10
Image segmentation

10.1 Image segmentation

Segmentation is the name given to the generic process by which an image is subdivided into its constituent regions or objects. In general, completely autonomous segmentation is one of the most difficult tasks in the design of computer vision systems and remains an active field of image processing and machine vision research. Segmentation occupies a very important role in image processing because it is so often the *vital first step* which must be successfully taken before subsequent tasks such as feature extraction, classification, description, etc. can be sensibly attempted. After all, if you cannot identify the objects in the first place, how can you classify or describe them?

The basic goal of segmentation, then, is to partition the image into mutually exclusive regions to which we can subsequently attach meaningful labels. The segmented objects are often termed the foreground and the rest of the image is the background. Note that, for any given image, we cannot generally speak of a single, 'correct' segmentation. Rather, the correct segmentation of the image depends strongly on the types of object or region we are interested in identifying. *What relationship must a given pixel have with respect to its neighbours and other pixels in the image in order that it be assigned to one region or another?* This really is the central question in image segmentation and is usually approached through one of two basic routes:

- *Edge/boundary methods* This approach is based on the detection of edges as a means to identifying the boundary between regions. As such, it looks for sharp differences between groups of pixels.

- *Region-based methods* This approach assigns pixels to a given region based on their degree of mutual similarity.

10.2 Use of image properties and features in segmentation

In the most basic of segmentation techniques (intensity thresholding), the segmentation is used only on the absolute intensity of the individual pixels. However, more sophisticated properties and features of the image are usually required for successful segmentation. Before

Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab Chris Solomon and Toby Breckon
 ISBN: 978-0-470-84472-4

we begin our discussion of explicit techniques, it provides a useful (if somewhat simplified) perspective to recognize that there are three basic properties/qualities in images which we can exploit in our attempts to segment images.

(1) *Colour* is, in certain cases, the simplest and most obvious way of discriminating between objects and background. Objects which are characterized by certain colour properties (i.e. are confined to a certain region of a colour space) may be separated from the background. For example, segmenting an orange from a background comprising a blue tablecloth is a trivial task.

(2) *Texture* is a somewhat loose concept in image processing. It does not have a single definition but, nonetheless, accords reasonably well with our everyday notions of a 'rough' or 'smooth' object. Thus, texture refers to the 'typical' spatial variation in intensity or colour values in the image over a certain spatial scale. A number of texture metrics are based on calculation of the variance or other statistical moments of the intensity over a certain neighbourhood/spatial scale in the image. We use it in a very general sense here.

(3) *Motion* of an object in a sequence of image frames can be a powerful cue. When it takes place against a stationary background, simple frame-by-frame subtraction techniques are often sufficient to yield an accurate outline of the moving object.

In summary, most segmentation procedures will use and combine information on one of more of the properties colour, texture and motion.

Some simple conceptual examples and possible approaches to the segmentation problem are summarized in Table 10.1.

What has led us to suggest the possible approaches in Table 10.1? Briefly: an aeroplane in the sky is (we hope) in motion. Its colour is likely to be distinct from the sky. Texture is unlikely to be a good approach because the aeroplane (being a man-made object) tends to have a fairly smooth texture – so does the sky, at least in most weather conditions. Some form of shape analysis or measurement could certainly be used, but in this particular case would tend to be superfluous. Of course, these comments are highly provisional (and may

Table 10.1 Simple conceptual examples and possible approach to segmentation

Object(s)	Image	Purpose of segmentation	Preferred approach
Aeroplane	Aeroplane in the sky	Tracking	Motion, colour
Human faces	Crowded shopping mall	Face recognition surveillance system	Colour, shape
Man-made structures	Aerial satellite photograph	Intelligence acquisition from aircraft	Texture, colour
Cultivated regions	Uncultivated regions	LandSAT survey	Texture, shape
Granny Smiths apples, pears	Various fruits on a conveyor belt	Automated fruit sorting	Colour, shape

actually be inaccurate in certain specific instances), but they are intended to illustrate the importance of carefully considering the approach which is most likely to yield success.

We will begin our discussion of segmentation techniques with the simplest approach, namely intensity thresholding.

10.3 Intensity thresholding

The basic idea of using intensity thresholding in segmentation is very simple. We choose some threshold value such that pixels possessing values greater than the threshold are assigned to one region whilst those that fall below the threshold are assigned to another (adjoint) region. Thresholding creates a binary image $b(x, y)$ from an intensity image $I(x, y)$ according to the simple criterion

$$b(x,y) = \begin{cases} 1 & \text{if } I(x,y) > T \\ 0 & \text{otherwise} \end{cases} \tag{10.1}$$

where T is the threshold.

In the very simplest of cases, this approach is quite satisfactory. Figure 10.1 (top right) (produced using the Matlab® code in Example 10.1) shows the result of intensity thresholding on an image of several coins lying on a dark background; all the coins are successfully identified. In this case, the appropriate threshold was chosen manually by trial and error. In a limited number of cases, this is an acceptable thing to do; for example, certain inspection tasks may allow a human operator to set an appropriate threshold before automatically processing a sequence of similar images. However, many image processing tasks require full automation, and there is often a need for some criterion for selecting a threshold automatically.

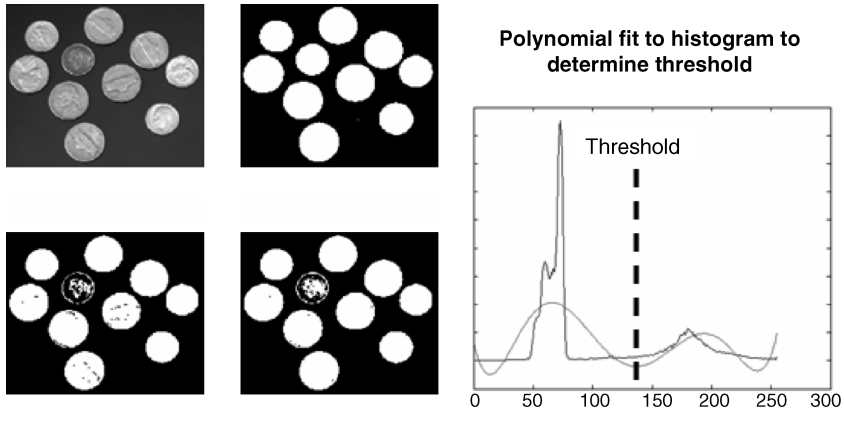

Figure 10.1 Top left: original image. Top right: result after manual selection of threshold. Bottom left: result of automatic threshold selection by polynomial fitting to image histogram. Bottom right: result of automatic threshold selection using Otsu's method. The image on the right shows the histogram and the result of fitting a sixth-order polynomial

Example 10.1

Matlab code	What is happening?
I = imread('coins.png');	%Read in original
subplot(2,2,1), imshow(I);	%Display original
subplot(2,2,2),im2bw(I,0.35);	%Result of manual threshold
[counts,X]=imhist(I);	%Calculate image hIstogram
P = polyfit(X,counts,6); Y=polyval(P,X);	%Fit to histogram and evaluate
[V,ind]=sort(abs(diff(Y))); thresh=ind(3)./255;	%Find minimum of polynomial
subplot(2,2,3), im2bw(I,thresh);	%Result of Polynomial theshold
level = graythresh(I);	%Find threshold
subplot(2,2,4), im2bw(I,level);	%Result of Otsu's method
figure; plot(X,counts); hold on, plot(X,Y,'r-');	%Histogram and polynomial fit

Comments
Matlab functions: ***im2bw***, ***imhist***, ***polyfit***, ***polyval***, ***sort***, ***graythresh***.

Automated threshold selection is essentially based on a conceptual or actual consideration of the image histogram. In those situations where thresholding can successfully segment objects from the background, the 1-D histogram of the image will typically exhibit two modes or peaks: one corresponding to the pixels of the objects and one to the pixels of the background. This is, in fact, the case with our chosen example; see Figure 10.1 (image on right). The threshold needs to be chosen so that these two modes are clearly separated from each other.

One simple approach is to calculate the image histogram and fit a polynomial function to it. Provided the order of the polynomial function is chosen judiciously and the polynomial adequately fits the basic shape of the histogram, a suitable threshold can be identified at the minimum turning point of the curve.[1]

A more principled approach to automatic threshold selection is given by Otsu's method. Otsu's method is based on a relatively straightforward analysis which finds that threshold which *minimizes the within-class variance* of the thresholded black and white pixels. In other words, this approach selects the threshold which results in the tightest clustering of the two groups represented by the foreground and background pixels. Figure 10.1 shows the results obtained using both a manually selected threshold and that calculated automatically by Otsu's method.

10.3.1 Problems with global thresholding

There are several serious limitations to simple global thresholding:

- there is no guarantee that the thresholded pixels will be contiguous (thresholding does not consider the spatial relationships between pixels);

[1] Two peaks require a minimum of fourth order to model (two maxima, one minimum), but it is usually better to go to fifth or sixth order to allow a bit more flexibility. If there is more than one minimum turning point on the curve, then it is usually straightforward to identify the appropriate one.

- it is sensitive to accidental and uncontrolled variations in the illumination field;
- it is only really applicable to those simple cases in which the entire image is divisible into a foreground of objects of similar intensity and a background of distinct intensity to the objects.

10.4 Region growing and region splitting

Region growing is an approach to segmentation in which pixels are grouped into larger regions based on their similarity according to predefined *similarity criteria*. It should be apparent that specifying similarity criteria *alone* is not an effective basis for segmentation and it is necessary to consider the adjacency spatial relationships between pixels. In region growing, we typically start from a number of seed pixels randomly distributed over the image and append pixels in the neighbourhood to the same region if they satisfy *similarity criteria* relating to their intensity, colour or related statistical properties of their own neighbourhood. Simple examples of similarity criteria might be:

(1) the absolute intensity difference between a candidate pixel and the seed pixel must lie within a specified range;

(2) the absolute intensity difference between a candidate pixel and the running average intensity of the growing region must lie within a specified range;

(3) the difference between the standard deviation in intensity over a specified local neighbourhood of the candidate pixel and that over a local neighbourhood of the candidate pixel must (or must not) exceed a certain threshold – this is a basic roughness/smoothness criterion.

Many other criteria can be specified according to the nature of the problem.

Region splitting essentially employs a similar philosophy, but is the reverse approach to region growing. In this case we begin the segmentation procedure by treating the whole image as a single region which is then successively broken down into smaller and smaller regions until any further subdivision would result in the differences between adjacent regions falling below some chosen threshold. One popular and straightforward approach to this is the *split-and-merge* algorithm.

10.5 Split-and-merge algorithm

The algorithm divides into two successive stages. The aim of the region splitting is to break the image into a set of disjoint regions each of which is regular within itself. The four basic steps are:

- consider the image as a whole to be the initial area of interest;
- look at the area of interest and decide if all pixels contained in the region satisfy some similarity criterion;

- if TRUE, then the area of interest (also called a block) corresponds to a region in the image and is labelled;

- if FALSE, then split the area of interest (usually into four equal sub-areas) and consider each of the sub-areas as the area of interest in turn.

This process continues until no further splitting occurs. In the worst-case scenario, this might happen when some of the areas are just one pixel in size. The splitting procedure is an example of what are sometimes referred to as divide-and-conquer or top-down methods. The process by which each block is split into four equal sub-blocks is known as *quadtree* decomposition.

However, if only splitting is carried out, then the final segmentation would probably contain many neighbouring regions that have identical or similar properties. Thus, a merging process is used after each split which compares adjacent regions and merges them if necessary. When no further splitting or merging occurs, the segmentation is complete. Figure 10.2 illustrates the basic process of splitting via quadtree decomposition and merging.

Figure 10.3 (produced using the Matlab code in Example 10.2) shows the result of a split-and-merge algorithm employing a similarity criterion based on local range (split the block if the difference between the maximum and minimum values exceeds a certain fraction of the image range).

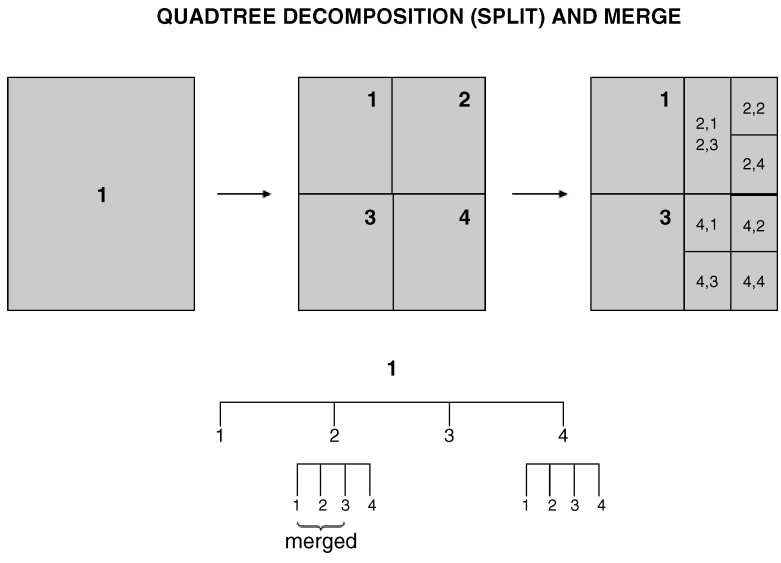

Figure 10.2 The basic split-and-merge procedure. The initial image is split into four regions. In this example, regions 1 and 3 satisfy the similarity criterion and are split no further. Regions 2 and 4 do not satisfy the similarity criterion and are then each split into four sub-regions. Regions (2,1) and (2,3) are then deemed sufficiently similar to merge them into a single region. The similarity criterion is applied to the designated sub-regions until no further splitting or merging occurs

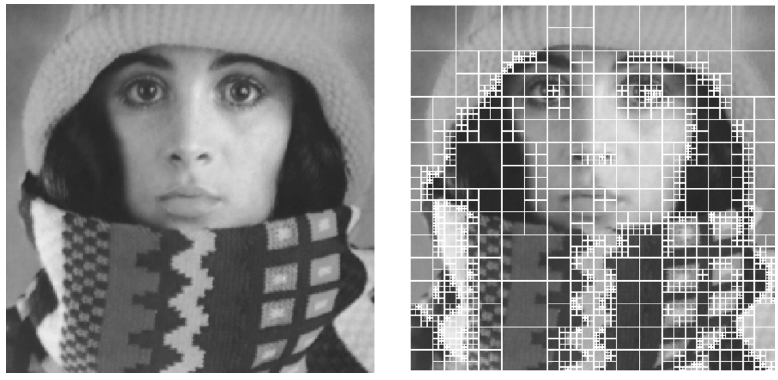

Figure 10.3 The image on the right delineates the regions obtained by carrying out a quadtree decomposition on the image using a similarity criterion based on the range of the intensity values within the blocks

Example 10.2

Matlab code	**What is happening?**
I=imread('trui.png');	%Read in image
S = qtdecomp(I,.17);	%Do quadtree decomposition
blocks = repmat(uint8(0),size(S));	%Create empty blocks
for dim = [512 256 128 64 32 16 8 4 2 1];	%Loop through successively smaller blocks
numblocks = length(find(S==dim));	
if (numblocks > 0)	
values = repmat(uint8(1),[dim dim numblocks]);	
values(2:dim,2:dim,:) = 0;	
blocks = qtsetblk(blocks,S,dim,values);	
end	
end	
blocks(end,1:end) =1;	
blocks(1:end,end) = 1;	
subplot(1,2,1), imshow(I);	
k=find(blocks==1);	%Find border pixels of regions
A=I; A(k)=255;	%Superimpose on original image
subplot(1,2,2), imshow(A);	%Display

Comments

Matlab functions: ***qtdecomp***, ***repmat***, ***length***, ***qsetblk***, ***find***.

The function ***qtdecomp*** performs a quadtree decomposition on the input intensity image and returns a sparse array S. S contains the quadtree structure such that if $S(k, m) = P \neq 0$ then (k, m) is the upper-left corner of a block in the decomposition, and the size of the block is given by P pixels.

10.6 The challenge of edge detection

Edge detection is one of the most important and widely studied aspects of image processing. If we can find the boundary of an object by locating all its edges, then we have effectively segmented it. Superficially, edge detection seems a relatively straightforward affair. After all, edges are simply regions of intensity transition between one object and another. However, despite its conceptual simplicity, edge detection remains an active field of research. Most edge detectors are fundamentally based on the use of gradient differential filters. The most important examples of these (the Prewitt and Sobel kernels) have already been discussed in Chapter 4. However, these filters do not find edges *per se,* but rather only give some indication of where they are most likely to occur. Trying to actually find an edge, several factors may complicate the situation. The first relates to edge strength or, if you prefer, the *context* – how large does the gradient have to be for the point to be designated part of an edge? The second is the effect of noise – differential filters are very sensitive to noise and can return a large response at noisy points which do not actually belong to the edge. Third, where exactly does the edge occur? Most real edges are not discontinuous; they are smooth, in the sense that the gradient gradually increases and then decreases over a finite region. These are the issues and we will attempt to show how these are addressed in the techniques discussed in the following sections.

10.7 The laplacian of Gaussian and difference of Gaussians filters

As we saw earlier in this book, second-order derivatives can be used as basic discontinuity detectors. In Figure 10.4, we depict a (1-D) edge in which the transition from low to high intensity takes place over a finite distance. In the ideal case (depicted on the left), the second-order derivative $\mathrm{d}^2 f/\mathrm{d}x^2$ depicts the precise onset of the edges (the point at which the gradient suddenly increases giving a positive response and the point at which it suddenly decreases giving a negative response). Thus, the location of edges is, *in principle*, predicted well by using a second-order derivative. However, second-derivative filters do not give the strength of the edge and they also tend to exaggerate the effects of noise (twice as much as first-order derivatives). One approach to getting round the problem of noise amplification is first to smooth the image with a Gaussian filter (reducing the effects of the noise) and then to apply the Laplacian operator. Since the Gaussian and Laplacian filtering operations are linear, and thus interchangeable, this sequence of operations can actually be achieved more efficiently by taking the *Laplacian of a Gaussian* (LoG) and then filtering the image with a suitable discretized kernel. It is easy to show that the Laplacian of a radially symmetric Gaussian is

$$\nabla^2 \left\{ \mathrm{e}^{-r^2/2\sigma^2} \right\} = \frac{r^2 - \sigma^2}{\sigma^4} \mathrm{e}^{-r^2/2\sigma^2} \tag{10.2}$$

and a discrete approximation to this function over a local neighbourhood can then easily be achieved with a filter kernel. The use of the LoG filter as a means of identifying edge locations was first proposed by Marr and Hildreth, who introduced the principle of the

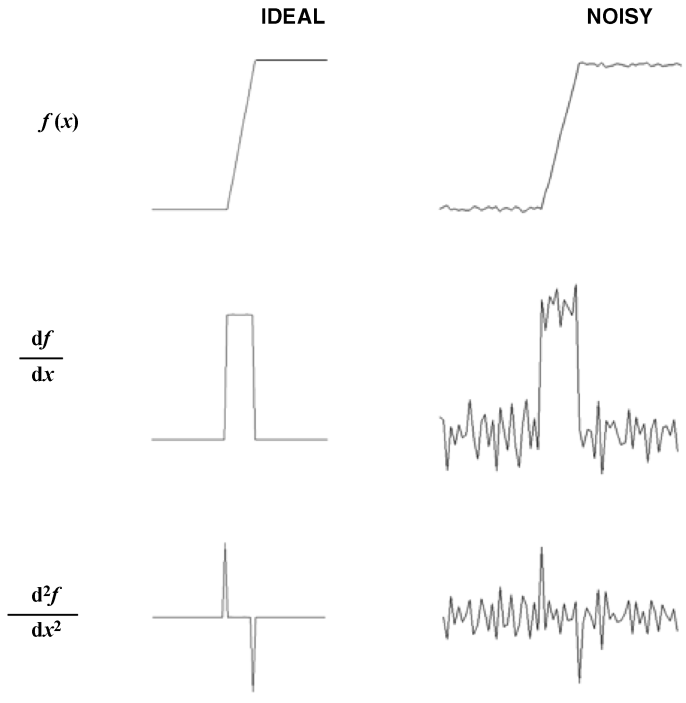

Figure 10.4 Demonstrating the sensitivity of differential operators to noise. In the ideal case (depicted on the left), the second-order derivative identifies the precise onset of the edge. However, even modest fluctuations in the signal result in the amplification of the noise in the first- and second-order derivatives (depicted on the right)

zero-crossing method. This relies on the fact that the Laplacian returns a high positive and high negative response at the transition point and the edge location is taken to be the point at which the response goes through zero. Thus, the points at which the LoG-filtered image go to zero indicate the location of the candidate edges.

A closely related filter to the LoG is the *difference of Gaussians* (DoG). The DoG filter is computed as the difference of two Gaussian functions having different standard deviations. It is possible to show formally that, when the two Gaussians have relative standard deviations of 1 and 1.6, the DoG operator closely approximates the LoG operator. Figure 10.5 shows the basic shape of the 2-D LoG filter (colloquially known as the Mexican hat function) together with the results of log filtering of our image and identifying edges at the points of zero crossing.

10.8 The Canny edge detector

Although research into reliable edge-detection algorithms continues, the Canny method is generally acknowledged as the best 'all-round' edge detection method developed to date. Canny aimed to develop an edge detector that satisfied three key criteria:

- A *low error rate*. In other words, it is important that edges occuring in images should not be missed and that there should be no response where edges do not exist.

Figure 10.5 The basic shape of the LoG filter is shown on the left. The flat surrounding region corresponds to value 0. Filtering of the centre image with a discrete 5×5 kernel approximation to the LoG function followed by thresholding to identify the zero crossings results in the image on the right

- The detected edge points should be *well localized*. In other words, the distance between the edge pixels as found by the detector and the actual edge should be a minimum.
- There should be *only one response to a single edge*.

It is outside the scope of this discussion to present the detailed mathematical analysis and reasoning undertaken by Canny, but the basic procedure can be summarized in the following steps:

(1) *The image is first smoothed using a Gaussian kernel*: Gradient operators are sensitive to noise and this preliminary step is taken to reduce the image noise. The greater the width of the kernel, the more smoothing (i.e. noise reduction) is achieved. However, larger kernels result in a greater error in the edge location.

(2) *Find the edge strength*: This is achieved by taking the gradient of the image with the *Sobel operators* in the horizontal and vertical directions and then adding the magnitude of these components as a measure of the 'edge strength'. Thus $E(x, y) = |G_x(x, y)| + |G_y(x, y)|$.

(3) *Calculate the edge direction*: This is easily calculated as

$$\theta = \tan^{-1} \frac{G_y(x, y)}{G_x(x, y)}$$

(4) *Digitize the edge direction*: Once the edge direction is known, we approximate it to an edge direction that can be traced in a digital image. Considering an arbitrary pixel, the direction of an edge through this pixel can take one of only four possible values - 0° (neighbours to east and west), 90° (neighbours to north and south), 45° (neighbours to north-east and south-west) and 135° (neighbours to north-west and south-east). Accordingly, we approximate the calculated θ by whichever of these four angles is closest in value to it.[2]

[2] It should be noted that in some formulations of this step in the process, edge orientation is calculated to sub-pixel accuracy using 8 connectivity, SEE Canny, J., A Computational Approach To Edge Detection, IEEE Trans. Pat. Anal. and Mach. Intel., 8(6):679–698, 1986.

(5) *Nonmaximum suppression*: After the edge directions are known, nonmaximum suppression is applied. This works by tracing along the edge in the edge direction and suppressing any pixel value (i.e. set it equal to zero) that is not considered to be an edge. This will give a thin line in the output image.

(6) *Hysteresis*: After the first five steps have been completed, the final step is to track along the remaining pixels that have not been suppressed and threshold the image to identify the edge pixels. Critical to the Canny method, however, is the use of two distinct thresholds – a higher value T_2 and a lower value T_1. The fate of each pixel is then determined according to the following criteria:

- if $|E(x,y)| < T_1$, then the pixel is rejected and is not an edge pixel;
- if $|E(x,y)| > T_2$, then the pixel is accepted and is an edge pixel;
- if $T_1 < |E(x,y)| < T_2$, the pixel is rejected except where a path consisting of edge pixels connects it to an unconditional edge pixel with $|E(x,y)| > T_2$.

Two comments are pertinent to the practical application of the Canny edge detector. First, we are, in general, interested in identifying edges on a particular scale. We can introduce the idea of feature scale by filtering with Gaussian kernels of various widths. Larger kernels introduce greater amounts of image smoothing and the gradient maxima are accordingly reduced. It is normal, therefore, to specify the size of the smoothing kernel in the Canny edge detection algorithm to reflect this aim. Second, we see that the Canny detector identifies weak edges ($T_1 < |E(x,y)| < T_2$) only *if* they are connected to strong edges ($|E(x,y)| > T_2$).

Figure 10.6 demonstrates the use of the LoG and Canny edge detectors on an image subjected to different degrees of Gaussian smoothing. The Matlab code corresponding to Figure 10.6 is given in Example 10.3.

Example 10.3

Matlab code	**What is happening?**
A=imread('trui.png');	%Read in image
subplot(3,3,1), imshow(A,[]);	%Display original
h1=fspecial('gaussian',[15 15],6);	
h2=fspecial('gaussian',[30 30],12);	
subplot(3,3,4), imshow(imfilter(A,h1),[]);	%Display filtered version sigma=6
subplot(3,3,7), imshow(imfilter(A,h2),[]);	%Display filtered version sigma=12
[bw,thresh]=edge(A,'log');	%Edge detection on original – LoG filter
subplot(3,3,2), imshow(bw,[]);	
[bw,thresh]=edge(A,'canny');	%Canny edge detection on original
subplot(3,3,3), imshow(bw,[]);	%Display
[bw,thresh]=edge(imfilter(A,h1),'log');	%LoG edge detection on sigma=6
subplot(3,3,5), imshow(bw,[]);	

```
[bw,thresh]=edge(imfilter(A,h1),'canny');      %Canny edge detection on sigma=6
subplot(3,3,6), imshow(bw,[]);
[bw,thresh]=edge(imfilter(A,h2),'log');        %LoG edge detection on sigma=12
subplot(3,3,8), imshow(bw,[]);
[bw,thresh]=edge(imfilter(A,h2),'canny');      %Canny edge detection on sigma=12
subplot(3,3,9), imshow(bw,[]);
```

Comments

Matlab functions: ***edge***.

The ***edge*** function can be called according to a variety of syntaxes. In the examples above, appropriate thresholds are estimated *automatically* based on calculations on the input image. See the Matlab documentation doc edge for full details of use.

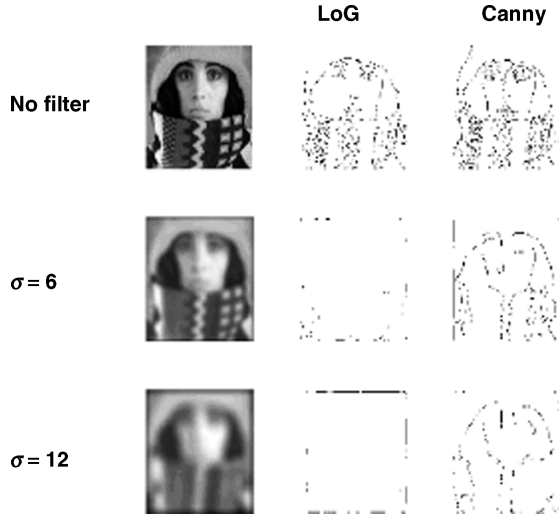

Figure 10.6 Edge detection using the zero-crossing method (LoG filter) and the Canny method for an image at three different degrees of blur. In these examples, the LoG threshold and both the lower and higher threshold for the Canny detector were calculated automatically on the basis of an estimated signal/noise ratio. The superior performance of the Canny filter is evident

10.9 Interest operators

In simple terms, a point is deemed *interesting* if it is distinct from all closely adjacent regions, i.e. all those points in its surrounding neighbourhood. Interest operators are filters which attempt to identify such points of interest automatically. Interest operators do not feature as a 'mainstream' tool for segmentation but are, nonetheless, useful in certain special kinds of problem. They also find application in problems concerned with image matching or registration by geometric transformation, which presupposes the existence of so-called tie points, i.e. known points of correspondence between two images. These are typically points such as spots or corners which are easily distinguishable from their surroundings. The identification of such points of interest is also desirable in other applications, such as

3-D stereographic imaging. The theoretical basis for identification of interesting points is given below.

Consider the change in intensity in an image $I(x, y)$ by displacing from the point (x, y) by some small distance$(\nabla x, \nabla y)$. A first-order Taylor series approximation gives

$$\nabla I = I(x + \nabla x, y + \nabla y) - I(x, y) = \nabla x f_x + \nabla y f_y \tag{10.3}$$

where the image gradient at point (x, y) is $[f_x, f_y]$.

We note from Equation (10.3) that the change in intensity is *greatest* when the displacement is in the same direction as the gradient vector but is least when the displacement is perpendicular to it. Haralick's criterion deems a point to be interesting when the quantity $(\nabla I)^2$ (squared because negative changes are just as interesting as positive and we do not want them to cancel) summed over a local neighbourhood is large for *any displacement direction*. This condition then satisfies, in the general sense, our constraint that the point be distinct from its surroundings. Using Equation (10.3), we may write a matrix expression for ∇I^2

$$(\nabla I)^2 = (\nabla x f_x + \nabla y f_y)^2 = (\nabla x \quad \nabla y) \begin{pmatrix} \sum_{\Omega} f_x^2 & \sum_{\Omega} f_x f_y \\ \sum_{\Omega} f_x f_y & \sum_{\Omega} f_y^2 \end{pmatrix} \begin{pmatrix} \nabla x \\ \nabla y \end{pmatrix} = \mathbf{v}^{\mathrm{T}} \mathbf{F} \mathbf{v} \tag{10.4}$$

where we have denoted the displacement vector

$$\mathbf{v} = \begin{pmatrix} \nabla x \\ \nabla y \end{pmatrix}, \quad \mathbf{F} = \begin{pmatrix} \sum_{\Omega} f_x^2 & \sum_{\Omega} f_x f_y \\ \sum_{\Omega} f_x f_y & \sum_{\Omega} f_y^2 \end{pmatrix}$$

and the elements inside the matrix $\mathbf{F}$ are summed over a local pixel neighbourhood Ω. Summation over a local neighbourhood is desirable in practice to reduce the effects of noise and measure the characteristics of the image over a small 'patch' with the given pixel at its centre. For convenience (and without imposing any restriction on our final result), we will assume that the displacement vector we consider in Equation (10.4) has unit length, i.e. $\mathbf{v}^{\mathrm{T}}\mathbf{v} = 1$.

Equation (10.4) is a quadratic form in the symmetric matrix $\mathbf{F}$ and it is straightforward to show that the minimum and maximum values of a quadratic form are governed by the eigenvalues of the matrix with the *minimum* value being given by $\|\mathbf{v}\|\lambda_{\min} = \lambda_{\min}$, with $\lambda_{\min}$ the smallest eigenvalue of the matrix $\mathbf{F}$. Equation (10.4) is readily interpetable (in its 2-D form) as the equation of an ellipse, the larger eigenvalue corresponding to the length of the major axis of the ellipse and the smaller eigenvalue to the minor axis. The Haralick method designates a point as interesting if the following two criteria are satisfied:

(1) that $\frac{\lambda_1 + \lambda_2}{2}$ is 'large'. This is roughly interpretable as the 'average radius' of the ellipse or average value of the quadratic over all directions.

(2) that $\frac{4\det(\mathbf{F})}{\mathrm{Tr}\{\mathbf{F}\}} = 1-\left(\frac{\lambda_1-\lambda_2}{\lambda_1+\lambda_2}\right)^2$ is 'not too small'. This second criterion translates to the requirement that the maximum and minimum values of the quadratic not be too different from each other or, in geometric terms, that the ellipse not be too elongated. It is apparent that a dominant eigenvalue such that $\lambda_1 \gg \lambda_2$ corresponds to a large value in the direction of the first principal axis but a small value in the direction of the second axis.

The question of what is 'large' and 'not too small' in the discussion above is, of course, relative and must be placed in the direct context of the particular image being analysed. 'Large' usually relates to being a member of a selected top percentile, whereas 'not too small' equates to an image-dependent cut-off value which can be systematically adjusted to reduce or increase the number of detected points.

Harris developed a slightly different criterion. He independently defined an interest function given by:

$$R = \lambda_1\lambda_2 - k(\lambda_1+\lambda_2)^2 \tag{10.5}$$

which from the basic properties of the trace and determinant of a square matrix can be written as:

$$R = \det[\mathbf{F}] - k(\mathrm{Tr}\{\mathbf{F}\})^2 = \mathbf{F}_{11}\mathbf{F}_{22} - \mathbf{F}_{12}\mathbf{F}_{21} - k(\mathbf{F}_{11}+\mathbf{F}_{22})^2 \tag{10.6}$$

or explicitly as:

$$R(x,y) = (1-2k)\sum_{\Omega} f_x^2 \sum_{\Omega} f_y^2 - k\left[\left(\sum_{\Omega} f_x^2\right)^2 + \left(\sum_{\Omega} f_y^2\right)^2\right] - (f_x f_y)^2 \tag{10.7}$$

and the constant k *is* set to 0.04.

One significant advantage of the Harris function is that it may be evaluated without explicit calculation of the eigenvalues (which is computationally intensive). Note from Equation (10.5), the following limiting forms of the Harris function:

- as $\lambda_2 \to \lambda_1 = \lambda_{\max}$ so $R \to (1-2k)\lambda_{\max}^2$ (its maximum possible value)
- as $\lambda_2 \to \lambda_1 \to 0$ so $R \to 0$ (no response – a smooth region)
- if $\lambda_1 \gg \lambda_2$ then $R \to -k\lambda_{\max}^2$.

Thus, a large positive value of R indicates an 'interesting' point, whereas a negative response suggests a dominant eigenvalue (a highly eccentric ellipse) and is 'not interesting'. The basic recipe for detection of interest points is as follows.

Detection of Interest Points

(1) Smooth the image by convolving the image with a Gaussian filter.

(2) For each pixel, compute the image gradient.

(3) For each pixel and its given neighbourhood Ω, compute the 2×2 matrix $\mathbf{F}$.

(4) For each pixel, evaluate the response function $R(x, y)$.

(5) Choose the interest points as local maxima of the function $R(x, y)$ (e.g. by a nonmaximum suppression algorithm).

For the final step, the non-maximum suppression of local minima, there are several possible approaches. One simple but effective approach (employed in the Example 10.4) is as follows:

- Let R be the interest response function.
- Let S be a filtered version of the interest function in which every pixel within a specified (nonsparse) neighbourhood is replaced by *the maximum value in that neighbourhood*.
- It follows that those pixels whose values are left *unchanged* in S correspond to the local maxima.

We end our discussion with two comments of practical importance. First, image smoothing reduces the size of the image gradients and, consequently, the number of interest points detected (and vice versa). Second, if the size of the neighbourhood Ω (i.e. the *integrative scale*) is too small, then there is a greater probability of rank deficiency in matrix $\mathbf{F}$.[3] Larger integrative scale increases the probability of a full-rank matrix. However, it also smoothes the response function $R(x, y)$; thus, too large an integrative scale can suppress the number of local maxima of $R(x, y)$ and, hence, the number of interest points detected.

The example given in Figure 10.7 (produced using the Matlab code in Example 10.4) illustrates the detection of interest points in two images on two different integrative scales using the Harris function. Note in both examples the tendency of smooth regions to have values close to zero and, in particular, of straight edges to return negative values. Straight-edge pixels, by definition, have a significant gradient in one direction (perpendicular to the dark–light boundary) but a zero (or at least very small) gradient response parallel to the edge. This situation corresponds to a dominant eigenvalue in the matrix $\mathbf{F}$.

[3] Rank deficient matrices have at least one zero eigenvalue.

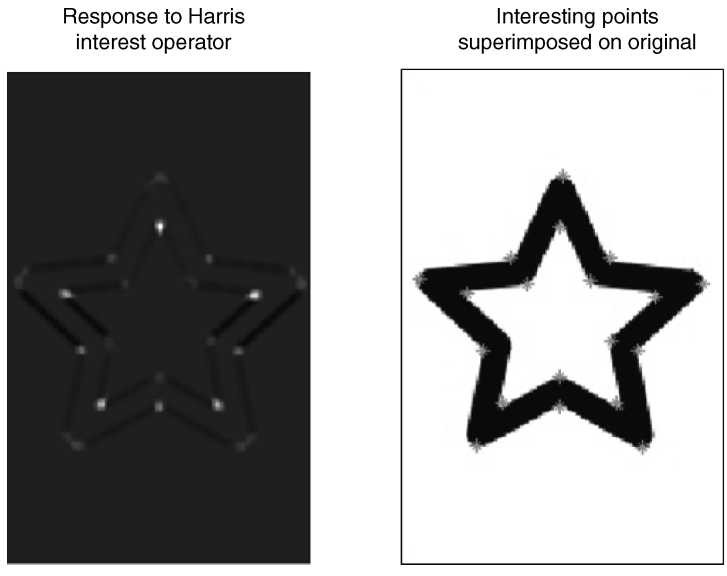

Figure 10.7 Left: the Harris response function to an image comprising a dark star on a plain background. Right: the significant local maximum points are shown superimposed on the original image

Example 10.4

Matlab code	What is happening?
I=imread('zener_star.jpg');	%Read in image and convert to
I=double(rgb2gray(I));	%intensity
wdth=5; sdvn=2;	%Fix smoothing parameters
k=0.04;	%Fix Harris constant
hsmall=fspecial('gaussian',[wdth wdth],sdvn);	%Define Gaussian filter
[Fx,Fy]=gradient(I);	%Calculate gradient
Fx_sq=Fx.^2; Fy_sq=Fy.^2; Fx_Fy=Fx.*Fy;	%Define terms in Harris function
Fx_sq=filter2(hsmall,Fx_sq);	%Perform neighbourhood smoothing
Fy_sq=filter2(hsmall,Fy_sq);	%on each term
Fx_Fy=filter2(hsmall,Fx_Fy);	
R=(1-2.*k).*Fx_sq.*Fy_sq - k.*(Fx_sq.^2 + Fy_sq.^2) - Fx_Fy.^2;	%Calculate Harris function
S=ordfilt2(R,wdth.^2,ones(wdth));	%Maximum filtering over %neighbourhood
[j,k]=find(R>max(R(:))./12 & R==S);	%Get subscript indices of local %maxima
subplot(1,2,1), imagesc(R); axis image; axis off; colormap(gray);	%Display Harris response
subplot(1,2,2), imshow(I,[]);	%Display original image
hold on; plot(k,j,'r*');	%Interest points superimposed
bw=zeros(size(R)); bw([j,k])=1;	%Return logical array of interest
bw=logical(bw);	%locations

Comments
Matlab functions: ***ordfilt2***, ***fspecial***, ***find***, ***logical***.
The interest response is combined with a local maximum filter to ensure that only one local maximum is identified. A subsequent hierarchical thresholding returns the points of maximum interest.

10.10 Watershed segmentation

Watershed segmentation is a relatively recent approach which has come increasingly to the fore in recent years and tends to be favoured in attempts to separate touching objects – one of the more difficult image processing operations. In watershed segmentation, we envisage the 2-D, grey-scale image as a topological surface or 'landscape' in which the location is given by the *x*,*y* image coordinates and the *height* at that location corresponds to the image intensity or grey-scale value.

Rain which falls on the landscape will naturally drain downwards, under the action of gravity, to its nearest minimum point. A *catchment basin* defines that connected region or area for which any rainfall drains to the *same* low point or minimum. In terms of a digital image, the catchment basin thus consists of a group of connected pixels. Maintaining the analogy with a physical landscape, we note that there will be points on the landscape (local maxima) at which the rainfall is equally likely to fall into two adjacent catchment basins; this is analogous to walking along the ridge of a mountain. Lines which divide one catchment area from another are called watershed ridges, watershed lines or simply watersheds. An alternative viewpoint is to imagine the landscape being gradually flooded from below with the water entering through the local minima. As the water level increases, we construct dams which prevent the water from the catchment basins spilling or spreading into adjacent catchment basins. When the water level reaches the height of the highest peak, the construction process stops. The dams built in this way are the watersheds which partition the landscape into distinct regions containing a catchment basin. The actual calculation of watersheds in digital images can be performed in several ways, but all fundamentally hinge on iterative morphological operations.

These basic concepts are illustrated for a 1-D landscape in Figure 10.8.

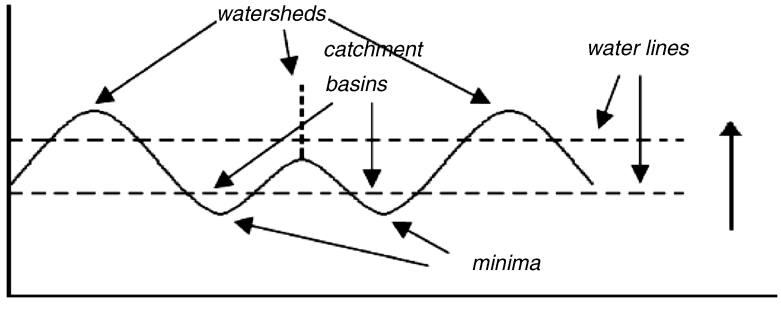

Figure 10.8 Watersheds and catchment basins. Basins are defined as connected regions with one local minimum to which any incident rainfall will flow. The watersheds can be visualized as those ridges from which 'water' would be equally likely to fall towards either of two or more catchment basins

Segmentation using watershed methods has some advantages over the other methods we have discussed in this chapter. A notable advantage is that watershed methods, unlike edge-detection-based methods, generally yield *closed contours* to delineate the boundary of the objects. A number of different approaches can be taken to watershed segmentation, but a central idea is that *we try to transform the initial image (the one we wish to segment) into some other image such that the catchment basins correspond to the objects we are trying to segment.*

10.11 Segmentation functions

A segmentation function is basically a function of the original image (i.e. another image derived from the original) whose properties are such that the catchment basins lie within the objects we wish to segment. The computation of suitable segmentation functions is not always straightforward but underpins successful watershed segmentation. The use of gradient images is often the first preprocessing step in watershed segmentation for the simple reason that the gradient magnitude is usually high along object edges and low elsewhere. In an ideal scenario, the watershed ridges would lie along the object edges.

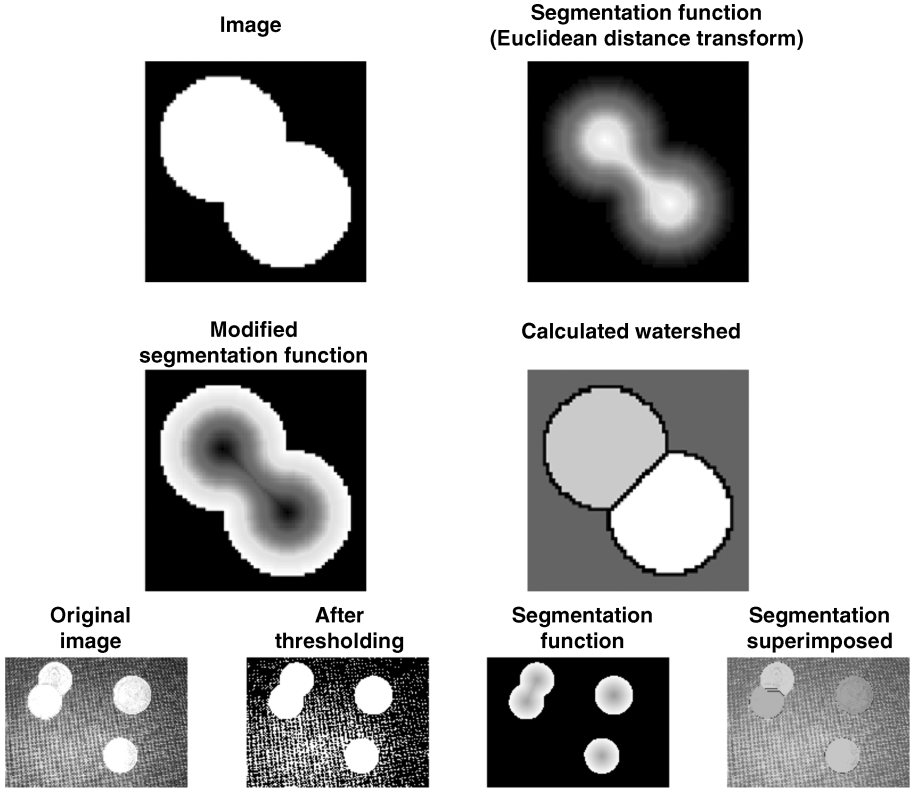

Figure 10.9 In the first, idealized example, the watershed yields a perfect segmentation. In the second image of overlapping coins, morphological opening is required first on the thresholded image prior to calculation of the watershed (*See colour plate section for colour version*)

However, noise and small-scale structures within the objects generally result in many local, small catchment basins (broadly analogous to puddles on the landscape). These spoil things and produce oversegmentation. Figure 10.9 (produced using the Matlab code in Example 10.5) illustrates this.

Example 10.5

Matlab code	What is happening?
center1 = -10;	%Create image comprising two
	%perfectly
center2 = -center1;	%smooth overlapping circles
dist = sqrt(2*(2*center1)^2);	
radius = dist/2 * 1.4;	
lims = [floor(center1-1.2*radius)	
eil(center2 + 1.2*radius)];	
[x,y] = meshgrid(lims(1):lims(2));	
bw1 = sqrt((x-center1).^2	
+ (y-center1).^2) <= radius;	
bw2 = sqrt((x-center2).^2	
+ (y-center2).^2) <= radius;	
bw = bw1 | bw2;	
D = bwdist(~bw);	%Calculate basic segmentation function
	%(Euclidean distance transform
	%of ~bw)
subplot(2,2,1), imshow(bw,[]);	%Display image
subplot(2,2,2), imshow(D,[]);	%Display basic segmentation function
	%Modify segmentation function
D = -D;	%Invert and set background pixels
	%*lower*
D(~bw) = -inf;	%than all catchment basin minima
subplot(2,2,3), imshow(D,[]);	%Display modified segmentation image
L = watershed(D); subplot(2,2,4), Imagesc(L);	%Calculate watershed of segmentation
	%function
axis image; axis off; colormap(hot); colorbar	%Display labelled image – colour
	%coded
A=imread('overlapping_euros1.png');	%Read in image
bw=im2bw(A,graythresh(A));	%Threshold automatically
se=strel('disk',10); bwo=imopen(bw,se);	%Remove background by opening
D = bwdist(~bwo);	%Calculate basic segmentation function
D = -D; D(~bwo) = -255;	%Invert, set background lower than
	%catchment basin minima
L = watershed(D);	%Calculate watershed
subplot(1,4,1), imshow(A);	%Display original
subplot(1,4,2), imshow(bw)	%Thresholded image
subplot(1,4,3), imshow(D,[]);	%Display basic segmentation function

```
ind=find(L==0); Ac=A; Ac(ind)=0;          %Identify watersheds and set=0 on
                                            original
subplot(1,4,4), Imagesc(Ac); hold on      %Segmentation superimposed on
                                            original
Lrgb = label2rgb(L, 'jet', 'w', 'shuffle'); %Calculate label image
himage = imshow(Lrgb); set(himage,        %Superimpose transparently on original
  'AlphaData', 0.3);
```

Comments

Matlab functions: ***bwdist***, ***watershed***.

The Euclidean distance transform operates on a binary image and calculates the distance from each point in the background to its nearest point in the foreground. This operation on the (complement of) the input binary image produces the basic segmentation function. This is modified to ensure the minima are at the bottom of the catchment basins. Computation of the watershed then yields the segmentation. Note the watershed function labels each uniquely segmented region with an integer and the *watersheds are assigned a value of 0.*

In Figure 10.9, the first (artificial) image is smooth and in this case, calculation of a suitable segmentation function is achieved by the Euclidean distance transform.[4] The corresponding image of some real coins in the second example beneath it requires some morphological pre-processing but essentially yields to the same method. However, the segmentation is substantially achieved here (apart from the overlapping coins) by the thresholding. In Figure 10.10 (produced using the Matlab code in Example 10.6) we attempt a gradient-based segmentation function, but this results in the over-segmentation shown. This outcome is typical in all but the most obliging cases.

An approach which can overcome the problem of over-segmentation in certain cases is marker-controlled segmentation, the basic recipe for which is as follows:

(1) Select and calculate a suitable segmentation function. Recall that this is an image whose catchments basins (dark regions) are the objects you are trying to segment. A typical choice is the modulus of the image gradient.

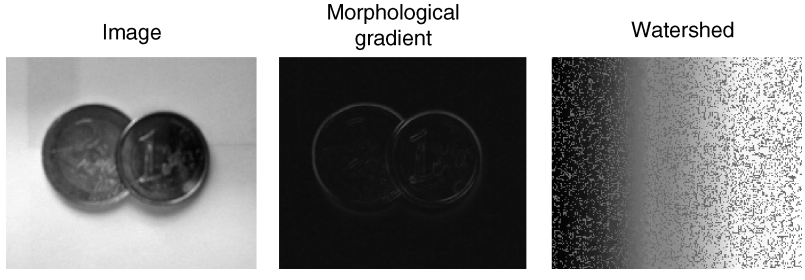

Figure 10.10 Direct calculation of the watershed on gradient images typically results in oversegmentation

[4] The Euclidean distance transform codes each foreground pixel as the shortest distance to a background pixel.

Example 10.6

Matlab code	**What is happening?**
A=imread('overlapping_euros.jpg');	%Read image
Agrad=ordfilt2(A,25,ones(5))-ordfilt2(A,1,ones(5));	%Calculate basic %segmentation function
figure, subplot(1,3,1), imshow(A,[]);	%Display image
subplot(1,3,2), imshow(Agrad,[]);	%Display basic segmentation %function
L = watershed(Agrad), rgb = label2rgb(L,'hot',[.5 .5 .5]);	%Calculate watershed
subplot(1,3,3), imshow(rgb,'InitialMagnification','fit')	%Display labelled image

Comments

Matlab functions: ***ordfilt2***, ***watershed***.

Watershed on the raw gradient image results in oversegmentation.

(2) Process the original image to calculate so-called foreground markers. Foreground markers are basically connected blobs of pixels lying within each of the objects. This is usually achieved via grey-scale morphological operations of opening, closing and reconstruction

(3) Repeat this process to calculate background markers. These are connected blobs of pixels that are not part of any object.

(4) Process the segmentation function so that it *only possesses minima at the foreground and background marker* locations.

(5) Calculate the watershed transform of the modified segmentation function.

Figure 10.11 shows a (largely) successful marker-controlled segmentation of a relatively complex image.[5] The rationale for this example is explained in Example 10.7.

Marker-controlled segmentation is an effective way of avoiding the problem of over-segmentation. As is evident from the aforementioned example, it generally needs and exploits a priori knowledge of the image and the methods required for marker selection are strongly dependent on the specific nature of the image. Thus, its strength is that it uses the specific context effectively. Conversely, its corresponding weakness is that it does not generalize well and each problem must be treated independently.

[5] This example is substantially similar and wholly based on the watershed segmentation example provided by the image processing demos in Matlab version 7.5. The authors offer grateful acknowledgments to The MathWork Inc. 3 Apple Hill Dr. Natick, MA, USA.

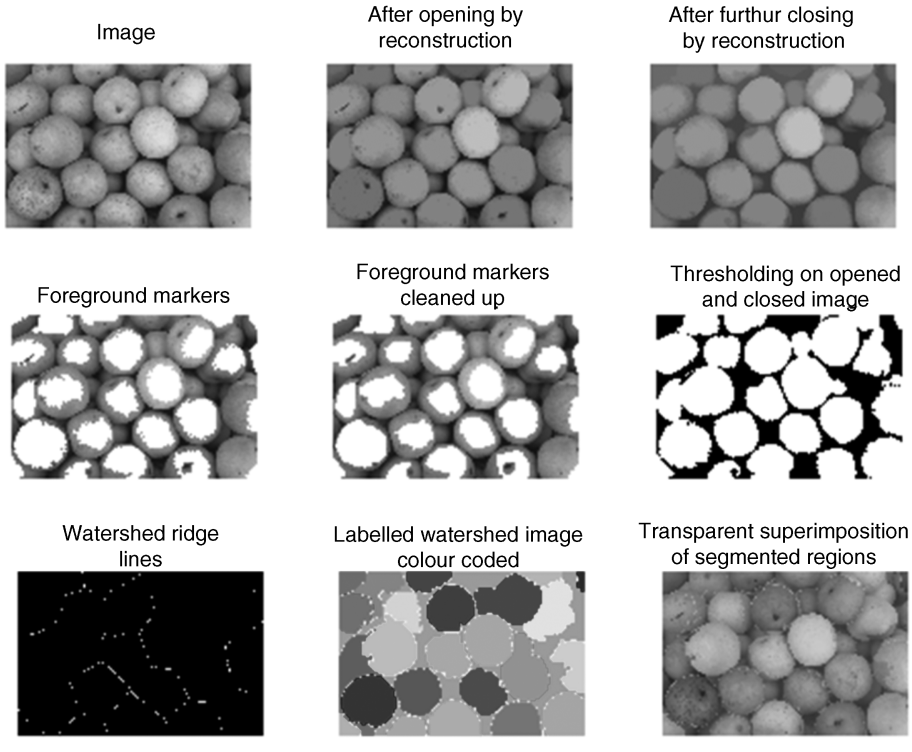

Figure 10.11 Marker-controlled watershed segmentation (*See colour plate section for colour version*)

Example 10.7

Matlab code	**What is happening?**
rgb = imread('pears.png');	% Step 1. Read image and
I = rgb2gray(rgb);	% use the Gradient Magnitude as
subplot(1,3,1), imshow(I)	% the basic segmentation Function
hy = fspecial('sobel');	
hx = hy';	
Iy = imfilter(double(I), hy, 'replicate');	
Ix = imfilter(double(I), hx, 'replicate');	
gradmag = sqrt(Ix.^2 + Iy.^2);	
se = strel('disk', 20);	% Step 2. Mark the Foreground Objects using
Ie = imerode(I, se);	% morphological techniques called
Iobr = imreconstruct(Ie, I);	% "opening-by-reconstruction" and
subplot(1,3,2), imshow(Iobr),	% "closing-by-reconstruction" to "clean"
	% up the image. These operations will
	% create flat maxima inside each object
	% that can be located using imregionalmax.
Iobrd = imdilate(Iobr, se);	% Following the opening with a closing
Iobrcbr = imreconstruct(imcomplement(Iobrd),	

```
imcomplement(Iobr));
Iobrcbr = imcomplement(Iobrcbr);      % to remove the dark spots and stem marks.
subplot(1,3,3); imshow(Iobrcbr);      % Notice you must complement the image
                                      % inputs and output of imreconstruct.

fgm = imregionalmax(Iobrcbr);         % Calculate the regional maxima of Iobrcbr
                                      % to obtain good foreground markers.

I2 = I; I2(fgm) = 255;                % To help interpret the result, superimpose
figure; subplot(1,3,1); imshow(I2);   % these foreground marker image on the
                                      % original image.

se2 = strel(ones(5,5));               % Some of the mostly-occluded and shadowed
fgm2 = imclose(fgm, se2);             % objects are not marked, which means that
fgm3 = imerode(fgm2, se2);            % these objects will not be segmented
                                      % properly in the end result.  Also, the
                                      % foreground markers in some objects go
                                      % right up to the objects' edge.  That means
                                      % we must clean the edges of the marker
                                      % blobs and then shrink them a bit.

fgm4 = bwareaopen(fgm3, 20);          % This procedure leaves some stray isolated
I3 = I; I3(fgm4) = 255;               % pixels that must be removed.  Do this
subplot(1,3,2), imshow(I3);           % using bwareaopen, which removes all blobs
                                      % that have fewer than a certain number of
                                      % pixels.

bw = im2bw(Iobrcbr,                   %Step 3: Compute Background Markers
   graythresh(Iobrcbr));
subplot(1,3,3), imshow(bw);           % Now we need to mark the background.  In
                                      % the cleaned-up image, Iobrcbr, the
                                      % dark pixels belong to the background,
                                      % so you could start with a thresholding
                                      % operation.
D = bwdist(bw);                       % The background pixels are in black,
DL = watershed(D);                    % but ideally we don't want the background
bgm = DL == 0;                        % markers to be too close to the edges of
figure; subplot(1,3,1);imshow(bgm);   % the objects we are trying to segment.
                                      % We'll "thin" the background by computing
                                      % the "skeleton by influence zones", or SKIZ,
                                      % of the foreground of bw. This can be done
                                      % by computing the watershed transform of the
                                      % distance transform of bw, and
                                      % then looking for the watershed ridge
                                      % lines (DL == 0) of the result.

gradmag2 = imimposemin                % Step 4: Compute the modified
   (gradmag, bgm | fgm4);
                                      % segmentation function.
                                      % The function imimposemin is used to modify
```

```
                                % an image so that it has regional minima
                                % only in certain desired locations.
                                % Here you can use imimposemin to modify
                                % the gradient magnitude image so that its
                                % regional minima occur at foreground and
                                % background marker pixels.

L = watershed(gradmag2);        % Step 5: Now compute the Watershed
                                % Transform of modified function
Lrgb = label2 rgb(L, 'jet',     % A useful visualization technique is to display the
   'w', 'shuffle');             % label matrix L as a color
subplot(1,3,2),imshow(Lrgb)     % image using label2rgb. We can then superimpose this
subplot(1,3,3),imshow(I),       %pseudo-color label
   hold on                      % matrix on top of the original intensity image.
himage = imshow(Lrgb);
   set(himage, 'AlphaData', 0.3);
```

10.12 Image segmentation with markov random fields

Except in the most obliging of situations, intensity thresholding alone is a crude approach to segmentation. It rarely works satisfactorily because it does not take into account the spatial relationships between pixels. Segmentation via region growing or splitting enables pixels to be labelled based on their similarity with neighbouring pixels. The use of Markov random field models is a more powerful technique which determines how well a pixel belongs to a certain class by modelling both the variability of grey levels (or other pixel attributes) within each class and its dependency upon the classification of the pixels in its neighbourhood.

Segmentation using Markov random fields is not a simple technique. Our aim here will be strictly limited to discussing the essence of the approach so that an interested reader is then better equipped to go on to examine the literature on this subject. We shall start by stating Bayes' law of conditional probability:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} \tag{10.8}$$

In our context of image segmentation, we shall suppose we are given an image containing a range of grey levels denoted by y and a set of possible labels denoted by x. In the simplest segmentation scenarios, there may, for example, be just two labels, referred to as *object* and *background*.

Our basic task is to assign to each pixel the most probable label x given our set of observations (i.e. our image) y. In other words, we wish to maximize $p(x|y)$ which, by Bayes'

theorem, is equivalent to maximizing the right-hand side of Equation (10.8). Let us then carefully consider each term in Equation 10.8.

- First, $p(y)$ refers to the probability distribution of pixel grey levels (i.e. the distribution which would be obtained from a grey-level histogram). *For a given image*, this quantity is fixed. Therefore, if $p(y)$ is a constant for a given image, then maximizing $p(x|y)$ reduces to maximising $p(y|x)p(x)$.

- Second, the conditional density $p(y|x)$ refers to the probability distribution of pixel grey levels y for a given class x (e.g. background or object). In the ideal case, a background would have one grey level and an object would have a second distinct grey level. In reality, the grey levels for both classes will exhibit a spread. The densities are often modelled as normal distributions:

$$p(y|x) = \frac{1}{\sqrt{2\pi}\sigma_x} \exp\left[-\frac{(y-\mu_x)^2}{2\sigma_x^2}\right] \tag{10.9}$$

  where μ_x and σ_x respectively refer to the mean and standard deviation of the pixel intensity in the class x.

- Third, the term $p(x)$, known as the prior distribution, refers to the level of agreement between the pixel's current label and the label currently assigned to its neighbouring pixels. This requires some elaboration. Intuitively, we would reason that $p(x)$ should return a low value if the pixel's label is *different* from all of its neighbours (since it is very unlikely that an isolated pixel would be surrounded by pixels belonging to a different class). By the same line of reasoning, $p(x)$ should increase in value as the number of identically labelled neighbouring pixels increases.

The whole approach thus hinges on the prior term in Bayes' law, because it is this term which must build in the spatial context (i.e. the correlation relations between a pixel and its neighbours).

To the question 'How exactly is the prior information built into our model?', we cannot give a full explanation here (this would require a long and complex detour into mathematical concepts traditionally associated with statistical physics), but the basic approach is to model the relationship between a pixel and its neighbours as a Markov random field. A Markov random field essentially models the probabilities of certain combination of intensity values occurring at various pixel locations in a neighbourhood region. To specify such probabilities directly is generally a difficult and even overwhelming task because there are a huge number of possible combinations. Fortunately, a powerful mathematical result exists which demonstrates that the right-hand side of Equation (10.8) can be written in the form of an exponential which is known as a Gibbs distribution:

$$p(x|y) = \frac{\exp[-U(x|y)]}{Z_y} \tag{10.10}$$

where Z_y is a normalizing constant which is included to prevent the above equation from returning a probability of greater than one. The exponential part of this equation turns out to be:

$$U(x|y) = \sum_i \left[\frac{1}{2} \ln \sigma_{x_i}^2 + \frac{(y - \mu_{x_i})^2}{2\sigma_{x_i}^2} + \sum_{n=1}^{N} \theta_n J(x_i x_{i+n}) \right] \tag{10.11}$$

By analogy with arguments from statistical physics, $U(x|y)$ can be loosely interpreted as a 'potential energy' term and it is clear from Equation (10.10) that the maximization of $p(x|y)$ reduces to minimizing the energy term $U(x|y)$.

In Equation (10.11), the outer summation covers all pixels in the image (i.e. the contents are applied to *every* pixel in the image). The first two terms of the outer summation refer to $p(y|x)$ as given in Equation (10.9), which expresses how well the pixel's grey level fits its presumed class label. The third term represents $p(x)$ and expresses the level of agreement between the pixel's label and those of its neighbours. This is itself expressed in the form of a summation over the pixel's immediate neighbours. The function $J(x_i, x_{i+n})$ returns a value of -1 whenever the label for the nth neighbourhood pixel x_{i+n} is the same as the label for the central pixel x_i. The term θ_n is included to adjust the relative weighting between the $p(x)$ and $p(y|x)$ terms.

To complete our formulation of the problem, we need to resolve two questions:

(1) How do we model the class-conditional densities $p(y|x)$ (the parameter estimation problem)?

(2) How do we select the neighbourhood weighting parameters θ_n to properly balance the two contributions to the energy function?

10.12.1 Parameter estimation

Assuming the Gaussian distribution for the class-conditional densities $p(y|x)$ given by Equation (10.9), the two free parameters of the distribution (the mean and variance) can be found by fitting Gaussian curves to a representative (model) image which has been manually segmented into the desired classes. Where such a representative image is not available, the parameters can be estimated by fitting such curves to the actual image histogram. For a two-class segmentation problem (object and background), we would thus fit a Gaussian mixture (sum of two Gaussian functions) to the image histogram $h(y)$, *adjusting the free parameters* to achieve the best fit:

$$
\begin{aligned}
h(y) &= \sum_{i=1}^{2} \frac{1}{\sqrt{2\pi}\sigma_{x_i}} \exp\left[-\frac{(y-\mu_{x_i})^2}{2\sigma_{x_i}^2}\right] \\
&= \frac{1}{\sqrt{2\pi}\sigma_{\text{Obj}}} \exp\left[-\frac{(y-\mu_{\text{Obj}})^2}{2\sigma_{\text{Obj}}^2}\right] + \frac{1}{\sqrt{2\pi}\sigma_{\text{Back}}} \exp\left[-\frac{(y-\mu_{\text{Back}})^2}{2\sigma_{\text{Back}}^2}\right]
\end{aligned}
\tag{10.12}
$$

The extension to more than two classes follows naturally.

10.12.2 Neighbourhood weighting parameter θ_n

The parameter θ_n represents the weighting which should be given to the local agreement between pixel labels. For example, a pixel which is presently labelled as 'object' should have a higher probability of being correct if, for example, six of its neighbouring pixels were

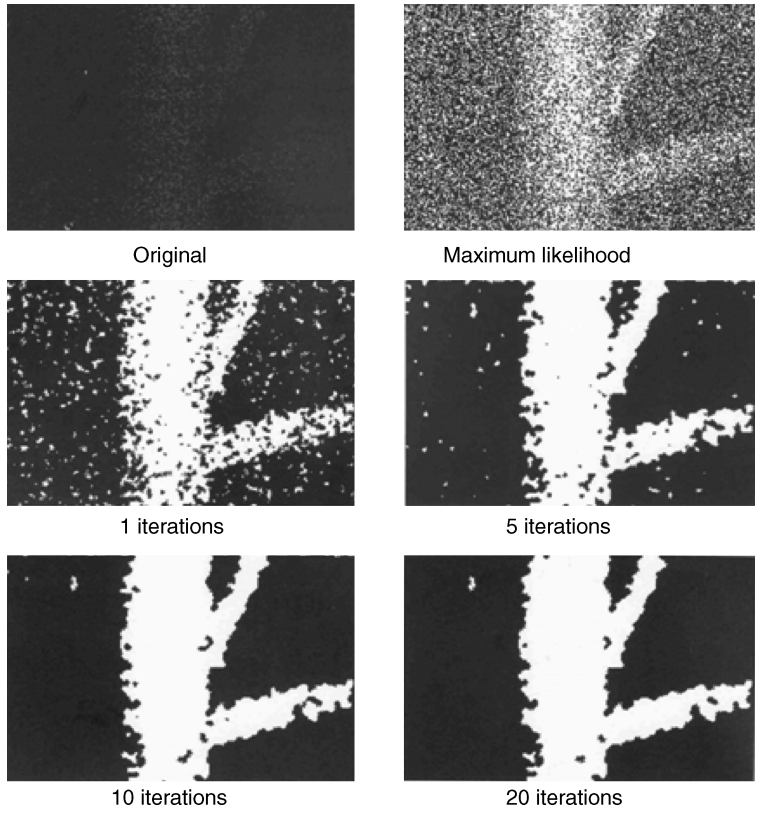

Figure 10.12 Segmentation of a noisy, underwater image of a cylindrical pipe structure. The sequence from left to right and top to bottom shows the original image, the maximum likelihood estimate and the results of segmentation using increasing numbers of iterations of the ICM algorithm. (Image sequence courtesy of Dr Mark Hodgetts, Cambridge Research Systems Ltd.)

identically labelled as 'object' as opposed to three. The analytical approach to determining θ_n would involve initially labelling all pixels according to pixel attribute alone (i.e. in the absence of any contextual information). Every labelled pixel in the image would then be examined and a note would be made of the number of identically labelled neighbouring pixels. Counts would be made for every combination of pixel label and number of agreeing neighbouring pixel labels. These would be used to infer how the probability of the central pixel label being correct varies with the number of consistent neighbouring labels, which would in turn be used to generate estimates for θ_n. In practice, most implementations treat θ_n as a constant θ which is initially estimated at around unity and adjusted until optimal results are obtained. A higher value of θ will tend to emphasize local agreement of pixel labels over the class suitability according to pixel attribute and vice versa.

10.12.3 Minimizing U(x | y): the iterated conditional modes algorithm

Having formulated the problem and estimated the parameters of the class-conditional densities and the neighbourhood weighting parameter, the final step in the segmentation is to actually find a viable means to minimize $U(x|y)$ in Equation (10.11). We briefly describe the iterated conditional modes (ICM) algorithm, as it is conceptually simple and computationally viable. The ICM algorithm is an example of a 'greedy algorithm', as it works on a pixel-by-pixel basis, accepting *only* those changes (i.e. changes from background to foreground or vice versa) which take us successively nearer to our goal of minimizing the potential. This is in contrast to other approaches, such as simulated annealing (Geman and Geman) which allow temporary increases in the potential function to achieve the overall goal of minimization. The ICM algorithm decomposes Equation (10.11) so that it is applied to single pixels in the image

$$U_i(x|y) = \frac{1}{2}\ln\sigma_{x_i}^2 + \frac{(y_i - \mu_{x_i})^2}{2\sigma_{x_i}^2} + \sum_{n=1}^{N} \theta_n J(x_i x_{i+n}) \tag{10.13}$$

- Step 1: label each pixel with its most probable class based on its pixel attribute only, i.e. according to whichever of the class-conditional densities yields the maximum value. This is termed the maximum-likelihood estimate.
- Step 2: for each pixel in the image, update it to the class which minimizes Equation 10.13.
- Repeat Step 2 until stability occurs.

Figure 10.12 shows an example of Markov random field segmentation using the ICM algorithm.

For further examples and exercises see http://www.fundipbook.com