Chapter 2

Image Measurements

In this chapter we shall consider various kinds of images such as might be produced by a camera or scanner, and the methods which may be employed to code them as points in $\mathbb{R}^n$ so as to accomplish recognition of the objects in the images. The underlying assumption is that a robot is looking at some collection of objects and needs to make decisions based on a pre-defined human classification of the types. The robot has to analyse a set of pixels in order to make a classification. In the next chapters I shall discuss statistical and ANN methods of recognition of the resulting vectors, but for now I attend to the matter of how to extract vectors from static images.

This may strike the reader as a bit limited, since there are other sources of information than cameras. On the other hand, the general problems are best approached through concrete instances, and much of the conceptual apparatus required for thinking about measuring images is capable of extension to thinking about measuring other things.

The process of performing an operation on an image to obtain a vector of numbers is often referred to as *feature extraction*. Each number in the vector is sometimes said to describe a 'feature'; this somewhat mystical terminology might leave the student wondering what a 'feature' actually is, and the answer would appear to be that anything you can do to an image to get a number out is detecting a 'feature'. In various parts of the literature, a 'feature' may mean a dark pixel, a bright pixel, a lot of bright (or dark) pixels, an edge, or some combination of the above. Some of this confusion will be resolved in the chapter on syntactic pattern recognition. Because I don't know what a feature is and therefore feel shy about using the term, I shall simply talk about making measurements on an image (or indeed any other objects), and a suite of measurements on such an object produces a sequence of numbers. A measurement of an object is any operation performed on the object which yields a real number. An integer or boolean value is treated as a particular case of a real number for these purposes, but we shall have a preference for real real numbers, expressed, of course, to some finite number of decimals.

For most of this chapter, I shall deal with *binary* images, that is to say images where the pixel is either black (0) or white (1) because the recognition problems are sharpest here. Colour images or monochrome greyscale images are of course more common, but the recognition problems become overshadowed by the image processing problems- which belong in a different book. In the later sections I shall discuss the complexities introduced by having greyscale and colour images to recognise.

Since there are an awful lot of different things which can be found in an image, even a binary image, we approach the subject by stages, dealing with the complications progressively. I shall focus on a particular problem so as to concentrate the mind; I shall investigate the problem of *Optical Character Recognition*, so as to sharpen the issues to concrete cases. Virtually everything that applies to recognising characters applies to other types of binary image object, except that the other objects are usually harder to deal with. It might be useful to contemplate Chinese or Arabic characters occasionally, or a handful of nuts and bolts in silhouette, so as to avoid parochialism,

Quite a lot of what I shall describe here is treated in books on Image Processing. This is *not* a book on Image Processing, but it will be necessary to outline the ideas and relevance of Image Processing material. My treatment will therefore be sketchy, and the bibliography should be used to fill out the details. Since many books have been devoted to what is covered in this chapter, you may take it that the treatment will be lacking in that depth which the intelligent reader quite properly craves; in exchange you have here the opportunity to get an overview of what the stuff can be used for. An honest attempt to do the Exercises at the end of the chapter, if necessary with the aid of some of the books referred to in the bibliography, will provide a crash introductory course in Image Processing.

The first chapter gave an overview from such an exalted height that the reader may have been left feeling a touch of vertigo. It is now necessary to take one's feet down, pour the drink into the cat's bowl, lace up one's running shoes and sober up. A certain amount of the nitty-gritty is waiting further down the alley with a sand-filled sock.

2.1 Preliminaries

2.1.1 Image File Formats

There are currently many different ways in which images are regularly stored inside a computer, and for each of them there has to be a way of reading the image into some sort of viewing program to actually *look* at it, and the pixels have to be stored in an array in memory in order to process the image or subimage. If we wish to restrict ourselves to binary images, then a single bit can be used to store the value of a pixel at a particular location. A binary file can therefore be stored as a long list of bits, but it also needs some information to say how many there are and what the dimensions of the array are.

The graphics systems of different computers also vary a lot. This presents the writer of a text book on image processing or pattern recognition with certain problems. What machine does one suppose the reader has access to? What programming languages does one assume he can use, and what make of scanner or camera is he going to be able to get his hands on?

It is possible to get around these matters by writing at a level of abstraction so high that the reader will not actually connect up what he reads with actual images or patterns, but I find this unappealing. I have agonised over these issues and concluded that there is not much use tackling serious pattern recognition without a UNIX workstation, and that X-Windows is currently the *de facto* standard for graphics. At the same time, PC's are widely available and many graphics acquisition cards are available for these machines at low price. Rather than choose between these competing alternatives, I have opted for both of them. I shall start off by supposing that the scanner is attached to a PC and that images can be stored in the common *.tif* files.

TIFF, short for Tag Image File Format, is a convention for storing a variety of images, and since it is a common one and most scanners can read and write such files, I shall refer only to such files. Other standards such as GIF, JPG, MPG and the page description languages can generally be converted, for a particular image, and I do not want to have to write programs for them all.

One of the many programs available with this splendid book, at no extra cost, is a TIF reader, written in Borland's excellent C, which will read a small binary TIF file and display it on the screen of a PC. It stores the image internally in an array, which means that the enthusiast can monkey about with it to his hearts content. At a later stage in the book, the reader will be introduced to a somewhat beefier X-Windows variant of this program for reading larger images in many colours into a workstation. Also included on disk accompanying this book are some image files, but the reader should certainly acquire the art of scanning documents; it is easy and gives a satisfying sense of power for very low expenditure of effort.

I shall suppose then that the images that follow can be obtained as *.tif* files output from a scanner, and that the transformation to an array of pixels is accomplished.

2.2 Generalities

Let us start then by supposing the simplest sort of image, a *binary* array of pixels, say 256 by 256. Each pixel is either black (0) or white (1). Hardly any modern equipment actually produces such images unless you count photocopiers, and I have doubts about them. Video cameras and scanners usually produce larger arrays in many colours or grey levels. Monochrome (grey scale) cameras are getting hard to find, and scanners will be following this progression soon. So the images I shall start with are rather simple. The image in *Fig.1.13* is an example of the kind of thing we might meet.

In some forms of image recognition, particularly in *Optical Character Recognition (OCR)* the first operation performed upon the image is called *thresholding* and what it does is to take a monochrome image and rewrite all the light grey (above threshold) pixels to white, and all the dark grey (below threshold) pixels to black. How to make a judicious choice of threshold is dealt with in image processing courses, and will not be considered here. Although a little unphysical, because real life comes with grey levels[1], the binary image is not without practical importance.

The result of digitising and thresholding a piece of handwriting is not unlike *Fig.2.1.*, where the word /the/ was hand drawn by mouse using the unix *bitmap* utility. It is reasonable to want to be able to read the characters, /t/, /h/, /e/ by machine. For handwriting of the quality shown, this is not too difficult. It is, however, far more difficult than reading printed script, and so we shall treat the printed case first.

In order to make the problem moderately realistic, we look at the output of a scanner, or at least a bit of such output, when applied to two pages of text taken from a book. (*Fig.2.2.*). The quality of the image is not, we regret to say, up to the quality of the book. Such images are, however, not uncommon, and it is worth noting a number of features.

[1] And for the lucky among us, colour.

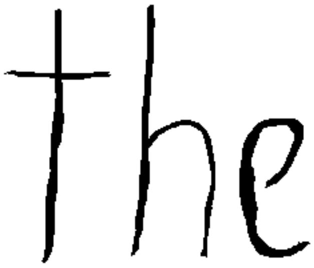
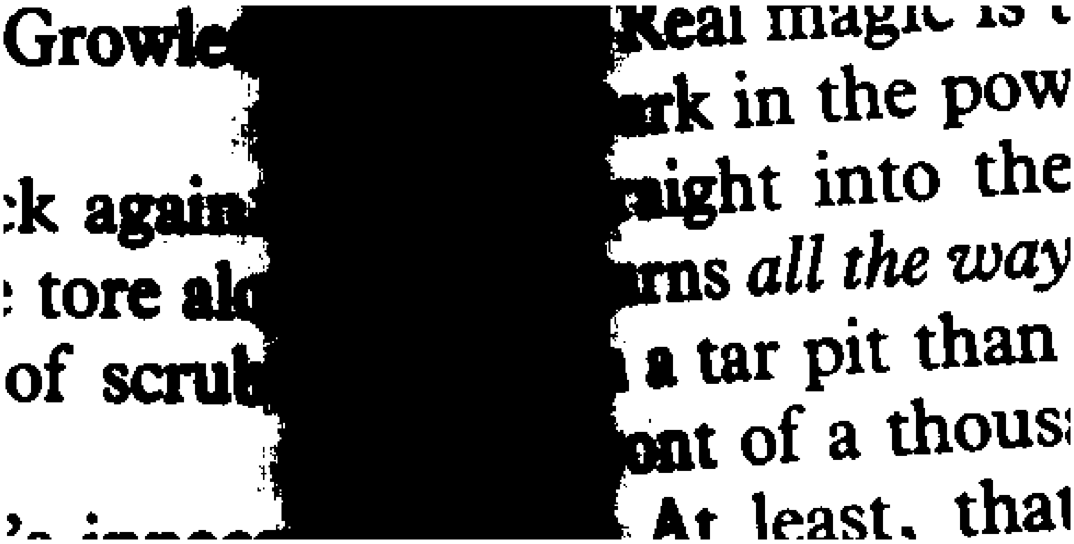


Figure 2.1: A handwritten word.



Growle Real magic is
rk in the pow
:k again aight into the
: tore al rns *all the way*
of scrut a tar pit than
ont of a thous
At least, that

Figure 2.2: A sample of scanned text from a book.

- The most manifest oddity is the black band down the centre between the pages, where the scanner had trouble with the curvature of the book. The result is that (1) the leftmost characters of several words are unreadable even by human beings and (2) there is some distortion of those characters which *are* readable.

- There is noise consisting of black pixels scattered around the boundary of the central black band and intruding onto the characters.

- The quantisation at 72 dots per inch is sufficient to ensure that no two characters are actually identical. A comparison of the letters /h/ or /o/ for example in their several occurrences shows significant differences in the actual array of pixels.

- Italicisation of part of the text means that we are not, as might be hoped, dealing with a single font.

- The lines of text are not horizontal, the left page and the right present different angles for what, in an ideal world, would be horizontal lines of text.

- Our first problem, before trying to recognise a character, is isolating it from the other characters and the noise, and since we want to preserve the order of the characters when we translate from pixels into ASCII, we need to find lines and order *them* top down, then we need to scan a line from left to right and isolate the characters, including the white spaces between the 'objects', since a translationthatlookedlikethis would be, rightly, thought objectionable.

The problem of reading such text is therefore far from trivial, even if we stipulate somewhat higher quality reproduction than is present in *Fig.2.2.* In doing some serious reading of a document such as a newspaper, large and uncomfortable problems of working out which bits of text belong together, which bits of an image *are* text and which bits are the pin-up girl, and similar high level segmentation issues arise. Any illusion that the problem of reading text automatically is easy should be abandoned now.

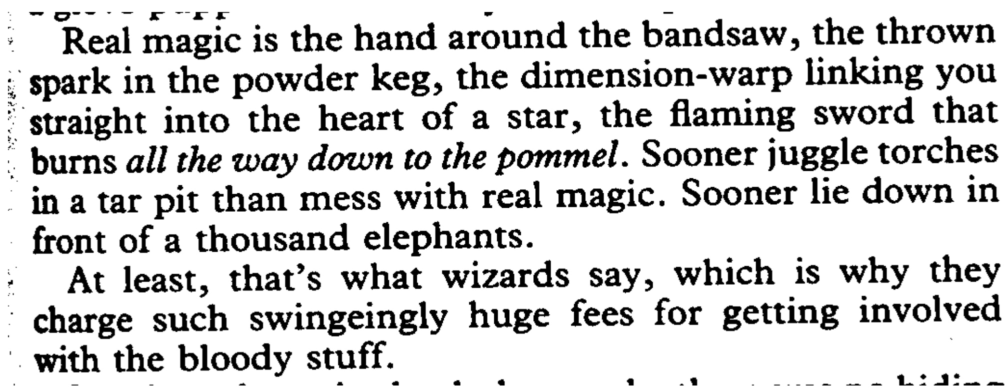
Figure 2.3: Postcode digits 6512

2.3 Image segmentation: finding the objects

How can we tackle the problem of writing a program which will read the text of *Fig.2.2.*? When I say 'read the text', I mean of course that the input will be an image file, such as that from which *Fig.2.2.* was extracted, and the output will be a sequence of ASCII characters, possibly with some additional formatting information to say where the characters are on the page. This is a clear case of a Pattern Recognition problem of an eminently practical type. Typically image files are very big, and text files which contain the data from which an image of text was obtained by printing are very much smaller. Also the latter can be read into your favourite word processor and the spelling checked, something which cannot be done with an array of pixels.

There are clearly pre-processing problems which must be gone through in order to identify lines of text in the image, and to work out where the characters actually *are*. These are not negligible and something must be said about them.

The general problem of image segmentation, chopping an array of pixels up into different bits, is fraught with difficulty: It is easy to run two letters together in an image so that they overlap, and for the eye to still be able to read them as two separate things. A program which kindly lumped the two together as one, would not make automatic recogni-


Real magic is the hand around the bandsaw, the thrown spark in the powder keg, the dimension-warp linking you straight into the heart of a star, the flaming sword that burns *all the way down to the pommel*. Sooner juggle torches in a tar pit than mess with real magic. Sooner lie down in front of a thousand elephants.

At least, that's what wizards say, which is why they charge such swingeingly huge fees for getting involved with the bloody stuff.

Figure 2.4: A bigger and better sample from the same book.

tion any easier. The proposition that the objects are going to come neatly separated by white spaces around them is simply not true. The postcode in *Fig.2.3* is handwritten, it is easy to read, but segmenting it automatically is best done in hindsight once you have worked out what the characters are!

By choosing to deal with printed text of sufficiently high quality we can make our lives much simpler, but there has to be something intrinsically unsatisfactory about a method which cannot generalise to cases which human beings find easy.

With the cautionary note out of the way, and bearing in mind for later chapters that there has to be a better method, let us proceed to examine the standard means of proceeding with the image of *Fig.2.4*, which is a somewhat cleaner and larger version of *Fig.2.2.* Anyone who can identify the source of the text is entitled to a prize, but already has it.

First we find the lines of text. Moving a long way from *Fig.2.4* and squinting at it with eyes half closed, one sees more or less horizontal bands where the words blur together; these are our lines of text. One way of finding them is to 'fuzz' the image by what are known as *Mathematical Morphology* techniques. The ideas of Mathematical Morphology are very simple (and usually made difficult by being expressed in formalism, a favourite method by which esoterrorists try to impress the innocent). We proceed to elucidate the easiest case; references to more comprehensive accounts may be found in the bibliography at the chapter's end.

2.3.1 Mathematical Morphology

Let A and B be subsets of $\mathbb{R}^2$. Then the *dilation* of A by B is the set C defined by

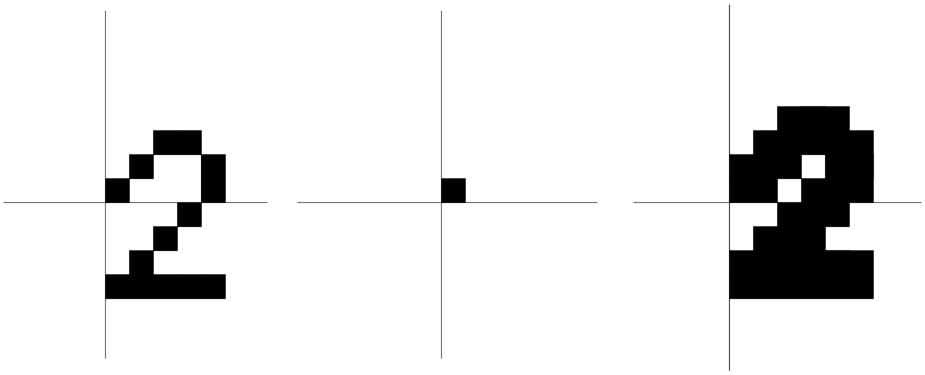
$$C = A \oplus B = \{\mathbf{x} \in \mathbb{R}^2 : \exists \mathbf{a} \in A, \exists \mathbf{b} \in B, \mathbf{x} = \mathbf{a} + \mathbf{b}\}$$

where the addition is, of course, the vector addition in that well known vector space $\mathbb{R}^2$.

In applications, A and B will be finite sets of pixels. We obviously have that $A \oplus B = B \oplus A$ but we shall tend to choose a particular B and think of it as operating on a lot of different A's. In this case, B is referred to as the *structuring element.*

Example: If A and B are the sets shown in *Fig.2.5*, where the origin and axes are as marked, we see that $A \oplus B$ is the 'thickened' version shown as C. By making B bigger and approximately disk shaped, we can fatten A quite out of recognition. In this example, each black square is a pixel, and if we think of the corners of the square which is B being at the points $\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$, every point $\mathbf{x}$ of A is expanded to a copy of the unit square stuck onto $\mathbf{x}$ at its lower left corner. So A is 'blobbified' by B; equally, B has been blobbified by A.

For digital images, the dilation procedure is somewhat simpler; we have the zero which is usually a part of the structuring element, and some points at integer co-ordinates. We simply move the structuring element B and place the zero at a pixel of A, and then fill in all the other pixels of B. Then we repeat for each of the pixels of the original A.



A B C

Figure 2.5: $A \oplus B = C$

Similarly, it is possible to have an *erosion* operation, $A \ominus B$ which treats B as a mask with holes in it where the pixels are. Then to get $A \ominus B$, place the origin of B at a pixel of A, and look to see which of the pixels of A can be seen through the holes in B. If there are any holes in B without points of A visible through them, mark the pixel of A visible through the origin of B for deletion. Repeat for each pixel of A, and finally remove all the pixels marked for deletion. Formally:

$$C = A \ominus B = \{\mathbf{x} \in \mathbb{R}^2 : \forall \mathbf{b} \in B, \mathbf{x} + \mathbf{b} \in A\}$$

Note that I have assumed in my explanation of how erosion is carried out that the origin is in B.

Exercise: What difference does it make if it isn't?

Repeated application of erosion by a suitable structuring element B can be useful for *thinning* sets. Carried too far, sets can easily be eliminated altogether.

Exercise: Draw a grid of lines, label a horizontal line in the middle somewhere as the X-axis, a suitable vertical line as the Y-axis, and draw pixel dots on the intersections at enough points to pick out a big fat character like the blobbified /2/ of *Fig.2.5.* This is A. Now take B to be the origin and its 8 nearest neighbours, on a separate grid drawn on transparent film. Do the dilation operation, then the erosion operation on the result. Start again and do it in the opposite order. Do the two operations commute? If the origin is omitted from B, and we use the formal definition, what effect does this have?

Remark There are hardware implementations of many such operations which are extremely fast. See Wojciech Kuczborski's Ph.D. thesis, coming real soon to a store near you, and referenced in the bibliography at this chapter's end. Well, you can write to Wojciech care of The Centre for Intelligent Information Processing Systems (CIIPS) at the University of Western Australia and ask nicely for a copy.

If we blobbify (or dilate) the set of pixels in the image *Fig.2.4* by a horizontal bar of pixels, we 'stretch' pixels horizontally. This will join up adjacent letters and blur them in a horizontal direction without having any effect vertically. It is childishly simple to write a program which goes over an array of pixels and dilates them horizontally: every time a black pixel is found, the new image has, say, black pixels inserted to the left and right of the location of the given pixel, as well as copying the original pixel. I assume, of course, that we are dealing with black images on a white ground.

Doing this to *Fig.2.4*, if necessary several times on the result of doing it previously, gives something close to horizontal bands. The page has its text at an angle, so the bands are not very horizontal, but it is now easy to find the spaces between the lines and also to determine the height of the bands that used to be characters. The lines of text can be numbered consecutively from the top and their direction specified.

Of course, there are ways of finding the lines of text which do not require us to apply morphology methods, but the methods will have other uses, and this seemed like a good place to introduce them.

I said it was easy to find the lines, and it is, if you don't mind doing the sort of ghastly hack that occurs naturally to the more depraved kind of programmer. If your aesthetic sensibilities are revolted by this sort of thing, good. There are better ways and we shall come to them in due course.

2.3.2 Little Boxes

Next we can start at the left hand end of each line (of the original image, not the dilated one!) and locate the first black pixel. This may be the bottom half of a character of text in the line above, the page number of the page, some noise from the book spine via an out of focus scanner, or a bit of a thumb-print or soup stain or worse. Since we have estimates of the character height obtained from the thickening, we can make an estimate of where the bottom of the character should be if we actually have the top left hand pixel of a character, and also have some ideas about where the rest of it should be. These will be very crude, because we might have an /o/, or a /b/ or a /p/ or a /q/, not to mention capitals or punctuation. If we have confidence in our knowledge of the font, we can look for vertical (relative to the direction of the line as horizontal) and horizontal separating lines between which our character may be found.

It is common practice in the case of the Roman alphabet to fit slightly slanted lines from South-South-West to North-North East because, as a careful examination of the italic text shows, there is not in general a vertical separating line between well spaced characters in italic fonts. This can also happen with well spaced non-italic fonts as when TeX places the characters in a word such as

WAVE

and it should be noted that there may be no line slanted like: '/' separating characters, either.

It should be plain that separating out characters from each other and putting boxes around them is not usually the simple procedure one might have hoped for.

2.3.3 Border Tracing

A more cautious approach than trying to find a box around each character, is to trace the exterior boundary of the black pixels. Then we can work out from the length of the boundary something about the size and shape of the object we have found. This should distinguish noise from characters, given reasonable luck and a good quality image. It also puts a box around the object, although rather an irregularly shaped one. Still, better a weird shaped box than no box at all. At least, we can segment the image into objects.

As we have noted, in the case of handwritten characters it could be a mistake to imagine that the segments we get this way are going to be characters. They could be bits of characters, or more often several characters joined together. For good quality printed text however, border tracing stands a reasonable chance of isolating characters. We therefore discuss the problem of finding the border pixels of a connected block of pixels. We then regard the character as the set of pixels lying inside this rather irregular 'box'.

The algorithm described briefly here (and which may be found, with some additional nuances, in some of the references below, particularly Haig, Attikiouzel and Alder) is one which the most innocent would devise unaided.

First we assume that the image has a white border around it, and we have to find the bounding pixels of clumps of black pixels. We therefore start at the top left of the image at a white pixel, and scan horizontally until we encounter the end of the row or a black pixel. In the former case we do a line feed carriage return to start at the left side of the next row down. If we are at the bottom row already and there is no

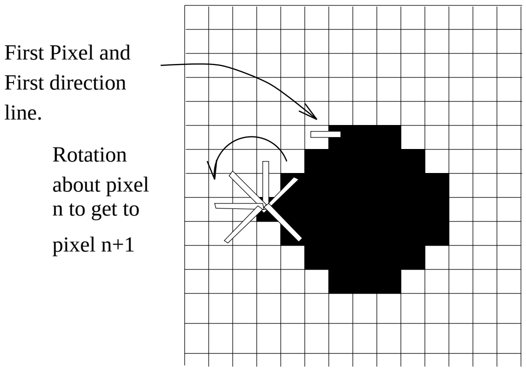


Figure 2.6: Border Tracing

next row, we have scanned the page and stop.

If we have found a black pixel, we draw an imaginary line from the pixel just to the left of it to the black pixel. We note the direction of this line, and the co-ordinates of the black pixel. Then we rotate the line, about the black pixel, in the positive direction (anticlockwise), until it meets another black pixel adjacent to the first black pixel, or returns to its original position. In the latter case the black pixel is isolated; we mark its location, delete it and start again.

If we find another black pixel then it is in one of the eight positions adjacent to the first black pixel. Call the black pixel we are currently standing on pixel n. When $n = 1$ we are at the first black pixel, but this works for all of them. We now move to pixel $n + 1$, note its location, note the direction from pixel n to pixel $n + 1$, and take the line joining pixels n to $n+1$ and rotate it, about pixel $n+1$, in the positive direction, until it finds an adjacent black pixel which becomes pixel $n + 2$. This process is repeated until pixel $n + 1$ is a previously listed pixel and the line from pixel n to pixel $n+1$ is the same previously listed direction. This last condition is very necessary, as it is easy to re-enter a pixel from different directions without circumnavigating the set. The reader should examine carefully the border tracing program and try it out on some sets to see if it locates borders in a sensible way.

Once a black object has been found, the pixel set of its border is determined by this algorithm, and this set of pixels *together with every pixel which is inside this border* is stored as an object and then deleted from the page, and the scan starts again. Only when the page has been blanked does the procedure terminate.

Deleting the pixels inside the border sounds easy but may not be if the border is not convex. To see if a pixel is inside the border, draw a half ray starting at the pixel and extend to the boundary. Count the number of times this intersects the border. If the number is even, the starting pixel is outside, otherwise it is inside or on the border. If a pixel is inside the border and not on it, every adjacent pixel is either inside or on the border. A 'floodfill' operation stopping at the border and starting inside it can be used to label the points to be removed. Of course, a border may have no interior points.

There are more efficient ways, but it is relatively easy to prove rigorously that this method will work. Well, sort of. If the image has a black border, or a black box around a character, as in Θ, and if the border is broken, then the algorithm will fail to find the desired set. Similarly, it will separate out the components of colons, semi-colons and characters such as i and j which are disconnected.

Noise, black lines down the middle where the centre of the book got mis-copied, smudges and the like will be picked out and will have anomalous shapes: the single pixel dots can be eliminated at this stage. Knowing the approximate size of a character can help remove the odd spots, although it is easy to get rid of the punctuation as well.

If the input text comes in a wide variety of fonts, as for example the page of a newspaper or magazine, or is mixed with graphic images, then the complications of segmentation can be formidable and go beyond the scope of this book, and, indeed, most others.

2.3.4 Conclusions on Segmentation

The exercises at the end of the chapter will give you some practice at doing some of this pre-processing. Later we shall discuss less flagrantly *ad hoc* methods of tackling the problem.

Separating out characters by putting little boxes on them, little rectangular boxes or little parallelogram boxes, little rhomboid boxes, or even little irregular boxes obtained by border following, allows us to start thinking about how to code each character as a point in $\mathbb{R}^n$ for some n. Let us be optimistic and suppose these issues have been resolved without too much difficulty.

There are a number of approaches, we consider some of them.

2.4 Measurement principles

2.4.1 Issues and methods

Let us now suppose that we have isolated a character by some acceptable process, possibly only tentatively, and that we now wish to proceed to the recognition/classification stage. The intervening step is to measure some properties or characteristics or *features* of the character, so we will be able to compare and contrast the results of doing the same measurements on other characters. There are several well established methods which will shortly be described, but there are a few issues to be contemplated before going into a list of them.

If I suggest a suite of measurement operations, certain properties may or may not hold for my selection. For example, it is conceivable that my measurements would be quite unaffected by simply shifting the character by one pixel to the right. More generally, I might choose to make measurements that would be the same no matter where the character is on the page. In this case we say that the measurement process is *invariant* under shifts. Similarly, if a measurement process were invariant under scaling, it would yield a set of numbers which would be the same for any two characters which had one simply a scaled up version of the other, and if it were invariant under rotations, it would not matter how much you rotated a character, the measuring process would give the same output. These ideas derive from the continuous case, and have to be modified a bit for the case of pixel arrays, but clearly there is some vague and approximate notion of invariance in the discrete case.

Even if the measuring process were not shift invariant, it would certainly simplify recognition if the resulting vector of numbers did not change beyond recognition if the character were moved one pixel to the left. Since we may very well want to know where the character is, complete shift invariance is undesirable, but it would be useful to have the measurement process partitioned into a part which tells you where the character *is*, another part which tells you how big it is, another part where the numbers tell you if it has been rotated, another part perhaps telling you if it has been transformed in some other standard way, such as a change of font, and finally a component which would be the same no matter how much the character had been scaled or rotated or where it was. It is not too much to hope that some of these properties would be found to hold for a well chosen suite of measurements. Conversely, a suite of measurements which had every number changed completely by shifting the character one pixel sideways is going to be less appealing.

So it is possible to think a little bit about what constitutes a sensible sort of system of measurements rather than to jump on the first such system that occurs to you. This can save a good deal of time in the longer run.

2.4.2 Invariance in practice

Suppose we have put a box around a putative character. Suppose that the box is of a reasonable size to contain some character, and that the top and bottom edges are parallel and can confidently be declared as horizontal. This is a somewhat courageous commitment because, as inspection of many a photocopy will show, text lines are often curved near the spine of the book, but we proceed in a spirit of optimism.

The first prudent thing to do is to contract the resulting box by making sure the black pixels inside it are touching the edges. A big box with a lot of white space against the right hand edge has more pixels in it than necessary. The next prudent thing to do is to scale it to a standard size and shape. It is prudent because the characters may be deformed and this will normalise them, and also because it is easier to compare like with like. The effect of this on an /o/ will be very different from its effect on an /i/ of course. What constitutes a good size for the standard box will depend to some extent on the font. If the box was a parallelogram which fitted an italicised character, the transformation back may de-italicise it. And it may have rather an odd effect on some characters, making /V/ much to much like /U/ if taken to excess. Some opportunity to experiment with various transformations is given the reader in the exercises.

2.5 Measurement practice

2.5.1 Quick and Dumb

If we normalise into, say, an 11 by 9 array, we can rewrite the characters into standard form. Then we could, if desperate for ideas, take each character as a point in $\mathbb{R}^{99}$. This is not a good idea, although it has been done. The main reason it is not a good idea is because one extra pixel in the wrong place can give vectors which are totally unrelated: a single pixel can shift a vertical line one pixel to the right with no trouble at all.

It would be nice if a horizontal shift of a character by one pixel column were to have minimal effect on the placing of the point in $\mathbb{R}^n$. Another reason it is a bad idea, is that the dimension of the space should be kept as small as possible. The reasons for this are subtle; basically we want to use our data to estimate model parameters, and the bigger the ratio of the size of the data set to the number of parameters we have to estimate, the more we can feel we have genuinely modelled something. When, as with some neural net enthusiasts, there are more parameters than data, we can only groan and shake our heads. It is a sign that someone's education has been sadly neglected. How much faith would you have in the neural net B of *Fig.1.7* being able to do a decent job of predicting new points, based as it is on only two data points? How would you expect it to perform as more points are obtained? How much faith would you have in the rightness of something like B if the dimension were 99 instead of 2, more faith or less?

In the first chapter I remarked that the image on my computer screen could be regarded as a point in a space of dimension nearly four million, and that I didn't assert that this was a good idea. Suppose you wanted to write a program which could distinguish automatically between television commercials and programmes. Doing this by trying to classify ten second slices of images using a neural net is something which might be contemplated by a certain sort of depraved mind. It would have to be pretty depraved, though. I shall return to this issue later when discussing model complexity.

2.5.2 Scanline intersections and weights

Some thought on how to represent the characters in a way which will not be too badly affected by such things as translations, the deck transforms that are produced by italicisation and small angle rotations, may lead to taking horizontal and vertical scan lines across the rectangle, and measuring the number of distinct intersections. Or we can take the quantity of black pixels along a scan line, and list the resulting numbers in some fixed order.

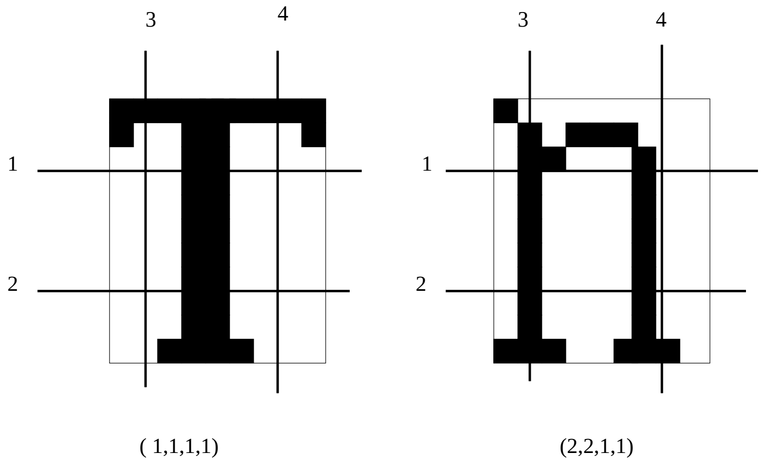


Figure 2.7: A coding of two characters by measuring scan line intersections

For example, the characters in *Fig.2.6* have been coded by listing the number of intersections along two horizontal and two vertical scan lines, as indicated, making it easy to distinguish them as vectors in $\mathbb{R}^4$.

The method is not likely to be satisfactory with only two horizontal and two vertical scanlines, but increasing the number can give more information. A little ingenuity can readily suggest variants of the method which may even give some degree of font invariance.

2.5.3 Moments

Suppose we have isolated a character as a set of pixels. Let us not draw a box around it, let us first find its centre of gravity. If A is the set of pixels in the figure, with integer co-ordinates x and y for each pixel p in A, then let f_A be the characteristic function of A, i.e. $f_A \begin{pmatrix} x \\ y \end{pmatrix} = 1$

iff there is a pixel of A at location $\begin{pmatrix} x \\ y \end{pmatrix}$, and otherwise f_A takes the value 0. Then we can write the count of the number of pixels in A as

$$\sum f_A \begin{pmatrix} x \\ y \end{pmatrix}$$

where the summation is taken over the whole plane of integer value pixel locations. We can write the mean of the x values of all the pixels in A in the form:

$$\bar{x} = \frac{\sum x f_A \begin{pmatrix} x \\ y \end{pmatrix}}{\sum f_A \begin{pmatrix} x \\ y \end{pmatrix}}$$

and similarly

$$\bar{y} = \frac{\sum y f_A \begin{pmatrix} x \\ y \end{pmatrix}}{\sum f_A \begin{pmatrix} x \\ y \end{pmatrix}}$$

where sums are taken over the whole plane. (Since f_A is zero over all but a bounded region, we don't really have to do as much arithmetic as this suggests.)

These numbers give us information about the set A. We can rewrite things slightly by defining the $(p,q)^{th}$*moment* of f_A (or of A) by

$$\mu_{[p,q]} = \sum x^p y^q f_A \begin{pmatrix} x \\ y \end{pmatrix}$$

for any natural numbers p,q,

and the *normalised* $(p,q)^{th}$*moment* of f_A (or of A) by

$$\mu_{[p,q]} = \frac{\sum x^p y^q f_A \begin{pmatrix} x \\ y \end{pmatrix}}{\sum f_A \begin{pmatrix} x \\ y \end{pmatrix}}$$

for any natural numbers p,q,

Then the moment $\mu_{[0,0]}$ is the pixel count of A,the mean x-value of A is the moment $\mu_{[1,0]}$ divided by the pixel count,or alternatively the normalised (1,0) moment, and the mean of the y-values of the pixels of A is the moment $\mu_{[0,1]}$ divided by the pixel count.

It is easy also to compute higher order moments. These give extra information about the distribution of the pixels in A.

The *central moments* are computed in the same way, except that the origin is shifted to the centroid $\begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix}$ for all the pixels of A.

All the moments $\mu_{[p,q]}$ where p+q takes the value v, are called *moments of order v.* A little thought and recollection of statistics will no doubt remind you that the *second order central moments, moments of order 2,* i.e. the (2,0), (0,2) and (1,1) central moments, are the elements of the covariance matrix of the set of points, except for a scaling dependent on the number of points. The use of central moments is clearly a way of taking out any information about where the set A actually *is.* Since this is a good idea, we have a distinct preference for the central moments. The three central moments of order 2 would allow us to distinguish between a disk and a thin bar rather easily. In the exercises we ask you to compute the central moments for some simple shapes.

Example We give an example for the easy case of sets A and B defined by

$$A = \{ \begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{Z}^2 : -5 \leq x \leq 5 \ \& -1 \leq y \leq 1\}$$

and

$$B = \{\begin{pmatrix} x \\ y \end{pmatrix} \in \mathbb{Z}^2 : -3 \leq x \leq 3 \;\&\; -2 \leq y \leq 2\}$$

Then A has 33 pixels and B has 35; B is squarer than A. Both have the origin as the centroid so as to save us some arithmetic.

A has three rows, so to calculate $\mu_{[2,0]}(A)$ which is the sum of the squares of all the x-values of the pixels in A we get

$$3(-5)^2 + 3(-4)^2 + 3(-3)^2 + 3(-2)^2 + 3(-1)^2 + \cdots + 3(5)^2$$

i.e.$\mu_{[2,0]}(A) = 330$. Similarly, $\mu_{[0,2]}(A) = 26$, and

$$\mu_{[1,1]}(A) = (-5)(-1) + (-4)(-1) + \cdots + (4)(-1) + (5)(-1)$$

$$+(-5)(0) + \cdots + (5)(0) + (-5)(1) + \cdots + (5)(1) = 0$$

So the three second order moments, in the right order, give the vector $\begin{pmatrix} 330 \\ 0 \\ 26 \end{pmatrix}$, while the same calculation for B gives $\begin{pmatrix} 140 \\ 0 \\ 70 \end{pmatrix}$.

So if we were to represent these objects as points in $\mathbb{R}^3$, it would be easy to tell them apart. Unfortunately, it takes rather higher dimensions, i.e. higher order moments, to discriminate between characters, but it can be done by the same methods.

2.5.4 Zernike moments and the FFT

The expression for the calculation of the moments in terms of a sum would be written by a mathematician in the more general form of an integral. This covers the case for a density function over the plane, or indeed over $\mathbb{R}^n$ for any positive integer n as well as the discrete case. The density function does not have to a characteristic function, we could allow grey levels, so f_P can be supposed to be an integrable function in $\mathbb{R}^n$, although we shall mainly be interested in the case $n = 2$. If we write for Natural Numbers p,q

$$\mu_{[p,q]} = \int x^p y^q f_A \begin{pmatrix} x \\ y \end{pmatrix}$$

where the integral is over the whole plane, it is clear that the numbers $\mu_{[p,q]}$ are the results of taking the inner product of f_A with the monomial $x^p y^q$ in the space $\mathcal{L}_2(\mathbb{R}^2)$. (If you need to brush up your linear algebra, particularly in respect of function spaces, see Kreider, Kuller, Ostberg and Perkins, subsequently KKOP for short, in the bibliography at the end of this chapter.) This is precisely the same idea as expressing a function by its Fourier coefficients, except that instead of using the sine and cosine functions, we have chosen to use monomials.

This leads to the obvious question as to whether there is a better basis of functions to use in order to express the characteristic function of a shape. In particular, why not take the Discrete Fourier Transform of the image? This may be done quickly and efficiently using the FFT or Fast Fourier Transform, and is described in Image Processing courses. Or are there better bases of functions, more suited for the shapes we get in character recognition?

A small amount of reflection on what you are doing when you do a two dimensional Fourier expansion of the characteristic function of a character suggests that this is not a particularly good way to go. Imagine doing it in one dimension: you have a set which is a couple of intervals and you have the characteristic function for the set, which looks like a rectangle, 1 over the set, 0 elsewhere. You approximate this by adding up sine and cosine waves until you get a reasonable fit. Now go to two dimensions and think of a character, say the letter /E/. You have a function which is of height 1 over the pixels in the character, and 0 outside the character. Now you want to build this function of two variables up out of sines and cosines in the x direction and in the y direction.

Closing your eyes and brooding about it quietly, leads you to the suspicion that it will be a messy business with rather a lot of terms before anything much like the /E/ emerges from the mess. Of course, the same considerations suggest using polynomials would be a mistake too, but at least it is easier to do the arithmetic for polynomials.

On the other hand, the trigonometric functions do have one advantage, they form an orthogonal set of functions, which makes the set of coefficients much more informative. So we might ask whether we can do anything to improve on the monomials.

In KKOP, page 277, the reader is shown how to use the *Gram-Schmidt Orthogonalisation Process* to make the sequence of basis functions

$$1, x, x^2, x^3, \cdots$$

orthogonal on the interval $[-1, 1]$, thus producing the *Legendre Polynomials*. It is a straightforward application of linear algebra ideas to spaces of functions, something all serious scientists and engineers should know. (If you were brought up to believe that engineers could memorise the recipes and not trouble to figure out why they worked, you should sue your educators for your fees. You wuz robbed. There are programs that can do what you were trained to do, and they probably do it better. So why should anyone hire you, when he can buy a program cheaper?)

If we normalise all our shapes so that they are scaled to lie in the unit disk in the plane, and have their centroid at the origin, and if we use polar co-ordinates, we can expand any shape in the plane into the set of basis functions, indexed by the natural numbers p,q:

$$\mathcal{V}_{p,q} = r^p e^{jq\theta}$$

where j denotes the square root of -1, and (r, θ) are the polar co-ordinates of points in the unit disk,

$$D^2 = \{ \begin{pmatrix} r \\ \theta \end{pmatrix} : 0 \leq r \leq 1 \ \& \ 0 \leq \theta < 2\pi \}$$

The result of applying the Gram-Schmidt Orthogonalisation Process to the above basis of functions gives us the *Zernike basis*, $\mathcal{Z}_{[p,q]}$.

Although it is fairly unpleasant to compute these by hand, some programming in a good symbolic package such as MAPLE or MATHEMATICA or MACSYMA helps. Alternatively one can get closed form expressions for the functions by reading the right papers. See Alireza Khotanzad and Jiin-Her Lu in the bibliography. They former give the formula:

$$\mathcal{Z}_{[p,q]}(\rho, \theta) = R_{[p,q]}(\rho) exp(jq\theta)$$

where	
p	is a positive integer or zero,
q	is a positive or negative integer, subject to the constraints $(p - \lvert q\rvert)$ is even and $\lvert q\rvert \leq p$,
ρ	is the distance from the origin to the pixel,
θ	is the angle between the radius line to the pixel and the x-axis,
$R_{[p,q]}(\rho)$	is the radial polynomial defined by:

$$R_{[p,q]}(\rho) = \sum_{s=0}^{(p-|q|)/2} \frac{(-1)^s \ [(p-s)!]\rho^{p-2s}}{s!(\frac{p+|q|}{2} - s)!(\frac{p-|q|}{2} - s)!}$$

Note that $R_{[p,-q]}(\rho) = R_{[p,q]}(\rho)$.

Then the *Zernike moment of order* p with repetition q for a function f defined in the unit disk (possibly the characteristic function of a set) is

$$A_{[p,q]} = \frac{p+1}{\pi} \int_0^{2\pi} \int_0^1 f(\rho, \theta) \mathcal{Z}^*_{[p,q]}(\rho, \theta) \rho d\rho d\theta.$$

The obvious changes are made to the digitised case, replacing the integrals by sums. * denotes the complex conjugate.

Projecting down on this basis in the space $\mathcal{L}_2(D^2)$, the set of square integrable functions defined on the interior and boundary of the unit disk in $\mathbb{R}^2$, or the discrete pixelised approximation to it, gives all the *Zernike moments*. They are complex numbers in general.

We have ensured that the objects we are going to expand in this way have a representation which is shift invariant by reducing it to the centroid, and scale invariant by normalising it to lie inside the unit disk. If we observe that the expansion in terms of the basis functions expressed radially will have a complex number arising out of the angular component, then by simply taking the modulus of this number we ensure that the resulting coefficients give a representation of the shape which is rotation invariant as well. Naturally, this leads to confusing commas and sixes and nines and single quote marks. The measurement suite has rather more invariance than is good for it, but it is a simple matter to add in a few extra terms which tell you where the character is, how big it is, and which way is up.

Experimentally it is found that it is feasible to recover almost complete information about the shape of a point set in $\mathbb{R}^2$ using less than 20 terms in the infinite collection of moments, when the point set is something of complexity comparable with a printed character. The general use of moments is favoured because, happily, only a small number of moments often suffice to give adequate approximations.

Historical Note

I have looked up the 1934 paper in *Physica* by Zernike. In the same issue is a paper by Van der Pol, for those of you who remember his oscillator. I cannot find any reference in Zernike's paper to the moments that bear his name. Mind you, its in German, so I may be mistaken. There are Bessel functions and Gamma functions galore, but not the Zernike basis. Since you can expand in more or less any old functions, using the functions given above can't do any harm, but since I haven't verified orthogonality in D^2, I don't know that this is actually correct. You can do it for me, quite quickly.

2.5.5 Masks and templates

The earliest way of reading characters automatically was to look at each character through a family of masks, basically holes in a piece of metal, and see how much was visible in each case. This is not essentially different from measuring intersections with scan lines, except that the mask holes don't have to be lines, they can be any shape. Nor is it very different in principle from *Exercise 1.6.3*, where we have weights of 1 and 0 on the arcs joining input cells to the neural unit. The older terminology also leads to terms like *template matching*, where you have a mask exactly the shape of the character you are trying to detect, and you measure, in effect the distance from the template mask. To give an example, suppose you wanted to detect a vertical bar in the middle of a three by three grid of cells, say a white bar on a black background. Then you could arrange the weights so as to give a zero to every cell of a three by three array of cells where the actual cell value was the same as the desired pattern, and a plus one to every cell where there is a difference. Then this simply gives as sum the Hamming distance between the two images. If zero, they are identical, if 9 they are negatives of each other. Or you can change things round so that the score is plus 1 when they are the same and zero when they differ, which means you go for the high score instead of the lowest. The difference between the two systems is rather trivial. So masks and templates are just old fashioned language for measuring distances between points in $\mathbb{R}^n$ where n is the number of pixels in the image and the entries in the vectors are usually just 0 or 1.

There are some muddled folk around who say derogatory things about template matching methods and then proceed to use Multi-Layer Perceptrons to do exactly the same thing with the names changed, so it is worth being clear about this. You measure similarity between image and template, you measure a distance in some representation space. If you do this for several templates, you get a vector of distances. That gives a new representation in which your choice of the 'right' interpretation is just the vector component which is smallest. This can be interpreted in terms of a somewhat more complicated network than the standard Multi-Layer Perceptron, but the differences are small.

2.5.6 Invariants

It is clearly desirable to select between different methods carefully on the basis of the ease of final classification. We can say something about this at an early stage: if there is a method of getting from a pixel array to a vector of numbers which is invariant with respect to the transformations which occur in images of characters, for example, shifting, scaling, the 'deck transform' that goes from normal font to italic, then it is probably more likely to give a satisfactory result than one which isn't. So it is worth asking, for each measurement scheme, what are its invariants, and are the transformations which get you from one character to a different version of 'the same' character going to preserve them? It is intuitively obvious that if I could give a 'template' character for a /5/ say, and I could list all the ways of transforming this particular point set into every other point set representing a /5/ and to no other point set, then I would have a way of deciding if something is a /5/ or not. If I have some coding into vectors (or anything else) which is *invariant* under these transformations and only these, then I have a good way of telling a /5/ from anything else. This way of looking at things may seem bizarre at first encounter but is rather powerful. It has proved its worth in Physics.

This has led some workers to look for topological invariants, presumably in a spirit of pure enquiry. The topological invariants are very basic indeed, since in topology we allow some very considerable stretchings and deformations of an object: it has been said that a topologist is the sort of man who can't tell a coffee cup from a doughnut. The reason is that you can transform one continuously into the other, if you don't mind looking rather silly when the handle falls off your doughnut while you are using it to hold coffee.

The only topological invariants for characters of the English language are the number of holes in them, and the number of components in a character (two for /;/, for instance). So hole counting allows one to distinguish /1/ from /0/ from /8/, but not /1/ from /5/ or /4/ from /9/. This is (a) fairly useless and (b) can go badly wrong with quite small amounts of noise, which can turn an /S/ into an /8/ or /9/ with no great difficulty. The use of topological invariants therefore stems from a misunderstanding of why invariants are important. The most important transformation between versions of the 'same' character is frequently that of adding noise, which is not a topological transformation at all. So any measuring process, any procedure for getting from pixel sets to vectors, which is corrupted by adding noise is not an attractive bet.

2.5.7 Simplifications and Complications

There are some mildly intelligent things one can do to reduce the computational load in getting from an image to a vector of numbers. One is to reduce the number of pixels we have to put into computations by ignoring the interior and simply looking at the boundary. This will only work if the noise levels are low, since reducing the number of pixels will probably put up the Signal to Noise ratio. If the boundary is corrupted, it won't matter so much if we have a lot of other pixels used in estimating the shape. On the other hand, there are ways of tracing the boundary and then smoothing it in case the odd pixel has been tacked on or gouged out, and since we can tell each character by looking at its boundary and boundaries have fewer pixels, some amount of boundary smoothing followed by an analysis of the boundary shape may be an attractive option in some cases.

A boundary is a relatively simple set and can be specified by *chain code*, where we pick a starting pixel and then specify the direction of

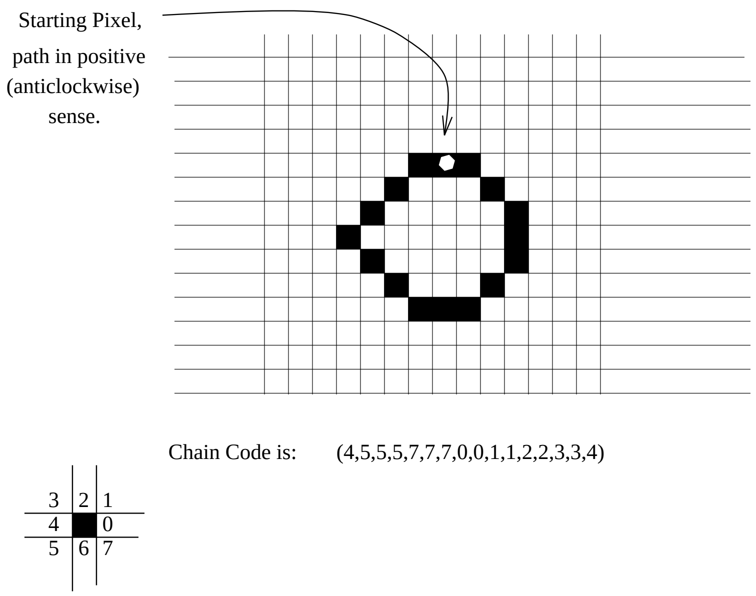


Figure 2.8: Chain Coding of a boundary.

the next by labelling the adjacent pixels, as in *Fig. 2.8*.

Either by such a coding or by linescan methods, it is possible to do such things as count convexities. If circumnavigating a letter /E/ for example, the changes of direction can be determined, and much useable information can be extracted. Many variants of this will occur to the ingenious student. Smoothing boundaries is properly part of an image processing course and there are many ways of doing this; an erosion followed by a dilation can accomplish something of this.

Again, much depends on the kind of data with which one is confronted, but hand written characters are harder to process in this way than printed characters; there is less consistency in general. For printed characters, it is possible to segment the boundary, smooth it, identify strokes and generally extract complex relationships. These methods are not generally robust under change of font, still less do they work for handwritten characters.

2.6 Syntactic Methods

The line of thought which starts off with segmentation of a page of text into lines, then into characters by checking that the set of points is separated by white pixels from other sets, and then tries to recognise the characters by looking at their shape, is liable to several sorts of criticism. Contemplation of *Fig.2.3*, the postcode digits, suggests that the shape based methods might work well with printed characters where there is a high degree of consistency and where, except when noise levels get very high, characters are isolatable by finding boundaries. Up to a point this is correct, as some experimenting with the exercises will show you. On the other hand, for hand-written characters, there is something unsatisfactory about this whole approach. As a person who writes with a pen, I am conscious that I make strokes with more or less care and attention; that the changes of direction are pretty important, but that once I have embarked upon generating a stroke, the thickness is of minor significance. I can easily tell by looking at *Fig.2.3* that the horizontal stroke belongs with the second character, i.e. is part of a /5/ and not of a /7/. I have no trouble about the lower end of the /5/ intruding into the /6/, nor the /2/ intersecting the horizontal stroke of the /5/.

Some sort of syntactic processing is going on here, whereby the image is decomposed by the eye not into pixels and then into characters in one step; there is an intermediate entity, the stroke, made up out of pixels, and itself a component part of a character. The eye has no difficulty joining the two components of the /5/; it expects a horizontal stroke because of the other two strokes making up the /5/. If the horizontal stroke were moved two centimetres to the left, it would have been a /3/, but it has to be one or the other. And so the arrangement of some of the strokes leads to hypotheses about the remaining strokes, hypotheses which are confirmed and lead to identifications of other elements.

Attempts to articulate this view of optical pattern recognition have tended into a quagmire. The Artificial Intelligentsia have tried (in such programs as HEARSAY, from Carnegie Mellon University) to articulate the notion of competing hypotheses resolved by some higher level adjudicator, but their methods of representing hypotheses have been mathematically naive and have had only limited success. To be fair, the problem is far from trivial.

What is known of the physiology of the the lower level processing in the mammalian eye leads us to the notion of a hierarchy of levels, proceeding from essentially pixels, retinal cells triggered by rhodopsin release, to edges or blobs or motion. It is known that a layer away from the retinal cells, there are cells which respond to edges in specific orientations. It is not too hard to believe that a little further back again there are cells which respond to strokes, sequences of edges. And so on, until, conceivably, there are cells responding to digits. Whether this continues to the level of the mythical 'grandmother neuron' which fires when and only when you see your grandmother, is a matter into which we shall not enquire. What is plausible from neurophysiology is that some aggregation process builds 'molecules' out of 'atoms', and then supermolecules out of molecules. In the same way, one might say, as letters aggregate to make words, words are agglutinated to make clichés, and so on. By analogy with the structure of language, whereby a sentence is considered to be derived from more primitive entities by means of rewrite rules, we talk of a *grammar* and of *syntax* of the components.

The diagram in *Fig.2.8* is, more or less, a *parse* of a sentence in terms of a phrase structure grammar, showing how each symbol (like 'sentence', Noun phrase') at any level from the top down, is rewritten to a string of symbols at the level below it, terminating in symbols which are words of the English language. Actually, we could go one level further down into a string of letters; we don't do this because meddling grammarians a few centuries ago regularised the spelling of words, so there is only one way to spell a word, which makes the rewrite rather uninteresting. Around Shakespeare's day, when an English Gentleman spelled the way he fancied and took no nonsense from grammarians, there was a choice. Winston Churchill was notorious for being atavistic in this and other respects, and if your spelling is not all it should be, console yourself with the thought that you have an eminent rôle model.

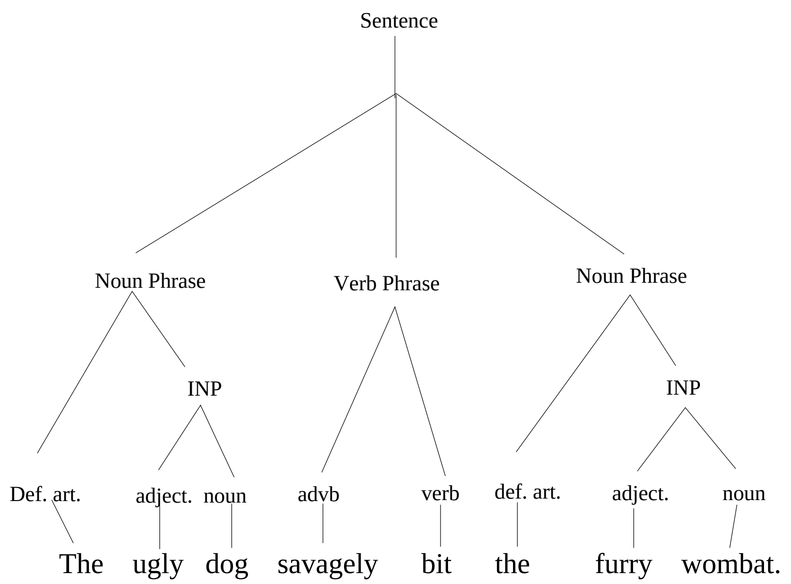


Figure 2.9: The generation of a sentence of words by applications of rewrite rules

The question 'how can one decompose objects like *Fig.2.3* into strokes and then characters?' leads us into interesting places which will be explored more thoroughly when we get to Syntactic Pattern Recognition. The motivation for doing so is worth reflecting on; strokes are produced by people and recognised by people as comprising characters. This is true not just in European scripts of course; Chinese characters and Japanese characters as well as Arabic and the many other scripts are usually decomposed in this way. Cuneiform is not, but you don't see a lot of *that* about these days. The idea that it might be good policy to unravel the problem of reading characters by studying the processes at work in human recognition methods produces interesting results upon which I shall expand later.

2.7 Summary of OCR Measurement Methods

So far in this chapter we have focussed on the recognition of characters, principally on printed characters, and we have been exclusively concerned with getting from a point set in the plane to a vector of real numbers which describes that set. The first difficulty is the problem of segmenting an image of text into separate characters, and I remarked that this approach, in which the character is regarded as the pixel set of interest, cannot be expected to work well for cursive hand-written text, suggesting that the methodology needs attention. We then examined three methods of turning the point set into a real vector. The first was to put the character into a box with a fixed number of pixels in a rectangular array (after some transformation), and then raster scan the array. This method was not recommended. The second was to apply a family of masks to the set and extract information about intersections between the mask and the pixel set. And the third was a family of methods which included taking a Fourier series expansion as a special case, but is more commonly described as the use of moments. This entails computing inner products in the function space $\mathcal{L}_2(D^2)$, and hence projecting down onto a basis for the set of square integrable functions defined on the unit disk. Choosing polynomials in the radius and trigonometric functions in the angle and orthogonalising gives us the *Zernike moments.*

In deciding between different methods, we were much concerned with the problems of having the representation *invariant* with respect to the kinds of transformations which occur in practice with characters in text- shifting, scaling and the 'deck transformation' were mentioned, as was rotational transforms. Making the system robust, or invariant under low levels of noise, can also be thought of as part of the same framework of looking at the problem.

It was observed that these methods could be applied to just the boundary set with some possible saving in computation but at the risk of being more vulnerable to noise.

Specialised methods arising from boundary tracing exist, such as fitting functions to the border of a character and counting convexities, curvatures, or other quantities.

We concluded by observing that the possibility of going through intervening steps, so that a character should more usefully be regarded as being built up out of *strokes*, and strokes being built up possibly out of something else, giving a hierarchy of structures and substructures, was attractive as a generic model for human information processing. Again, there were promises to keep.

2.8 Other Kinds of Binary Image

I have gone to some trouble to discuss binary images of characters because it is now useful to ask oneself, which of the methods discussed can be expected to generalise to other kinds of image? For binary images, the answer is most of the techniques mentioned are still useful.

Binary images, or sufficiently good approximations thereto to allow thresholding to binary to give reliable results, arise in the world naturally, and range from bar codes which are designed to be machine readable and give problems to people, to cartoon sketches of politicians the recognition of which by human beings is just this side of miraculous and by machine currently out of the question. Handprinted characters, signatures, writing on cheques and the results of filling in government forms, all can be captured as, or thresholded to, binary images.

The problem of segmentation arises whenever it makes sense to say there are several different objects in an image, and they are disjoint. Of course, this need not happen at all. Images obtained by pointing a camera at machine parts or other physical objects for the purposes of sorting or counting them can easily look like *Fig.2.10.*

Here, the rotational invariance of measurements is of much greater practical importance than with images of characters which tend to come

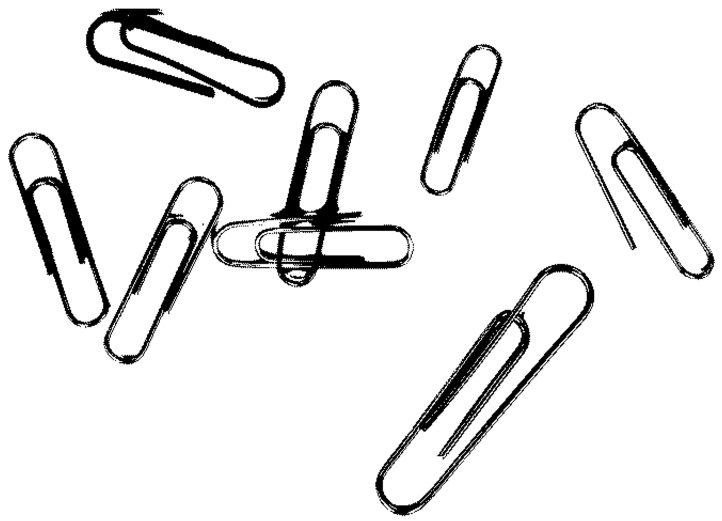


Figure 2.10: Binary image of real objects

reasonably consistently aligned. The spacings are not as consistent as with printed characters, and may not exist at all. (Some OCR enthusiasts have been known, in their cups, to bemoan the passing of the typewriter, which produced output which is much easier to read automatically than properly printed material. This is in conformity with Alder's Law of artificiality, which states that if it is quick and easy for a machine to do something, then whatever it is will be pretty damned ugly.)

Preprocessing by morphology techniques is commonly used; we erode the objects until they are separated, and this allows us to count them and isolate them. This can be something of a failure, as with biological images of, say chromosomes, included as a .tif file on disk, or the paper clips in *Fig.2.10.* Erosion can split objects into two parts, particularly objects which have narrow waists, and naturally fails with high degrees of overlap as in the case of the paper clips. I shall have more to say about the segmentation problem in the chapter on Syntactic Pattern Recognition, where it plainly belongs. It is clear to the reflective mind that we do not in fact recognise handwritten words by first identifying the letters and then recognising the word. We do not read that way. We recognise the words before we have recognised all the letters, and then use our knowledge of the word to segment into letters. It ought to be possible to do the same thing with images of objects having regular structure, or objects which have a multiplicity of occurrences with more or less the same form. In particular, the human eye has no problem with counting the number of paper clips in the image of *Fig.2.10*, but a program has to be written which contains information about the shape of a paper clip. Even a Trobriand Islander who has never seen a paper clip in his life has no great difficulty counting to eight objects instead of six or seven. It is plain that information is extracted from part of the image in order to decide what constitutes an object, and this information applied elsewhere. This is a clever trick which involves learning; how it may be implemented in a program will be discussed later.

For monochrome images with many grey levels, some techniques still survive as useful in specific classes of image. For full colour images, life is harder again, and except for rather simple images, or images which can be converted into grey scale images with only a few grey levels, we have to consider other methods.

I shall discuss particular images and their provenance and outline the kind of modifications which may still give us a vector of real numbers.

2.9 Greyscale images of characters

It is common practice in manufacturing steel to stamp an identification code on each bar of the stuff. This is done with a metal stamp and a hammer or equivalent. The result may be several digits punched into one end of a bar of steel, and it would be nice in some applications to be able to read these digits automatically by pointing a camera at the bars as they roll by. Variants of this problem occur throughout industry; sometimes the only action taken is to make stock entries,

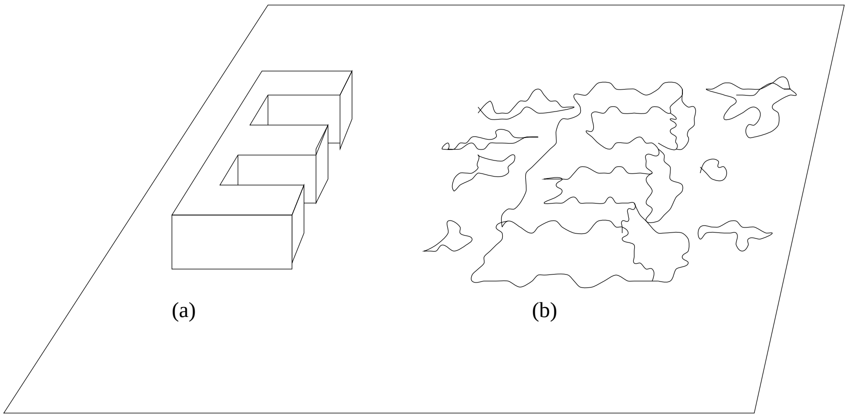


Figure 2.11: A letter 'E' regarded as a function; (a) binary, (b) grey scale

sometimes other automation responses are appropriate. Sometimes, the stamping process is itself automated and needs to be verified.

The image now is a grey scale one, some typical images are provided as TIF files on the accompanying disk. I am grateful to Chris de Silva and Gek Lim of CIIPS, The University of Western Australia, for image files and also many programs used in image processing.

The first thing to do with a greyscale image of a character, if at all possible, is to threshold it back to a binary image. Inspection of an image will, sometimes, allow you to decide if this is feasible. In the case of the stamped characters, thresholding to a binary image does not work well here, since information relevant to the recognition is lost. Ideally one would have a thresholding process which adapted the threshold to the local environment rather than one which merely set a value and stuck with it. This sort of consideration leads us to the theory of *filtering*, which is properly part of an image processing course rather than a pattern recognition book, and some pointers to the literature are provided in the bibliography at chapter's end. Removing 'textural noise' can be done to some extent, but it still leaves us normally with a grey scale image.

In *Fig.2.11* we indicate two images, of a letter E in terms of the function from $\mathbb{R}^2$ to the brightness value in $\mathbb{R}$, one is a binary image and the other a greyscale image. It is useful to look at images as functions defined in some region of the plane in much of what follows.

2.9.1 Segmentation: Edge Detection

The segmentation of greyscale characters is rather more complex because we cannot just trace the white spaces between objects; the separating background will be textured. It makes sense to try to get a characterisation of the background texture and identify that. How this may be done will be outlined below.

Additional information may be available which can assist in segmentation, such as the number of characters and the alphabet. Life is simpler if we are looking for digits than for digits together with alphabetics, and if we know there are precisely four of them and how big they are, and possibly how separated they are, the problem of segmentation becomes easier. If I know that there is going to be a set of parallel texture bands of known width, orientation and number, distinguishable from the digits themselves at the texture level, then finding such separating bands may not be too hard a problem.

In general, the finding of edges in grey scale or colour images is a relatively complicated business; again this is properly part of an image processing course rather than a pattern recognition course, so I shall merely sketch the elements of the methods available.

Regarding the function which picks out a letter /E/ in *Fig.2.11*, we see that the bright regions (for I assume that we have here a bright letter against a black background) are the tops of mountains, and the dark regions are the background. For a black image on a white ground, the signs are reversed but no essential difference occurs. Now if an edge can be said to exist, it is a region where the gradient of the two dimensional function is steep. So we need to go around looking for the places where the gradient is high. This may be done by looking along a horizontal line and measuring the difference in pixel values for consecutive pixels. If this exceeds some threshold, you have a (vertical) edge at the pixel in the centre of the rapid change. Similarly one may look along vertical or diagonal lines. If you have found a possible edge by scanning left to right, some time may be saved by looking at adjacent pixels in the row underneath. Edges that have no extension in a direction transverse to the scan line are probably just noise. This method can be quite fast, and can be altered by making the threshold vary according to whether or not adjacent pixels in a preceding row or column have produced 'micro' edges. This is a direct and obvious method which can work poorly on some images- but then, for each method there is an image it will perform badly on.

We can actually differentiate an image with respect to a direction, horizontal vertical or either of the diagonals, by simply rewriting each pixel to the difference between consecutive values: if in a scanline we have pixels x_n followed by pixel x_{n+1}, then in the differentiated image, we put $y_n = x_{n+1} - x_n$ and the result is, approximately, the partial derivative of the function in the horizontal direction if x_n is to the left of x_{n+1}. Similarly we could partially differentiate in the vertical or diagonal directions with the obvious modifications. When one of the partial derivative values exceed some threshold, we have a putative edge. Real edges are putative edges that have some other putative edges adjacent to them, and the longer the chain of putative edges, the realer the edge becomes. This intuitively appealing idea can be formalised and made respectable.

Making the above system slightly more complicated, we can express it in the following terms: run a step function $g(n)$ along each row of the image, x_n which for consistency ought to be written as $x(n)$. Let g be defined by $g(0) = -1, g(-1) = 1, g(n) = 0$ for $n \neq 0, -1$. Now define $y(n) = \sum_{t=-\infty}^{\infty} x(n-t)g(t)$. Now those depressing infinities vanish since g is zero except at 0 and -1, so $\forall n, y(n) = x(n)g(0)+x(n+1)g(-1)$ which gives us the partial derivative in whichever direction n is increasing. Or at least, a discrete approximation to it.

The expression $\sum_{t=-\infty}^{\infty} x(n-t)g(t)$ is said to be the *convolution* of x with g, and so we have treated a one dimensional convolution which outputs a directional derivative. It is convenient to think of this as taking a sort of generalised 'window' over the function x, and applying a kind of *weighted average* operation to x, weighting by g, in order to get y. If, for example we had $g(0) = 0.5, g(1) = 0.25 = g(-1)$, then we would be doing a smoothing operation to x. The essential idea is simple: we input a function x, and replace $x(t)$ by some linear combination of the values of x in a neighbourhood of t. The set of coefficients in the linear combinations is specified by a function g. It is distinctly reminiscent of Morphology operations, but linear.

In two dimensions, it is common to impose a three by three array of pixels on which g is defined; everywhere else it is supposed to take the value zero. In the nine squares, the window, g, can have any set of values whatever, although we often normalise to ensure that they add up to 1. Then we generate a new image by plonking the window down on the (p, q) pixel of the old image, summing over the neighbourhood of this pixel with the weights of g applied, and then taking the resulting value and assigning it to the (p, q) pixel of the *filtered* output image. Suitable choices of g can find edges in different directions, in essentially the same way as the example of partial differentiation. Three by three windows are on the small side, and larger windows are more common.

Working through a few cases by hand beats the idea into ones skull very rapidly, and you should explore the exercises at the end of the chapter.

Using bigger windows or to put it in another way, taking a convolution with a function having a bigger support, can give improved results at slightly higher cost in time. Image convolutions may be done in hardware on some image processing boards.

And *that* is your very brief introduction to convolution filters. For more details, see any good book on image processing, several are cited in the bibliography.

2.10 Greyscale Images in general

2.10.1 Segmentation

Everything said about greyscale images of characters holds for greyscale images in general, the extra degree of awfulness in segmentation is just as for binary images: there is less information about the extent to which things are separated and separation is not uniform in most applications. The very best thing to do with a greyscale image is to threshold it back to being a binary image if you possibly can. Unfortunately, this often gives poor quality results and more devious methods have to be used.

In *Fig.2.12* we have some grey scale objects, reproduced rather unconvincingly on a laser printer. It will give you some idea of what to expect on the *.tif* files supplied on disk. My thanks to Miss Gek Lim for producing *Fig.2.12* at very short notice.

Note that in this case we may not have a knowledge of what classes of objects are likely to turn up in the image, and that even the Trobriand Islander will have no difficulty counting seven objects, one of which is very different from the others, and two of which are overlapping. One of the six similar objects is bent, and one is short. One has the thread missing, but this is hard to see in the image on paper. This image was obtained by pointing a standard camera at the parts thrown down on a sheet of paper; although it has suffered in its many transmogrifications between camera and page, it is a higher quality image than others used in inspection in industry.

Mathematical morphology techniques can be extended to a limited extent to the greyscale case, see any of the books in the bibliography on

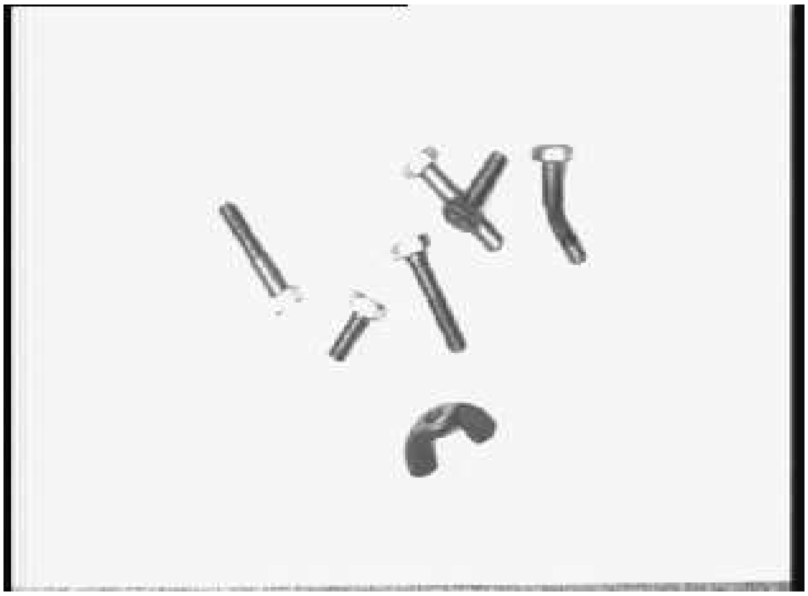


Figure 2.12: Nuts and bolts.

the subject. Finding boundaries of objects by differentiating images sometimes works or can be made to work. There is no substitute for experience here, and if you do the exercises you will get plenty.

2.10.2 Measuring Greyscale Images

Let us suppose that, by finding edges instead of borders or by locating well defined spaces between the objects, we have succeeded in putting a box, perhaps an irregular quasi-border, around the object. The assumptions that went into measuring the 'contents of a box' for binary images have to be examined anew for grey-scale images.

Transforms to normalise and register the image in some standard location proceed without essential alteration. In the case where we are lucky enough to get regular boxes we may enclose our objects in standard rectangles if this looks to be plausible; much depends on what can be said *a priori* about the objects which may be found. If any edge boundary can be found, then at least the pixel array can be reduced to a centroid and brought inside the unit disc in $\mathbb{R}^2$. The computation of the centroid now has to be taken over real rather than integer values, but that is straightforward enough. Everything we learnt about moments is still applicable, except that f_A is no longer a characteristic function of the set, it is the function giving the pixel values. We may conveniently normalise these to lie between 0 (black) and 1 (white). Moment methods are therefore popular and moderately robust in general.

It is useful to think of mask methods for binary images as an attempt to look at different regions of the image simultaneously, so as to do some parallel (or more accurately concurrent) processing. If we take some shaped window and look at the image through it, we have the basic idea of mask processing. As a first development of that idea, we can measure what we see in a number of ways: one would be to collect a few central moments. Another would be to count regions, or pixels. Mask methods which measure the intersections with scan lines, or with some other kind of window on the image, can be just as effective with grey scale images as with binary images when they are estimating the total intensity in the field of the mask, but may fail due to incorrect thresholding when used to decide if something occurs or not. So the system of *Fig. 2.7* which simply counts intersections is not robust. What can be done is to extend the mask into higher dimensions: instead of regarding a mask as a sort of hole through which one looks at an image, a hole with its own characteristic function, one regards it as a continuous function defined over a region, and then makes a comparison between the actual function which is the image, and the mask function. There are several different sorts of comparison which can be made. I leave it to the ingenuity of the reader to devise some: I shall return to the issue later in the book.

Mask methods tend to become, for computational convenience, convolutions at selected locations, and tend to be specific to the classes of objects to be classified. Fortunate is the man who can say in advance what kind of sub-images he is going to get after he has done some edge tracing. A basic problem with mask based methods is that it may be difficult to tell when you have something odd in the image which is different from anything your program has ever seen before.

The best way to find out about mask methods is to invent a few and see what they do. The exercises will give you an opportunity to do this. Differentiating images is easy to do and perfectly intelligible after a little thought, and the results on greyscale images can be quite satisfying. See the disk files for some examples.

2.10.3 Quantisation

Thresholding a greyscale image to a binary image is a particular case of reducing the range of levels. There are many applications where a reduction in the range has advantages. Quantisation over the greyscale is particularly important in image processing associated with compressing the image into fewer bits. A common technique of image compression is to first do a Discrete Cosine Transform of the image, which is just the even part of a Fourier Transform, and then take the resulting new image and quantise it judiciously. Then this quantised DCT image is stored. When the inverse DCT is performed, the original image is restored to quite high approximations, since the eye is relatively insensitive to very high and very low spatial frequencies. The same technique can, of course, be regarded as a measurement method.

Since one wants to make regions of the image which are close both in space and in grey level more likely to be assigned the same quantised value than regions of the image which are separate in either space or in grey level, it is convenient to work in the space of the graph of the function, as with *Fig.2.11.*

Again, this is more likely to be treated in a good book on image processing than one on Pattern Recognition, but the issue must be mentioned.

2.10.4 Textures

It is worth while taking some colour pens and making strawberries, by the simple means of first colouring in a heart shaped region in red, and then going over it drawing lots of little green 'V' shapes. The resulting object, seen from a suitable distance, has a rather curious speckled look to it. The artist Cannaletto made a whole heap of money out of his discovery that you could make green paint look a bit like water if you painted lots of white ripples on top of it. Similar results can be obtained by playing with the background stipple patterns on the control panel of a Macintosh computer. Such a result is called a *texture* and may be obtained, as in the case of the Mac, with binary images.

If we take a rectangular grid and move it around the image, we find that the pattern seen at one location strikingly resembles the pattern seen when it is shifted by some more or less fixed number of pixels in any fixed direction. In general, this pattern is statistical rather than deterministic, but we can extract some statistics for the values in windows in this case too. For example, on a piece of reflective steel, upon which some characters have been incised, there is no regularity apart from that in the characters, but the mean grey level and the variance may be fairly constant. The eye can quite easily distinguish between regions having the same mean grey level if one has a much higher variance than the other, providing the variation is within some kind of visual acuity. Similarly, other kinds of correlation between pixel values may appear as a 'texture' variation in the image, and this information may be extracted and put to use in obtaining a 'feature' of an image. In the exercise at the end of chapter one where you were asked to distinguish pictures of underclad ladies from pictures of trees, even the most glorious of New England Fall Maple trees can easily be distinguished automatically from naked flesh, not by colour but by texture. The variation in a square block of 25 pixels is bigger for leaves than for flesh; try it if you don't believe me. I have studied such images with assiduous attention to detail and am confident of my ground.

2.11 Colour Images

2.11.1 Generalities

A colour image is normally three greyscale images. This may indeed be how the colour image was obtained: some of the astronomical pictures of planets and satellites of the solar system seen from spacecraft were obtained by putting a green filter and taking a greyscale image, then repeating with red and blue filters.

2.11.2 Quantisation

Quantising the graph of an image from $\mathbb{R}^2$ to $\mathbb{R}^3$ is only a little more difficult than quantising the graph of an image from $\mathbb{R}^2$ to $\mathbb{R}$, but the space in which the points sit is now five-dimensional. This ought not to worry those heroic enough to be reading this book. We are going to be looking for clusters in this space, and how this may be done will be described later, for we need to be very interested in clustering methods for other reasons.

2.11.3 Edge detection

If an edge is going to be really there and not some accident of brightness, then it ought to have some extension in space and also in colour, that is to say, if we think we have a sharp change in colour at some point, then neighbouring pixels ought to show similar changes. This means that it suffices to ensure that a putative edge in the red image ought to have spatial extension, likewise the green and blue images. So edge detection is slightly easier, because there are often simultaneous changes in more than one of the greyscale images at the same pixel. It would not be surprising if there were changes in the different greyscale images at nearby pixels, and smarter methods can look for this effect. Again, this sort of issue belongs in Image Processing courses, but it is worth doing a little meditation on what is possible before reading the books or attending the courses.

2.11.4 Markov Random Fields

As with greyscale images, texture effects may be described as local correlations between pixel values. Various models of this kind of phenomenon have been studied, including the theory of markov Fields. The work of Geman and Geman is notable in this regard: these are two people, not clones. See the bibliography for more details; I shall have a little more to say on this topic later but it represents a degree of technicality inappropriate for the present book. [2]

2.11.5 Measurements

If we can actually identify an object by some segmentation process involving boundaries, either of brightness or colour or texture, then again we can use mask methods or moments to try to obtain information about the shape of the object in the three colours. This gives us three separate sets of mask values or moments, and they may correlate. Indeed it is to be hoped that they do.

Finding objects out of correlations or textures alone is at the present time a convenience and tends to be image specific, as with the girly pix, rather than a pressing need. This may change as the objects to be recognised get more complicated and more interesting.

---

[2] I promised that this book would be user friendly and am trying to keep my word. Maybe my next book will be academically respectable and properly inscrutable.

2.12 Spot counting

One of the more fundamental things to be done in pattern recognition is simply counting the objects in an image. Going back to *Fig.2.10*, the paper clips, we see that the objects may overlap, and that this can very effectively clobber segmentation methods. If the objects are simple in shape, say disk shaped, then something can be done, but it is mostly nasty and *ad hoc* and therefore gives a thrill only to debauched computer programmers.

Some of the images on disk are microphotographs of cells, and it is frequently necessary to count the cells of a given type or colour. They usually overlap. Erosion of the cells until they fall into separate points, boundary smoothing and colour quantisation have been used by Dr. Chris deSilva and his students at the Centre for Intelligent Information Processing at the University of Western Australia, to obtain reliable cell counts. More advanced methods will be discussed later. It should be noted that there are an awful lot of cells to count, and automating the counting process is a critical bottleneck of mass screening schemes, in, for example, the early detection of breast cancer.

2.13 IR and acoustic Images

Not all images are produced by visible light, and images produced by acoustic or Infra-Red means are common, as are X-Rays and Tomographic images. Because there are a lot of people in this world who want to live forever (despite having no better idea of how to spend a weekend than watching football on television), a great deal of money is being poured into medical research, and some of this goes into automating the task of looking at images of people's insides. Since cutting them up in order to take visual images is considered intrusive and has side effects, anything which can give an image of a patients insides and allows him to walk away afterwards is a **Good Thing**. Making sense of the resulting image, and correlating it to what you would have got if you'd opened the patient up with a hack-saw, is a task too complicated to be left to the medical profession, and hence the interest in PET scans, CAT scans, MRI scans, Acoustic Imaging, and generally the results of pumping into patients anything too small to leave visible scars.

The military have what might be termed the converse interest in other than visual images; being able to kill people and blow up buildings on a dark night is of obvious importance if you get your kicks that way. So being able to tell a tank from a Lamborghini by inspection of an Infra-Red image has its uses. If instead of being a military analyst safe in a bunker you are a missile trying to make up your mind whether or not to destroy yourself and your target, the automation issue arises.

Existing methods of analysis are essentially of the sort described for other images, but the source sometimes means that the funding situation is improved. It is for this reason that some energy and time has been dedicated to this class of data. Since many images are structured, syntactic methods have particular applicability.

Flaw detection in pipes, rails and machine parts by ultrasonic imaging, has been accomplished successfully using the methods I shall discuss in the chapter on Syntactic Pattern Recognition.

2.14 Quasi-Images

In the case of a binary image, we are dealing with a pixel array in two dimensions, which it is convenient to describe as a point set in $\mathbb{R}^2$. There are many measurement processes which give point sets in higher dimensional spaces. For example, a standard method of sorting precious stones by colour is to place them on a tray and shine a light through the stone, as in *Fig.2.13.* The light is refracted to some extent by the stone, and the emergent beam is coloured. By placing a prism in the path of the emergent beam and an array of photocells across the spread out beam, the energy at different frequencies may be measured.

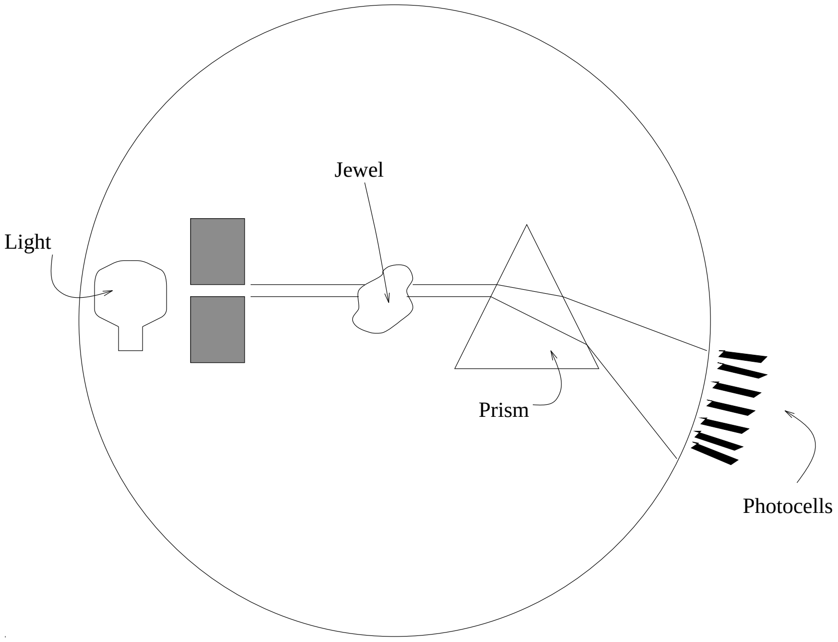


Figure 2.13: Measuring coloured glass.

If there are twelve such photoreceptors, the stone is transformed by the measuring process to a point in $\mathbb{R}^{12}$. The usual methods of cluster analysis and recognition may then be applied.

2.15 Dynamic Images

A video-recording of a football match shows figures moving around in ways which are, for short periods, relatively predictable. Using temporal information to identify objects in a time series of images is possible using such techniques as differencing consecutive images to distinguish the moving objects from the background. It is, of course, very simple to difference consecutive images and simply measure the amount of change. A simple intruder alarm which accomplishes this is currently obtainable for under fifty dollars. Unfortunately it counts newspapers blown by the wind and passing cars as intruders. Actually being able to distinguish between a man, a sheet of newspaper, and a passing car is possible by the techniques described. More sophisticated recognition issues, such as distinguishing a man from a dog, naturally arise. Because of the huge variety of images which a human being can confidently label as 'man' or 'dog', the existing methods are not satisfactory. It is possible to submit the whole image to a huge neural net, and this has been tried by optimists in need of an education, but any passing statistician would give a pitying smile if informed of this plan. It is something like trying to recognise text a page at a time because it is hard to segment into letters. You'd have to be pretty desperate to try it.

2.16 Summary of Chapter Two

This chapter has discussed, in outline, material which can be found in detail in books on Image Processing; references to such books may be found in the bibliography following. It has, I hope, put the material into some kind of perspective at the expense of being sketchy and superficial. The exercises, if done diligently, will also fill in some details which have been left out.

The usual problem when confronted with an image is that it contains rather a lot of things not just one. This leads to the segmentation problem; trying to chop up the image so each part of it contains just one recognisable object. Having isolated the parts, it is then possible to move on to the task of identifying them. It is irritating in the extreme that human beings seem to do this in the reverse order.

Segmentation will often entail boundary tracing which in turn involves edge detection for greyscale images. This may be conveniently accomplished by convolution filters. I have sketched out the underlying idea here, but to find out the details, more specific texts are indicated.

Having segmented the image, which in easy cases may be done by finding clear spaces between the objects and in hard cases cannot be done at all, we may throw away the bits that cannot correspond to objects we can recognise. We may then choose to normalise the result of this; this might be a matter of inserting the object in a standard array, or moving it and scaling it so that it just fits into the unit disk.

We then have two basic methods of turning the image into a vector; one is to use a 'mask' family which may inspect bits of the image and compare the fit to some standard mask value. This does a 'local' shape analysis of some sort. The alternative is a moment based method, which includes FFTs, DCTs and Zernike moments as special cases of orthogonal basis functions.

Images may be obtained from a wide variety of sources, and simply counting the objects in an image may present formidable problems.

2.17 Exercises

1. Generate a square grid such as that in the background of *Fig.2.6*, if necessary using pen and paper, and mark in some pixels. Repeat with another grid and mark the origin in both grids. Now compute carefully the dilation and erosion of one by the other. Write a program to dilate and or erode one pixel array by another and display all three arrays.

2. With the program of the last exercise, explore alternate erosions and dilations on some .tif files obtained from a scanner or from disk.

3. Use erosion techniques to count the objects in *Fig.2.10* or *Fig.2.12.*

4. Write a program to find the lines of text and the spaces between them in some scanned image of text such as *Fig.2.4.* You may use any method you choose.

5. Extend the previous program so as to separate out each line of text into words.

6. Extend the previous program so as to isolate single characters. What does it do to an image such as *Fig.2.2*?

7. Write a boundary tracing program and try it out on a .tif file of your choice. Alternatively, you might like to start on a bitmap file if you have access to a unix workstation and some familiarity with the beast. See the *.c* files on disk for a helping hand.

8. By hook or by crook put a box around each of some characters in a printed word, either from *Fig.2.4* or somewhere else. Normalise the boxes to a standard size, having first worked out what a good standard size is. Print your normalised images and ask whether you could recognise them by eye! Recall that a 2 by 2 matrix applied to each pixel will give the effect of a linear transformation. It is recommended that you expand or shrink each axis separately to a standard size as your first move.

9. How many scan lines would you think necessary to distinguish the characters in your sample? Turn each character into a vector by scanline methods. Can you tell by looking at the vectors what the characters are? (It might be useful to start of with a set of digits only, and then to investigate to see how things change a the alphabet size increases.)

10. Are there any alternative mask shapes you feel tempted to use in order to convert each character into a vector?

11. Segment by hand an image such as *Fig.2.12* and normalise to occupy the unit disk with the centroid of the function representing the region at the origin. Divide the disk up into sectors and some the quantity of grey in each sector. Can the resulting vectors distinguish shapes by eye? Can you devise any modifications of this simple scheme to improve the recognition?

12. Obtain images of shapes which have some kind of structure which is complicated; orient them in some canonical way and look at them hard. Can you devise a suitable collection of masks to discriminate the shapes? Characters under grey levels present an interesting problem set. Try getting camera images of stamped

digits (easily made with modelling clay and a set of house numbers) under different illuminations.

13. Compute the central moments of one of your characters up to order two by hand. Write a program for doing it automatically and confirm it gives the right answer. Extend your program to compute central moments up to any order. Compute them up to order n for all your characters, and check by eye to see if you can tell them apart by looking at the vectors, for different values of n.

14. Write a program to normalise any array of characters into the unit disk with the centroid as origin.

15. Write a program to compute the first 12 Zernike moments for a pixel array which has been normalised into the unit disk.

16. Use a border following algorithm to extract external borders of your characters and then apply the last program to see if you can still recognise the characters by looking at the vectors obtained from taking moments of the boundaries.

17. Find an algorithm for finding edges in greyscale images and use it to differentiate an image, bringing the borders of objects into relief. The disk programs might be places to start looking, but you may find other books of value.

18. Write a program which smooths a time series of real numbers by applying a moving average filter. The program should ask for the filter coefficients from -n to n after finding out what n is. To be more explicit, let $x(n)$ be a time series of real numbers; to get a good one, take the daily exchange rate for the Dollar in terms of the Yen for the last few weeks from the past issues of any good newspaper. Let $g(n)$ be defined to be zero for $|n| > 5$ and assign positive values summing to 1 for $-5 \leq n \leq 5$. You could make them all equal, or have a hump in the middle, or try some negative values if you feel brave.

Your program should generate a new time series $y(n)$ defined by

$$y(n) = \sum_{j=-\infty}^{j=\infty} x(n-t)g(t)$$

and plot both x and y.

19. Modify your program to deal with a two dimensional filter and experiment with it; get it to remove 'salt and pepper' noise introduced into a grey scale image by hand.

20. Beef up the above program even further, and use it for differentiating images. Try to segment images like *Fig.2.12* by these means.

21. Generate a binary waveform $x(n)$ as follows: if the preceding value is black, make the current value white, and if the preceding value is white, make the current one black. Initialise at random. This gives a trivial alternating sequence, so scrap that one and go back the preceding two values.If the two preceding values are both black, make the current value white; if the last two values were black then white, make the current value white; if the last two values were white make the current value black, and if the last two values were white then black, make the current value black. Well, this is still pretty trivial, so to jazz it up, make it probabilistic. Given the two previous values, make a random choice of integer between 0 and 9. If both the preceding values were black, and the integer is less than 8 make the current value white, if it is 8 or 9 make it black. Make up similar rules for the other three cases. Now run the sequence from some initial value and see what you get.

22. Could you, given a sequence and knowing that it was generated according to probabilities based on the preceding k values as in the last problem, work out what the probabilities are?

23. Instead of having k = 2, does it get more or less interesting if k is increased?

24. Instead of generating a sequence of binary values, generate a sequence of grey scale values between 0 and 1 by the same idea.

25. Can you generalise this to the case of two dimensional functions? On a rectangular pixel array, make up some rules which fix the value of a pixel at 0 or 1 depending on the values of pixels in the three cells to the North, West and North-West. Put a border around the array and see if you can find rules for generating a chess-board pattern. Make it probabilistic and see if you can generate an approximate texture. Do it with grey levels instead of just black and white.

26. Make up a probabilistic rule which decides what a cell is going to have as its value given all the neighbouring pixel values. Generate a pixel array at random. Now mutate this initial array by putting a three by three mask on, and using the surrounding cells to recompute the middle one. Put your mask down at random repeatedly. Do the following: always make the centre pixel the average of the surrounding pixels. Now make the top and left edge black, the bottom and right edge white, and randomise the original array. Can you see what must happen if you repeat the rewrite operation indefinitely? Run this case on your program if you can't see it.

27. Can you, in the last set of problems, go back and infer the probabilistic rules from looking at the cell values? Suppose one part of an array was obtained by using one set of values and another by using a different set, can you tell? Can you find a way of segmenting regions by texture?

28. Take an image of some gravel on white paper, then sieve the mixture and take another image of what gets through the sieve and another of what gets left in the sieve. Repeat six times. Can you sort the resulting images by any kind of texture analysis? Try both high and low angle illumination in collecting your data.

2.18 Bibliography

1. Terry Pratchett *Moving Pictures* Gollancz 1990.

2. Kreider, Kuller, Ostberg and Perkins, *An Introduction to Linear Analysis*, Addison Wesley 1966.

3. Wojciech Kuczborski, *Real Time Morphological Processes based on Redundant Number Representation*, Ph.D. thesis, The University of Western Australia, 1993.

4. Tony Haig, *Reconstruction and Manipulation of 3-D Biomedical Objects from Serial Tomographs*Ph.D. thesis, University of Western Australia 1989.

5. Tony Haig, Yianni Attikiouzel and Mike Alder, *Border Following: New Definition Gives Improved Border* IEE Proc part I, Vol 139 No 2 pp206-211 April 1992.

6. Jim R. Parker, *Practical Computer Vision using C* New York: Wiley, 1994.

7. Edward R. Dougherty and Charles R. Giardina, *Image Processing Continuous to Discrete, Vol.1.* Prentice-Hall 1987.

8. Edward R. Dougherty and Charles R. Giardina, *Matrix Structured Image Processing* Prentice-Hall 1987.

9. Charles R. Giardina and Edward R. Dougherty, *Morphological methods in image and signal processing* Prentice-Hall, 1988.

10. Robert M. Haralick and Linda G. Shapiro, *Computer and Robot Vision* Addison-Wesley 1991

11. IEEE Transactions on Image Processing : a publication of the IEEE Signal Processing Society, from: Vol. 1, no. 1 (Jan. 1992)

12. John C. Russ, *The image processing handbook* CRC Press, 1992.

13. Theo Pavlidis, *Algorithms for graphics and image processing* Berlin Springer-Verlag, 1982.

14. Azriel Rosenfeld,(Ed) *Image modeling* New York : Academic Press, 1981.

15. Azriel Rosenfeld, Avinash C. Kak. *Digital picture processing* New York Academic Press, 1982.

16. Laveen N. Kanal and Azriel Rosenfeld,(Eds) *Progress in pattern recognition* Elsevier North-Holland, 1981

17. Jacob Beck, Barbara Hope, Azriel Rosenfeld, (Eds) *Human and machine vision* New York Academic Press, 1983.

18. Azriel Rosenfeld, editor *Human and machine vision II* Boston : Academic Press, 1986.

19. Rosenfeld, Azriel, *Picture processing by computer* New York : Academic Press, 1969.

20. Julius T. Tou and Rafael C. Gonzalez, *Pattern recognition principles* Addison-Wesley Pub. Co., 1974

21. Rafael C. Gonzalez, Richard E. Woods, *Digital image processing 3rd ed* Addison-Wesley, 1992.

22. Serra, Jean Paul, *Image analysis and mathematical morphology.* New York : Academic Press, 1982.

23. Shimon Ullman and Whitman Richards, (Eds) *Image understanding* Norwood, N.J. : Ablex Pub. Corp., 1984.

24. Narendra Ahuja, Bruce J. Schachter, *Pattern models* New York : Wiley, 1983 .

25. Wang, Patrick Shen-pei, (Ed) *Computer vision, image processing and communications systems and applications : proceedings of the 1985 NEACP Conference on Science and Technology, Boston, USA, Nov. 30-Dec. 1, 1985* Philadelphia : World Scientific, 1986.

26. Wang, Patrick Shen-pei, (Ed)*Array grammars, patterns and recognizers* World Scientific, 1989.

27. K. S. Fu and A. B. Whinston, (Eds) *Pattern recognition theory and applications [proceedings of the NATO Advanced Study Institute on Pattern Recognition-Theory and Application, Bandol, France, September 1975]* Leyden Noordhoff, 1977.

28. K.S. Fu, (Ed) *Applications of pattern recognition* CRC Press, 1982.

29. King-Sun Fu and T.L. Kunii, (Eds) *Picture engineering* Springer-Verlag, 1982.

30. K.S. Fu , with contributions by T.M. Cover, *Digital pattern recognition* Springer-Verlag, 1980.

31. Tzay Y. Young and King-Sun Fu, *Handbook of pattern recognition and image processing* Academic Press, 1986.

32. George Robert Cross *Markov random field texture models [microform]* Ann Arbor, Mi. University Microfilms, 1981, 3 microfiche (Ph.D. thesis)

33. Alireza Khotanzad and Jiin-Her Lu, *Classification of Invariant Image Representations Using a Neural Network.* IEEE Trans. Speech and Sig.Proc. Vol.38 No.6, June 1990.

34. F.Zernike, *Beugungstheorie des Schneidenverfahrens und Seiner Verbesserten Form, der Phasenknotrastemethode* Physica Vol.1.pp 689-704, 1934.