Blobs & Texture

Most vision applications ultimately require the analysis of binary images because they are easier to measure than colour or grey-scale. Images, captured from a camera must be processed first, to yield a binary image, which may then be processed further. This pre-processing might, for example, be edge-detection filtering, followed by thresholding. Finally, measurements are taken on the resulting binary image. The observed and expected values of a range of measurements are then compared. This may consist of a very simple check, for example, that a single metric value lies within defined limits. However, it might be very much more sophisticated, involving advanced logical reasoning.

Figure 5.1 illustrates the broad range of binary images that we need to consider. There are several important types, including

- Simple blob (e.g.silhouette of a con-rod, slice of bread, cam, leaf)
- Complicated blobs (e.g. cake-decoration patterns)
- Multiple blobs (e.g. structured lighting, shoe parts,)
- Images with fragmented features and a noisy background.(e.g. crack)
- Texture (e.g. bread texture, rattan cane)

A figure in a binary image, is properly called a *binary object*, but for convenience, it will be called a *blob*. The formal definition of a binary object requires that there is a continuous path of white pixels joining any two given white points in that blob. (Figure 5.2) What constitutes a *"continuous path"* requires some explanation. This relies on the concept of *connectivity* and what constitutes *neighbouring pixels*. (Figure 5.3)

Blobs

Your own informal idea of what constitutes a blob is perfectly satisfactory for our present purposes. A binary image can, of course, contain one or more blobs. It is customary to discuss the processing of *white* blobs. This is merely a convention; black blobs often arise naturally. If we wish to analyse black blobs, we simply negate the image first. Throughout this chapter, we will assume that we are analysing and measuring white blobs. The analysis of a blob can often be simplified by measuring each of its component parts, such as "holes" and "bays" separately. (Figure 5.4)

Texture

Many natural and artificial surfaces generate textured binary images after processing. (Figure 5.5) It is not sensible to try and measure each of the constituent blobs an such an image. Instead statistical analysis methods are more appropriate. We will see how the analysis of binary-image texture can be made simpler by selective filtering, based on concepts of distance (within an image) and neighbourhood.

Special type of grey-scale image

A binary image can be regarded as a special type of grey-scale image. So, some of the image processing operations described in the previous chapter can be applied with good effect to binary images. We will see several instances of this as we progress. However, binary images can be processed in other ways that have no counterpart for grey-scale images. Without resorting to mathematical terminology, it is sometimes very difficult to explain *how* binary image processing operators work, although describing *what* they do may be trivial. For example, we will illustrate a procedure, called the *Convex Hull*, that effectively places a rubber band around a blob. It is not nearly so easy to explain how this can be achieved as it is to describe its function.

Array and edge-based representations of a blob

In Chapter 3, we described a binary image as a 2-dimensional array whose elements are allowed to have only two possible values. This is a natural format for images derived from a camera containing an array of photo-sensors. However, as we will see later that, sometimes, other image representations are more convenient. Some of these rely on describing the path followed by a "bug" as it travels around the edge of a blob. This exploits the fact that only the edge pixels are needed to define a blob completely. Edge-following generates less data than the array representation requires. For some operations, computer programs to manipulate edge-sequence representations are often simpler to write and understand. For example, detecting corners and edge smoothing are better defined in terms of edge following than an array of 0s and 1s. However, remember that video cameras initially generate image data in array format and this must be converted to an edge-tracing description before the benefits of the latter are enjoyed.

Separating blobs

We will concentrate on processing images containing a single blob. In order to justify this, we need to show that we are not restricting the scope of our discussion unduly. There are several techniques for separating the component parts of a multi-blob image. (See Figure 5.6.} Again, the details of the calculation needed to do this are beyond the scope of this chapter. Let it suffice to say that it is possible to isolate complex inter-twined blobs. (Figure 5.7) A program can isolate each blob in turn very quickly, discarding small/large ones at will. For example, programs can find the blob with the largest area, or discard those consisting of fewer than a given size. No human being can match the speed of a program! We can even use the same methods to plot a route through a complicated maze, give its plan view. (Figure 5.8) There is an obvious application for this in planning conductor routing on printed circuit boards

A note on the illustrations in this chapter: As in Figure 5.8, we will sometimes display a multi-level image, or pseudo-colour image, so that several different binary images can be superimposed and compared to each other. For a similar reason, an outline will sometimes be superimposed on other blob features.

Applying Grey-scale operators to Binary Images

There is only one binary monadic operator: the binary equivalent of grey-scale negation. This is called *Inversion*, or sometimes *Negation*. As would be expected, white pixels become black; and black pixels are mapped to white. (Figure 5.9)

There are just three binary dyadic operators:

- **AND** Each pixel in the output image is white if the pixels at the corresponding positions in the two input images are both white. Otherwise, it is set to black.
- **OR** Each pixel in the output image is white if either of the pixels at the corresponding positions in the two input images is white. Otherwise, it is set to black.
- **Exclusive OR (XOR)** Each pixel in the output image is white, if the pixels at the corresponding positions in the two input images are different. If they are the same, it is set to black.

Sometimes, it is helpful to think of a binary image as being a simplified grey-scale image in which there are just two levels: 0 (black)and 255 (white) Applying a blurring operator to such an image results in a grey-scale image that has intensity values covering the whole range [0,255]. By thresholding the result, a new binary image is created in which the edges are smoother than in the original. This will be illustrated and discussed in more detail later, where it will be compared to other methods for edge smoothing.

The grey-scale operators *row-integration* and *row-maximum* functions can also perform important functions when they are applied to binary images, (Figure 5.10) Two related operators are also illustrated here. The first of these allows the length of each horizontal chord across the blob to be measured. (Figure 5.10[BL]) The second finds the centre of each horizontal chord. (Figure 5.10[BR]) Figure 5.11 demonstrates a naive procedure for distinguishing two similar shapes using row-integration. This example is important because fitting the wrong car brake-pad could lead to a fatal accident.

We turn our attention now to demonstrate how edges in binary images can be found using operators that we have already encountered. (Figure 5.12) Shifting the image, subtracting the original from the result and then thresholding can be used to highlight horizontal, vertical, or diagonal edges. However, here are more efficient ways to achieve the same result and we will turn our attention now to consider these as members of a much broader class of operator.

Neighbourhood Functions

Like grey-scale local operators, *Binary Neighbourhood Functions (BNFs)* calculate the intensity of each output pixel from the values in a group of input pixels. Again, the intensity for a single output pixel is calculated from a compact group of nearby pixels. The same process is repeated, once for each output pixel. Initially, consider the 9 pixels within a 3x3 processing window, called a *Structuring Element (SE)*. We can easily describe several functions of this type quite succinctly. Each of the functions described below, sets the output pixel white under the conditions specified. All other output pixels are set to black. (Figure 5.13)

- *Dilating white areas:* Any one, or more, of the 8-neighbours is white. [This expands all white objects by one pixel in each direction. Two blobs that are very close together may "fuse" into a single entity.]
- *Eroding white areas:* All 8-neighbours are white. [This shrinks all white objects by one pixel on each side. Very small blobs may disappear altogether and narrow streaks begin to disintegrate.]
- *Isolating black pixels:* The central pixel is black and its 8-neighbours are all white. All other pixels are set to black.
- *Isolating white pixels:* The central pixel is white and its 8-neighbours are all black. All other pixels are set to black.
- *Detecting limb ends of matchstick-figures:* The central pixel is white and exactly one 8-neighbour is white. [We will discuss the generation and importance of matchstick-figures later.]
- *Dismembering matchstick-figures:* The central pixel is white and exactly two 8-neighbours are white. Cut the match-stick here
- *Detecting joints of matchstick-figures:* The central pixel is white and three or more 8-neighbours are white.
- *Majority filtering:* The majority of the pixels within the 3x3 neighbourhood are white. [This provides an effective filter for reducing the effects of camera "noise".]
- *Detecting inside-edge:* The central pixel is white and at least one of its 8-neighbours is black. [This detects pixels adjacent to the black-white boundary but lying inside the blob.]
- *Detecting outside-edge:* The central pixel is black and at least one of its 8-neighbours is white. [This detects background pixels adjacent to the edge.)

Recall that, in the previous chapter, we saw that a grey-scale local-averaging operator can effectively count the white pixels in a 3x3 region. Another 3x3 grey-scale operator can detect any given logical pattern of 0s and 1s. There are more efficient ways to do this but it is helpful to keep this idea in mind when reading Figure 5.13.

Dilation and erosion

These two important operators require special attention. As described above

- *Dilation* is the process of making the output pixel white if it or any of its 8-neighbours is white.
- *Erosion* is the process of making the output pixel white only if it and all of its 8-neighbours are white.

Notice the following points. (See Figures 5.14 - 5.18.)

- Repeating dilation with a 3x3 SE, produces the same result as that from dilation with a 5x5 pixel SE. (Figure 5.14)
- Performing 3x3 dilation N times produces the same result as that from dilation with a (2N+1)x(2N+1) pixel SE.
- Performing 3x3 erosion N times produces the same result as that from erosion with a (2N+1)x(2N+1) pixel SE.
- Dilation, or erosion, followed by *exclusive OR* draws an edge contour for each blob. (Figure 5.s 15 & 16)
- Dilation followed by erosion (called *closing*) eliminates small black spots. Larger spots can be deleted by increasing the size of the SE. (Figure 5.17)
- Erosion followed by dilation (called *opening*) eliminates white dark spots. Larger spots can be deleted by increasing the size of the SE.
- Negating a binary image then dilating it has the same effect as erosion followed by negation.
- Performing dilation, erosion and then XORing the result with the original detects thin black arcs and small black spots. (Figure 5.18)

Generalising the Scope

Dilation and erosion can be made even more useful by generalising them by reforming the shape of the structuring element. Here is a generalised version of erosion. (Figure 5.19)

1. *Construct a "mask" consisting of a single small blob. This is a more general use of the term Structuring Element (SE).*
2. *An SE designed to detect the apex of a sans serif, upper-case letter "**A**" might resemble "∧" or "∩", while two SEs (⊢ and ⊣) would be required to identify the joints on such a letter.*
3. *Scan the image, in turn placing the SE in every possible position over the input image.*
4. *At each position, if every point in the SE lies over a white pixel in the input image, the pixel that is closest to the centre of the SE in the output image is set to white. Otherwise, it is set to black.*

The equivalent procedure for a generalised version of dilation replaces Step 4 by the following:

4. At each position, if one or more points in the SE lies over a white pixel in the image, the pixel that is closest to the centre of the SE is set to white. Otherwise, it is set to black.

Erosion and dilation and combinations of them are called *Morphological operators*. (*Morphology*: study of form.) Figures 5.20 - 5.26 indicate some possible uses. Figure 5.20 hints at techniques that are useful for *Optical Character Recognition (OCR)*, which is required for reading printed text. Figure 5.21 shows how morphology can be used as part of an inspection process for cake-decoration patterns, while Figure 5.22 demonstrates its use as a prelude to counting circular objects (peas and soft-drink cans) Figure 5.23 uses morphology to detect ring-shaped features (components connecting pads) on printed circuit boards. Figure 5.24 demonstrates how an SE can be derived by processing the input image. Also, notice that erosion was not applied directly to the original binary image but to a dilated version of it. (Figure 5.24[CL]) This makes the process of detecting the elephant motif less susceptible to noise and hence more reliable. In Figure 5.25 morphology is applied to identifying playing-card suit symbols. Figure 5.26[TL]-[CR] shows how the teeth on a gear can be isolated so that they can be counted, or inspected. The same method was used to detect the "valleys" on an external screw thread. (Figure 5.26[BL] - [BR])

Position & Orientation

There is no unique measurement of position that suits every situation. (Figure 5.27[T]) For example, when a person picks up a hammer, the position of the handle is important but, when it is being used, what happens as the other end is the thing that really matters. One end of a venomous snake (Figure 5.27[C]) is clearly of far greater significance than the other, even though it may be very difficult to decide which it is visually. The garden shears, Figure 5.27[B], have four important points of interest: two handles and two blade tips. Similarly, there is no uniquely agreed "significant point of interest" on a blob that defines “its position". The point is that there are many ways for a vision system to analyse the silhouette of a real-world object in order to guide a robot; every application requires individual consideration.

One very popular way to measure blob position is to calculate its *geometric centroid*, or simply *centroid*. This is the centre of gravity of a thin uniform sheet of material that has the same outline as the given blob. The position of the centroid can be calculated from a binary array, using a simple formula.

Although “the orientation” of a blob has no unique definition, the *Principal Axis* is often useful for this purpose. It is also easy (and fast) to compute. The principal axis is properly called the *axis of minimum second moment*. Figure 5.28 shows two objects (gear and cam) for which the principal axis is unsuitable. Let us return again to the con-rod. A safe lifting point for a *small* vacuum or magnetic gripper is at the centroid. (Figure 5.29) A large suction gripper would not manage a safe lift because there would be an air leak outside the object boundary. A 2-finger gripper can be positioned using the centroid and principal axis but neither of these is sufficient on its own; further calculation is needed. Figure 5.29[BR] shows two well-defined reference points for position. The position of the lower point was found using erosion using a large circular SE. The upper point is the centroid of the "hole", properly called the *lake*. The line joining them can be used to determine orientation.

Edge-Based Techniques

So far in this chapter, we have concentrated on processing binary images represented as a 2-dimensional array of 0s and 1s. As explained in Chapter 2, tracing the boundary of a blob can lead to a major reduction in the amount of data needed to define it exactly. In addition to saving storage space, edge-based methods make certain calculations easier, including:

- Separating objects in a *multi-blob binary image*. (Figure 5.30)
- *Edge smoothing*
- Detecting sharp changes of edge direction. (*Corners*. Figure 5.31[BR])
- Feature detection: finding long *straight-line edge segments*.
- Calculating the position of the centroid
- Calculating the perimeter. (Figure 5.31) Measuring perimeter is always fraught with difficulty!, Figure 5.32)
- Calculating *extreme points*: top, bottom, left-most & right-most points
- Fitting the *minimum-area bounding rectangle*. (Figure 5.33[TL] & 34)
- Fitting the minimum-area bounding octagon. Sides at 0˚,45˚,90˚,135˚, ..
- Fitting the minimum-area bounding hexadecagon. Sides at 0˚,22.5˚,45˚,67.5˚,90˚, ..
- Fitting the *minimum-area bounding circle*. (Figure 5.33[TR])
- Computing the *convex hull* of a blob. (This is the area enclosed by a rubber band stretched around the blob. It encloses the object itself and all of its bays. (Figure 5.33[CL] & Figure 5.34)
- Fitting an ellipse (Figure 5.33[BR] and Figure 5.35)
- Reducing the blob to a *match-stick* **(** Figure 5.36) *Thinning* and *skeletonisation* are different ways to do this. (Figure 5.35)
- Calculating the *grassfire transform*. (Figure 5.36[CR])
- Fitting internal circles. (Figure 5.37)
- Comparing two shapes (Figure 5.38)
- Locating the edge points at the ends of the *largest diameter* (Figure 5.39[BL])
- Finding the maximum & minimum distance from any given fixed point (e.g. blob centroid, Figure 5.39[BR])
- Finding the maximum & minimum distance from any given straight line.
- Finding the position and radius of a circular arc fitted to part of the edge. (*Local radius of curvature*)
- Fitting a curve (e.g. polynomial) to part of the edge.

Blob Measurements

Figure 5.39 & Figure 5.40 suggest ways by which a blob can be analysed to yield a variety of "anchor points" from which a range of measurements can be derived. Using simple geometry, distances, angles and areas can be computed from the coordinates of thesenanchor points. For an inspection system, it probably does not really matter whether or not these relate directly to the primary features that the application engineer specifies as important. Remember the vision engineer's maxim

If a product feature is not as it should be, it is wrong!

Suppose, for example, that one of the round "leaves" of the club symbol (Figure 5.39[BR]) were bent, one or more of the red and yellow crosses would be misplaced. In this one example, it is not difficult to formulate multiple (over 50) different measurements: distance, angle and area. (Figure 5.41) However, this approach is not systematic. The following idea tries to make it so.

Concavity Trees

A *concavity tree (CT)* provides a way to describe blob shapes using a recursive construction process. The *convex hull (CH)* is central to this approach. (Figures 5.40 - 5.42) The CH of a blob is unchanged by moving or rotating the blob it encloses. The difference between a blob and its CH is called its *convex deficiency (CD)*. It consists of a number of separate *bays* and *lakes.* Together, these are referred to as *concavities*. Since Figure 5.42 contains no lakes, we will refer only to bays from now on but lakes are treated in just the same way

Let us consider each bay separately and derive its CH. Think of the various shapes created in this way as being cardboard cut-outs, A crude approximation to the original shape can be produced by "cutting out" the CHs of each bay from the CH of the original blob. To obtain a better approximation, we "stick in" the CHs of the CD of each bay. This process continues recursively indefinitely. Alternating levels in the CT are “cut-outs”, while the levels in between are “stick ins”

Figure 5.42 shows a truncated CT in which each node represents a convex shape that has two associated measurements: its area and the area of its CH. As we saw earlier, there is no shortage of ideas for possible measurements. The labels attached to the nodes in a concavity tree can be multi-element vectors, whatever is convenient.

Concavity trees have some interesting and desirable features:

- We can normalise the CT so that is independent of the orientation of the original blob.
- The structure of the CT is independent of the size of the original blob
- The CT is independent of the position of the original blob.
- The CT combines local and global shape measurements in a systematic way.
- A wide variety of shape measurements can be incorporated in a CT, to label its nodes.
- It is possible to use a CT to determine whether a blob is *"heads up"* or *"tails up"*. (Think of the CT as representing parts of a glove.) This defines the *chirality*. (Figure 5.41)
- A fully developed CT allows the original blob to be reconstructed exactly, using only convex polygons.
- A CT can be truncated, if necessary, to whatever level of precision is needed for the application

Inspecting objects represented by matching concavity trees involves some interesting Artificial Intelligence techniques but these are beyond the scope of our present discussion.

Three Final Remarks

- Numeric shape descriptors do not always provide a complete or satisfactory representation of a binary image. Figure 5.43 shows four situations where symbolic output signals are required.
- A good vision engineer exploits all available *Application Knowledge.* Figures 5.44 & Figure 5.45
- We still need subtle, crafty, cunning, artful, devious human intelligence!