derived from a piece of dress fabric: (a) original image; (b) Roberts gradient; (c) Sobel gradient

Prewitt edge detector

The Prewitt edge-detector is similar to the Sobel operator, but is more sensitive to noise as it does not possess the same inherent smoothing. This operator uses the two 3×3 weight matrices shown below to determine the edge gradient,

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|} \hline -1 & -1 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 1 & 1 \\ \hline \end{array} & & \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \\
 P_1 & & P_2
 \end{array}$$

where P_1 and P_2 are the values calculated from each mask respectively. The Prewitt gradient magnitude is defined as:

$$E \propto \sqrt{P_1^2 + P_2^2}$$

Frei and Chen edge detector

This operator uses the two 3×3 masks shown below to determine the edge gradient,

$$\begin{array}{ccc}
 \begin{array}{|c|c|c|} \hline -1 & -2 & -1 \\ \hline 0 & 0 & 0 \\ \hline 1 & 2 & 1 \\ \hline \end{array} & & \begin{array}{|c|c|c|} \hline -1 & 0 & 1 \\ \hline -2 & 0 & 2 \\ \hline -1 & 0 & 1 \\ \hline \end{array} \\
 F_1 & & F_2
 \end{array}$$

where F_1 and F_2 are the values calculated from each mask respectively. The Frei and Chen gradient magnitude is defined as: $E \leftarrow \sqrt{F_1^2 + 2 F_2^2}$

Rank filters [mdf, rid]

The generalised 3×3 rank filter is:

$$c(i, j) \leftarrow k1.(A W1 + B W2 + C W3 + D W4 + E W5 + F W6 + G W7 + H W8 + I W9) + k2$$

where

$A \leftarrow \text{LARGEST}(A, B, C, D, E, F, G, H, I)$

$B \leftarrow \text{SECOND_LARGEST}(A, B, C, D, E, F, G, H, I)$

$C \leftarrow \text{THIRD_LARGEST}(A, B, C, D, E, F, G, H, I)$

*

$I \leftarrow \text{NINTH_LARGEST}(A, B, C, D, E, F, G, H, I)$

and $k1$ and $k2$ are the normalisation constants defined previously. With the appropriate choice of weights ($W1, W2, \dots, W9$), the rank filter can be used for a range of operations including edge detection, noise reduction, edge sharpening and image enhancement.

Direction codes [dbn]

This function can be used to detect the *direction* of the intensity gradient. A direction code function DIR_CODE is defined thus:

$$\text{DIR_CODE}(A, B, C, D, F, G, H, I) \leftarrow \begin{cases} 1 & \text{if } A \geq \text{MAX}(B, C, D, F, G, H, I) \\ 2 & \text{if } B \geq \text{MAX}(A, C, D, F, G, H, I) \\ 3 & \text{if } C \geq \text{MAX}(A, B, D, F, G, H, I) \\ 4 & \text{if } D \geq \text{MAX}(A, B, C, F, G, H, I) \\ 5 & \text{if } F \geq \text{MAX}(A, B, C, D, G, H, I) \\ 6 & \text{if } G \geq \text{MAX}(A, B, C, D, F, H, I) \\ 7 & \text{if } H \geq \text{MAX}(A, B, C, D, F, G, I) \\ 8 & \text{if } I \geq \text{MAX}(A, B, C, D, F, G, H) \end{cases}$$

Using this definition the operator *dbn* may be defined as:

$$E \leftarrow \text{DIR_CODE}(A, B, C, D, F, G, H, I)$$

2.2.6 N-tuple Operators

The N-tuple operators are closely related to the local operators and have a large number of linear and non-linear variations. N-tuple operators may be regarded as generalised versions of local operators. In order to understand the N-tuple operators, let us first consider a *linear* local operator, which uses a large processing window, (say $r.s$ pixels) with most of its weights equal to zero. Only N of the weights are non-zero, where $N \ll r.s$. This is an N-tuple filter (Figure 2.6.). The N-tuple filters are usually designed to detect specific patterns. In this role, they are able to locate a simple feature, such as a corner, annulus, the numeral “2”, in any position etc. However, they are sensitive to changes of orientation and scale. The N-tuple can be regarded as a sloppy template, which is convolved with the input image.

Non-linear tuple operators may be defined in a fairly obvious way. For example, we may define operators which compute the average, maximum, minimum or median values of the intensities of the N pixels covered by the N-tuple. An important class of such functions is the *morphological operators*. (Sections 2.4 and 2.5.) Figure 2.7 illustrates the recognition of the numeral ‘2’ using an N-tuple. Notice how the goodness of fit varies with the shift, tilt, size, and font. Another character (‘Z’ in this case) may give a score that is close to that obtained from a ‘2’, thus making these two characters difficult to distinguish reliably.

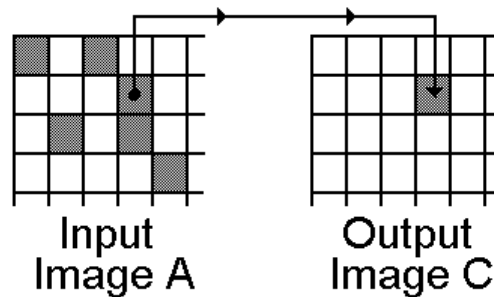


Figure 2.6. An N-tuple filter operates much like a local operator. The only difference is that the pixels whose intensities are combined together do not form a compact set. A linear N-tuple filter can be regarded as being equivalent to a local operator which uses a large window and in which many of the weights are zero.

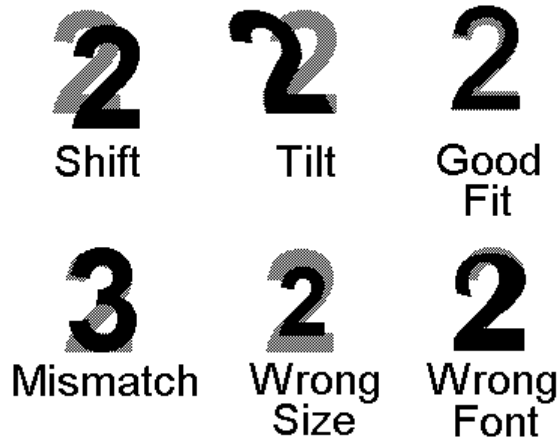


Figure 2.7. Recognising a numeral '2' using an N-tuple.

2.2.7 Edge Effects

All local operators and N-tuple filters are susceptible to producing peculiar effects around the edges of an image. The reason is simply that, in order to calculate the intensity of a point near the edge of an image, we require information about pixels outside the image, which of course are simply not present. In order to make some attempt at calculating values for the edge pixels, it is necessary to make some assumptions, for example that all points outside the image are black, or have the same values as the border pixels. This strategy, or whatever one we adopt, is perfectly arbitrary and there will be occasions when the edge effects are so pronounced that there is nothing that we can do but to remove them by masking [*edg*]. Edge effects are important because they require us to make special provisions for them when we try to patch several low-resolution images together.

2.2.8 Intensity Histogram [*hpi*, *hgi*, *hge*, *hgc*]

The intensity histogram is defined in the following way:

- a. let

$$s(p, i, j) \leftarrow \begin{cases} 1 & a(i, j) = p \\ 0 & \text{otherwise} \end{cases}$$

- b. let $h(p)$ be defined thus: $h(p) \leftarrow \sum_{i, j} s(p, i, j)$

It is not, in fact, necessary to store each of the $s(p,i,j)$, since the calculation of the histogram can be performed as a serial process in which the estimate of $h(p)$ is updated iteratively, as we scan through the input image. The *cumulative histogram*, $H(p)$, can be calculated using the following recursive relation:

$$H(p) = H(p-1) + h(p), \text{ where } H(0) = h(0).$$

Both the cumulative and the standard histograms have a great many uses, as will become apparent later. It is possible to calculate various intensity levels which indicate the occupancy of the intensity range $[pct]$. For example, it is a simple matter to determine that intensity level, $p(k)$, which when used as a threshold parameter ensures that a proportion k of the output image is black, $p(k)$ can be calculate using the fact that $H(p(k)) = m.n.k$. The *mean intensity* $[avg]$ is equal to:

$$\sum_p (h(p).p) / (m.n)$$

while the *maximum intensity* $[gli]$ is equal to $\text{MAX}(p \mid h(p) > 0)$ and the *minimum intensity* is equal to $\text{MIN}(p \mid h(p) > 0)$.

One of the principal uses of the histogram is in the selection of threshold parameters. It is useful to plot $h(p)$ as a function of p . It is often found from this graph that a suitable position for the threshold can be related directly to the position of the “foot of the hill” or to a “valley” in the histogram.

An important operator for image enhancement is given by the transformation:

$$c(i,j) \leftarrow [W.H(a(i,j))] / (m.n)$$

This has the interesting property that the histogram of the output image $\{c(i,j)\}$ is flat, giving rise to the name *histogram equalisation* $[heq]$ for this operation. Notice that histogram equalisation is a *data-dependent* monadic, point-by-point operator.

An operation known as “*local area histogram equalisation*” relies upon the application of histogram equalisation within a small window. The number of pixels in a small window that are darker than the central pixel is counted. This number defines the intensity at the equivalent point in the output image. This is a powerful filtering technique, which is particularly useful in texture analysis applications. (Section 2.7.).

2.3 Binary Images

For the purposes of this description of binary image processing, it will be convenient to assume that $a(i,j)$ and $b(i,j)$ can assume only two values: 0 (black) and 1 (white). The operator “+” denotes the Boolean **OR** operation, “•” represents the **AND** operation and where ‘ \otimes ’ denotes the Boolean **Exclusive OR** operation. Let $\#(i,j)$ denote the number of white points addressed by $N(i,j)$, including (i,j) itself.

Inverse [not]

$$c(i,j) \Leftarrow \text{NOT}(a(i,j))$$

AND white regions [and]

$$c(i,j) \Leftarrow a(i,j) \cdot b(i,j)$$

OR [lor, max]

$$c(i,j) \Leftarrow a(i,j) + b(i,j)$$

Exclusive OR [xor] (Find differences between white regions.)

$$c(i,j) \Leftarrow a(i,j) \otimes b(i,j)$$

Expand white areas [exw]

$$c(i,j) \Leftarrow a(i-1,j-1) + a(i-1,j) + a(i-1,j+1) + a(i,j-1) + a(i,j) + a(i,j+1) \\ + a(i+1,j-1) + a(i+1,j) + a(i+1,j+1)$$

Notice that this is closely related to the local operator *lnb* defined earlier. This equation may be expressed in the simplified notation: $E \Leftarrow A + B + C + D + E + F + G + H + I$

Shrink white areas [skw]

$$c(i,j) \Leftarrow a(i-1,j-1) \cdot a(i-1,j) \cdot a(i-1,j+1) \cdot a(i,j-1) \cdot a(i,j) \cdot a(i,j+1) \\ \cdot a(i+1,j-1) \cdot a(i+1,j) \cdot a(i+1,j+1)$$

or more simply $c(i,j) \Leftarrow A \cdot B \cdot C \cdot D \cdot E \cdot F \cdot G \cdot H \cdot I$

Edge detector [bed]

$$c(i,j) \Leftarrow E \cdot \text{NOT}(A \cdot B \cdot C \cdot D \cdot F \cdot G \cdot H \cdot I)$$

Remove isolated white points [wrm]

$$c(i,j) \Leftarrow \begin{cases} 1 & a(i,j) \cdot (\#(i,j) > 1) \\ 0 & \text{otherwise} \end{cases}$$

Count white neighbours [cnw]

$$c(i,j) \Leftarrow \#(a(i,j) = 1).$$

Where $\#(Z)$ is the number of times Z occurs. Notice that $\{c(i,j)\}$ is a grey-scale image.

Connectivity detector [cny].

Consider the following pattern:

1	0	1
1	X	1
1	0	1

If $X=1$, then all of the 1s are 8-connected to each other. Alternatively, if $X=0$, then they are not connected. In this sense, the point marked X is critical for connectivity. This is also the case in the following examples:

1	0	0
0	X	1
0	0	0

1	1	0
0	X	0
0	0	1

0	0	1
1	X	0
1	0	1

However, those points marked X below are not critical for connectivity, since setting $X=0$ rather than 1 has no effect on the connectivity of the 1s.

1	1	1
1	X	1
0	0	1

0	1	1
1	X	0
1	1	1

0	1	1
1	X	0
0	1	1

A connectivity detector shades the output image with 1s to indicate the position of those points which are critical for connectivity and which were white in the input image. Black points, and those which are not critical for connectivity, are mapped to black in the output image.

Euler number [eul]

The Euler number is defined as the number of connected components (blobs) minus the number of holes in a binary image. The Euler number represents a simple method of counting blobs in a binary image, provided they have no holes in them. Alternatively, it can be used to count holes in a given object, providing they have no “islands” in them. The reason why this approach is used to count blobs, despite the fact that it may seem a little awkward to use, is that the Euler number is

very easy and fast to calculate. It is also a useful means of classifying shapes in an image. The Euler number can be computed by using three local operators. Let us define three numbers $N1$, $N2$ and $N3$, where $N\alpha$ indicates the number of times that one of the patterns in the pattern set α ($\alpha = 1, 2$ or 3) occur in the input image.

0	0	0	0	1	0	0	1
0	1	1	0	0	0	0	0

Pattern set 1 ($N1$)

0	1	1	0
1	0	0	1

Pattern set 2 ($N2$)

1	1	1	1	0	1	1	0
1	0	0	1	1	1	1	1

Pattern set 3 ($N3$)

The *8-connected* Euler number, where holes and blobs are defined in terms of 8-connected figures, is defined as: $(N1 - 2.N2 - N3)/4$. It is possible to calculate the *4-connected* Euler number using a slightly different formula, but this parameter can give results which seem to be anomalous when we compare them to the observed number of holes and blobs.

Filling holes [blb]

Consider a white blob-like figure containing a hole (lake), against a black background. The application of the hole-filling operator will cause all of the holes to be *filled-in*; by setting all pixels in the holes to white. This operator will not alter the outer edge of the figure.

Region labelling [ndo]

Consider an image containing a number of separate blob-like figures. A region-labelling operator will shade the output image so that each blob is given a separate intensity value. We could shade the blobs according to the order in which they are found, during a conventional raster scan of the input image. Alternatively, the blobs could be shaded according to their areas; the biggest blobs becoming the brightest. This is a very useful operator, since it allows objects to be separated and analysed individually (Figure 2.8). Small blobs can also be eliminated from an image using this operator. Region labelling can also be used to count the number of distinct binary blobs in an image. Unlike the Euler number, counting based on region labelling is not effected by the presence of holes.

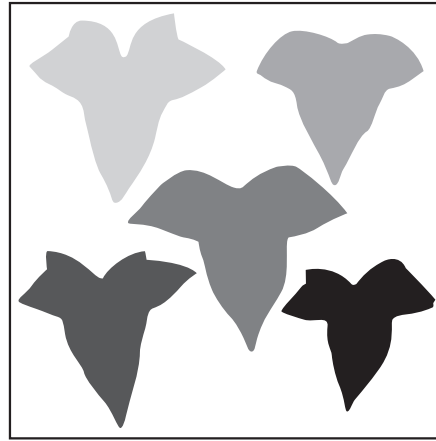


Figure 2.8. Shading blobs in a binary according to the order in which they are found during a raster scan (left to right; top to bottom).

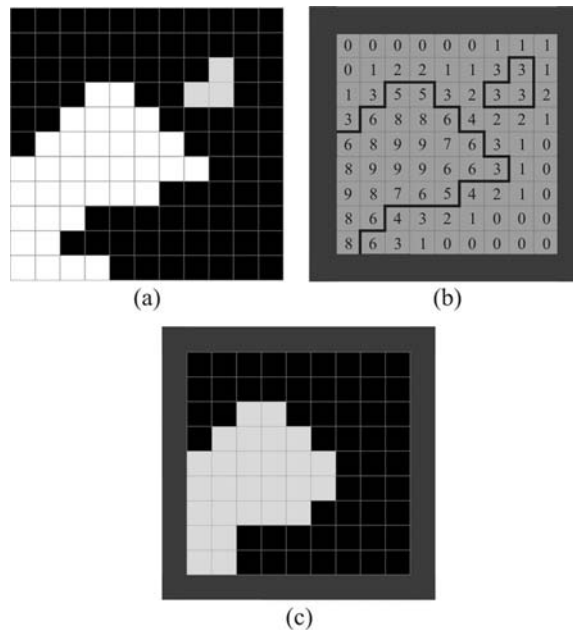


Figure 2.9. Using a grey-scale low pass (blurring) filter to remove noise from a binary image (a) Original image. (b) Applying a local averaging filter. Integers represent the number of white pixels within the 3×3 neighbourhood. (c) Thresholding the filtered image at level 5.

Other methods of detecting/removing small spots

A binary image can be represented in terms of a grey-scale image in which only two grey levels, 0 and W, are allowed. The result of the application of a conventional low-pass (blurring) filter to such an image is a grey-scale image in which there is a larger number of possible intensity values. Pixels which were well

inside large white areas in the input image are mapped to very bright pixels in the output image. Pixels which were well inside black areas are mapped to very dark pixels in the output image. However, pixels which were inside small white spots in the input image are mapped to mid-grey intensity levels (Figure 2.9). Pixels on the edge of large white areas are also mapped to mid-grey intensity levels. However, if there is a cluster of small spots, which are closely spaced together, some of them may also disappear.

Based on these observations, the following procedure has been developed. It has been found to be effective in distinguishing between small spots and, at the same time, achieving a certain amount of edge smoothing of the large bright blobs which remain:

```
raf(11,11),           % Low-pass filter using a 11×11 local operator
thr(128),             % Threshold at mid-grey
```

This technique is generally easier and faster to implement than the blob shading technique described previously. Although it may not achieve the desired result exactly, it can be performed at high speed.

An N-tuple filter having the weight matrix illustrated below can be combined with simple thresholding to distinguish between large and small spots. Assume that there are several small white spots within the input image and that they are spaced well apart. All pixels within a spot which can be contained within a circle of radius three pixels will be mapped to white by this particular filter. Pixels within a larger spot will become darker than this. The image is then thresholded at white to separate the large and small spots.

			-1	-1	-1			
		-1				-1		
	-1						-1	
-1								-1
-1				20				-1
-1								-1
	-1						-1	
		-1				-1		
			-1	-1	-1			

Grass-fire transform and skeleton [gfa, mdl, mid]

Consider a binary image containing a single white blob (Figure 2.10). Imagine that a fire is lit at all points around the blob's outer edge and the edges of any holes it may contain. The fire will burn inwards, until at some instant, advancing fire lines meet. When this occurs, the fire becomes extinguished locally. An output image is

generated and is shaded in proportion to the time it takes for the fire to reach each point. Background pixels are mapped to black.

The importance of this transform, referred to as the *grass-fire* transform, lies in the fact that it indicates distances to the nearest edge point in the image [1]. It is therefore possible to distinguish thin and fat limbs of a white blob. Those points at which the fire lines meet are known as *quench* points. The set of quench points form a “match-stick” figure, usually referred to as a *skeleton* or *medial axis transform*. These figures can also be generated in a number of different ways [2] (Figure 2.11).

One such approach is described as *onion-peeling*. Consider a single white blob and a “bug” which walks around the blob’s outer edge, removing one pixel at a time. No edge pixel is removed if by doing so we would break the blob into two disconnected parts. In addition, no white pixel is removed if there is only one white pixel amongst its 8-neighbours. This simple procedure leads to an undesirable effect in those instances when the input blob has holes in it; the skeleton which it produces has small loops in it which fit around the holes like a tightened noose. More sophisticated algorithms have been devised which avoid this problem.

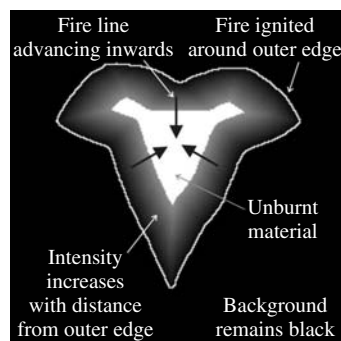


Figure 2.10. Grass-fire transform.

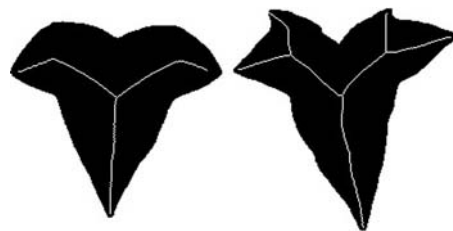


Figure 2.11. Application of the Medial Axis Transform.

Edge smoothing and corner detection

Consider three points B1, B2 and B3, which are placed close together on the edge of a single blob in a binary image (Figure 2.12). The perimeter distance between B1 and B2 is equal to that between B2 and B3. Define the point P to be that at the centre of the line joining B1 and B3. As the three points now move around the

edge of the blob, keeping the spacing between them constant, the locus of P traces a smoother path than that followed by $B2$ as it moves around the edge. This forms the basis of a simple edge smoothing procedure. A related algorithm, for corner detection, shades the edge according to the distance between P and $B2$. This results in an image in which the corners are highlighted, while the smoother parts of the image are much darker.

Many other methods of edge smoothing are possible. For example, we may map white pixels which have fewer than, say, three white 8-neighbours to black. This has the effect of eliminating “hair” around the edge of a blob-like figure. One of the techniques described previously for eliminating small spots offers another possibility. A third option is to use the processing sequence: $[exw, skw, skw, exw]$, where exw represents expand white areas and skw denotes shrink white areas.

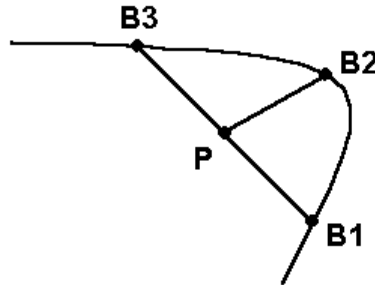


Figure 2.12. Edge smoothing and corner detection.

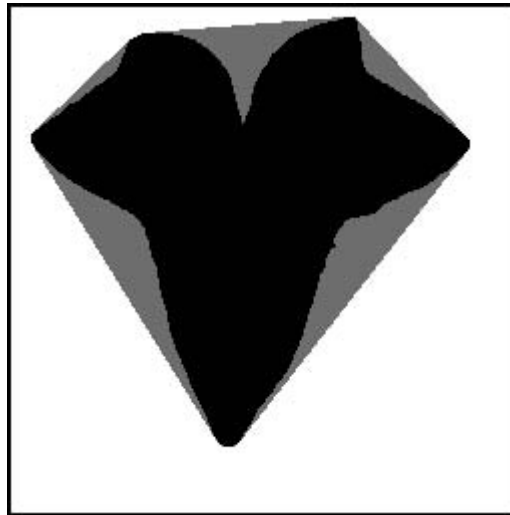


Figure 2.13. Convex hull of an ivy leaf. The lightly shaded region indicates the shape's convex deficiency.

Convex hull [chu]

Consider a single blob in a binary image. The convex hull is that area enclosed within the smallest convex polygon which will enclose the shape (Figure 2.13). This can also be described as the region enclosed within an elastic string, stretched around the blob. The area enclosed by the convex hull, but not within the original blob is called the *convex deficiency*, which may consist of a number of disconnected parts, and includes any holes and indentations. If we regard the blob as being like an *island*, we can understand the logic of referring to the former as *lakes* and the latter as *bays*.

2.3.1 Measurements on Binary Images

To simplify the following explanation, we will confine ourselves to the analysis of a binary image containing a single blob. The area of the blob can be measured by the total number of object (white) pixels in the image. However, we must first define two different types of edge points, in order to measure an object's perimeter.

The *4-adjacency* convention (Figure 2.14) only allows the four main compass points to be used as direction indicators, while *8-adjacency* uses all eight possible directions. If 4-adjacency convention is applied to the image segment given in Figure 2.14c, then none of the four segments (two horizontal and two vertical) will appear as touching, i.e., they are not connected. Using the 8-adjacency convention, the segments are now connected, but we have the ambiguity that the inside of the shape is connected to the outside. Neither convention is satisfactory, but since 8-adjacency allows diagonally connected pixels to be represented, it leads to a more faithful perimeter measurement.

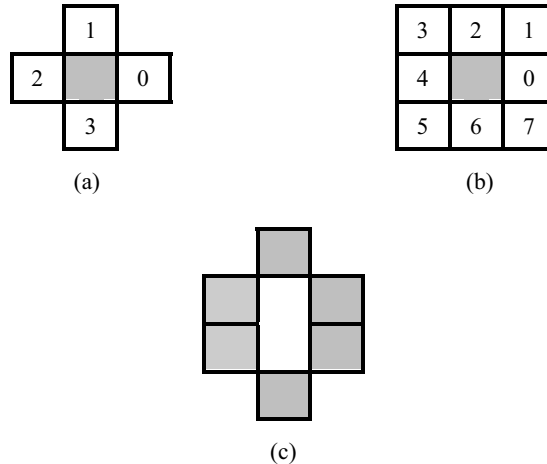


Figure 2.14. Chain code: (a) 4-adjacency coding convention; (b) 8-adjacency coding convention; (c) image segment.

Assuming that the 8-adjacency convention is used, we can generate a coded description of the blob's edge. This is referred to as the *chain code* or *Freeman*

code [*fcc*]. As we trace around the edge of the blob, we generate a number, 0–7, to indicate which of the eight possible directions we have taken (i.e., from the centre, shaded pixel in Figure 2.14(b)). Let N_o indicate how many *odd-numbered* code values are produced as we code the blob's edge, and N_e represent the number of *even-numbered* values found. The *perimeter* of the blob is given *approximately* by the formula: $N_e + \sqrt{2}.N_o$

This formula will normally suffice for use in those situations where the perimeter of a smooth object is to be measured. The *centroid* of a blob [*cgr*] determines its position within the image and can be calculated using the formulae:

$$I \leftarrow \sum_j \sum_i (a(i,j).i) / N_{i,j} \quad \text{and} \quad J \leftarrow \sum_j \sum_i (a(i,j).j) / N_{i,j}$$

where $N_{i,j} \leftarrow \sum_j \sum_i a(i,j)$

Although we are considering images in which the $a(i,j)$ are equal to 0 (black) or 1 (white), it is convenient to use $a(i,j)$ as an ordinary arithmetic variable as well.

2.3.2 Shape Descriptors

The following are just a few of the numerous shape descriptors that have been proposed:

- a. the distance of the furthest point on the edge of the blob from the centroid;
- b. the distance of the closest point on the edge of the blob from the centroid;
- c. the number of protuberances, as defined by that circle whose radius is equal to the average of the parameters measured in a. and b.;
- d. the distances of points on the edge of the blob from the centroid, as a function of angular position. This describes the silhouette in terms of polar co-ordinates. (This is not a single-valued function.);
- e. Circularity = $\text{Area} / \text{Perimeter}^2$. This will tend to zero for irregular shapes with ragged boundaries, and has a maximum value ($=1/4\pi$) for a circle;
- f. the number of holes. (Use *eul* and *ndo* to count them.);
- g. the number of bays;
- h. Euler number;
- i. the ratio of the areas of the original blob and that of its convex hull;
- j. the ratio of the areas of the original blob and that of its circumcircle;
- k. the ratio of the area of the blob to the square of the total limb-length of its skeleton;
- l. distances between joints and limb ends of the skeleton;
- m. the ratio of the projections onto the major and minor axes.

2.4 Binary Mathematical Morphology

The basic concept involved in mathematical morphology is simple: an image is probed with a template shape, called a structuring element, to find where the structuring element fits, or does not fit within a given image [3]. (Figure 2.15). By marking the locations where the template shape fits, structural information about the image can be gleaned. The structuring elements used in practice are usually geometrically simpler than the image they act on, although this is not always the case. Common structuring elements include points, point pairs, vectors, lines, squares, octagons, discs, rhombi and rings. Since shape is a prime carrier of information in machine vision applications, mathematical morphology has an important role to play in industrial systems [4].

The language of binary morphology is derived from that of set theory [5]. General mathematical morphology is normally discussed in terms of Euclidean N -space, but in digital image analysis we are only interested in a discrete or digitised equivalent in two-space. The following analysis is therefore restricted to binary images, in a digital two-dimensional integer space, Z^2 . The image set (or scene) under analysis will be denoted by A , with elements $a = (a1, a2)$. The shape parameter, or structuring element, that will be applied to scene A will be denoted by B , with elements $b = (b1, b2)$. The primary morphological operations that we will examine are dilation, erosion, opening and closing.

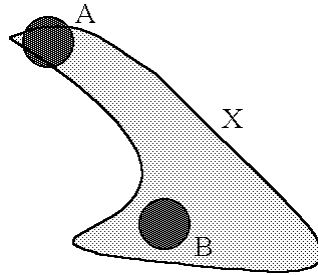


Figure 2.15. A structuring element fitting, B , and not fitting, A , into a given image scene X [3].

Dilation

Dilation (also referred to as filling and growing) is the expansion of an image set A by a structuring element B . It is formally viewed as the combination of the two sets using vector addition of the set elements. The dilation of an image set A by a structuring element B , will be denoted $A \# B$, and can be represented as the union of translates of the structuring element B [5]:

$$A \# B = \bigcup_{a \in A} B_a$$

where \bigcup represents the union of a set of points and the translation of B by point a is given by, $B_a = \{c \in \mathbb{Z}^2 \mid c = b + a \text{ for some } b \in B\}$. This is best explained by visualising a structuring element B moving over an image A in a raster fashion. Whenever the origin of the structuring element touches one of the image pixels in A , then the entire structuring element is placed at that location. For example, in Figure 2.16 the grid image is dilated by a cross-shaped structuring element, contained within a 3×3 pixel grid.

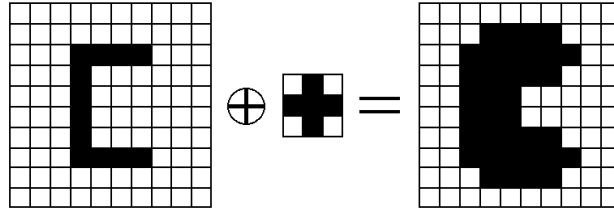


Figure 2.16. Dilation of a grid image by a cross structuring element.

Erosion

Erosion is the dual morphological operation of dilation and is equivalent to the shrinking (or reduction) of the image set A by a structuring element B . This is a morphological transformation, which combines two sets using vector subtraction of set elements [5]. The erosion of an image set A by a structuring element B , denoted $A \circ B$, can be represented as the intersection of the negative translates:

$$A \circ B = \bigcap_{b \in B} A_{-b}$$

where \bigcap represents the intersection of a set of points. Erosion of the image A by B is the set of all points for which B translated to a point x is contained in A . This consists of sliding the structuring element B across the image A , and where B is fully contained in A (by placing the origin of the structuring element at the point x) then x belongs to the eroded image $A \circ B$. For example, in Figure 2.17 the grid image is eroded by a cross-shaped structuring element, contained within a 3×3 pixel grid.

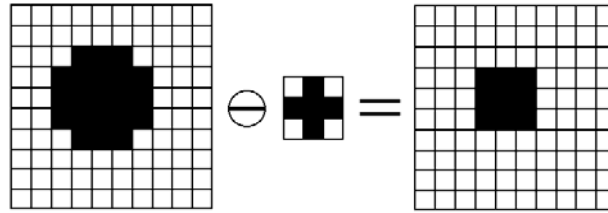


Figure 2.17. Erosion of a grid image by a cross structuring element

A duality relationship exists between certain morphological operators, such as erosion and dilation. This means that the equivalent of such an operation can be performed by its dual on the complement (negative) image and by taking the complement of the result [6]. Although duals, erosion and dilation operations are not inverses of each other. Rather they are related by the following duality relationships:

$$(A \ominus B)^c = A^c \oplus \bar{B} \quad \text{and} \quad (A \oplus B)^c = A^c \ominus \bar{B}$$

Where A^c refers to the complement of the image set A and $\bar{B} = \{x \mid \text{for some } b \in B, x = -b\}$ refers to the reflection of B about the origin. (Serra [7,8] refers to this as the *transpose* of the structuring element.)

2.4.1 Opening and Closing Operations

Erosion and dilation tend to be used in pairs to extract, or impose, structure on an image. The most commonly found erosion-dilation pairings occur in the *opening* and *closing* transformations.

Opening

Opening is a combination of erosion and dilation operations that have the effect of removing isolated spots in the image set A that are smaller than the structuring element B and those sections of the image set A narrower than B . This is also viewed as a geometric *rounding* operation (Figure 2.18). The opening of the image set A by the structuring element B , is denoted $A \circ B$, and is defined as $(A \ominus B) \oplus B$.

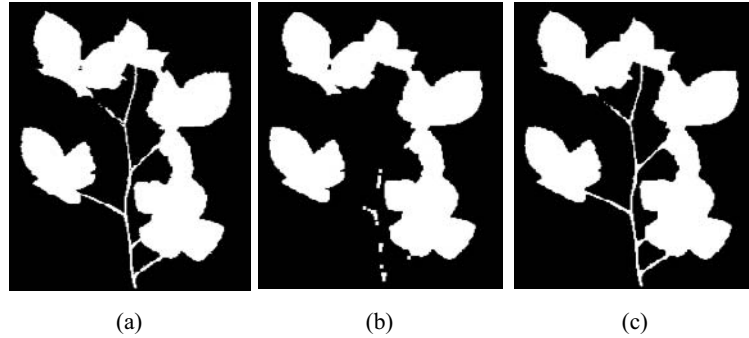


Figure 2.18. Application of a 3×3 square structuring element to a binary image of a small plant: (a) original image; (b) result of morphological opening; (c) result of morphological closing.

Closing

Closing is the dual morphological operation of opening. This transformation has the effect of filling in holes and blocking narrow valleys in the image set A , when a structuring element B (of similar size to the holes and valleys) is applied (Figure 2.18). The closing of the image set A by the structuring element B , is denoted $A \bullet B$, and is defined as $(A \oplus B) \ominus B$.

One important property that is shared by both the opening and closing operations is *idempotency*. This means that successful reapplication of the operations will not change the previously transformed image [4]. Therefore, $A \circ B = (A \circ B) \circ B$ and $A \bullet B = (A \bullet B) \bullet B$.

Unfortunately, the application of morphological techniques to industrial tasks, which involves complex operations on “real-world” images, can be difficult to implement. Practical imaging applications tend to have structuring elements that are unpredictable in shape and size. In practice, the ability to manipulate arbitrary structuring elements usually relies on their decomposition into component parts.

2.4.2 Structuring Element Decomposition

Some vision systems [9,10,11] can perform basic morphological operations very quickly in a parallel and/or pipelined manner. Implementations that involve such special-purpose hardware tend to be expensive, although there are some notable exceptions [12]. Unfortunately, some of these systems impose restrictions on the shape and size of the structuring elements that can be handled. Therefore, one of the key problems involved in the application of morphological techniques to industrial image analysis is the generation and/or decomposition of large structuring elements. Two main strategies are used to tackle this problem.

The first technique is called *dilation* or *serial decomposition*. This decomposes certain large structuring elements into a sequence of successive erosion and dilation operations, each step operating on the preceding result. Unfortunately, the decomposition of large structuring elements into smaller ones is not always

possible. Also, those decompositions that are possible are not always easy to identify and implement.

If a large structuring element B can be decomposed into a chain of dilation operations, $B = B_1 \oplus B_2 \oplus \dots \oplus B_N$ (Figure 2.19), then the dilation of the image set A by B is given by:

$$A \oplus B = A \oplus (B_1 \oplus B_2 \oplus \dots \oplus B_N) = (((A \oplus B_1) \oplus B_2) \dots) \oplus B_N.$$

Similarly, using the so-called chain rule [13], which states that $A \ominus (B \oplus C) = (A \ominus B) \ominus C$, the erosion of A by B is given by:

$$A \ominus B = A \ominus (B_1 \oplus B_2 \oplus \dots \oplus B_N) = (((A \ominus B_1) \ominus B_2) \dots) \ominus B_N$$

A second approach to the decomposition problem is based on “breaking up” the structuring element, B , into a union of smaller components, B_1, \dots, B_N . We can think of this approach as ‘tiling’ of the structuring element by sub-structuring elements (Figure 2.20). Since the ‘tiles’ do not need to be contiguous or aligned, any shape can be specified without the need for serial decomposition of the structuring element, although the computational cost of this approach is proportional to the area of the structuring element [11]. This is referred to as *union* or *parallel decomposition*. Therefore, with B decomposed into a union of smaller structuring elements, $B = B_1 \cup B_2 \cup \dots \cup B_N$, then the dilation of an image A by the structuring element B can be rewritten as:

$$\begin{aligned} A \oplus B &= A \oplus (B_1 \cup B_2 \cup \dots \cup B_N) \\ &= (A \oplus B_1) \cup (A \oplus B_2) \cup \dots \cup (A \oplus B_N) \end{aligned}$$

Likewise, the erosion of A by the structuring element B can be rewritten as:

$$\begin{aligned} A \ominus B &= A \ominus (B_1 \cup B_2 \cup \dots \cup B_N) \\ &= (A \ominus B_1) \cap (A \ominus B_2) \cap \dots \cap (A \ominus B_N) \end{aligned}$$

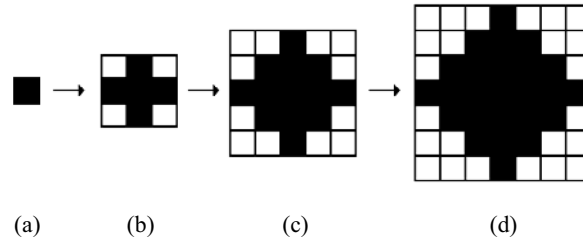


Figure 2.19. Construction of a 7×7 structuring element by successive dilation of a 3×3 structuring element: (a) initial pixel; (b) 3×3 structuring element and the result of the first dilation; (c) result of the second dilation; (d) result of the third dilation [11]

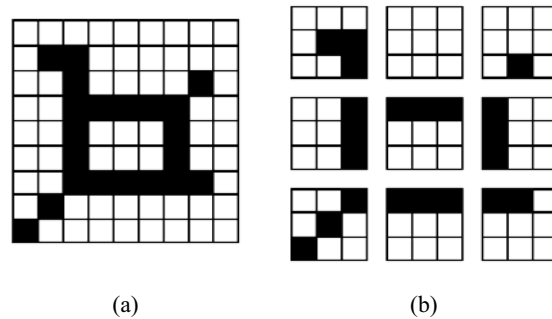


Figure 2.20. Tiling of a 9×9 arbitrary structuring element: (a) the initial 9×9 structuring element; (b) tiling with nine 3×3 sub-structuring elements [11].

This makes use of the fact that $A \ominus (B \cup C) = (A \ominus B) \cap (A \ominus C)$ [4]. Due to the nature of this decomposition procedure, it is well suited to implementation on parallel computer architectures.

Waltz [11] compared these structural element decomposition techniques, and showed that the serial approach has a 9:4 speed advantage over its parallel equivalent. (This was based on an arbitrarily specified 9×9 pixel-structuring element, when implemented on a commercially available vision system.) However, the parallel approach has a 9:4 advantage in the number of degrees of freedom. (Every possible 9×9 -structuring element can be achieved with the *parallel decomposition*, but only a small subset can be realised with the serial approach.) Although slower than the serial approach, it has the advantage that there is no need for serial decomposition of the structuring element.

Classical parallel and serial methods mainly involve the numerous scanning of image pixels and are therefore inefficient when implemented on conventional computers. This is so, because the number of scans depends on the total number of pixels (or edge pixels) in the shape to be processed by the morphological operator. Although the parallel approach is suited to some customised (parallel) architectures, the ability to implement such parallel approaches on *serial* machines is discussed by Vincent [14].

2.5 Grey-scale Morphology

Binary morphological operations can be extended naturally to process grey-scale imagery, by the use of neighbourhood minimum and maximum functions [4]. Heijmans [15], presents a detailed study of grey-scale morphological operators, in which he outlines how binary morphological operators and thresholding techniques can be used to build a large class of useful grey-scale morphological operators. Sternberg [16], discusses the application of such morphological techniques to industrial inspection tasks.

In Figure 2.21, a one-dimensional morphological filter, operates on an analogue signal (equivalent to a grey-scale image). The input signal is represented by the

thin curve and the output by the thick black curve. In this simple example, the structuring element has an approximately parabolic form. In order to calculate a value for the output signal, the structuring element is pushed upwards, from below the input curve. The height of the top of the structuring element is noted. This process is then repeated, by sliding the structuring element sideways. Notice how this particular operator attenuates the intensity peak but follows the input signal quite accurately everywhere else. Subtracting the output signal from the input would produce a result in which the intensity peak is emphasised and all other variations would be reduced.

The effect of the basic morphological operators on two-dimensional grey-scale images can also be explained in these terms. Imagine the grey-scale image as a landscape, in which each pixel can be viewed in 3-D. The extra height dimension represents the grey-scale value of a pixel. We generate new images by passing the structuring element above/below this landscape. (See Figure 2.21.)

Grey-scale dilation

This is computed as the maximum of translations of the grey surface. Grey-level dilation of image A by the structuring element B produces an image C defined by:

$$C(r,c) = \text{Max}_{(i,j)} \{A(r-i, c-j) + B(i,j)\} = (A \oplus B)(r,c)$$

where A , B and C are grey level images. Commonly used grey-level structuring elements include rods, discs, cones and hemispheres. This operation is commonly used to smooth small negative contrast grey-level regions in an image.

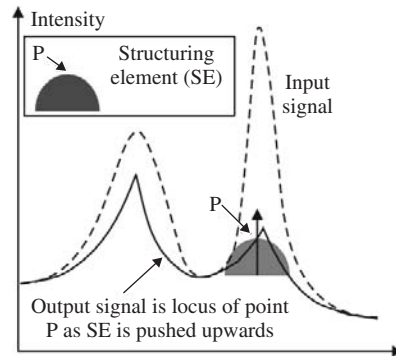


Figure 2.21. A 1-dimensional morphological filter, operating on an analogue signal.

Grey-scale erosion

The grey value of the erosion at any point is the *maximum* value for which the structuring element centred at that point still fits entirely within the foreground under the surface. This is computed by taking the *minimum* of the grey surface

translated by all the points of the structuring element (Figure 2.21). Grey-level erosion of image A by the structuring element B produces an image C defined by:

$$C(r,c) = \text{Min}_{(i,j)} \{A(r+i, c+j) - B(i,j)\} = (A \ominus B)(r,c)$$

This operation is commonly used to smooth small positive contrast grey level regions in an image.

Grey-scale opening

This operation is defined as the grey level erosion of the image followed by the grey-level dilation of the eroded image. That is, it will cut down the peaks in the grey-level topography to the highest level for which the elements fit under the surface.

Grey-scale closing

This operation is defined as the grey-level dilation of the image followed by the grey-level erosion of the dilated image. Closing fills in the valleys to the maximum level for which the element fails to fit above the surface. For a more detailed discussion on binary and grey-scale mathematical morphology, see Haralick and Shapiro [17] and Dougherty [3].

2.6 Global Image Transforms

An important class of image processing operators is characterised by an equation of the form $B \Leftarrow f(A)$, where $A = \{a(i,j)\}$ and $B = \{b(p,q)\}$. Each element in the output picture, B , is calculated using all or, at least a large proportion of the pixels in A . The output image B may well look quite different from the input image A . Examples of this class of operators are: lateral shift, rotation, warping, Cartesian to polar co-ordinate conversion, Fourier and Hough transforms.

Integrate intensities along image rows [rin]

This operator is rarely of great value when used on its own, but can be used with other operators to good effect, for example detecting horizontal streaks and edges. The operator is defined recursively:

$$b(i,j) \Leftarrow b(i,j-1) + a(i,j)/n \text{ where } b(0,0) = 0$$

Row maximum [rox]

This function is often used to detect local intensity minima: $c(i,j) \Leftarrow \text{MAX}(a(i,j), c(i,j-1))$

Geometric transforms

Algorithms exist by which images can be shifted [*psh*], rotated [*tur*], undergo axis conversion [*ctr*, *rtc*], magnified [*pex* and *psq*] and warped. The reader should note that certain operations, such as rotating a digital image, can cause some difficulties

because pixels in the input image are not mapped exactly to pixels in the output image. This can cause smooth edges to appear stepped. To avoid this effect, interpolation may be used, but this has the unfortunate effect of blurring edges. (See [18] for more details.)

The utility of axis transformations is evident when we are confronted with the examination of circular objects, or those displaying a series of concentric arcs, or streaks radiating from a fixed point. Inspecting such objects is often made very much easier, if we first convert from *Cartesian* to *polar* co-ordinates. Warping is also useful in a variety of situations. For example, it is possible to compensate for *barrel*, or *pincushion* distortion in a camera. Geometric distortions introduced by a wide-angle lens, or trapezoidal distortion due to viewing the scene from an oblique angle can also be corrected. Another possibility is to convert simple curves of known shape into straight lines, in order to make subsequent analysis easier.

2.6.1 Hough Transform

The Hough transform provides a powerful and robust technique for detecting lines, circles, ellipses, parabolas, and other curves of pre-defined shape, in a binary image. Let us begin our discussion of this fascinating topic, by describing the simplest version, the basic Hough Transform, which is intended to detect straight lines. Actually, our objective is to locate *nearly linear* arrangements of disconnected white spots and “*broken*” lines. Consider that a straight line in the input image is defined by the equation $r = x \cos \phi + y \sin \phi$, where r and ϕ are two unknown parameters, whose values are to be found. Clearly, if this line intersects the point (x_i, y_i) , then $r = x_i \cos \phi + y_i \sin \phi$ can be solved for many different values of (r, ϕ) . So, each white point (x_i, y_i) in the input image may be associated with a *set* of (r, ϕ) values. Actually, this set of points forms a *sinusoidal curve* in (r, ϕ) space. (The latter is called the *Hough Transform (HT)* image.) Since each point in the input image generates such a sinusoidal curve, the whole of that image creates a multitude of overlapping sinusoids, in the HT image. In many instances, a large number of sinusoidal curves are found to converge on the same spot in the HT image. The (r, ϕ) address of such a point indicates the slope, ϕ , and position, r , of a straight line that can be drawn through a large number of white spots in the input image.

The implementation of the Hough transform for line detection begins by using a two-dimensional accumulator array, $A(r, \phi)$, to represent quantised (r, ϕ) space. (Clearly, an important choice to be made is the step size for quantising r and ϕ . However, we shall not dwell on such details here.) Assuming that all the elements of $A(r, \phi)$ are initialised to zero, the Hough Transform is found by computing a set $S(x_i, y_i)$ of (r, ϕ) pairs satisfying the equation $r = x_i \cos \phi + y_i \sin \phi$. Then, for all (r, ϕ) in $S(x_i, y_i)$, we increment $A(r, \phi)$ by one. This process is then repeated for all values of i such that the point (x_i, y_i) in the input image is white. We repeat that bright spots in the HT image indicate “linear” sets of spots in the input image. Thus, line detection is transformed to the rather simpler task of finding local maxima in the accumulator array, $A(r, \phi)$. The co-ordinates (r, ϕ) of such a local

maximum give the parameters of the equation of the corresponding line in the input image. The HT image can be displayed, processed and analysed just like any other image, using the operators that are now familiar to us.

The robustness of the HT techniques arises from the fact that, if part of the line is missing, the corresponding peak in the HT image is simply darker. This occurs because fewer sinusoidal curves converge on that spot and the corresponding accumulator cell is incremented less often. However, unless the line is almost completely obliterated, this new darker spot can also be detected. In practice, we find that “near straight lines” are transformed into a cluster of points. There is also a spreading of the intensity peaks in the HT image, due to noise and quantisation effects. In this event, we may conveniently threshold the HT image and then find the centroid of the resulting spot, to calculate the parameters of the straight line in the input image. Pitas [19] and Davies [20] give a more detailed description of this algorithm. Figure 2.22 illustrates how this approach can be used to find a line in a noisy binary image.

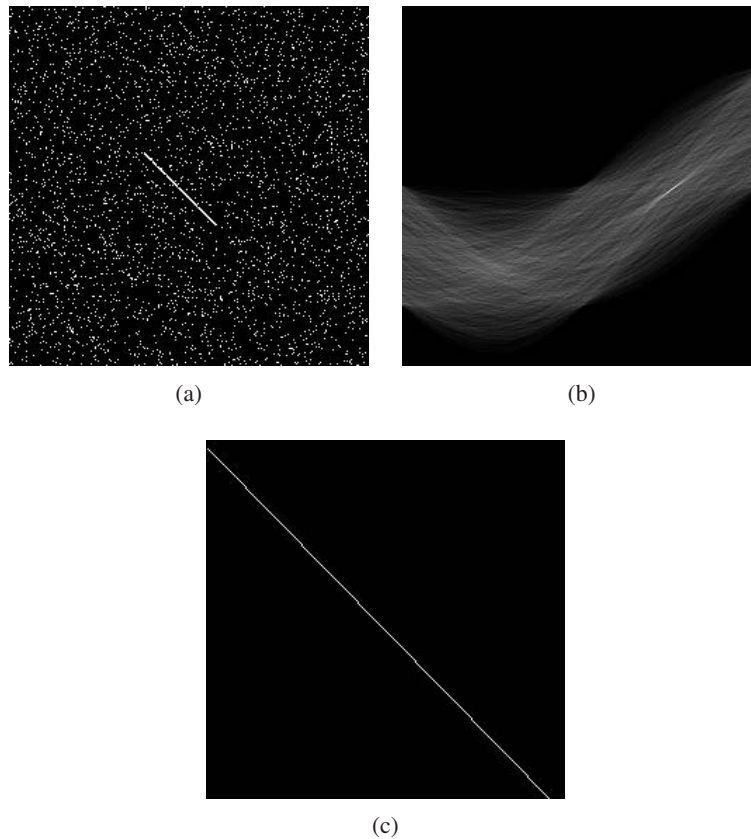


Figure 2.22. Hough transform: (a) original image; (b) Hough transform; (c) inverse Hough transform applied to a single white pixel located at the point of maximum intensity in (b). Notice how accurately this process locates the line in the input image, despite the presence of a high level of noise.

The Hough transform can also be generalised to detect groups of points lying on a curve. In practice, this may not be a trivial task, since the complexity increases very rapidly with the number of parameters needed to define the curve. For circle detection, we define a circle parametrically as:

$$r^2 = (x - a)^2 + (y - b)^2$$

where (a, b) determines the co-ordinates of the centre of the circle and r is its radius. This requires a three-dimensional parameter space, which cannot, of course, be represented and processed as a single image. For an arbitrary curve, with no simple equation to describe its boundary, a look-up table is used to define the relationship between the boundary co-ordinates an orientation and the Hough transform parameters. (See [21] for more details.)

2.6.2 Two-dimensional Discrete Fourier Transform

We have just seen how the transformation of an image into a different domain can sometimes make the analysis task easier. Another important operation to which this remark applies is the Fourier Transform. Since we are discussing the processing of images, we shall discuss the *two-dimensional Discrete Fourier Transform*. This operation allows spatial periodicities in the intensity within an image to be investigated, in order to find, amongst other features, the dominant frequencies. The two-dimensional Discrete Fourier Transform of an $N \times N$ image $f(x, y)$ is defined as follows: [22]

$$F(u, v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x, y) \exp[-j2\pi (ux + vy) / N]$$

where $0 \leq u, v \leq N-1$. The inverse transform of $F(u, v)$ is defined as:

$$f(x, y) = \frac{1}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u, v) \exp[j2\pi (ux + vy) / N]$$

where $0 \leq x, y \leq N-1$.

Several algorithms have been developed to calculate the two-dimensional Discrete Fourier Transform. The simplest makes use of the observation that this is a *separable transform* which can be computed as a sequence of two one-dimensional transforms. Therefore, we can generate the two-dimensional transform by calculating the one-dimensional Discrete Fourier Transform along the image rows and then repeating this on the resulting image but, this time, operating on the columns [22]. This reduces the computational overhead when compared to direct two-dimensional implementations. The sequence of operations is as follows:

$$f(x, y) \rightarrow \text{Row Transform} \rightarrow F_1(x, v) \rightarrow \text{Column Transform} \rightarrow F_2(u, v)$$

Although this is still computationally slow compared to other many shape measurements, the Fourier transform is quite powerful. It allows the input to be represented in the frequency domain, which can be displayed as a *pair* of images. (It is not possible to represent both amplitude and phase using a single monochrome image.) Once the processing within the frequency domain is complete, the inverse transform can be used to generate a new image in the original, so-called, *spatial* domain.

The Fourier power, or amplitude, spectrum plays an important role in image processing and analysis. This can be displayed, processed and analysed as an intensity image. Since the Fourier transform of a real function produces a complex function: $F(u,v) = R(u,v) + iI(u,v)$, the frequency spectrum of the image is the magnitude function

$$|F(u,v)| = \sqrt{R^2(u,v) + I^2(u,v)}$$

and the power spectrum (spectral density) is defined as $P(u,v) = |F(u,v)|^2$.

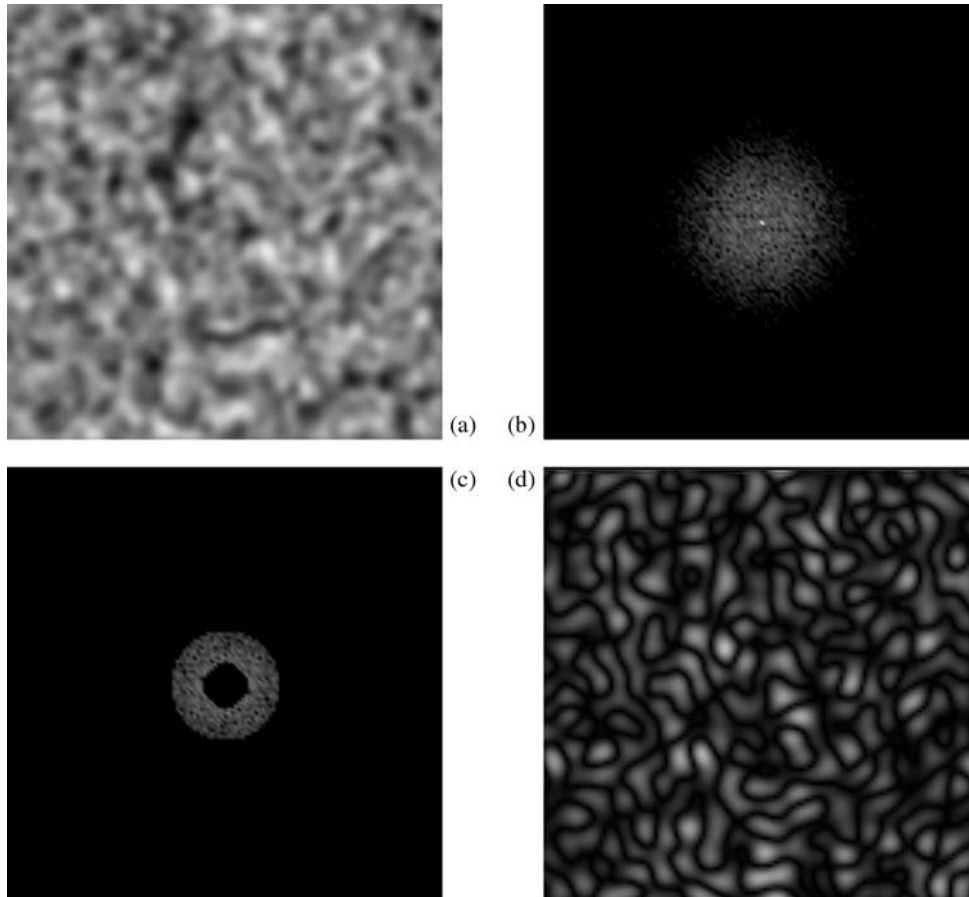


Figure 2.23. Filtering a textured image in the frequency domain: (a) original textured image; (b) resultant transformed image in the frequency domain after using the two-dimensional Discrete Fourier Transform (the image is the frequency spectrum shown as an intensity function); (c) resultant frequency domain image after an ideal band-pass filter is applied to image; (d) the resultant spatial domain image after the inverse two-dimensional discrete Fourier transform is applied to the band-pass filtered image in (c).

Figure 2.23 illustrates how certain textured features can be highlighted using the two-dimensional Discrete Fourier Transform. The image is transformed into the frequency domain and an ideal band-pass filter (with a circular symmetry) is applied. This has the effect of limiting the frequency information in the image. When the inverse transform is calculated, the resultant textured image has a different frequency content which can then be analysed. For more details on the Fourier transform and its implementations, see [19] and [22].

2.7 Texture Analysis

Texture is observed in the patterns of a wide variety of synthetic and natural surfaces (e.g., wood, metal, paint and textiles). If an area of a textured image has a large *intensity* variation then the dominant feature of that area would be *texture*. If this area has little variation in *intensity* then the dominant feature within the area is *tone*. This is known as the *tone-texture concept*. Although a precise formal definition of texture does not exist, it may be described subjectively using terms such as *coarse*, *fine*, *smooth*, *granulated*, *rippled*, *regular*, *irregular* and *linear*, and of course these features are used extensively in manual region segmentation. There are two main classification techniques for texture: *statistical* and *structural*.

2.7.1 Statistical Approaches

The statistical approach is well suited to the analysis and classification of random or natural textures. A number of different techniques have been developed to describe and analyse such textures [23], a few of which are outlined below.

Auto-correlation Function (ACF)

Auto-correlation derives information about the basic 2-D tonal pattern that is repeated to yield a given periodic texture. Although useful at times, the ACF has severe limitations. It cannot always distinguish between textures, since many subjectively different textures have the same ACF, which is defined as follows:

$$A(\delta x, \delta y) = \sum_{i,j} [I(i,j) \cdot I(i + \delta x, j + \delta y)] / \sum_{i,j} [I(i,j)]^2$$

where $\{I(i,j)\}$ is the image matrix. The variables (i,j) are restricted to lie within a specified window outside which the intensity is zero. Incremental shifts of the image are given by $(\delta x, \delta y)$. It is worth noting that the ACF and the power spectral density are Fourier transforms of each other.

Fourier spectral analysis

The Fourier spectrum is well suited to describing the directionality and period of repeated texture patterns, since they give rise to high-energy narrow peaks in the power spectrum (Section 2.6 and Figure 2.23). Typical Fourier descriptors of the power spectrum include: the location of the highest peak, mean, and variance and the difference in frequency between the mean and the highest value of the spectrum. This approach to texture analysis is often used in aerial/satellite and medical image analysis. The main disadvantage of this approach is that the procedures are not invariant even, under monotonic transforms of its intensity.

Edge density

This is a simple technique in which an edge detector or high-pass filter is applied to the textured image. The result is then thresholded and the edge density is

measured by the average number of edge pixels per unit area. Two-dimensional, or directional filters/edge detectors may be used as appropriate.

Histogram features

This useful approach to texture analysis is based on the intensity histogram of all or part of an image. Common histogram features include: *moments*, *entropy* *dispersion*, *mean* (an estimate of the average intensity level), *variance* (this second moment is a measure of the dispersion of the region intensity), *mean square value* or *average energy*, *skewness* (the third moment which gives an indication of the histogram's symmetry) and *kurtosis* (cluster prominence or "peakness"). For example, a narrow histogram indicates a low-contrast region, while two peaks with a well-defined valley between them indicates a region that can readily be separated by simple thresholding.

Texture analysis, based solely on the grey-scale histogram, suffers from the limitation that it provides no information about the relative position of pixels to each other. Consider two binary images, where each image has 50% black and 50% white pixels. One of the images might be a checkerboard pattern, while the second one may consist of a salt and pepper noise pattern. These images generate exactly the same grey-level histogram. Therefore, we cannot distinguish them using first-order (histogram) statistics alone. This leads us naturally to the examination of the co-occurrence approach to texture measurement.

2.7.2 Co-occurrence Matrix Approach

The co-occurrence matrix technique is based on the study of *second-order grey-level spatial dependency statistics*. This involves the study of the grey-level spatial interdependence of pixels and their spatial distribution in a local area. Second-order statistics describe the way grey levels tend to occur together, in pairs and therefore provide a description of the type of texture present. A *two-dimensional* histogram of the spatial dependency of the various grey-level picture elements within a textured image is created. While this technique is quite powerful, it does not describe the shape of the primitive patterns making up the given texture.

The co-occurrence matrix is based on the estimation of the second-order joint conditional probability density function, $f(p, q, d, a)$, for angular displacements, a , equal to 0, 45, 90 and 135 degrees. Let $f(p, q, d, a)$ be the probability of going from one pixel with grey level p to another with grey level q , given that the distance between them is d and the direction of travel between them is given by the angle a . (For N_g grey levels, the size of the co-occurrence matrix will be $N_g \cdot N_g$.) For example, assuming the intensity distribution shown in the sub-image given below, we can generate the co-occurrence matrix for $d = 1$ and a is taken as 0 degrees.

2	3	3	3
1	1	0	0
1	1	0	0
0	0	2	2
2	2	3	3

Sub-image with four grey-levels

Grey Scale	0	1	2	3
0	6	2	1	0
1	2	4	0	0
2	1	0	4	2
3	0	0	2	6

Co-occurrence matrix $\{f(p,q,1,0)\}$ for the sub-image

A co-occurrence distribution that changes rapidly with distance, d , indicates a fine texture. Since the co-occurrence matrix also depends on the image intensity range, it is common practice to normalise the textured image's grey scale prior to generating the co-occurrence matrix. This ensures that first-order statistics have standard values and avoids confusing the effects of first- and second-order statistics of the image.

A number of texture measures (also referred to as texture attributes) have been developed to describe the co-occurrence matrix numerically and allow meaningful comparisons between various textures [23] (Figure 2.24). Although these attributes are computationally intensive, they are simple to implement. Some sample texture attributes for the co-occurrence matrix are given below.

Energy

Energy, or angular second moment, is a measure of the homogeneity of a texture. It is defined thus,

$$Energy = \sum_p \sum_q [f(p,q,d,a)]^2$$

In a uniform image, the co-occurrence matrix will have few entries of large magnitude. In this case the *Energy* attribute will be large.

Entropy

Entropy is a measure of the complexity of a texture and is defined thus:

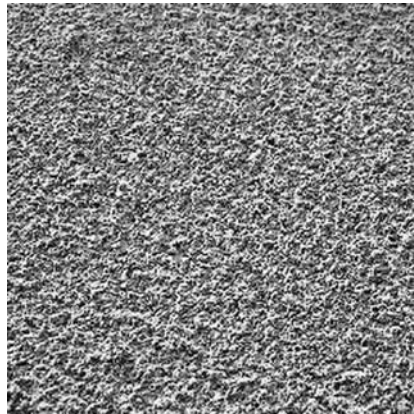
$$Entropy = - \sum_p \sum_q [f(p,q,d,a) \cdot \log(f(p,q,d,a))]$$

It is commonly found that what a person judges to be a complex image tends to have a higher *Entropy* value than a simple one.

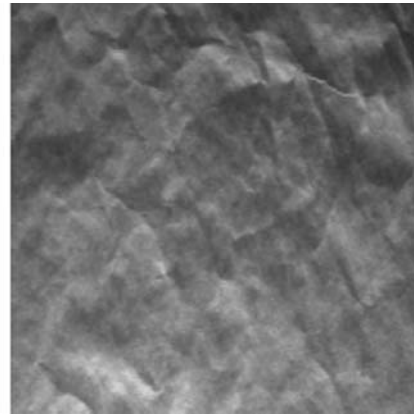
Inertia

Inertia is the measurement of the moment of inertia of the co-occurrence matrix about its main diagonal. This is also referred as the *contrast* of the textured image. This attribute gives an indication of the amount of local variation of intensity present in an image.

$$Inertia = \sum_p \sum_q [(p-q)^2 \cdot f(p,q,d,a)]$$



(a)



(b)

	Sand		Paper	
	$f(p,q,1,0)$	$f(p,q,1,90)$	$f(p,q,1,0)$	$f(p,q,1,90)$
Energy ($\times 10^6$)	1.63	1.7	3.49	3.42
Inertia ($\times 10^8$)	5.4	6.5	0.181	0.304

(c)

Figure 2.24. Co-occurrence based texture analysis: (a) sand texture; (b) paper texture; (c) texture attributes.

2.7.3 Structural Approaches

Certain textures are deterministic in that they consist of identical *texels* (basic texture element), which are placed in a repeating pattern according to some well-defined but unknown placement rules. To begin the analysis, a texel is isolated by identifying a group of pixels having certain invariant properties, which repeat in the given image. A texel may be defined by its: grey level, shape, or homogeneity of some local property, such as size or orientation. Texel spatial relationships may be expressed in terms of adjacency, closest distance and periodicities.

This approach has a similarity to language; with both image elements and grammar, we can generate a syntactic model. A texture is labelled *strong* if it is defined by deterministic placement rules, while a *weak* texture is one in which the texels are placed at random. Measures for placement rules include: edge density, run lengths of maximally connected pixels and the number of pixels per unit area showing grey levels that are locally maxima or minima relative to their neighbours

2.7.4 Morphological Texture Analysis

Textural properties can be obtained from the erosion process (Sections 2.4 and 2.5) by appropriately parameterising the structuring element and determining the number of elements of the erosion as a function of the parameters value [3]. The number of white pixels of the morphological opening operation as a function of the size parameter of the structuring element, H , can determine the size distribution of the *grains* in an image. Granularity of the image F is defined as:

$$G(d) = 1 - (\# [F \circ H_d] / \#F)$$

Where H_d is a disc structuring element of diameter d or a line structuring element of length d , and $\#F$ is the number of elements in F . This measures the proportion of pixels participating in grains smaller than d .

2.8 Implementation Considerations

Of course, all of the image processing and analysis operators that have been mentioned above can be implemented using a conventional programming language, such as C or C++. However, it is important to realise that many of the algorithms are time-consuming when realised in this way. The monadic, dyadic and local operators can all be implemented in time $K.m.n$ seconds, where K is a constant that is different for each function and (m,n) define the image resolution. However, some of the global operators require $O(m^2.n^2)$ time. With these points in mind, we see that a low-cost, slow but very versatile image processing system can be assembled, simply by embedding a *frame-grabber* into a conventional desk-top

computer. (A *frame-grabber* is a device for digitising video images and displaying computer-processed/generated images on a monitor.)

The monadic operators can be implemented using a look-up table, which can be realised simply in a ROM or RAM. The dyadic operators can be implemented using a straightforward *Arithmetic and Logic Unit* (ALU), which is a standard item of digital electronic hardware. The linear local operators can be implemented, nowadays, using specialised integrated circuits. One manufacturer sells a circuit board which can implement an 8×8 linear local operator in real time on a standard video signal. Several companies market a broad range of image processing modules that can be plugged together, to form a very fast image processing system that can be tailored to the needs of a given application. Specialised architectures have been devised for image processing. Among the most successful are parallel processors, which may process one row of an image at a time (vector processor), or the whole image (array processor). Competing with these are *systolic array*, *neural networks* and field programmable gate arrays. See Dougherty and Laplante [24] for a discussion on the considerations that need to be examined in the development of real-time imaging systems.

2.8.1 Morphological System Implementation

While no single image processing operation is so important that all others can be ignored, it is interesting to consider the implementation of the morphological operators, since it reflects the range of hardware and software techniques that can be applied to achieve high speed.

There are two classical approaches to the implementation of morphological techniques on computer architectures, parallel and sequential (serial) methods (Section 2.4). Morphological operations with 3×3 pixel structuring elements, are easily implemented by array architectures, such as CLIP [9]. Other system implementations include Sternberg [10]. Waltz [11,12] describes examples of a near real-time implementation of binary morphological processing using large (up to 50×50 pixels), arbitrary structuring elements, based on commercially available image processing boards. The success of this approach, referred to as SKIPSM (*Seperated-Kernal Image Processing using Finite State Machines*), was achieved by reformulating the algorithm in such a way that it permitted high-speed hardware implementation. Similar algorithmic methods allow fast implementation of these operators in software.

2.9 Commercial Devices

In this section, we discuss generic types of computing sub-systems for machine vision, rather than giving details of existing commercial products, since any review of current technology would become out of date quite quickly. The discussion will concentrate on the computing aspects of machine vision systems, rather than the

remaining systems issues, such as lighting and optics. Also, there are numerous trade magazines that deal with commercial machine vision products.

For the purposes of this book, we have classified commercial systems into three main categories:

- plug-in board-based systems (frame-stores, dedicated function);
- self-contained vision systems;
- turnkey systems.

2.9.1 Plug-in Boards: Frame-grabbers

The imaging engine in many low-cost machine vision systems consists of a host computer working in conjunction with single or multiple plug-in boards. The most common example of these systems consists of a personal computer, or workstation, and a frame-grabber card, which allows an image to be captured from a standard CCD camera (array image format) and displayed. Many of the current, extensive range of frame-store cards also offer on-board processing. Plug-in accelerator cards which enable certain functions to be implemented in real time are available as daughter boards for many frame-stores. Some frame-stores have slow-scan capabilities and the ability to interface to line-scan cameras. When used in conjunction with the current range of high-speed personal computers, such a vision system is an attractive option for small to medium applications of low complexity. Even personal computers now offer direct video input via USB or IEEE 1394 (“firewire”) ports, without the need for an additional plug-in frame-grabber cards.

Such systems offer a number of significant advantages, most important of which is their relatively low cost. Another significant advantage is their ease of use and familiarity. This is especially the case when used in conjunction with standard personal computers, which have become commonplace both in the home and the workplace. The fact that the host computer for the imaging system is a widely available commercial product also widens the base for software applications and maximises the use of the frame-grabber. Many of the software packages available today use 'point and click' interaction with the user, making it easy for him to investigate image processing ideas. (Unfortunately, the majority of these packages are for image processing, rather than image analysis.) The majority of the plug-in frame-store boards can be programmed using commonly used high-level languages, such as C, C++ or Java! This is important, since the use of standard programming languages can have a major impact on program development costs.

A common disadvantage with frame-grabber cards is that they rely on the power of the host computer to do all of the required imaging tasks. Since the host computer is generally not tuned for imaging applications, the system operation may be too slow, despite the constantly increasing performance of commercial computers. So, for many high-speed industrial applications, such systems are not suitable. Many machine vision integrators would not consider personal computer systems as robust enough for industrial applications. The use of industrial PCs in conjunction with a wide range of dedicated processing and interface cards counters this argument to a certain extent. Despite these disadvantages, the use of frame-grabber plug-in cards offers a low-cost

introduction to machine vision, and is suitable for educating, training, system design and other less-demanding applications.

2.9.2 Plug-in Boards: Dedicated Function

For greater speed and ability, engineers often turn to plug-in boards that have a specific functionality, such as real-time edge detection, binary correlation, and convolution. Typically the host computer for such boards would be a VME rack fitted with a CPU card. Quite often, such special-purpose boards are pipelined. That is, they perform different operations on the image, in a sequential manner that allows a new image to be captured while the previous image is still undergoing processing. The main advantage of such systems is their speed and the ability to increase the systems image throughput rate by the addition of extra plug-in boards. The disadvantage of such systems is that they can be difficult to program and quite often require programmers with highly specialist skills. There is also a significant cost factor involved in the capital equipment, along with the application development costs.

While the majority of dedicated plug-in boards for pipelined systems are tuned to deal with array CCD cameras, newer systems have appeared on the market that are specifically designed for a line-scan camera.

2.9.3 Self-contained Systems

Some system manufacturers have taken the option of designing specific machine vision engines that are not tuned for a specific application, but rather designed for their general functionality. Such systems may be totally self-contained and ready to install in an industrial environment. That is, they contain the imaging optics, camera, imaging engine and interfaces for various mechanical actuators and sensors. They differ from turnkey systems in that the software is supplied with the self-contained system has yet to be moulded into a form that would solve the vision application. Such systems have significant advantages, the main one being speed. The majority of self-contained systems are custom designed, although they may contain some plug-in boards and are tuned to provide whatever functionality is required by the application. The self-contained nature of the mechanical and image acquisition and display interfaces is also a significant benefit when installing vision systems. However, it can be difficult to add further functionality at a later date without upgrading the system.

2.9.4 Turnkey Systems

Turnkey vision systems are self-contained machine vision systems, designed for a specific industrial use. While some such systems are custom designed, many turnkey systems contain commercially available plug-in cards. Turnkey systems tend to be designed for a specific market niche, such as can-end inspection, high-speed print

recognition and colour print registration. So, not only is the hardware tuned to deal with high-speed image analysis applications, it is also optimised for a specific imaging task. While the other systems discussed usually require significant development to produce a final solution for an imaging application, turnkey systems are fully developed, although they need to be integrated into the industrial environment. This should not be taken lightly, as this can often be a difficult task. It may not be possible to find a turnkey system for a specific application.

While we have avoided the discussion of any specific commercial devices, there are a number of valuable information sources available, some of these are provided by commercial organisations but some of the most valuable are free! One resource that is well worth considering is the “Machine Vision Resources” website operated by P.F. Whelan [25]. This is a machine vision database that gives details of a large number of machine vision vendors and their products and services.

2.9.5 Software

As was mentioned earlier, there is a large number of image processing, and analysis, packages available, for a wide range of computing platforms. Several of these packages are freely available over the Internet. Some of these packages are tightly tied to a given vision system, while others are compiled for a number of host computers and operating systems. The majority of the software packages have interactive imaging tools that allow ideas to be tested prior to coding for the efficient operation. For more information on the hardware and software aspects of real-time imaging, including a survey of commonly used languages, see [24].

2.10 Further Remarks

The low-level image processing operators described in this chapter are always used in combination with one another, since none can, on its own, provide the kind of quantitative information needed to solve practical applications. Very often, high-level operators can be expressed simply, in terms of sequences of the basic procedures described above. (The high-level procedure may be implemented in this way, or by reformulating it and coding it directly, to improve computational efficiency.) However, it is sometimes necessary to combine the basic operators with sophisticated decision-making, search and control techniques, based on Artificial Intelligence and Pattern Recognition. These are the subjects of the next chapter, where we emphasise that product variation almost invariably requires that we employ more intelligent algorithms.

2.11 References

- [1] Borgefors, G. (1986) Distance transformations in digital images, *Computer Vision, Graphics and Image Processing*, vol. 34, pp. 344–371.

- [2] Gonzalez, R.C. and Wintz, P. (1987) *Digital Image Processing*, Addison Wesley, Reading, MA.
- [3] Dougherty, E.R (1992) *An Introduction to Morphological Image Processing*, Tutorial Textvol. TT9, SPIE Press.
- [4] Haralick, R.M. (1987) Image analysis using mathematical morphology, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 9, no. 4, pp. 532–550.
- [5] Haralick, R.M. and Shapiro L.G. (1992) *Computer and Robot Vision: Volumes I and II*, Addison Wesley, Reading MA.
- [6] Vogt, R.C. (1989) *Automatic Generation of Morphological Set Recognition Algorithms*, Springer-Verlag.
- [7] Serra, J. (1982) *Image Analysis and Mathematical Morphology Vol. 1*, Academic Press, New York.
- [8] Serra, J. (1988) *Image Analysis and Mathematical Morphology Vol. 2, Theoretical Advances*, Academic Press, New York.
- [9] Duff, M.J.B, Watson, D.M., Fountain, T.M., and Shaw, G.K. (1973) A cellular logic array for image processing, *Pattern Recognition*, volume/issue/pp.,
- [10] Sternberg, S.R. (1978) Parallel architectures for image processing, Proc. IEEE Conf. Int.Computer Software and Applications, Chicago, pp. 712–717.
- [11] Waltz, F.M., Hack, R., and Batchelor, B.G. (1998) Fast, efficient algorithms for 3×3 ranked filters using finite-state machines, Proc. SPIE Conf. on Machine Vision Systems for Inspection and Metrology VII, Vol. 3521, Paper No. 31, Boston.
- [12] Waltz, F.M. and Garnaoui, H.H. (1994) Application of SKIPSM to binary morphology, Proc. SPIE Conf. on Machine Vision Applications, Architectures, and Systems Integration III, Vol. 2347, Paper No. 37, Boston.
- [13] Zhuang, X. and Haralick, R.M. (1986) Morphological structuring element decomposition, *Computer Vision, Graphics and Image Processing*, vol. 35, pp. 370–382.
- [14] Vincent, L. (1991) *Morphological transformations of binary images with arbitrary structuring elements*, *Signal Processing*, vol. 22, pp. 3–23.
- [15] Heijmans, H.J.A.M. (1991) Theoretical aspects of grey-scale morphology, *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 568–582.
- [16] Sternberg, S.R. (1986) Grey-scale morphology, *Computer Vision, Graphics and Image Processing*, vol. 35, pp. 333–355.
- [17] Haralick, R.M. (1979) Statistical and structural approaches to texture, *Proc. IEEE*, vol. 67, no. 5, p. 768–804.
- [18] Batchelor, B.G. (1991) *Intelligent Image Processing in Prolog*, Springer-Verlag, Berlin, ISBN 3-540-19647-1.
- [19] Pitas, I (1993) *Digital Image Processing Algorithms*, Prentice-Hall, Englewood Cliffs NJ.
- [20] Davies, E.R (1990) *Machine Vision: Theory, Algorithms, Practicalities*, Academic Press, London.
- [21] Sonka, M., Hlavac, V., and Boyle, R. (1993) *Image Processing, Analysis and Machine Vision*, Chapman and Hall.

- [22] Gonzalez and Wintz, P. (1987) *Digital Image Processing*, Addison Wesley, Reading MA.
- [23] Haralick, R.M. (1979) Statistical and structural approaches to texture, *Proc. IEEE*, vol. 67, no. 5, pp. 768–804.
- [24] Dougherty, E.R. (1992) *An Introduction to Morphological Image Processing*, Tutorial Text vol. TT9, SPIE Press.
- [25] www.eng.dcu.ie/~whelanp/resources/resources.html

Machine Vision for the Inspection of Natural Products

Graves, M.; Batchelor, B. (Eds.)

2003, XX, 471 p., Hardcover

ISBN: 978-1-85233-525-0