# 14

# Image Segmentation

## 14.1 Introduction

Image segmentation is the identification and isolation of an image into regions that, one hopes, correspond to structural units. It is an especially important operation in biomedical image processing since it is used to isolate physiological and biological structures of interest. The problems associated with segmentation have been well studied and a large number of approaches have been developed, many specific to particular image features. The general approaches to segmentation can be grouped into four classes: pixel-based, regional or continuity-based, edge-based, and morphological methods. Pixel-based methods are the easiest to understand and to implement, but are also the least powerful and, since they operate on one element at a time, are particularly susceptible to noise. Continuity-based and edge-based methods approach the segmentation problem from opposing sides: edge-based methods search for differences while continuity-based methods search for similarities. Morphological methods use information on shape to constrain or define the segmented image. Recently, information concerning the mechanics and dynamics of the imaged tissue has been adapted in advanced approaches to segmentation. In biomedical imaging, segmentation is often very challenging, and multiple approaches are used.

## 14.2 Pixel-Based Methods

The most straightforward and common of the pixel-based segmentation methods is *thresholding* in which all pixels having intensity values above or below some level are classified as part of the segment. Thresholding is an integral part of converting an intensity image into a binary image as demonstrated in Example 14.2. Even when preceded by other approaches, thresholding is usually required to produce a segmentation *mask*, a black-and-white image of the feature of interest.

Thresholding is usually quite fast and can be done in real time, allowing for interactive adjustment of the threshold. The basic concept of thresholding can be extended to include both upper and lower boundaries, an operation termed *slicing* since it isolates a specific range of pixel intensities. Slicing can be generalized to include a number of different upper and lower boundaries, each encoded into a different number. An example of multiple slicing was presented in Chapter 12 using MATLAB's `gray2slice` routine. Finally, when an RGB image is involved, thresholding can be applied to each color plane separately. The resulting image can be either a thresholded RGB image or a single image composed of a logical combination (AND or OR) of the three image planes after thresholding. An example of this approach is seen in the problems.

A technique that can aid in all image analysis approaches, but is particularly useful in pixel-based methods, is intensity remapping. In this global procedure, pixel values are rescaled so as to extend over different maximum and minimum values. Usually, the rescaling is linear; so, each point is adjusted proportionally with a possible offset. MATLAB supports rescaling with the routine `imadjust` described below, which also provides a few common nonlinear rescaling options. Of course, any rescaling operation is possible using MATLAB code if the intensity images are in double format or the image arithmetic routines described in Chapter 12 are used.

### 14.2.1 Threshold Level Adjustment

A major concern in pixel-based methods is setting the threshold or slicing level(s) appropriately. Usually, these levels are set by the program, although in some situations, they are set interactively by the user. Finding an appropriate threshold level can be aided by a plot of the distributions of pixel intensity over the image, regardless of whether you adjust pixel level interactively or automatically. Such a plot is termed the *intensity histogram* and is supported by the MATLAB routine `imhist` described below. Figure 14.1 shows an x-ray image of the spine with its associated intensity histogram. Figure 14.1 also shows the binary image obtained by applying a threshold at a specific point on the histogram indicated by the vertical line. When RGB color images are being analyzed, intensity histograms can be obtained from all three color planes and different thresholds can be established for each color plane with the aid of the corresponding histogram.

Intensity histograms can be very helpful in selecting threshold levels, not only for the original image, but also for images produced by various segmentation algorithms described later. Intensity histograms can also be useful in evaluating the efficacy of different processing schemes: as the separation between structures improves, histogram peaks should become more distinctive. This relationship between separation and histogram shape is demonstrated in Figure 14.2 and, more dramatically, in Figures 14.7 and 14.11.

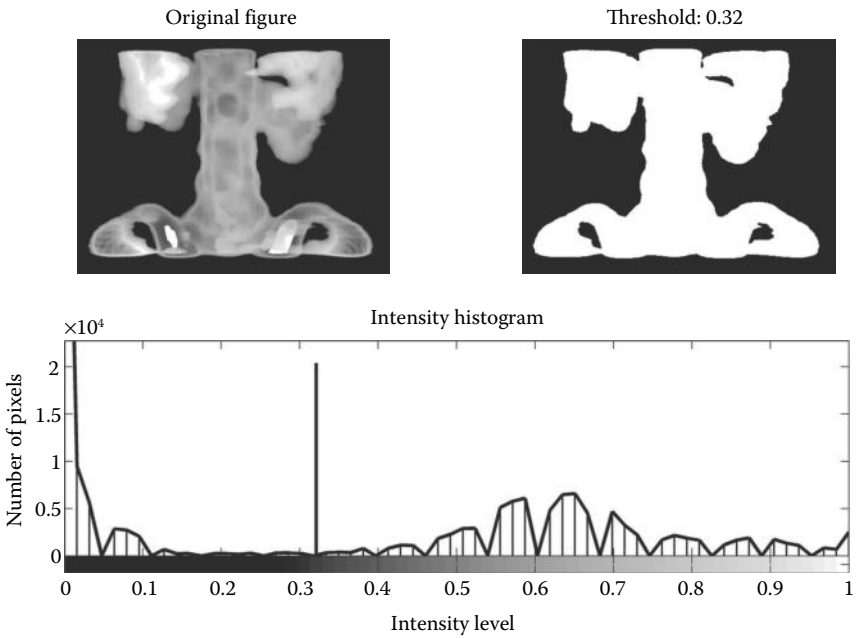


Figure 14.1 X-ray image of the spine, upper left, and its associated intensity histogram, lower plot. The upper right image is obtained by thresholding the original image at a value corresponding to the vertical line on the histogram plot. (Image courtesy of MATLAB.)

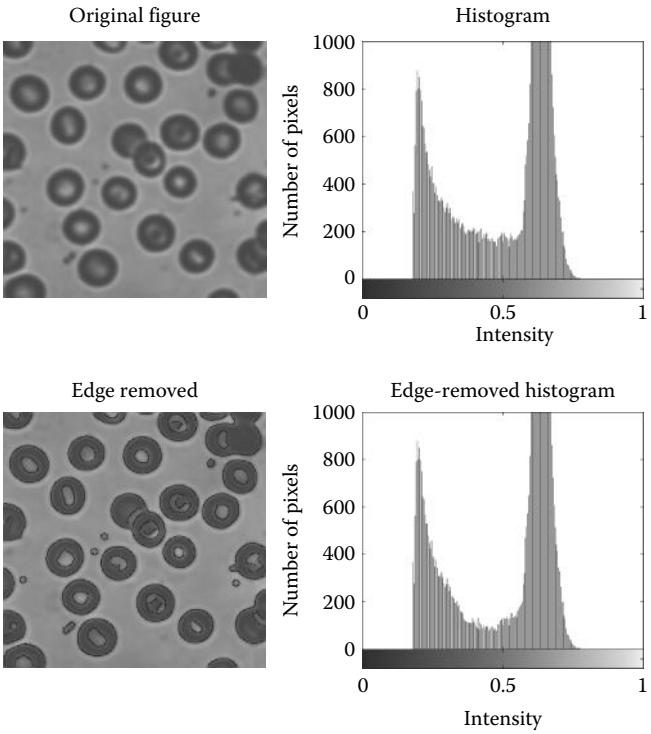


Figure 14.2 Image of bloods cells with (upper) and without (lower) intermediate boundaries removed. The associated histograms (right side) show improved separability when the boundaries are eliminated. The code that generated these images is given in Example 14.1.

Although intensity histograms contain no information on position, they can still be useful for segmentation, particularly for estimating threshold(s) from the histogram (Sonka et al. 1993). If the intensity histogram is, or is assumed to be, bimodal (or multimodal), a common strategy is to search for low points or minima in the histogram. This is the strategy used in Figure 14.1 where the threshold is set at 0.32, the intensity value where the histogram shows a minimum. Such points represent the fewest number of pixels, but often, the histogram minima are difficult to determine visually.

An approach to improve the determination of histogram minima is based on the observation that many boundary points carry values intermediate to those on either side of the boundary. These intermediate values lie between the actual boundary values and may mask the optimal threshold value. However, these intermediate points also have the highest gradient, and it should be possible to identify them using a gradient-sensitive filter such as the Sobel or Canny filter. After these boundary points are identified, they can be eliminated from the image and a new histogram can be computed that has a more definitive distribution. This strategy is used in Example 14.1, and Figure 14.2 shows the images and associated histograms before and after the removal of boundary points identified using the Canny filtering. A slight reduction in the number of intermediate points can be seen in the middle of the histogram (around 0.45). As shown in Figure 14.3, this leads to somewhat better segmentation of the blood cells when the threshold is based on the histogram as explained in Example 14.1.

Another histogram-based strategy that can be used if the distribution is bimodal is to assume that each mode is the result of a unimodal, Gaussian distribution. An estimate is then made of the underlying distributions, and the point at which the two estimated distributions intersect

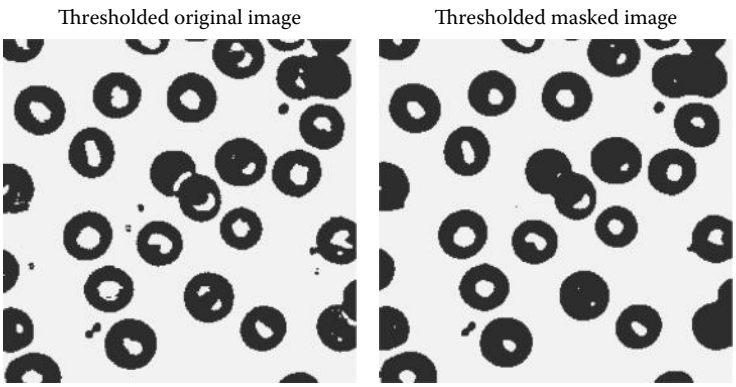


Figure 14.3 Thresholded blood cell images. Thresholds based solely on the histogram (as explained below) were applied to the blood cell images in Figure 14.2 with (left) and without (right) boundary pixels removed (i.e., set to zero). Somewhat fewer inappropriate pixels are seen in the right image.

should provide the optimal threshold. The principal problem with this approach is that the distributions are unlikely to be Gaussian.

A threshold strategy that does not use the histogram directly is based on the concept of minimizing the variance between presumed foreground and background elements. Although the method assumes two different gray levels, it works well even when the distribution is not bimodal (Sonka et al. 1993). The approach termed *Outso's method* uses an iterative process to find a threshold that minimizes the *variance* between the intensity values on either side of the threshold level. This approach is implemented using the MATLAB routine `graythresh` described below and used in Example 14.1.

A pixel-based technique that provides a segment boundary directly is *contour mapping*. Contours are lines of equal intensity and, in a continuous grayscale image, they are necessarily continuous: they cannot end within the image, although they can branch or loop back on themselves. In digital images, these same properties exist, but the value of any given contour line generally requires interpolation between adjacent pixels. To use contour mapping to identify image structures requires accurate setting of the contour levels, and this carries the same burdens as thresholding. Nonetheless, contour maps do provide boundaries directly and, if "subpixel" interpolation is used in establishing the contour position, they may be more spatially accurate. Contour maps are easy to implement in MATLAB, as shown in the next section. Figure 14.4 shows

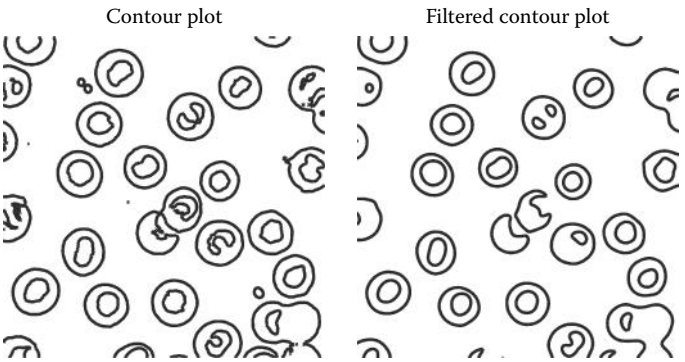


Figure 14.4 Contour maps drawn from the blood cell image of Figures 14.2 and 14.3. The right image was prefiltered with a Gaussian lowpass filter (alpha = 3) before the contour lines were drawn. The contour values were set manually to provide good outlines.

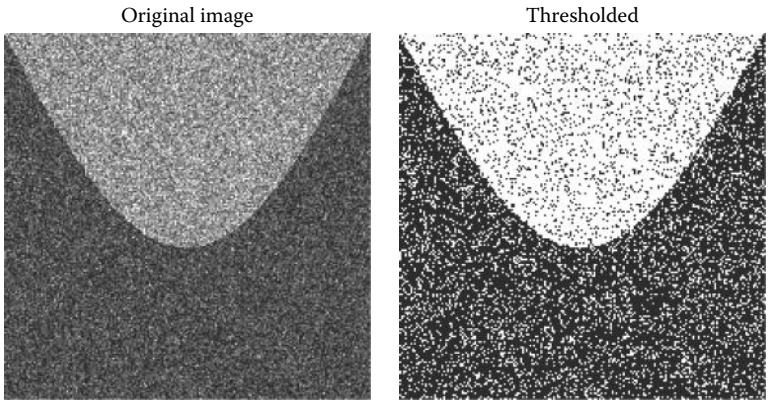


Figure 14.5 Image with two regions that have different average gray levels. The two regions are clearly distinguishable, but it is not possible to accurately segment the two regions using thresholding alone because of noise.

contour maps for the blood cell images shown in Figure 14.2. The right image was prefiltered with a Gaussian lowpass filter that reduces noise slightly and improves the resultant contour image.

Pixel-based approaches can lead to serious errors, even when the average intensities of the various segments are clearly different, due to noise-induced intensity variation within the structure. Such variation could be acquired during image acquisition, but could also be inherent in the structure itself. Figure 14.5 shows two regions with quite different average intensities. Even with optimal threshold selection, many inappropriate pixels are found in both segments due to intensity variations within the segments. The techniques for improving separation in such images are explored in Section 14.3.

### 14.2.2 MATLAB Implementation

Some of the routines for implementing pixel-based operations such as `im2bw` and `grayslice` have been described in the preceding chapters. The image intensity histogram routine is produced by `imhist` without the output arguments:

```
[counts, x] = imhist(I, N);
```

where `counts` is the histogram value at a given `x`, `I` is the image, and `N` is an optional argument specifying the number of histogram bins (the default is 255). As mentioned above, `imhist` is usually invoked without the output arguments to produce a plot directly. Several pixel-based techniques are presented in Example 14.1.

An automated thresholding operation is performed by the MATLAB routine `graythresh`. This program determines a threshold based on the intensity histogram using Outso's method. As mentioned above, this method iteratively adjusts the threshold to minimize the variance on either side. The routine is called as

```
thresh = graythresh(I); % Determine threshold using Outso's method
```

The output threshold can be used directly in `im2bw` to construct a thresholded image (in fact, the routine can be embedded in the call to `im2bw`).

**EXAMPLE 14.1**

An example of segmentation using pixel-based methods. Load the image of blood cells and display this image along with the intensity histogram. Remove the edge pixels from the image and

display the histogram of this modified image. Determine thresholds using the minimal variance iterative technique (Outso's method) and apply this approach to threshold both the original and modified images. Display the resultant thresholded images.

**Solution**

To remove the edge boundaries, first identify these boundaries using an edge detection scheme. While any of the edge detection filters described previously can be used, here, we use the Canny filter as it is the most robust to noise. This filter is described in Section 14.6 and is implemented as an option of MATLAB's `edge` routine, which produces a binary image of the boundaries. This binary image is converted into a boundary *mask* by inverting the image using the NOT operator. After inversion, the edge pixels will be zero while all other pixels will be one. Multiplying the original image by the boundary mask produces an image in which the boundary points are removed (i.e., set to zero or black). Then `graythresh` is used to find the minimum variance thresholds that are used in `im2bw` to obtain thresholded images. All the images involved in this process, including the original image, are then displayed.

```
% Example 14.1 Pixel-based segmentation
%
.....input image and convert to double.......
%
h = fspecial('gaussian',14,2);       % Gaussian filter
I_f = imfilter(I,h,'replicate');     % Filter image
%
I_edge = edge(I_f,'canny',0.3);      % Find edge points
% Complement edge points and mask using multiplication
I_rem = I_f .* ~ I_edge
    ...... Display images and histograms, new figure.....
%
% Get thresholds
t1 = graythresh(I);                  % Use minimum variance
t2 = graythresh(I_f);                % (Outso's) method
BW1 = im2bw(I,t1));                  % Threshold images
BW2 = im2bw(I_f,t2));
......display thresholded images.......
```

**Results**

The results have been shown previously in Figures 14.2 and 14.3 and the improvement in the histogram and threshold separation has been mentioned. The edge image and its complement used as a mask are shown in Figure 14.6. While the change in the histogram is fairly small (Figure 14.2), it does lead to a reduction in artifacts in the thresholded image as shown in Figure 14.3. This small improvement can be quite significant in some applications. The methods for removing the small artifacts remaining will be described in Section 14.5.

## 14.3 Continuity-Based Methods

Continuity-based approaches look for similarity or consistency in the search for feature elements. These approaches can be very effective in segmentation tasks, but they all tend to reduce edge definition. This is because they are based on neighborhood operations that operate on a local area and blur the distinction between edge and feature regions. The larger the neighborhood used, the more poorly edges will be defined. Increasing neighborhood size usually improves the power of any given continuity-based operation, setting up a compromise between identification ability and edge definition.

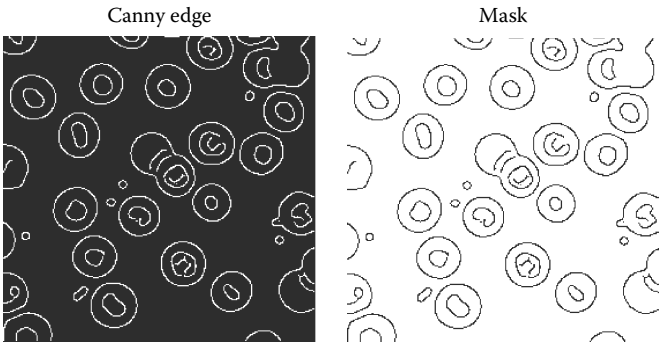


Figure 14.6 The BW image of edges produced by the MATLAB `edge` routine using the Canny filter (left). The inverted image, right, is used to mask (i.e., remove) boundaries in the cell image as seen in Figure 14.2. Eliminating these intermediate boundary values slightly improves threshold-based segmentation (Figure 14.3).

One simple continuity-based technique is lowpass filtering. Since a lowpass filter is a sliding neighborhood operation that takes a weighted average over a region, it enhances consistent features. Figure 14.7 shows histograms of the image in Figure 14.5 before and after filtering with a $10 \times 10$ Gaussian lowpass filter (alpha = 1.5).

After lowpass filtering, the two regions are evident in the histogram (Figure 14.7) and the boundary found by minimum variance results in perfectly isolated segments as shown in Figure 14.8. The thresholding uses the same minimum variance technique in both Figures 14.5 and 14.8 and the improvement brought about by simple lowpass filtering is remarkable.

Image features related to *texture* can be particularly useful in segmentation. Figure 14.9 shows three regions that have approximately the same average intensity values, but are readily distinguished visually because of differences in texture. Several neighborhood-based operations can be used to distinguish textures: the small-segment Fourier transform, local variance (or standard deviation), the *Laplacian* operator, the *range* operator (the difference between maximum and minimum pixel values in the neighborhood), the *Hurst* operator (maximum difference as a function of pixel separation), and the *Haralick* operator (a measure of distance moment). Many of these approaches are directly supported in MATLAB or can be implemented using the `nlfilter` routine described in Chapter 12. Example 14.2 attempts to separate the three regions shown in Figure 14.9 by applying one of these operators to convert the texture pattern into a difference in intensity that can then be separated using thresholding.

### 14.3.1 MATLAB Implementation

**EXAMPLE 14.2**

Separate out the three segments in Figure 14.9 that differ only in texture. Use one of the texture operators described above and demonstrate the improvement in separability through histogram plots. Determine the appropriate threshold levels for the three segments from the histogram plot.

**Solution**

Use the nonlinear range operator to convert the textural patterns into differences in intensity. The *range operator* is a sliding neighborhood procedure that sets the center pixel to the difference between the maximum and minimum pixel value with the neighborhood. Implement this operation using MATLAB's `nlfilter` routine with a $7 \times 7$ neighborhood. This neighborhood size was empirically found to produce good results. The three regions are then thresholded using

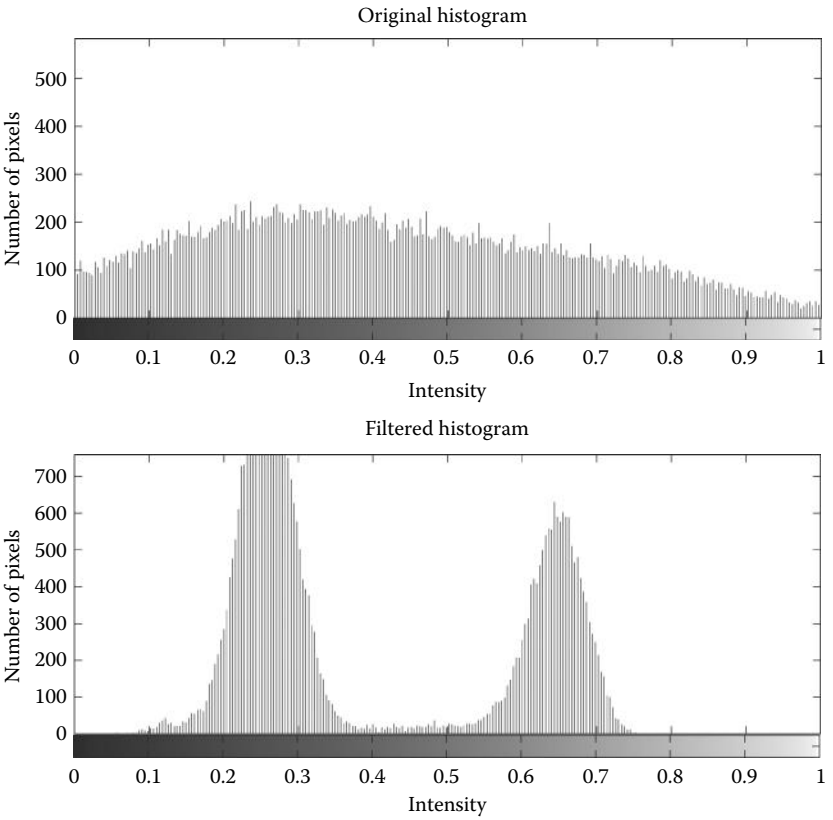


Figure 14.7 Histogram of the image shown in Figure 14.5 before (upper) and after (lower) lowpass filtering. Before filtering, the two regions overlap to such an extent that they cannot be identified in the histogram. After lowpass filtering, the two regions are evident and the boundary found by minimum variance is 0.45. The application of this boundary to the filtered image results in perfect separation as shown in Figure 14.8.

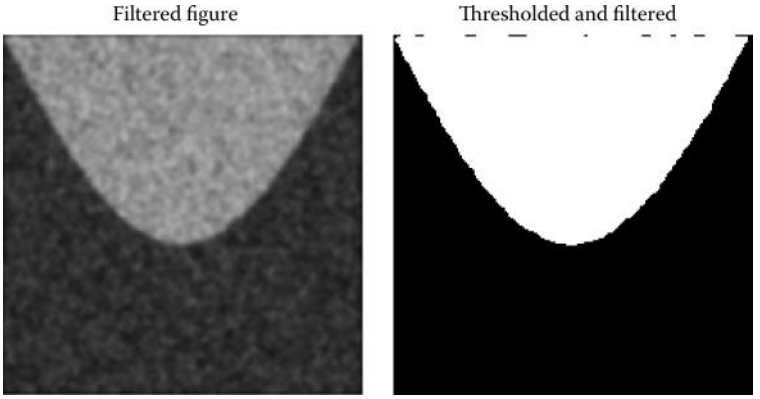


Figure 14.8 The same image as in Figure 14.5 after lowpass filtering. The two features can now be separated perfectly by thresholding as demonstrated in the right-hand image.

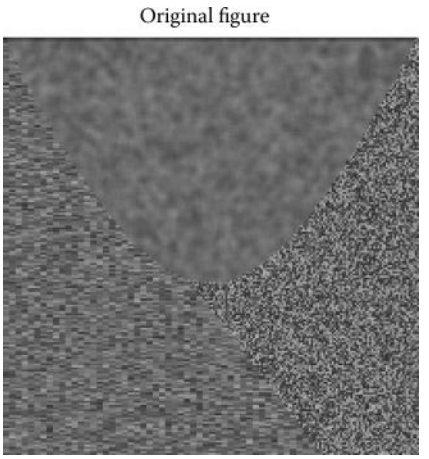


Figure 14.9 Image containing three regions having approximately the same intensity, but different textures. While these areas can be distinguished visually, separation based on intensity or edges will surely fail.

an empirically determined threshold of 0.22 and 0.55. The upper texture is segmented by inverting the BW image formed by the lower threshold and the right side texture is segmented by the upper threshold. The remaining texture can be found using a logical combination (the AND operation) applied to the two isolated images after inverting.

```
% Example 14.2 Segmentation of textures using the range operator
%
I = imread('texture3.tif');          % Load image and
I = im2double(I);             % Convert to double
%
range = inline('max(max(x)) − min(min(x))'); % Range operator
I_f = nlfilter(I,[7 7], range);
I_f = mat2gray(I_f);                 % Rescale intensities
   %
......Display range operation image and histograms.....
% Threshold and display segments, invert as needed, subplot as needed
BW1 = ~im2bw(I_f,.21);        % Isolate upper texture
BW2 = im2bw(I_f,.55);         % Isolate right-side texture
BW3 =~ BW1 & ~BW2;            % Isolate remaining texture
```

**Results**

The image produced by the range filter is shown in Figure 14.10 and a clear distinction in intensity level can now be seen between the three regions. This is also demonstrated in the histogram plots of Figure 14.11. The histogram of the original figure (upper plot) shows a single Gaussian-like distribution with no evidence of the three patterns.* After filtering, the three patterns emerge as three distinct distributions. Using this distribution, two thresholds were chosen at a minima between the distributions (at 0.21 and 0.55: the solid vertical lines in Figure 14.11) and the three segments isolated based on these thresholds. The upper and right side patterns are isolated using `im2bw` and the third pattern is found for a logical combination of the two.

* In fact, the distribution is Gaussian since the image patterns were generated by filtering an array filled with Gaussianly distributed numbers generated by `randn`.

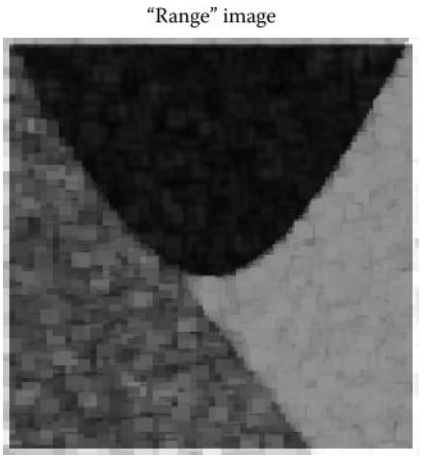


Figure 14.10 Texture pattern shown in Figure 14.9 after application of the nonlinear range operation. This operator converts the textural properties in the original figure into a difference in intensities. The three regions are now clearly visible as intensity differences and can be isolated using thresholding.

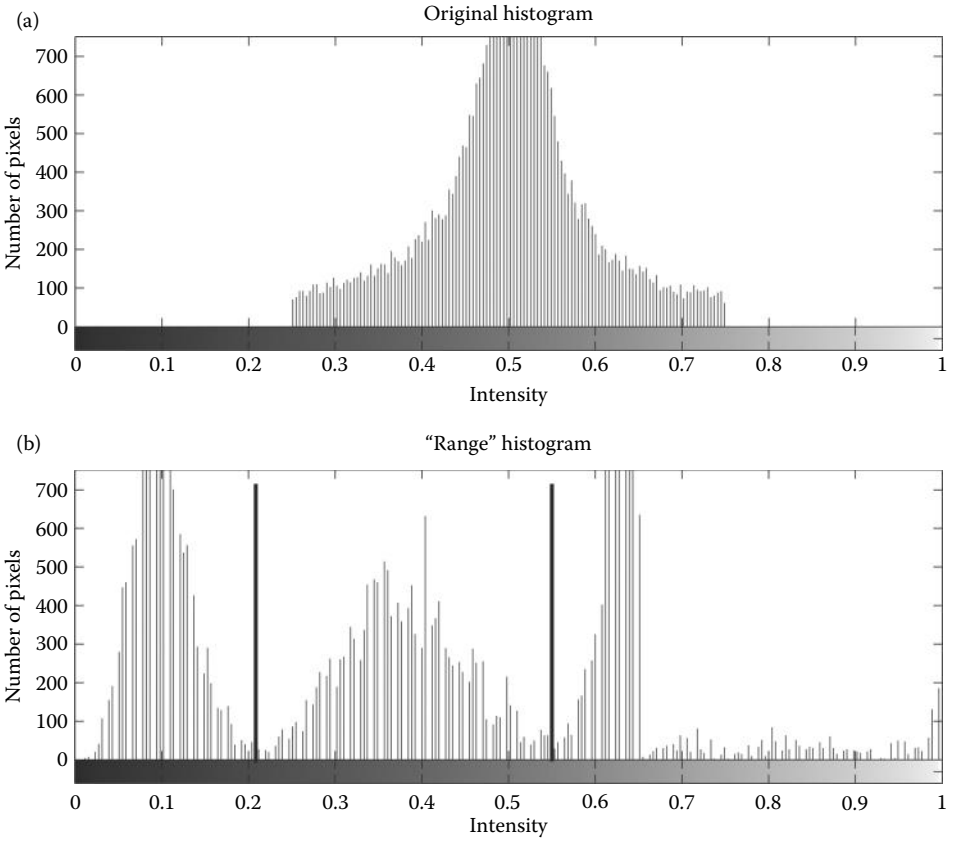


Figure 14.11 Histogram of the original texture pattern before (a) and after nonlinear filtering using the *range* operator (b). After filtering, the three intensity regions are clearly seen. The thresholds used to isolate the three segments are indicated.

Figure 14.12 Isolated regions of the texture pattern in Figure 14.9 produced by Example 14.2. Although there are some artifacts, the segmentation is quite good considering the original image. The methods for improving the segmented images are demonstrated in Section 14.5 and in the problems.

The three fairly well-separated regions are shown in Figure 14.12. A few artifacts remain in the isolated images and some of the morphological methods described in Section 14.5 could be used to eliminate or reduce these erroneous pixels. The separation can also be improved by applying lowpass filtering to the range image as demonstrated in one of the problems.

Occasionally, segments have similar intensities and textural properties, except that the texture differs in orientation. Such patterns can be distinguished using a variety of nonlinear operators that have orientation-specific properties. The local Fourier transform can also be used to distinguish orientation. Figure 14.13 shows a pattern with texture regions that are different only in terms of their orientation. To segment the three texture-specific features in this image, Example 14.3 uses a direction-specific operator followed by a lowpass filter that improves separation.

**EXAMPLE 14.3**

Isolate segments from a texture pattern that includes two patterns with the same textural statistical properties except for orientation. Also plot histograms of the original image and the image after application of the nonlinear filter.

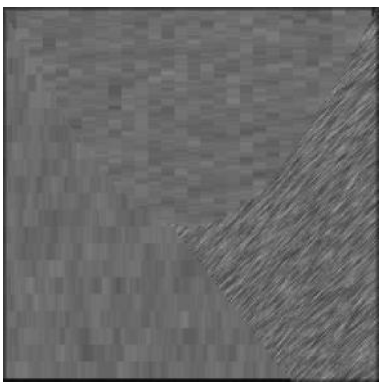

Figure 14.13 Textural pattern used in Example 14.3. The upper and left diagonal features have similar statistical properties and would not be separable with a standard range operator. However, they do have different orientations. As in the previous example, all three features have the same average intensity.

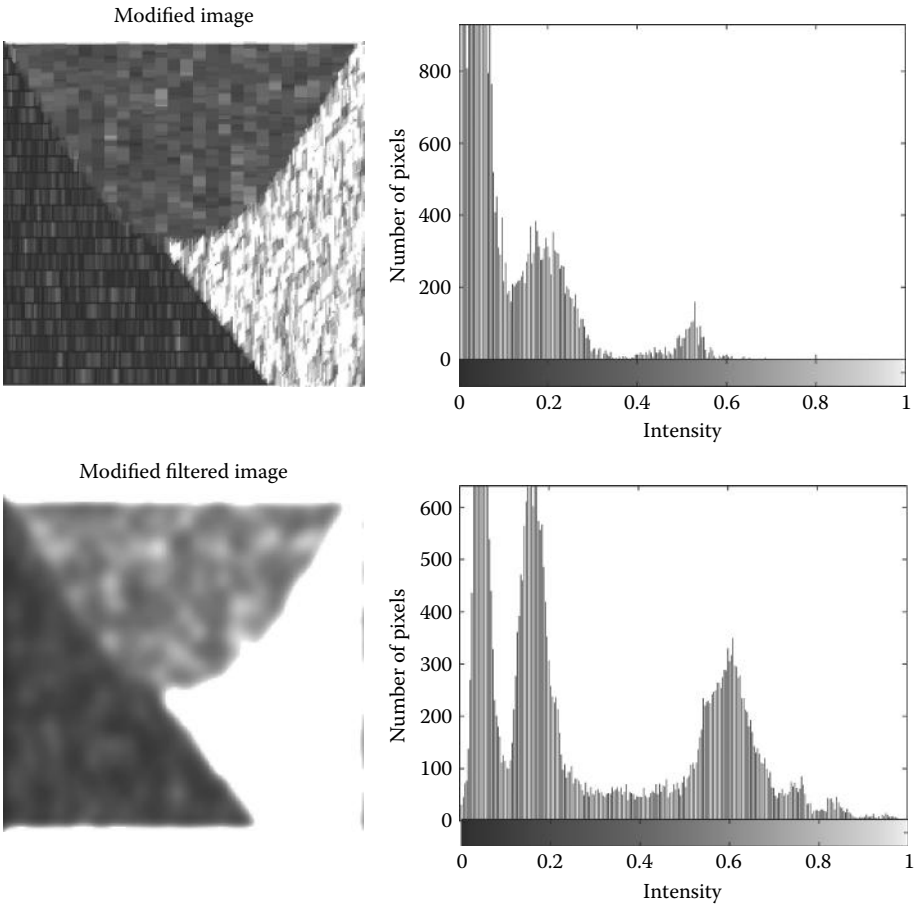


Figure 14.14 Images produced by the application of a directional range operator applied to the image in Figure 14.13 before (upper) and after (lower) lowpass filtering. The histograms demonstrate the improved separability of the filter image showing deeper minima in the filtered histogram.

### Solution

Because of the similarity in the statistical properties of the vertical and horizontal patterns, the standard range operator used in Example 14.2 will not provide good separation. Instead, we apply a filter that has directional sensitivity. A Sobel or Prewitt filter can be used, followed by the range or similar operator, or the operations can be done in a single step by using a directional range operator. The choice made in this example is to use a horizontal range operator implemented with `nlfilter`. This is followed by a strong lowpass filter ($20 \times 20$ Gaussian, alpha = 4) to improve separation by smoothing over intensity variations. Two segments are then isolated using standard thresholding. As in Example 14.2, the third segment is constructed by applying logical operations to the other two segments.

```
% Example 14.3 Analysis of texture pattern having similar textural
% characteristics but with different orientations..
%
I = imread('texture4.tif');   % Load texture
I = im2double(I);             % Convert to double
%
```

Figure 14.15 Isolated segments produced by thresholding the lowpass filtered image in Figure 14.14. The rightmost segment was found by applying logical operations to the other two images.

```
% Define filters and functions
range = inline('max(x) – min(x)');   % Range function
b_lp = fspecial('gaussian', 20, 4);  % Gaussian filter
I_nl = nlfilter(I, [9 1], range);    % Directional filter
I_h = imfilter(I_nl*2, b_lp);        % Scale image and filter
.......Display images and histograms........
%
BW1 = ~im2bw(I_f,.1);                % Isolate upper texture
BW2 = im2bw(I_f,.29);                % Isolate right-side texture
BW3 =~ BW1 & ~BW2;                   % Isolate remaining texture
.......Display image segments.......
```

**Results**

The image and histogram produced by the horizontal range operator with, and without, lowpass filtering are shown in Figure 14.14. The range operation produces a very dark image and it is multiplied by a scaling factor for better viewing; however, the BW masks were obtained by applying `im2bw` to the unscaled images. Note the improvement in separation produced by the lowpass filtering as indicated by the better defined peaks in the histogram. The thresholded images are shown in Figure 14.15. As in Example 14.3, the separation is not perfect, but it is quite good considering the challenges posed by the original image. Again, the separation could be further improved by morphological operations.

## 14.4 Multithresholding

The results of several different segmentation approaches can be combined either by adding the images together or, more commonly, by first thresholding the images into separate binary images and then combining them using logical operations. The AND, OR, or NOT ( ~ ) operator is used depending on the characteristics of each segmentation procedure. If each procedure identifies all the segments and also includes nondesired areas, the AND operator is used to reduce artifacts. An example of the use of the AND and NOT operations is given in Examples 14.2 and 14.3 where one segment is found using the inverse of a logical AND of the other two segments. Alternatively, if each procedure identifies some portion of the segment, then the OR operator is used to combine the various portions. This approach is illustrated in Example 14.4 where the first two, then three, thresholded images are combined to improve segment identification. The structure of interest is a cell that is shown on a gray background. Threshold levels above and below the gray background are combined (after one is inverted) to provide improved isolation. Including a third binary image obtained by thresholding a textured image further improves the identification.

**EXAMPLE 14.4**

Isolate the cell structures from the image of a cell shown in Figure 14.16.

**Solution**

Since the cell is projected against a gray background, it is possible to isolate some portions of the cell by thresholding above and below the background level. After inversion of the lower thresholded image (the one that is below the background level), the images are combined using a logical OR. Since the cell also shows some textural features, a textured image is constructed by taking the regional standard deviation. This textured image is then filtered with a lowpass filter ($20 \times 20$ Gaussian, alpha = 2) and is shown in Figure 14.16, right hand image. After thresholding, this texture-based image is also combined with the other two images.

```
% Example 14.4 Analysis of the image of a cell using combined
% texture and intensity information
%
I = imread('cell.tif');              % Load cell image
I = im2double(I);                    % Convert to double
%
b = fspecial('gaussian', 20, 2);     % Gaussian filter
%
I_std = (nlfilter(I,[3 3], 'std2'))*10; % Std operation
I_lp = imfilter(I_std, b);           % Apply filter
%
......Display original and filtered images, new figure......
%
BW_th = im2bw(I,.5);                 % Threshold image
BW_thc = ~im2bw(I,.42);              % and its complement
BW_std = im2bw(I_std,.2);            % Thresh. texture image
BW1 = BW_th | BW_thc;                % Combine two images
BW2 = BW_std | BW_th | BW_thc;       % Combine all images
    .......Display BW images and title...........
```

The original and texture/filtered images are shown in Figure 14.16. Note that the texture image has been scaled up by a factor of 10 to bring it within a nominal image range. The intensity

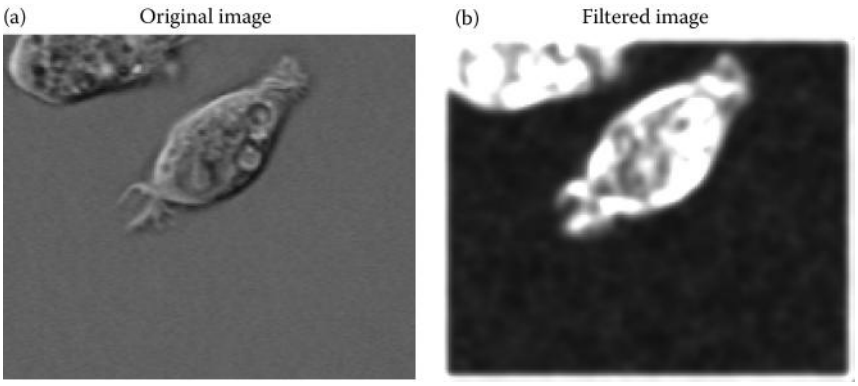


Figure 14.16 Image of cells (a) on a gray background. The textural image (b) was created based on local variance (standard deviation) followed by lowpass filtering and shows somewhat more definition. (Cancer cell from rat prostate, courtesy of Alan W. Partin, Johns Hopkins University School of Medicine.)

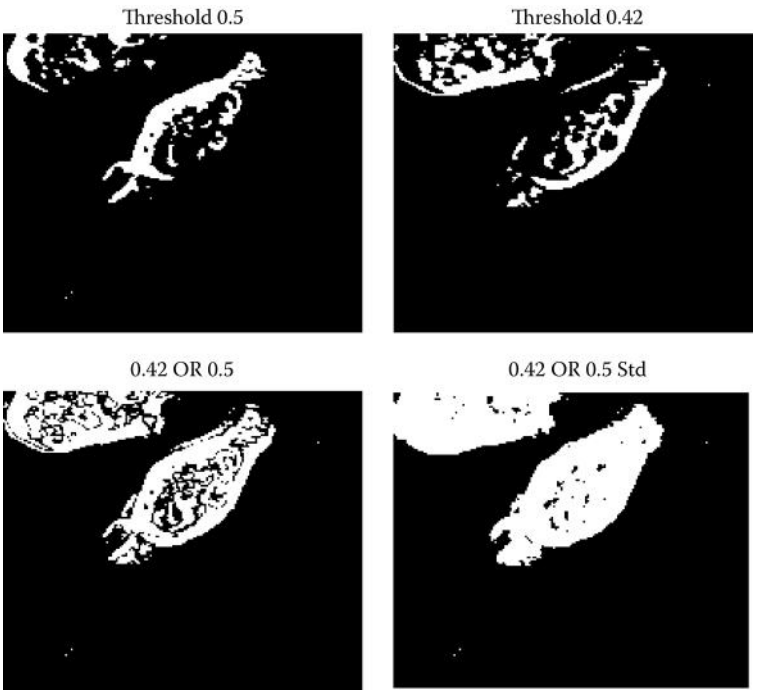


Figure 14.17 Isolated portions of the cells shown in Figure 14.16 (left image). The upper images were created by thresholding the intensity at the levels indicated. The lower left image is a combination (logical OR) of three images: the upper two BW images and a thresholded texture-based image.

thresholded images are shown in Figure 14.17 (upper images; the upper right image has been inverted) and the thresholds are shown. These images are ORed in the lower left image. The lower right image shows the OR of both thresholded images with a thresholded texture image. This method of combining images can be extended to any number of different segmentation approaches.

## 14.5 Morphological Operations

Morphological operations have to do with processing *shapes*. In this sense, they are both continuity based and may also operate on edges. In fact, morphological operations have many image-processing applications in addition to segmentation and they are well represented and supported in the MATLAB Image Processing Toolbox.

The two basic morphological operations are *dilation* and *erosion*. In dilation, the rich get richer, and in erosion, the poor get poorer. Specifically, in dilation, the center or active pixel is set to the maximum of its neighbors, and in erosion, it is set to the minimum of its neighbors. Since these operations are often performed on binary images, dilation tends to expand edges, borders, or regions, while erosion tends to decrease or even eliminate small regions. Obviously, the size and shape of the neighborhood used will have a very strong influence on the effect produced by either operation.

The two processes can be done in tandem over the same area. Since both erosion and dilation are nonlinear operations, they are not invertible transformations, that is, one followed by the other will not generally result in the original image. If erosion is followed by dilation, the operation is termed *opening*. If the image is binary, this combined operation will tend to remove small

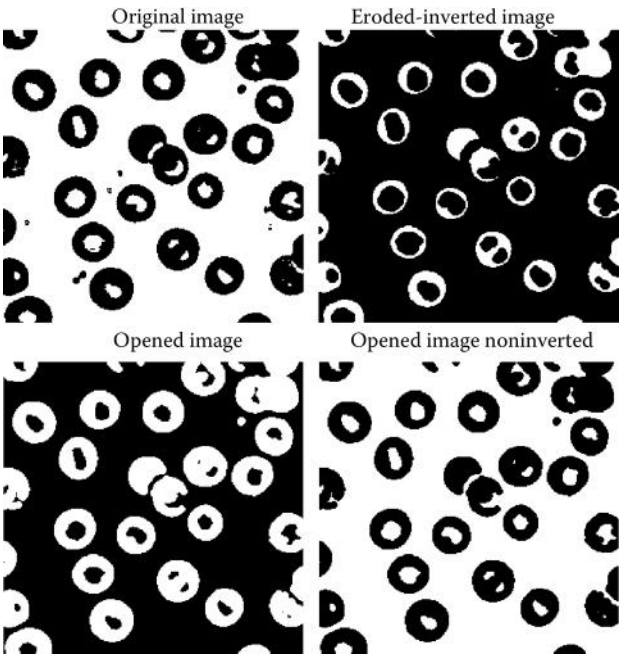


Figure 14.18 Example of the *opening* operation to remove small artifacts. Note that the final image has fewer background artifacts, but now one of the cells has a gap in the wall.

objects without changing the shape and size of larger objects. Basically, the initial erosion tends to reduce all objects, but some of the smaller objects will disappear altogether. The subsequent dilation will restore those objects that were not completely eliminated by erosion. If the order is reversed and dilation is performed first followed by erosion, the combined process is called *closing*. Closing connects objects that are close to each other, tends to fill up small holes, and smooths an object's outline by filling small gaps. As with the more fundamental operations of dilation and erosion, the size of objects removed by opening or filled by closing depends on the size and shape of the neighborhood that is selected.

An example of the opening operation is shown in Figure 14.18, including the erosion step. This is applied to the blood cell image after thresholding, the same image originally shown in Figure 14.3 (left side). Since we wish to eliminate black artifacts in the background, we first invert the image and erode as shown in Figure 14.18, upper right. As can be seen in the final, opened image, there is a reduction in the number of artifacts seen in the background, but now, there is also a gap created in one of the cell walls. The opening operation would be more effective on the image in which intermediate values were masked out (Figure 14.3, right side).

Figure 14.19 shows an example of closing applied to the same blood cell image. Again, the operation is performed on the inverted image. This operation tends to fill the gaps in the center of the cells, but it also fills in gaps between the cells. A much more effective approach to filling holes is to use the `imfill` routine described in Section 14.5.1.

Other MATLAB morphological routines provide local maxima and minima, and allow for manipulating the image's maxima and minima, which implement various fill-in effects.

### 14.5.1 MATLAB Implementation

The erosion and dilation could be implemented using the nonlinear filter routine `nlfilter`, although this routine limits the shape of the neighborhood to a rectangle. The MATLAB

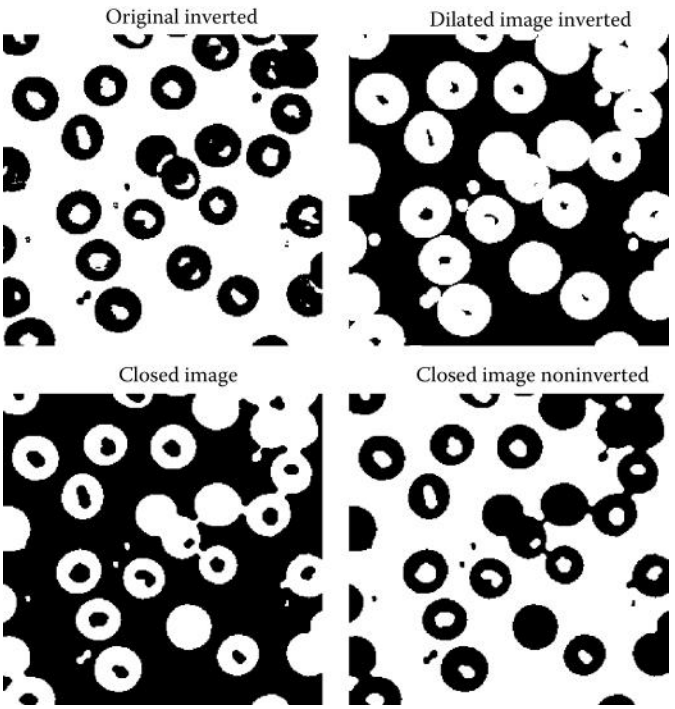


Figure 14.19 Example of *closing* to fill gaps. In the closed image, some of the cells are now filled, but some of the gaps between cells are also filled.

routines `imdilate` and `imerode` provide for a variety of neighborhood shapes and are much faster than `nlfilter`. As mentioned above, opening consists of erosion followed by dilation; closing is the reverse. MATLAB also provides routines for implementing these two operations in one statement.

To specify the neighborhood used by all these routines, MATLAB uses a *structuring element*.* A structuring element is defined by a binary array, where the 1s represent the neighborhood and the 0s are irrelevant. This allows for easy specification of neighborhoods that are nonrectangular: indeed, that can have any arbitrary shape. In addition, MATLAB makes a number of popular shapes directly available.

The routine to specify the structuring element is `strel` and is called as

```
structure = strel(shape, NH, arg);
```

where `shape` is the type of shape desired, `NH` usually specifies the size of the neighborhood, and `arg` is an argument that applies to `shape`. If `shape` is `'arbitrary'`, or simply omitted, then `NH` is an array that specifies the neighborhood in terms of the neighborhoods as described above. The prepackaged shapes include

```
'disk'          a circle of radius NH (in pixels)
'line'          a line of length NH and angle arg in degrees
```

* Not to be confused with a similar term, *structural unit* is used in the beginning of this chapter. This refers to an object of interest in the image.

```
'rectangle'     a rectangle where NH is a two-element vector specifying
                rows and columns
'diamond'       a diamond where NH is the distance from the center to each
                  corner
'square'        a square with linear dimensions NH
```

For many of these shapes, the routine `strel` produces a "decomposed" structure that runs significantly faster.

Given a structure designed by `strel`, the statements for dilation, erosion, opening, and closing are

```
I1 = imdilate(I, structure);
I1 = imerode(I, structure);
I1 = imopen(I, structuure);
I1 = imclose(I, structure);
```

where `I1` is the output image, `I` is the input image, and `structure` is the neighborhood specification given by `strel` as described above. In all cases, `structure` can be replaced by an array specifying the neighborhood as ones, bypassing the `strel` routine. In addition, `'imdilate'` and `'imerode'` have optional arguments that provide packing and unpacking of the binary input or output images.

**EXAMPLE 14.5**

Apply opening and closing to the thresholded blood cell images of Figure 14.3 in an effort to remove small background artifacts and to fill holes. Use a circular structure with a diameter of four pixels.

**Solution**

With the use of MATLAB routines, these morphological operations are straightforward. Since opening and closing are performed on BW images, we begin by using thresholding of the original image using the minimum variance method. This is followed by erosion and dilation to perform the opening operation. After the opened and intermediate image (i.e., the eroded image) are displayed, the closing operation is applied to the thresholded image by first dilating, then eroding the image, and the resulting images are displayed. Of course, both operations could have been performed with a single MATLAB command, but then the intermediate images would not have been available for display.

```
% Example 14.5 Demonstration of morphological opening to eliminate
% small artifacts and of morphological closing to fill gaps
%
I = imread('blood1.tif');     % Get image and threshold
I = im2double(I);
BW = ~im2bw(I,graythresh(I)); % Threshold image (Outso's method)
%
SE = strel('disk',4);         % Define structure
BW1= imerode(BW,SE);          % Opening operation: erode
BW2 = imdilate(BW1,SE);       % then dilate
%
    .......display images.....
%
BW3= imdilate(BW,SE);         % Closing operation:
BW4 = imerode(BW3,SE);        % dilate, then erode
    .......display images.....
```

### Results

This example produced the images shown in Figures 14.18 and 14.19. The next example shows how these morphological operations can be used to improve the segmentation produced by other methods. This example applies opening to the cell images in Figure 14.17 and to one of the textured images in Figure 14.14. The problems also address improving the segmentation of the textured images shown in Figure 14.12.

### EXAMPLE 14.6

Apply an opening operation to remove the dark patches seen in the thresholded cell image of Figure 14.17. Also remove *both* the black and white specks from the texture-segmented image of Figure 14.12 (right side).

### Solution

The opening operation is used in both figures since it tends to eliminate small objects. The opening operation acts on activated (i.e., white) pixels; so, it is necessary to invert the image using the NOT operation before opening to remove the black artifacts. After the opening operation, the image is inverted again (reinverted) to produce an image similar to the original.

In the textured image of Figure 14.14, both the light and dark specks are to be removed. The latter can be done by performing the opening operation on the inverted image as above, then, after reinverting, performing a second opening operation to eliminate the white specks.

```
% Example 14.6 Use opening to remove the dark patches in the
% thresholded cell image of Figure 14.17 and to
% remove both white and black specks from the
% texture image on the right side of Figure 14.14
%
load Ex14_4_data;            % Get cell image(BW2)
SE = strel('square',5);      % Define structure:
BW1= ~imopen(~BW2,SE);       % Opening/inverted operation
    .......Display images, new figure......
%
load Ex14_2_data;            % Repeat for texture BW image
BW = ~imopen(~BW3,SE);       % Opening/inverted operation
BWA = imopen(BW,SE);         % Opening operation
.......Display images......
```

### Results

Applying the opening operation to the inverted cell image using a $5 \times 5$ square structural element results in the elimination of all the dark patches within the cells as seen in Figure 14.20 (right). The size (and shape) of the structural element controls the size of the artifact removed and no attempt is made to optimize its shape. The size is set here as the minimum that would still remove all the dark patches.

As shown in Figure 14.21 (center), the opening operation applied to the texture-segmented image eliminates the dark specks from the white background,* but not the light specks in the isolated feature. A second opening applied to the center image removes most of these white specks (right-hand image). To use that as a segmentation mask to isolate the right-hand feature, the image would be inverted; so, the right-hand feature is white while the rest of the image is black. The original image would then be multiplied by this BW image.

* A dark speck in the corner remains left over from the border artifact. A larger structure element would be required to remove this artifact.

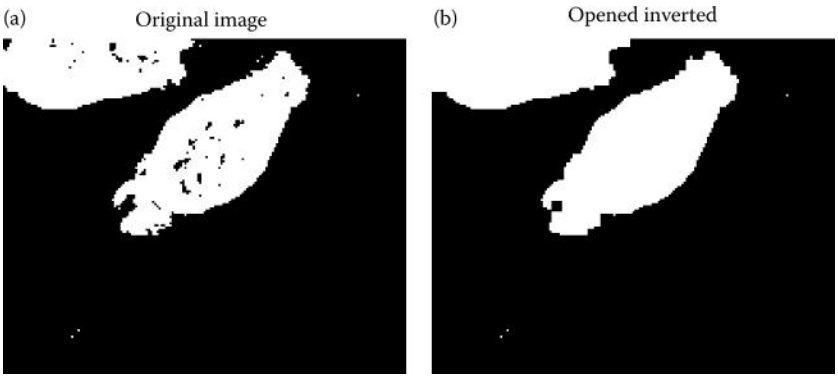


Figure 14.20 The opening operation performed on the thresholded cell image (a) eliminates the dark artifacts from the interior of the cells. (b) A $5 \times 5$ rectangular structure is used.

MATLAB morphology routines also allow for manipulation of maxima and minima in an image. This is useful for identifying objects and for filling. Of the many other morphological operations supported by MATLAB, only the `imfill` operation is described here. This operation begins at a designated pixel and changes the connected background pixels (0 s) to foreground pixels (1 s), stopping only when a boundary is reached. For grayscale images, `imfill` brings the intensity levels of the dark areas that are surrounded by lighter areas up to the same intensity level as the surrounding pixels. (In effect, `imfill` removes regional minima that are not connected to the image border.) The initial pixel can be supplied to the routine or can be obtained interactively. Connectivity can be defined as either *four connected* or *eight connected*. In four connectivity, only the four pixels bordering the four edges of the pixel are considered, while in eight connectivity, all pixels that touch the pixel are considered, including those that touch only at the corners.

The basic `imfill` statement is

```
I_out = imfill(I,[r c],con);
```

where `I` is the input image, `I _ out` is the output image, `[r c]` is a two-element vector specifying the beginning point, and `con` is an optional argument that is set to 8 for eight connectivity

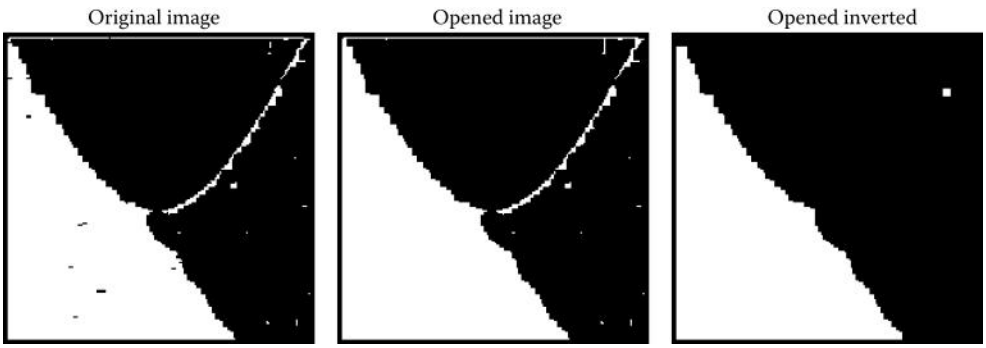


Figure 14.21 Applying opening to an inverted original image then inverting produces the center image which is free of dark speck in the white section of this image; however white artifacts remain in the dark section. Reapplying opening to the center image remove all but on white speck as shown in the right hand image.

(four connectivity is the default). (See the help file to use `imfill` interactively.) A special option of `imfill` is available specifically for filling holes. If the image is binary, a hole is a set of background pixels that cannot be reached by filling in the background from the edge of the image. If the image is an intensity image, a hole is an area of dark pixels surrounded by lighter pixels. To invoke this option, the argument following the input image should be `'holes'`. Example 14.7 shows the operation performed on the blood cell image by `imfill` using the `'holes'` option. This operation is followed by thresholding and opening to produce a good segmentation of the blood cell images.

**EXAMPLE 14.7**

Load the image of blood cells (`blood.tif`) and apply `imfill` to darken the center of the cells. Then apply thresholding to produce a mask of the cells followed by opening to remove any artifacts.

```
% Example 14.7 Demonstration of imfill with option 'holes' on a
% grayscale image followed by thresholding and opening to
% produce a segmented image of the blood cells.
%
I = imread('blood1.tif');     % Get image and
I = im2double(I);             % Convert to double
I1 = imcomplement(I);         % Invert original image
%
I2 = imfill(I1,'holes');
% Threshold and open the filled image
BW = im2bw(I2,.5);
SE = strel('disk',5);    % Define structure:
BW1 = imopen(BW,SE);
```

**Results**

The image produced by `imfill` is shown in Figure 14.22. This routine is applied to the grayscale image, and portions of the image that are enclosed by the circular cell walls become black. This image is then thresholded at 0.5 (Figure 14.23, left image). Opening is then used to remove the small white artifacts (Figure 14.23, right image), resulting in a well-segmented image of the cells.

The image in Figure 14.23 can be used as a mask to isolate only the cells. This is done simply by multiplying (point by point) the mask image in Figure 14.23 by the original blood cell image. The result is shown in Figure 14.24. This image has been scaled by 2 to enhance the cell interiors.

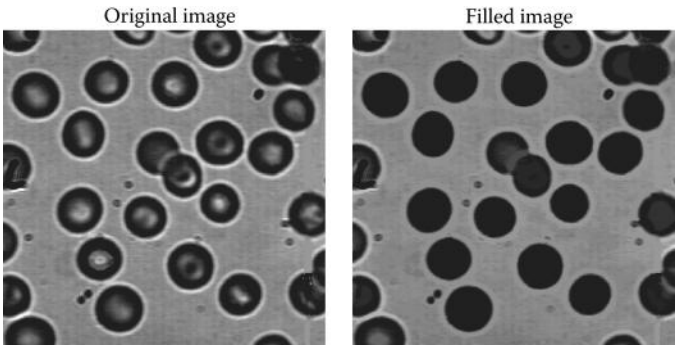


Figure 14.22 Hole-filling operation produced by `imfill`. Note that the edge cells and the overlapped cell in the center are not filled since they are not actually holes.

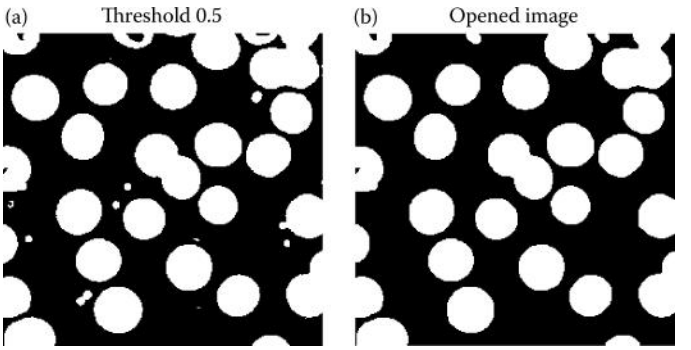


Figure 14.23 Filled image in Figure 14.22 after thresholding (a) and opening (b) to remove the artifacts.

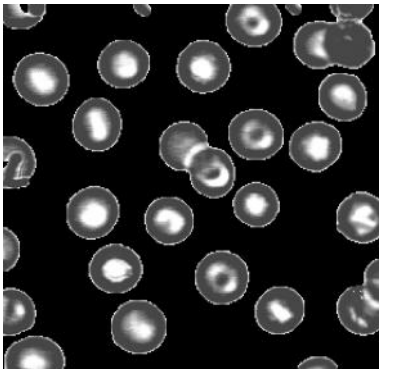

Figure 14.24 Image of the blood cells after the background is removed by masking (i.e., multiplying) the original image using the black-and-white image in Figure 14.23 (right). The image has been scaled (multiplied) by 2.0 to improve the visual appearance of the cell interiors.

## 14.6 Edge-Based Segmentation

Historically, edge-based methods were the first set of tools developed for segmentation. To move from edges to segments, it is necessary to group edges into chains that correspond to the structural boundaries, that is, the sides of structural units. The approaches vary in how much prior information they use, that is, how much is used of what is known about the possible shape. False edges and missed edges are two of the more obvious and more common problems associated with this approach.

The first step in edge-based methods is to identify edges that then become candidates for boundaries. Some of the filters presented in Chapter 13 perform edge enhancement, including the Sobel, Prewitt, Roberts, Log, and Canny filters. These edge detection filters can be divided into two classes: filters that use the gradient of the first derivative, and filters that use zero crossings of the second derivative. The Sobel, Roberts, and Prewitt filters are all examples of the derivative-based filter and differ in how they estimate the first derivative. The Log (that actually stands for "Laplacian of Gaussian") filter takes the spatial second derivative and is used in conjunction with zero crossings to detect edges. These filters are used with MATLAB's `edge` routine described below that sets thresholds that can be automatically adjusted.

The Canny filter, the most advanced edge detector filter supported by MATLAB's `edge` routine, is a type of zero-crossing filter. The approach finds edges by looking for local maxima of the

gradient of the image. The gradient is calculated using the derivative of a Gaussian filter, similar to that used in the Log filter. The method uses two thresholds to detect strong and weak edges and includes the weak edges in the output only if they are connected to strong edges. Therefore, this method is less likely than the others to be fooled by noise and is more likely to detect true weak edges. All these filters produce a binary output that can be a problem if a subsequent operation requires a graded edge image. These filter operations are explored in Example 14.8 and in the problems.

*Edge relaxation* is one approach used to build chains from edge candidate pixels. This approach takes into account the local neighborhood: weak edges positioned between strong edges are probably part of the edge, while strong edges in isolation are likely to be spurious. The Canny filter incorporates a type of edge relaxation. Various formal schemes have been devised under this category. A useful method, described in Sonka et al. (1993), establishes edges between pixels (the so-called crack edges) based on the pixels located at the end points of the edge.

Another more advanced method for extending edges into chains is termed *graph searching*. In this approach, the endpoints (that can both be the same point in a closed boundary) are specified and the edge is determined based on minimizing some cost function. The possible pathways between the endpoints are selected from candidate pixels, those that exceed some threshold. The actual path is selected using an optimization technique to minimize the cost function. The cost function can include features such as the strength of an edge pixel and total length, curvature, and proximity of the edge to other candidate borders. This approach allows for a great deal of flexibility.

The methods briefly described above use local information to build up the boundaries of the structural elements. The details of these methods can be found in Sonka et al. (1993). MATLAB supports a few of such local operations as described in Section 14.6.2.

Model-based edge detection methods can be used to exploit prior knowledge of the structural unit. These are some of the most powerful segmentation tools and are the subject of much of the current research. If the shape and size of the image are known, then a simple matching approach based on correlation can be used (matched filtering). When the general shape is known, but not the size, a model-based method described in the next section can be used. Another approach is to use the underlying physical properties of the structure of interest to set constraints on the shape of the boundaries. For example, possible deformations of the heart are limited by the mechanical properties of the cardiac tissue. Using a model of a structure's mechanical properties to guide the edge detection task is another area of research interest.

### 14.6.1 Hough Transform

The *Hough transform* approach was originally designed for identifying straight lines and curves, but can be expanded to other shapes provided the shape can be described analytically. The basic idea behind the Hough transform is to transform the image into a parameter space that analytically defines the desired shape. Maxima in this parameter space then correspond to the presence of the desired image in image space.

For example, if the desired object is a straight line (the original application of the Hough transform), one analytic representation for this shape is $y = mx + b$,* and such shapes can be completely defined by a 2-D parameter space of $m$ and $b$ parameters. All straight lines in image space map to points in parameter space (also known as the *accumulator array* for reasons that will become obvious). Operating on a binary image of edge pixels, all possible lines through a given pixel are transformed into $m,b$ combinations, which then increment the accumulator array. Hence, the accumulator array accumulates the number of *potential* lines that could exist in the image. Any active pixel will give rise to a large number of possible line slopes, $m$, but only a limited number of $m,b$ combinations. If the image actually contains a line, then the accumulator element that

* This representation of a line will not be able to represent vertical lines since $m \to \infty$ for a vertical line. However, lines can also be represented in two dimensions using cylindrical coordinates, $r$ and $\theta$: $r = a \sin \theta / \sin(\theta - \varphi)$.

corresponds to that particular line's *m,b* parameters will have accumulated a large number of "hits," at least relative to all the other accumulator elements. The accumulator array is searched for maxima, or a number of suprathreshold locations, and these locations identify a line, or lines, in the image.

This concept can be generalized to any shape that can be described analytically, although the parameter space (i.e., the accumulator) may have to include several dimensions. For example, to search for circles, a circle can be defined in terms of three parameters, *a*, *b*, and *r*:

$$(y - a)^2 + (x - b)^2 = r^2 \tag{14.1}$$

where *a* and *b* define the center point of the circle and *r* is the radius. Hence, the accumulator space must be 3-D to represent *a*, *b*, and *r*.

### 14.6.2 MATLAB Implementation

The edge detection filters described above can be implemented using `fspecial` and `imfilter`, but MATLAB offers an easier implementation using the routine edge:

```
BW = edge(I, 'method', threshold, options);
```

where `BW` is the binary output image, `I` is the input image, and `'method'` specifies the type of filter. The method can be the name (no caps) of any of the filters mentioned above. The threshold argument is optional and if it is not specified (or empty), Outso's method is used to automatically determine the threshold. The Canny method calls for two thresholds, both of which can be determined automatically or can be entered as a vector. There are a few other options and they will be described when used. The behavior of these filters is compared on a single image in the next example.

**EXAMPLE 14.8**

Load the blood cell image and detect the cell boundaries using the three derivative-based filters, `sobel`, `prewitt`, and `roberts`. Also compare the two zero-crossing filters `log` and `canny`. Select the threshold empirically to be the highest possible and still provide solid boundaries.

```
% Example 14.8 and Figure 14.25 and 14.26
% Apply various edge detection schemes to the blood
% cell image
%
clear all; close all;
.......Load and convert 'blood1'tif'.........
imshow(I);figure;
% Apply the 5 filters to the cell image
[BW1,thresh1] = edge(I,'sobel',.13,'nothinning');
[BW2,thresh2]= edge(I,'roberts',.09,'nothinning');
[BW5,thresh3]= edge(I,'prewitt',.13,'nothinning');
[BW3,thresh4] = edge(I,'log',.004);
[BW4,thresh5] = edge(I,'canny',[.04 .08]);
.......Display and label images..........
```

**Results**

The original image and the results of the derivative-based filters are shown in Figure 14.25. All these filters used the option `'nothinning'` to make the lines thicker to improve the display. The filter thresholds were adjusted to give approximately the same level of boundary

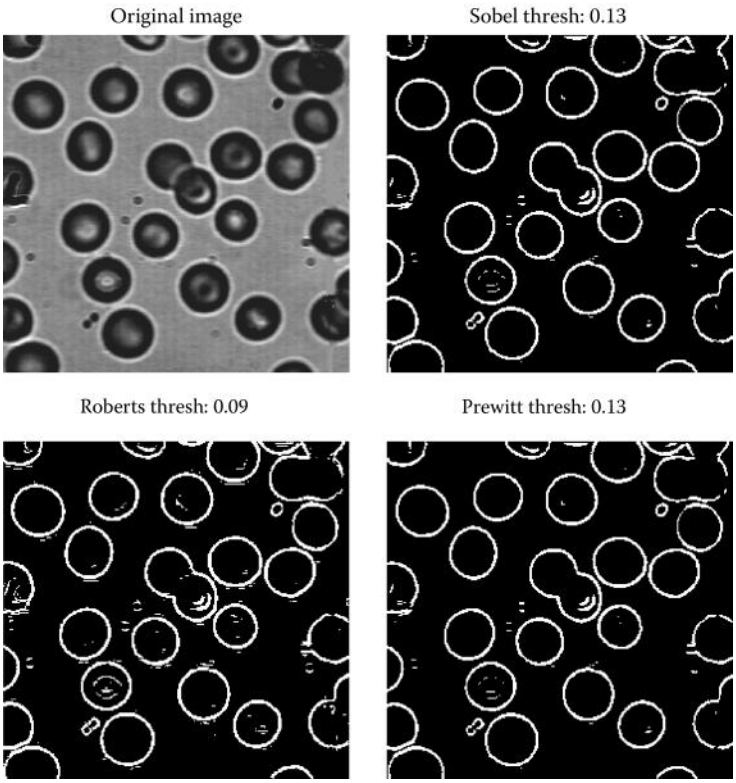


Figure 14.25 Blood cell edges determined from three derivative-based filters. Filter thresholds are indicated.

integrity. The performance of the three filters is comparable, although the Prewitt filter has slightly fewer or smaller artifacts.

The output of the two zero-crossing filters is shown in Figure 14.26. Again, filter thresholds were adjusted to give reasonably intact boundaries with minimal artifacts. These filters show two edges around each cell as they detect both the light-to-dark and dark-to-light transitions. This could be an advantage in other applications, but is not as useful if the goal is to segment the

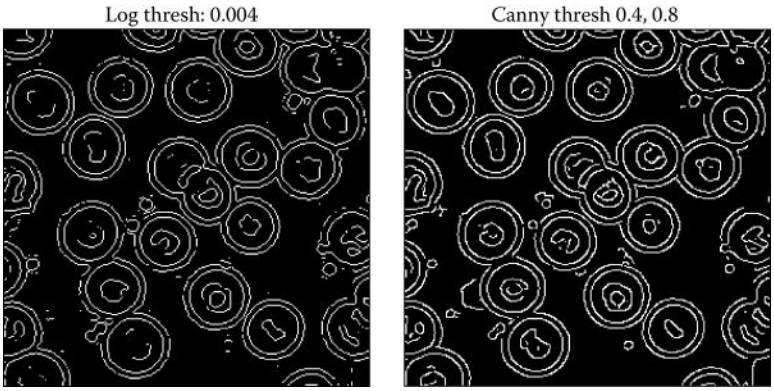


Figure 14.26 Blood cell edges determined by two filters based on zero crossing. The more advanced Canny filter that used two threshold criteria shows more complete boundaries.

blood cells. The more advanced Canny filter shows better edge identification and fewer artifacts than the Laplacian of Gaussian (Log) filter. Other operations on binary images are explored in the problems.

The Hough transform is supported by MATLAB image-processing routines, but only for straight lines. It is supported as the *Radon transform* that computes projections of the image along a straight line, but this projection can be done at any angle.* This transform produces a projection matrix that is the same as the accumulator array for a straight line Hough transform when expressed in cylindrical coordinates.

The Radon transform is implemented by the statement

```
[R, xp] = radon(BW, theta);
```

where `BW` is a binary input image and `theta` is the projection angle in degrees, usually a vector of angles. If not specified, `theta` defaults to (1:179). The output `R` is the projection array where each column is the projection at a specific angle. The second output, `xp`, is a vector that gives the radial value of each row of `R`. Hence, maxima in `R` correspond to the positions (encoded as an angle and distance) in the image. An example of the use of `radon` to perform the Hough transformation is given in Example 14.9.

**EXAMPLE 14.9**

Find the strongest line in the image of the Saturn in `saturn1.tif`. Plot that line superimposed on the image.

**Solution**

First convert the image into an edge array using MATLAB's `edge` routine with the Canny filter. Apply the Hough transform (implemented for straight lines using `radon`) to the edge image to build an accumulator array. Find the maximum point in that array (using `max`), which will give theta, the angle perpendicular to the line, and the distance along that perpendicular line of the intersection. Convert that line into rectangular coordinates, then plot the line superimposed on the image. Finding the maximum point in the transform array and converting it into rectangular coordinates are the most difficult aspects of the problem.

```
% Example 14.9 The Hough Transform implemented using radon
% to identify the strongest line in an image.
%
radians = 2*pi/360;            % Degrees to radians
I = imread('saturn1.tif');     % Get image of Saturn
BW = edge(I,'canny',.05);      % Threshold image
[R,xp] = radon(BW,theta);      % Hough transform
%
    ........Display original and thresholded images.....
[~, c] = max(max(R));          % Find maximum element
[~, r] = max(R(:,c));
[ri ci] = size(I);             % Size of image array
[ra ca] = size(R);             % and accumulator array
% Convert to rectangular coordinates
A = xp(r);                     % Get line displacement
phi = theta(c);                % and angle
m = tan((90-phi)*radians);     % Calculate slope and
b = A/cos((90-phi)*radians);   % intercept from
```

* The Radon transform is an important concept in CT as described in Chapter 15.

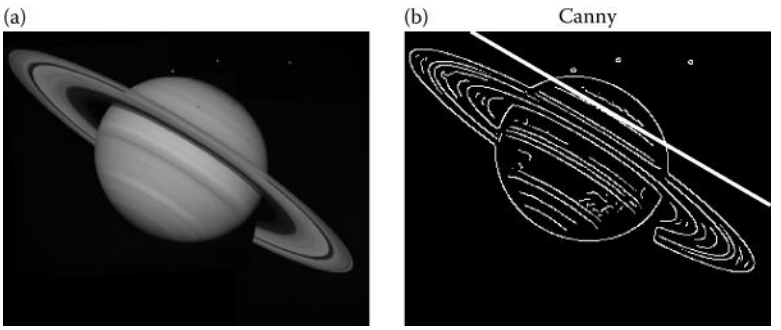


Figure 14.27 Normal (a) and thresholded (b) image of the Saturn with the dominant line found by the Hough transform. (The original image is a public domain image courtesy of NASA, Voyager 2 image, 08-24-1981.)

```
x = (0:ci);                    % basic trig.
y = mx + b;                    % Construct line
figure(fig1); subplot(1,2,2); hold on;
   plot(x,-y,'w');             % Plot line on Saturn
figure(fig2); hold on;
   plot(phi,A,'*b');           % Plot max accumulator point
   ......Labels.........
```

**Results**

This example produces the images shown in Figure 14.27. The broad white line superimposed on the edge image is the line found as the most dominant using the Hough transform. The location of the maximum coordinate is in the accumulator array (i.e., the parameter space) is indicated by a black * in the accumulator array (Figure 14.28). The location of the maximum point must be converted into a degree-and-offset position and then must be converted into rectangular coordinates to plot the line. Other points nearly as strong (i.e., bright) can be seen in the parameter array representing other lines in the image. Of course, it is possible to identify these lines as well by searching for maxima other than the global maximum. This is done in Problem 14.12.

## 14.7 Summary

Segmentation, the isolation of regions of interest, is often a critical first step in biomedical image analysis. In segmentation problems, the goal is to develop a "mask," a BW image that is 1.0 over the region of interest and 0.0 elsewhere. When this mask is multiplied by the original image, all features except for the region of interest are set to black (0.0). Sometimes, the task is so difficult that human intervention is required, but new techniques permit increasingly automated segmentation operations. A large number of basic and advanced segmentation operations exist. These approaches broadly fall into four categories: pixel-based, regional or continuity-based, edge-based, and morphological methods.

Pixel-based methods primarily involve separation by grayscale intensity and are the fastest and easiest to implement, but are also the least powerful. They operate on one element at a time and do not consider the larger aspects of the image; therefore, they are particularly susceptible to noise. Finding the best threshold to isolate a region is often aided by a plot of the number of pixels at a given intensity versus intensity, the intensity histogram. MATLAB routines that support threshold operations include an operation that automatically adjusts the threshold to minimize the total variance in intensity level on either side of the threshold boundary (Outso's method).

Regional methods identify similarities in the region of interest such as textural properties while edge-based methods search for differences that indicate boundaries. Filters, including nonlinear filters, are a mainstay of regional methods as they act on a group of pixels. Averaging or lowpass operations are the commonly used linear filters while nonlinear operations include the range, Laplacian, Hurst (maximum difference as a function of pixel separation), and Haralick operators.

Edge-based operations search for differences in image features to identify boundaries. These approaches also rely on filtering, but use derivative or derivative-like filters that enhance boundaries. More advanced methods involving multiple thresholds can be used to fill in gaps in the boundary. MATLAB's Canny filter is an example of such an advanced filter. Another advanced method uses an analytical description of the boundary shape and keeps track of all the shapes that meet the analytical criteria. MATLAB supports this procedure for straight line boundaries in the form of the Hough transform, developed for the analysis of CT imaging.

Morphological methods use information on shape to constrain or define the segmented image. Advanced morphological methods may include information on the biomechanics and biodynamics of the tissue of interest. Morphological operations supported by MATLAB include dilation, erosion, opening, and closing. These operations are applied to BW images. In dilation, the rich get richer so that areas surrounded by active pixels become active (i.e., set to 1.0). In erosion, the poor get poorer and active pixels are set to 0.0 if they have few active neighbors. Control of the size and shape of the region considered is a major design issue in the implementation of these operations. The two operations can be applied sequentially and, since they are nonlinear operations, the results are highly dependent on the sequence. In opening, erosion is followed by dilation; during the former, some points disappear altogether and thus are not restored by dilation. Opening is useful for removing small, isolated artifacts often caused by noise. In closing, dilation is followed by erosion: gaps that are filled by dilation remain that way after the subsequent erosion. Closing connects objects that are close to each other, fills small holes, and smooths boundaries.

Many challenging segmentation problems require the application of multiple approaches. In such cases, partial masks are created and combined using the logical AND/OR operations. Segmentation has received considerable attention by biomedical imagers and continues to progress with the development of new algorithms and more powerful, faster, hardware.

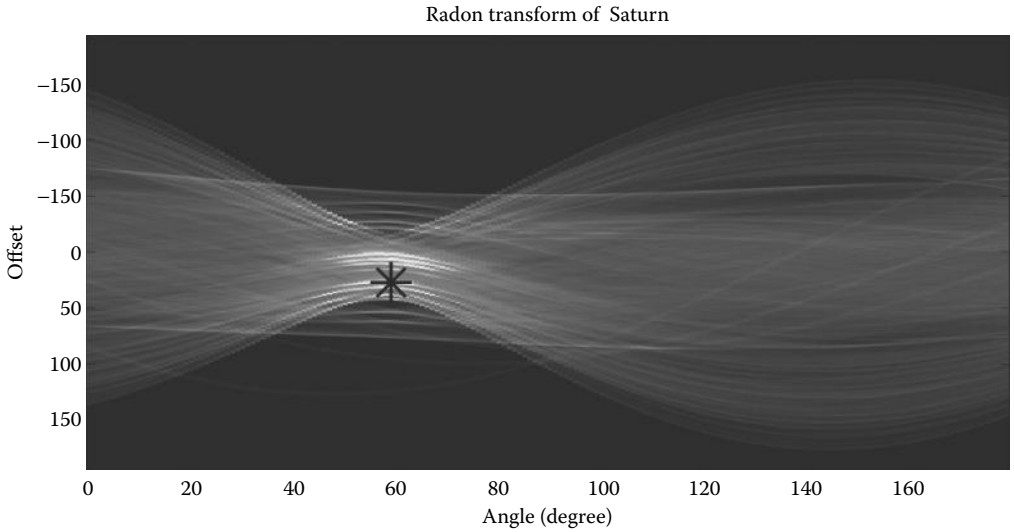


Figure 14.28 Accumulator array found from the Hough transform of the image of the Saturn in Example 14.9. The maximum point in this array provides the location of the strongest line and is indicated by an "*."

**PROBLEMS**

14.1 Load the bone marrow image in `bonemarr.tif`. Convert the image into a BW mask using `graythresh` to determine the threshold. Then construct a second mask with fewer artifacts by finding a threshold manually. You will find that the artifacts in the image found using `graythresh` can only be improved slightly before the smaller white segments are eliminated. Plot the original image and the two threshold images side by side.

14.2 Load the blood cell image in `blood1.tif`. Filter the image with two lowpass filters, both Gaussian $20 \times 20$, one having a low cutoff (an alpha of 0.5) and the other having a high cutoff (`alpha` $\geq 4$). Threshold the two filtered images using the minimum variance routine `graythresh`. Display the original and filtered images along with their histograms. Also display the two thresholded images. Note the improvement in artifact elimination with the strong filter. Regarding the histograms, also note the reduction in low-intensity pixels after strong lowpass filtering.

14.3 Repeat Problem 14.2 for the original image shown in Figure 14.5 and found in file `Fig14_5.tif`. As in Figure 12.8, there is a substantial improvement in the separation of peaks in the histogram with the stronger filter and a radical improvement in segmentation. The downside is that the border is less precisely defined in the heavily filtered image.

14.4 The Laplacian filter that calculates the second derivative can also be used to find edges. In this case, edges will be located where the second derivative is near zero. Load the image of the spine (`spine.tif`) and filter using the Laplacian filter obtained from the `fspecial` routine (use the default constant). Then, threshold this image. You will need to take a fairly low threshold (<0.02) since you are interested in the values near zero. Note the extensive tracing of subtle boundaries that this filter produces.

14.5 Load the image `texture3.tif` that contains three regions having the same average intensities but different textural patterns. Before applying the nonlinear range operator used in Example 14.2, preprocess with a Laplacian filter (`alpha` = 0.5). Apply the range operator as in Example 14.2 using `nlfilter`. Plot the original and "range" images along with their histograms. Threshold the range image to isolate the segments and compare with the figures in this book. [Hint: You may have to adjust the thresholds slightly, but you do not have to rerun the time-consuming range operator to adjust these thresholds. You can do that from the MATLAB command line.] You should observe a modest improvement over Example 14.2: one of the segments can now be perfectly separated.

14.6 Load the texture orientation image `texture4.tif`. Separate the segments by first preprocessing the image with a Sobel filter. Then apply a nonlinear filter using a standard deviation operation determined over a $2 \times 9$ pixel grid. Finally, postprocess the output image from the nonlinear filter using a strong Gaussian lowpass filter. (Note: You will have to multiply the output of the nonlinear filter by around 3.5 to get it into an appropriate range.) Plot the output image of the nonlinear filter and that of the lowpass filtered image and the two corresponding histograms. Separate the lowpass filtered image into three segments as in Examples 14.2 and 14.3. Use the histogram of the lowpass filtered image to determine the best boundaries for separating the three segments. Display the three segments as white objects (i.e., as segment masks).

14.7 Load the thresholded images of Figure 14.12 found as `BW1`, `BW2`, and `BW3` in file `Ex14_2_data.mat`. Invert the image in BW1 and apply opening to the inverted

image to eliminate as many points as possible in the upper field without affecting the lower field. Then use closing on the inverted image to try to blacken as many points as possible in the lower field without affecting the upper field. You should be able to whiten the upper field completely with opening and blacken the lower field completely with closing. You should pick the best structural element to accomplish the task. Plot the output of the two operations inverted so that the segmented section is white. [Hint: The spots you are trying to eliminate are small and round. You may need different structural elements for the upper and lower fields.]

14.8 As in Problem 14.7, load the thresholded images found as `BW1`, `BW2`, and `BW3` in file `Ex14_2_data.mat`. Apply opening to an inverted `BW3` to eliminate as many points as possible in the lower field without affecting the lower field. Then eliminate the artifacts in the upper field of the opened image in two different ways. Apply closing to the opened image and opening to an inverted opened image. You many need different structural elements for these different operations, but both approaches should completely eliminate the upper and lower artifacts from the image. Show the original image, the initial opened image, and the two artifact-free images. Invert the images as needed so that the lower diagonal field is white.

14.9 As in Problem 14.7, load the images from Figure 14.12 found in file `Ex14_2_data.mat`. Clear the artifacts in both portions of `BW2` in two ways: using opening applied twice (as in Problem 14.8) and closing applied twice. Invert the images as required to eliminate the artifacts and display with the right segment as white. For each approach, display the original and final images along with the intermediate image.

14.10 Load the image of the bacteria found in `bacteria.tif`. The objective of this problem is to segment the bacterial cell using a multistep process. First, apply the Canny edge filter with a primary threshold of 0.2. Then add this edge image to the grayscale image of the bacteria. Next, apply `imfill` to this combined image that will make the interior of all but one cell white. Finally, threshold this image at a very high level (0.99) to get the bacteria mask. One cell is not captured by this method because its border is not continuous. To capture this cell, erode the thresholded image using a fairly small structure. All the cells should now be captured by this method.

14.11 Load the image of the brain in `brain1.tif`. Convert that into double and use `mat2gray` to ensure proper intensity scaling. The objective is to segment the ventricles, the darker areas in the center of the brain. This is a multistep process somewhat similar to that used in Problem 14.10. Ultimately, the segmentation will rely on `imfill` and a careful separation based on a small intensity range; however, an important first step is to eliminate the bright areas around the outer edges of the brain.

To accomplish this, identify those bright regions using edge and a Canny filter with a fairly high threshold so that you detect only the outer edges of the brain. Then increase the size of those edges using dilation and subtract these highlighted images from the main image. (You could also invert the BW edge image and multiply the brain image by this mask.) For dilation, you should use a structural element that is just large enough to remove all the outer features from the subtracted (or masked) image.

If you then apply `imfill` to this modified image, it will outline the ventricles using a constant intensity level that you can isolate using two BW images generated with carefully selected thresholds. When combined appropriately, the ventricles will be highlighted in white. There will also be additional speckles in this image that can be removed using erosion. The same structural element used in dilation above

seems to work well. The result will be the segmented ventricles. Plot the final thresholded image (before erosion), the final segmented image, and the original image together. Also plot the dilated edge image, the original image after removing the outer features, and the filled image.

14.12 Modify Example 14.9 to plot the *fourth* strongest line in the Saturn image. Plot the original line, then plot the fourth strongest line on the same plot using a dashed line and/or a different color. Also plot the fourth largest point in the accumulator array using a different marker type and/or color. [Hint: Use a loop to set the accumulator entry corresponding to the first three maximum values to zero, then find the maximum value of the modified accumulator array. Also use the "hot" colormap to plot the accumulator array.]