CHAPTER

2 BINARY MACHINE VISION

Thresholding and Segmentation

2.1 Introduction

In the detection and recognition of two-dimensional or three-dimensional objects or object defects by a computer vision system, the input image is often simplified by generating an output image whose pixels tend to have high values if they are part of an object of interest and low values if they are not part of any object of interest. To actually recognize an object, regions on the image that have the potential for being some part of the object first need to be identified. The simplest, although not necessarily the best, way to identify these object regions is to perform a threshold-labeling operation in which each pixel that has a high enough value is given the value binary 1. The value binary 1 here designates that the pixel has some possibility of being part of an object of interest. Each pixel that does not have a high enough value is given the value binary 0. This designates that it has little possibility of being part of any object of interest. The generation and analysis of such a binary image is called *binary machine vision.*

The first step of binary machine vision is to threshold a gray scale image, thereby labeling each pixel as a binary 0 or a binary 1. The binary-1 label designates a pixel that is considered to be part of an object of interest. The binary-0 label designates a background pixel. Thresholding is a labeling operation.

Depending on the complexity of the objects and the nature of the shapes to be identified and their expected relative positions, the next stage of processing could be one of two midlevel vision grouping techniques: connected components labeling or signature segmentation. Both these techniques make a transformation on the kind of units being processed. The units of the image are the pixels. The units after the transformation are more complex; they are called regions or segments and are composed of groupings of pixels. After the regions are defined, a variety of measurements can be made on them. This constitutes an attribute labeling or property measurement step that we call *feature extraction.* The regions are finally

assigned an object class or an object defect class or category through a pattern recognition technique. This constitutes the matching and inferring steps.

The operation sequence of thresholding, connected components labeling, region property measurement, and statistical pattern recognition is called *connected components analysis*. This was the basis of what has come to be known as the SRI algorithm (Gleason and Agin, 1979; Agin, 1980). The operation sequence of thresholding, signature segmentation, region property measurement, and statistical pattern recognition is called *signature analysis*. This chapter covers the topics of thresholding and segmentation. Thresholding is discussed in Section 2.2; segmentation by connected components labeling, in Section 2.3; and signature segmentation and analysis, in Section 2.4.

2.2 · Thresholding

Thresholding is a labeling operation on a gray scale image. Thresholding distinguishes pixels that have higher gray values from pixels that have lower gray values. Pixels whose gray values are high enough are given the binary value 1. Pixels whose gray values are not high enough are given the binary value 0. Figure 2.1 illustrates a simple gray scale image. Figure 2.2 illustrates a thresholded image obtained by making all pixels having a value greater than 1 a binary 1 and all other pixels a binary 0.

The question of thresholding is a question of the automatic determination of the threshold value. What basis can be used? Since the threshold value separates the dark background from the bright object (or vice versa), the separation could ideally be done if the distribution of dark pixels were known and the distribution of bright pixels were known. The threshold value could then be determined as that separation value for which the fraction of dark pixels labeled binary 1 equals the fraction of bright pixels labeled binary 0. Such a threshold value would equalize the probability of the two kinds of errors: the error of assigning a pixel belonging to the background as a binary 1 and the error of assigning a pixel belonging to the object as a binary 0. The difficulty here is that the independent distributions of dark and bright pixels are often not known ahead of time. What is known is the image histogram, which tells how many pixels are associated with any gray value in the mixture distribution.

The *histogram* h of a digital image I is defined by $h(m) = \#\{(r,c) \mid I(r,c) = m\}$, where m spans each gray level value and $\#$ is the operator that counts the number of elements in a set. A histogram can be computed by using an array data structure and a very simple procedure. Let H be a vector array dimensioned from 0 to MAX, where 0 is the value of the smallest possible gray level value and MAX is the value of the largest. Let I be a two-dimensional array, dimensioned 1 to ROWSIZE by 1 to COLSIZE that holds a gray level image. The histogram procedure is given by

```
procedure Histogram(I,H);
"Initialize histogram to zero."
```

								5	8	9			
									9	8	9		
		5								7	9	8	
		3									6	9	9
		5				7							
	2					6							
	8						5			1	1		
3							6			1	1	1	
							7						
	4	3							3	2	6	2	
	2	4							3	8	4	3	
	5	9							7	2	3	9	

Figure 2.1 Original gray scale image. Pixels having no numbers have value of 0.

```
for i := 0 to MAX do
  H(i) := 0;
  "Compute values by accumulation."
  for r := 1 to ROWSIZE do
    for c := 1 to COLSIZE do
      begin
        grayval := I(r,c);
        H(grayval) := H(grayval) + 1
      end
    end for
  end for
end Histogram
```

If the distributions of dark pixels and bright pixels are widely separated, then the image histogram will be bimodal, one mode corresponding to the dark pixels and one mode corresponding to the bright pixels. With little distribution overlap, the threshold value is easily chosen as a value in the valley between the two dominant histogram modes. This is illustrated in Figs. 2.3 through 2.5. However, as the

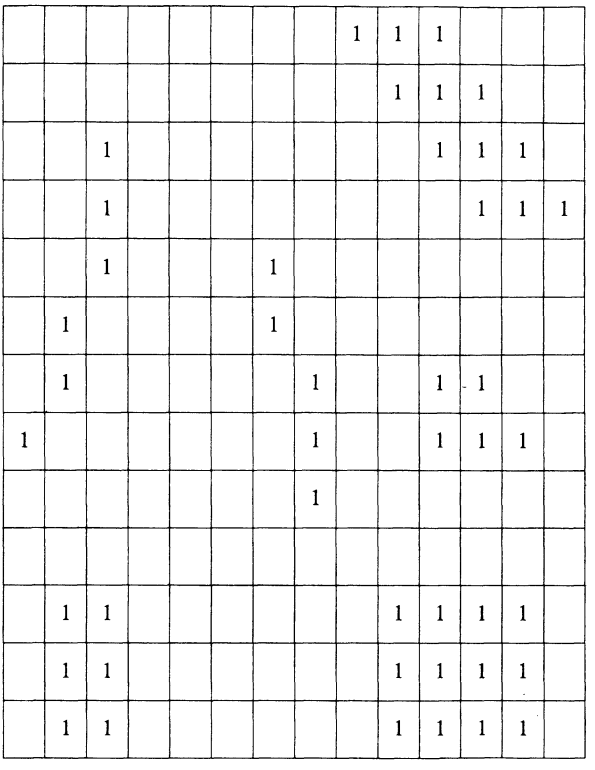


Figure 2.2 Thresholded gray scale image. All pixels greater than 0 are marked with a binary 1.

distributions for the bright and dark pixels become more and more overlapped, the choice of threshold value as the valley between the two histogram modes becomes more difficult, because the valley begins to disappear as the two distributions begin to merge together. Furthermore, when there is substantial overlap, the choice of threshold as the valley point is less optimal in the sense of minimizing classification error.

To make this point, we consider an example. Figure 2.6 illustrates an image of a BNC T-connector on a dark textured background. In such a simple image one might hope for a pixel to be either a part of the background or a part of the object of interest based purely on its gray value. But things are not so simple. Figure 2.7 shows a histogram of the BNC T-connector image of Fig. 2.6. For this image the histogram of the combination of the bright object on the dark background is not bimodal. The placement of the threshold is not so easy to determine.

Figure 2.8 illustrates different thresholds applied to the image of Fig. 2.6. Some are too low. Some are too high. Thresholds that are too low incorrectly label more pixels as bright (and therefore as part of the object of interest) than appropriate. Thresholds that are too high incorrectly label more pixels as dark (and therefore as not part of the object of interest) than appropriate.

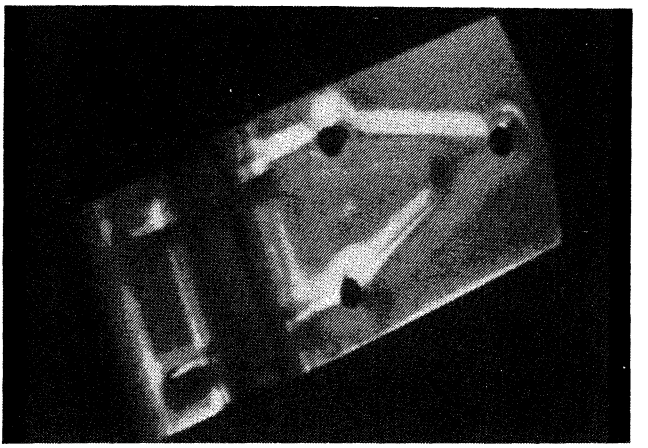


Figure 2.3 Image of a metal part.

Simple thresholding schemes such as that employed on the image of Fig. 2.6 and shown as the images of Fig. 2.8 compare each pixel's gray value with the same global threshold and make the distinction on the basis of whether the pixel's gray value is higher than the threshold. More complex thresholding schemes use a spatially varying threshold. Such techniques can compensate for a variety of local spatial context effects. Some of the more complex thresholding algorithms can be decomposed into two algorithms; the first algorithm produces the spatially varying threshold image, and the second subtracts the spatially varying threshold image from the original image and then performs a simple thresholding on the difference image. Such a spatially varying threshold image can be thought of as a means of performing

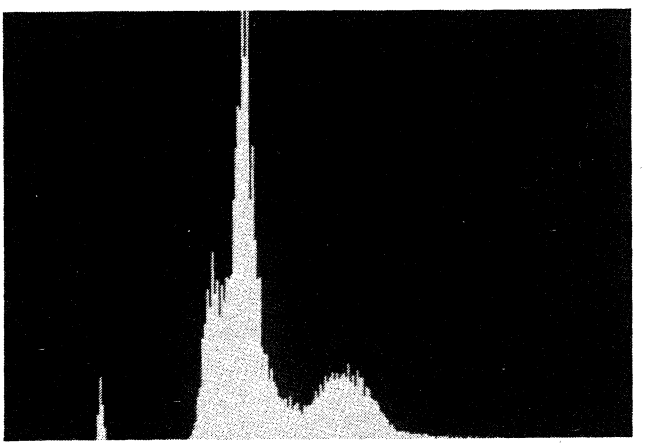


Figure 2.4 Histogram of the image of Fig. 2.3. The histogram shows two dominant modes. The small mode on the left tail is not significant.

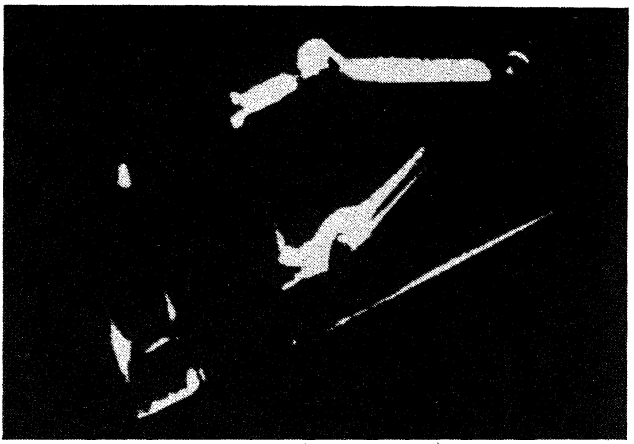


Figure 2.5 Metal-part image of Fig. 2.3 thresholded at gray level 148, which is in the valley between the two dominant modes.

background normalization. The technique of constructing a spatially varying threshold image arises naturally in the opening and closing residue operations, which are discussed in Chapter 5 on mathematical morphology.

In this section we give an introductory discussion of two methods for simple thresholding. The key issue is how to select the most appropriate threshold value. Since all pixels will be compared with the same global threshold, and no further information is available, it is natural to base the selection of the threshold on the image histogram.

The two techniques we discuss are based on finding a threshold that minimizes a criterion function. The first method minimizes the within-group variance. The second method minimizes the Kullback information measure (Kullback, 1959).

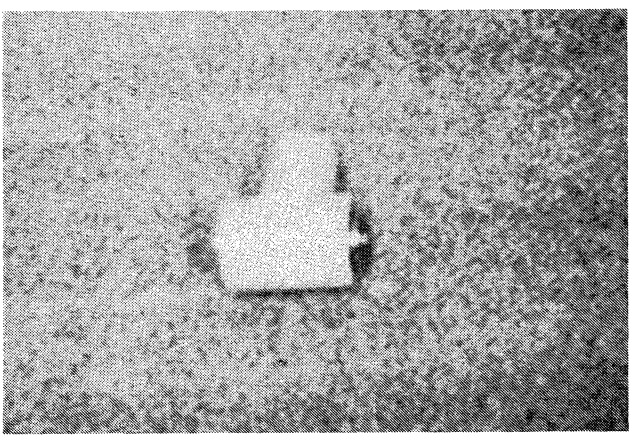


Figure 2.6 BNC T-connector against a dark background.

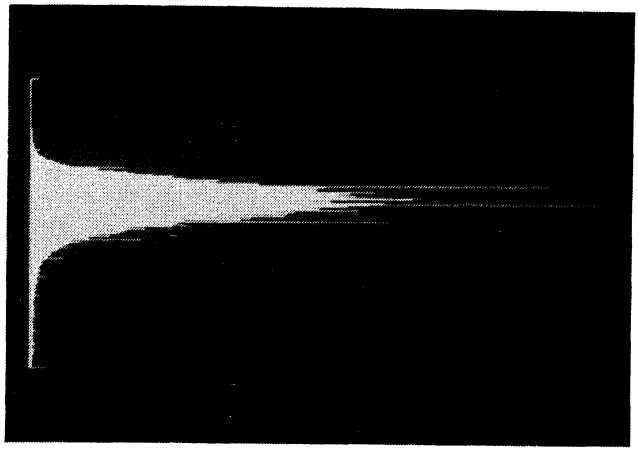


Figure 2.7 Histogram of the BNC T-connector image of Fig. 2.6.

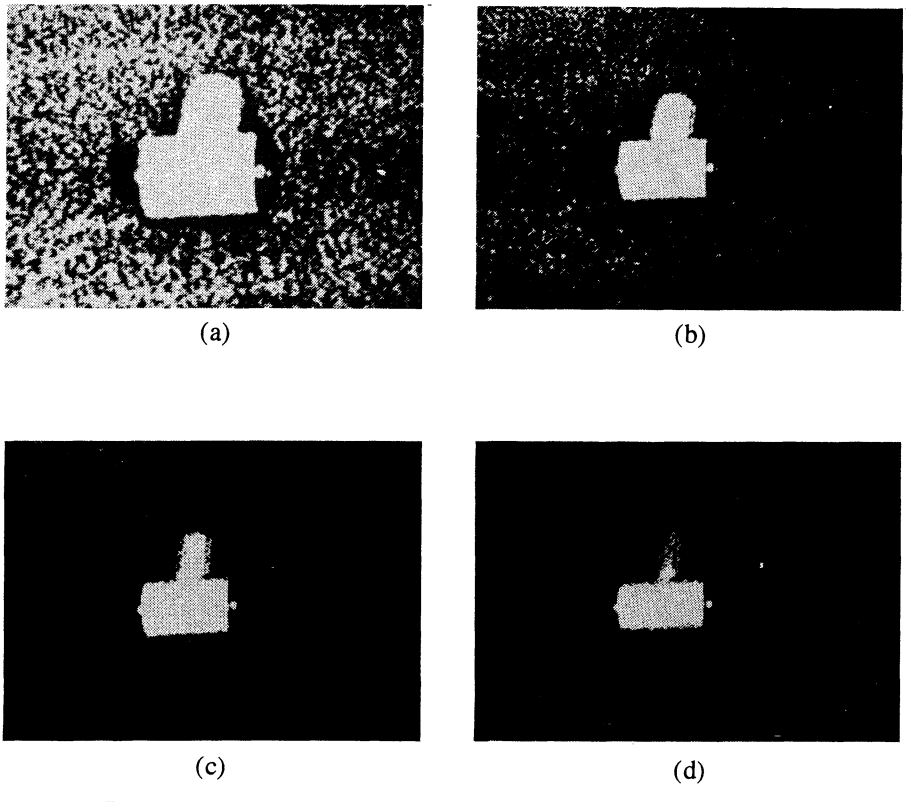


Figure 2.8 Image of the BNC T-connector thresholded at four levels: (a) 110, (b) 130, (c) 150, and (d) 170.

2.2.1 Minimizing Within-Group Variance

Let $P(1),\ldots,P(I)$ represent the histogram probabilities of the observed gray values $1,\ldots,I; P(i) = \#\{(r,c) \mid Image(r,c) = i\}/\#R \times C$, where $R \times C$ is the spatial domain of the image. If the histogram is bimodal, the histogram thresholding problem is to determine a best threshold t separating the two modes of the histogram from each other. Each threshold t determines a variance for the group of values that are less than or equal to t and a variance for the group of values greater than t. The definition for best threshold suggested by Otsu (1979) is that threshold for which the weighted sum of group variances is minimized. The weights are the probabilities of the respective groups.

We motivate the within-group variance criterion by considering the situation that sometimes happens in a schoolroom. An exam is given and the histogram of the resulting scores is bimodal. There are the better students and the worse students. Lectures that are aimed at the better students go too fast for the others, and lectures that are aimed at the level of the worse students are boring to the better students. To fix this situation, the teacher decides to divide the class into two mutually exclusive and homogeneous groups based on the test score. The question is to determine which test score to use as the dividing criterion. Ideally each group should have test scores that have a unimodal bell-shaped histogram, one around a lower mean and one around a higher mean. This would indicate that each group is homogeneous within itself and different from the other.

A measure of group homogeneity is variance. A group with high homogeneity will have low variance. A group with low homogeneity will have high variance. One possible way to choose a dividing criterion is to choose a dividing score such that the resulting weighted sum of the within-group variances is minimized. This criterion emphasizes high group homogeneity. A second way to choose the dividing criterion is to choose a dividing score that maximizes the resulting squared difference between the group means. This difference is related to the between-group variance. Both dividing criteria lead to the same dividing score because the sum of the within-group variances and the between-group variances is a constant.

Let σ_W^2 be the weighted sum of group variances, that is, the *within-group variance*. Let $\sigma_1^2(t)$ be the variance for the group with values less than or equal to t and $\sigma_2^2(t)$ be the variance for the group with values greater than t. Let $q_1(t)$ be the probability for the group with values less than or equal to t and $q_2(t)$ be the probability for the group with values greater than t. Let $\mu_1(t)$ be the mean for the first group and $\mu_2(t)$ the mean for the second group. Then the within-group variance σ_W^2 is defined by

$$\sigma_W^2(t) = q_1(t)\,\sigma_1^2(t) + q_2(t)\,\sigma_2^2(t)$$

where

$$q_1(t) = \sum_{i=1}^{t} P(i)$$

$$q_2(t) = \sum_{i=t+1}^{I} P(i) \tag{2.1}$$

$$\mu_1(t) = \sum_{i=1}^{t} i\ P(i)/q_1(t)$$

$$\mu_2(t) = \sum_{i=t+1}^{I} i\ P(i)/q_2(t) \tag{2.2}$$

$$\sigma_1^2(t) = \sum_{i=1}^{t} [i - \mu_1(t)]^2 P(i)/q_1(t)$$

$$\sigma_2^2(t) = \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 P(i)/q_2(t) \tag{2.3}$$

The best threshold t can then be determined by a simple sequential search through all possible values of t to locate the threshold t that minimizes $\sigma_W^2(t)$. In many situations this can be reduced to a search between the two modes. However, identification of the modes is really equivalent to the identification of separating values between the modes.

There is a relationship between the within-group variance $\sigma_W^2(t)$ and the total variance σ^2 that does not depend on the threshold. The total variance is defined by

$$\sigma^2 = \sum_{i=1}^{I} (i - \mu)^2 P(i)$$

where

$$\mu = \sum_{i=1}^{I} i\ P(i)$$

The relationship between the total variance and the within-group variance can make the calculation of the best threshold less computationally complex. By rewriting σ^2, we have

$$\sigma^2 = \sum_{i=1}^{t} [i - \mu_1(t) + \mu_1(t) - \mu]^2\ P(i) + \sum_{i=t+1}^{I} [i - \mu_2(t) + \mu_2(t) - \mu]^2\ P(i)$$

$$= \sum_{i=1}^{t} \{[i - \mu_1(t)]^2 + 2[i - \mu_1(t)][\mu_1(t) - \mu] + [\mu_1(t) - \mu]^2\} P(i)$$

$$+ \sum_{i=t+1}^{I} \{[i - \mu_2(t)]^2 + 2[i - \mu_2(t)][\mu_2(t) - \mu] + [\mu_2(t) - \mu]^2\} P(i)$$

But

$$\sum_{i=1}^{t} [i - \mu_1(t)][\mu_1(t) - \mu] P(i) = 0 \quad \text{and}$$

$$\sum_{i=t+1}^{I} [i - \mu_2(t)][\mu_2(t) - \mu)] P(i) = 0$$

Since

$$q_1(t) = \sum_{i=1}^{t} P(i) \quad \text{and} \quad q_2(t) = \sum_{i=t+1}^{I} P(i)$$

$$\begin{aligned}\sigma^2 &= \sum_{i=1}^{t} [i - \mu_1(t)]^2 P(i) + [\mu_1(t) - \mu]^2 q_1(t) \\ &\quad + \sum_{i=t+1}^{I} [i - \mu_2(t)]^2 P(i) + [\mu_2(t) - \mu]^2 q_2(t) \\ &= [q_1(t)\, \sigma_1^2(t) + q_2(t)\, \sigma_2^2(t)] \\ &\quad + \{q_1(t)\, [\mu_1(t) - \mu]^2 + q_2(t)\, [\mu_2(t) - \mu]^2\} \end{aligned} \tag{2.4}$$

The first bracketed term is called the within-group variance σ_W^2. It is just the sum of the weighted variances of each of the two groups. The second bracketed term is called the between-group variance σ_B^2. It is just the sum of the weighted squared distances between the means of each group and the grand mean. The between-group variance can be further simplified. Note that the grand mean μ can be written as

$$\mu = q_1(t)\, \mu_1(t) + q_2(t)\, \mu_2(t) \tag{2.5}$$

Using Eq. (2.5) to eliminate μ in Eq. (2.4), substituting $1 - q_1(t)$ for $q_2(t)$, and simplifying, we obtain

$$\sigma^2 = \sigma_W^2(t) + q_1(t)[1 - q_1(t)]\, [\mu_1(t) - \mu_2(t)]^2$$

Since the total variance σ^2 does not depend on t, the t minimizing $\sigma_W^2(t)$ will be the t maximizing the between-group variance $\sigma_B^2(t)$,

$$\sigma_B^2(t) = q_1(t)\, [1 - q_1(t)]\, [\mu_1(t) - \mu_2(t)]^2 \tag{2.6}$$

To determine the maximizing t for $\sigma_B^2(t)$, the quantities determined by Eqs. (2.1) to (2.3) all have to be determined. However, this need not be done independently for each t. There is a relationship between the value computed for t and that computed for the next $t : t + 1$. We have directly from Eq. (2.1) the recursive relationship

$$q_1(t + 1) = q_1(t) + P(t + 1) \tag{2.7}$$

with initial value $q_1(1) = P(1)$.

From Eq. (2.2) we obtain the recursive relation

$$\mu_1(t + 1) = \frac{q_1(t)\, \mu_1(t) + (t + 1)P(t + 1)}{q_1(t + 1)} \tag{2.8}$$

with the initial value $\mu_1(0) = 0$. Finally, from Eq. (2.5) we have

$$\mu_2(t + 1) = \frac{\mu - q_1(t + 1)\, \mu_1(t + 1)}{1 - q_1(t + 1)} \tag{2.9}$$

Figure 2.9 illustrates the binary image produced by the Otsu threshold. Kittler and Illingworth (1985) note that the between group variance σ_B^2 is not necessarily a unimodal criterion, even though Otsu had hypothesized it was. Also, when the fractions

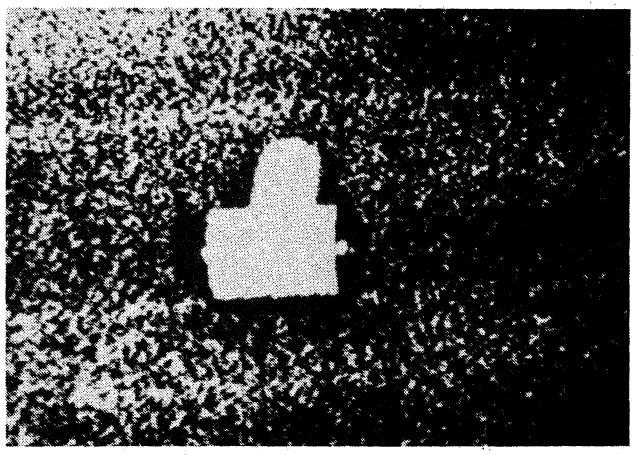


Figure 2.9 Binary image produced by thresholding the T-connector image of Fig. 2.6 with the Otsu threshold.

of pixels in each mode are far from being approximately equal, the minimization of σ_W^2 or the equivalent maximization of σ_B^2 will not necessarily produce the correct answer.

2.2.2 Minimizing Kullback Information Distance

Kittler and Illingworth (1985) suggest a different criterion from Otsu's. They assume that the observations come from a mixture of two Gaussian distributions having respective means and variances (μ_1, σ_1^2) and (μ_2, σ_2^2) and respective proportions q_1 and q_2. They determine the threshold T that results in $q_1, q_2, \mu_1, \mu_2, \sigma_1, \sigma_2$, which minimize the Kullback (1959) directed divergence J from the observed histogram $P(1), \ldots, P(I)$ to the unknown mixture distribution f. J is defined by

$$J = \sum_{i=1}^{I} P(i) \log \left[\frac{P(i)}{f(i)} \right]$$

A mixture distribution f having fraction q_1 of distribution h_1 and fraction q_2 of distribution h_2 can be represented as

$$f(i) = q_1 h_1(i) + q_2 h_2(i)$$

The mixture distribution of the two Gaussians reflected in the histogram therefore takes the form

$$f(i) = \frac{q_1}{\sqrt{2\pi}\sigma_1} e^{-\frac{1}{2}\left(\frac{i-\mu_1}{\sigma_1}\right)^2} + \frac{q_2}{\sqrt{2\pi}\sigma_2} e^{-\frac{1}{2}\left(\frac{i-\mu_2}{\sigma_2}\right)^2}$$

The meaning of J can be understood in the following way. Let H be the hypothesis that the observed outcomes follow probability distribution P. Let H' be the hypothesis that the observed outcomes follow probability distribution f. Let i

designate a value of an outcome. Then $Prob(i|H) = P(i)$ and $Prob(i|H') = f(i)$. Denote the prior probability of H by $Prob(H)$ and the prior probability of H' by $Prob(H')$.

By definition of conditional probability,

$$Prob(H|i) = \frac{Prob(i|H)Prob(H)}{Prob(i|H)Prob(H) + Prob(i|H')Prob(H')}$$
$$= \frac{P(i)Prob(H)}{P(i)Prob(H) + f(i)Prob(H')}$$

Similarly,

$$Prob(H'|i) = \frac{f(i)Prob(H')}{P(i)Prob(h) + f(i)Prob(H')}$$

Dividing the two equations and rearranging them yields:

$$\frac{Prob(H|i)Prob(H')}{Prob(H'|i)Prob(H)} = \frac{P(i)}{f(i)}$$

Hence

$$\log \frac{P(i)}{f(i)} = \log \frac{Prob(H|i)}{Prob(H'|i)} - \log \frac{Prob(H)}{Prob(H')} \tag{2.10}$$

The right-hand side of Eq. (2.10) is the difference between the logarithm of the odds in favor of H after observing outcome i and the logarithm of the odds in favor of H before observing outcome i. Therefore $\log P(i)/f(i)$ has the interpretation of the information in the outcome i for discrimination in favor of H against H'. Under the hypothesis H, the mean information in favor of H against H' is then

$$J(P;f) = \sum_{i=1}^{I} P(i) \log \frac{P(i)}{f(i)}$$

$J(P;f)$ has the property that (1) $J(P;f) \geq 0$ for all probability distributions P and f, and (2) $J(P;f) = 0$ if and only if $P = f$ (Kullback, 1959). However, J is not symmetric and does not satisfy the triangle inequality and is therefore not a metric.

The parameters of the mixture distribution can be estimated by minimizing J. Now J can be rewritten

$$J = \sum_{i=1}^{I} P(i) \log P(i) - \sum_{i=1}^{I} P(i) \log f(i)$$

Clearly the first term does not depend on the unknown parameters. The minimization can be done by minimizing the second term. Hence we take the information measure H to be minimized where

$$H = -\sum_{i=1}^{I} P(i) \log f(i)$$

To carry out the minimization, we assume that the modes are well separated. Hence for some threshold t that separates the two modes

$$f(i) \approx \begin{cases} q_1 \Big/ \left(\sqrt{2\pi}\sigma_1\right) e^{-\frac{1}{2}\left(\frac{i-\mu_1}{\sigma_1}\right)^2}, & i \leq t \\ q_2 \Big/ \left(\sqrt{2\pi}\sigma_2\right) e^{-\frac{1}{2}\left(\frac{i-\mu_2}{\sigma_2}\right)^2}, & i > t \end{cases}$$

Now

$$H(t) = -\sum_{i=1}^{t} P(i) \log \frac{q_1}{\sqrt{2\pi}\sigma_1} e^{-\frac{1}{2}\left(\frac{i-\mu_1}{\sigma_1}\right)^2} - \sum_{i=t+1}^{I} P(i) \log \frac{q_2}{\sqrt{2\pi}\sigma_2} e^{-\frac{1}{2}\left(\frac{i-\mu_2}{\sigma_2}\right)^2}$$

Upon simplifying we obtain

$$H = \frac{1 + \log 2\pi}{2} - q_1 \log q_1 - q_2 \log q_2 + \frac{1}{2}(q_1 \log \sigma_1^2 + q_2 \log \sigma_2^2) \qquad (2.11)$$

The assumption of well separated modes means that if t is the threshold that separates the modes, the mean and variance estimated from $P(1), \ldots, P(t)$ will be close to the true mean and variance μ_1 and σ_1. Likewise, the mean and variance estimated from $P(t+1), \ldots, P(I)$ will be close to the true mean and variance μ_2 and σ_2. Hence, using the estimated quantities for the unknown quantities, $H(t)$ can be evaluated for each threshold t. The value t that minimizes $H(t)$ is then the best threshold. Figure 2.10 illustrates the binary image of the BNC T-connector image produced by the Kittler-Illingworth technique. Figure 2.11 shows the histogram of the T-connector image and the places where the Otsu technique and the Kittler-Illingworth technique determine the threshold. The difference is substantial. In this case the Otsu technique detected all the pixels belonging to the connecter but at the

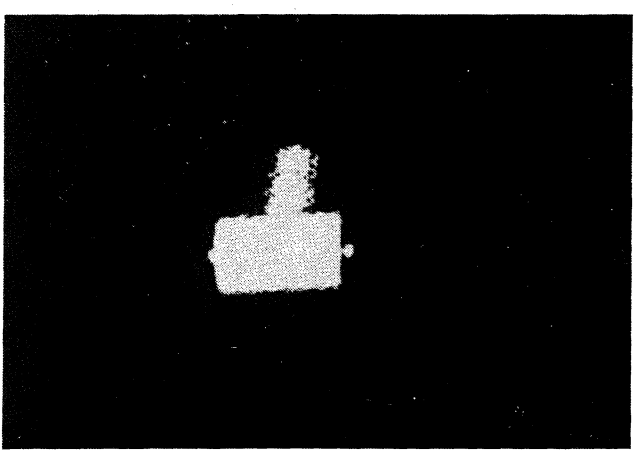


Figure 2.10 Binary image produced by thresholding the T-connector image of Fig. 2.6 with the Kittler-Illingworth threshold.

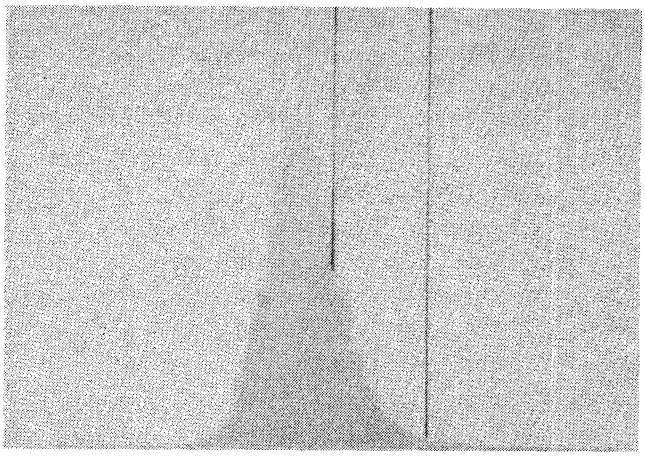


Figure 2.11 Histogram of the image of Fig. 2.6 showing where the Otsu and Kittler-Illingworth techniques choose the threshold value. The leftmost dark line is the Otsu threshold. The rightmost dark line is the Kittler-Illingworth threshold.

expense of many background pixels being falsely detected. In the case of the Kittler-Illingworth technique, no background pixels were falsely detected, but not quite all the connector pixels were detected. On balance, for gray scale images having bimodal histograms, machine vision techniques will have an easier job working with the binary images produced by the Kittler-Illingworth threshold than with the images produced by the Otsu threshold.

The evaluation of the best Kittler-Illingworth threshold t can be simplified by using the results of the previous t. From Eqs. (2.1) and (2.3), we can develop the recursive equations

$$\sigma_1^2(t+1) = \frac{q_1(t)\left\{\sigma_1^2(t) + [\mu_1(t) - \mu_1(t+1)]^2\right\} + P(t+1)\left[(t+1) - \mu_1(t+1)\right]^2}{q_1(t+1)}$$

$$\sigma_2^2(t+1) = \frac{[1 - q_1(t)]\left\{\sigma_2^2(t) + [\mu_2(t) - \mu_2(t+1)]^2\right\} - P(t+1)\left[(t+1) - \mu_2(t+1)\right]^2}{1 - q_1(t+1)}$$

which can be used in evaluating the $H(t)$ of Eq. (2.10). Then $\mu_1(t)$, $\mu_2(t)$, and $q_1(t)$ can be recursively computed using Eqs. (2.7), (2.8), and (2.9).

Instead of using the criterion of Kittler and Illingworth, we can use the more theoretically powerful criterion of probability of correct classification $P_c(t)$ that would be obtained by a threshold classifier under the assumption of class conditional Gaussian distributions. If we assume, without loss of generality, that $\mu_1 < \mu_2$, the probability of correct classification with threshold value t is given by

$$P_c(t) = q_1(t)\phi\left(\frac{t - \mu_1}{\sigma_1}\right) + q_2(t)\left[1 - \phi\left(\frac{t - \mu_2}{\sigma_2}\right)\right]$$

where

$$\phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{1}{2}u^2} du$$

This is illustrated in Fig. 2.12.

In using the criterion of probability of correct classification, the means μ_1 and μ_2 are each computed by using a distribution; one of whose tails has been truncated. The smaller-valued mean comes from a distribution whose right tail has been truncated, and the larger-valued mean comes from a distribution whose left tail has been truncated. Thus the smaller mean is biased too small and the larger mean is biased too high. It is possible to correct for these biases.

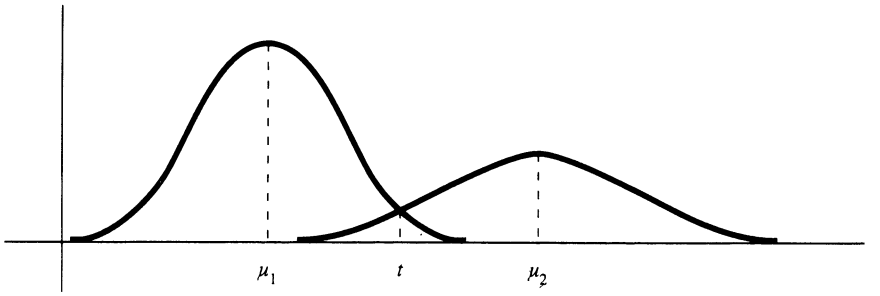
If μ denotes the mean of a normal distribution having variance σ^2, with a truncated right tail, and μ^* denotes the mean of the same normal distribution without its right tail, then it is easy to derive

$$\mu^* = \mu - \frac{\sigma}{\phi\left(\frac{t-\mu^*}{\sigma}\right)} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu^*}{\sigma}\right)^2}$$

where t is the truncation point.

As a first order correction to μ, we can estimate the mean of the normal without a truncated right tail by

$$\hat{\mu} = \mu + \frac{\sigma}{\phi\left(\frac{t-\mu}{\sigma}\right)} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}$$



$$f_1(x) = \frac{q_1}{\sqrt{2\pi}\sigma_1} e^{-\frac{1}{2}\left(\frac{x-\mu_1}{\sigma_1}\right)^2}$$

$$f_2(x) = \frac{q_2}{\sqrt{2\pi}\sigma_2} e^{-\frac{1}{2}\left(\frac{x-\mu_2}{\sigma_2}\right)^2}$$

Figure 2.12 Mixture of two Gaussians. If an assignment is made to background pixel whenever $x < t$ and an assignment is made to foreground pixel whenever $x > t$, the probability of observing an $x < t$ and classifying it as background will be the area under f_1 to the left of t. The probability of observing an $x > t$ and classifying it as foreground will be the area under f_2 to the right of t.

Similarly, if μ denotes the mean of a normal distribution with a truncated left tail, we can obtain

$$\hat{\mu} = \mu + \frac{\sigma}{1 - \phi\left(\frac{t-\mu^*}{\sigma}\right)} \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2}$$

Cho, Haralick, and Yi (1989) demonstrate how the use of these corrections can improve the performance of the Kittler-Illingworth technique.

Several other thresholding techniques are discussed in the vision literature. Some of them do not work as well as the Kullback information we do discuss. Unfortunately, there seems to be no uniform solution to the thresholding problem without simplifying the assumptions such as mixture of Gaussians.

Weszka and Rosenfeld (1978) discuss a variety of ways to evaluate thresholding techniques. Weszka (1978) and Sahoo et al. (1988) survey thresholding techniques. Tsai (1985) suggests thresholding based on preserving values of moments. Wu, Hong, and Rosenfeld (1982) suggest using a quadtree segmentation procedure. There, the histogram of the resulting larger near piecewise constant regions will be highly peaked, the various peaks corresponding to the modes.

A few papers have suggested combining histogram information with edge and gradient information: Weszka, Nagel, and Rosenfeld (1974); Weszka and Rosenfeld (1979); Milgram and Herman (1979); and Kittler, Illingworth, and Föglein (1985). Kirby and Rosenfeld (1979) combine gray level and local neighborhood gray level. Abutaleb (1989) uses the same combination along with an entropy criterion. Ahuja and Rosenfeld (1978) suggest using the distribution of spatially neighboring gray tones. Kohler (1981) selects a threshold to maximize the resulting contrast between the gray value coming from binary-1 pixels adjacent to binary-0 pixels.

2.3 Connected Components Labeling

Connected components analysis of a binary image consists of the connected components labeling of the binary-1 pixels followed by property measurement of the component regions and decision making. The connected components labeling operation performs the unit change from pixel to region or segment. All pixels that have value binary 1 and are connected to each other by a path of pixels all with value binary 1 are given the same identifying label. The label is a unique name or index of the region to which the pixels belong. The label is the identifier for a potential object region. Connected components labeling is a grouping operation.

The units of the image are pixels, and the filtering techniques of image processing transform pixels to pixels. Connected components labeling is one image-processing technique that can make a unit change from pixel to region. The region is a more complex unit than the pixel. The only properties a pixel has are its position and its gray level or brightness level. A region has a much richer set of properties. A region has shape and position properties as well as statistical properties of the gray levels of the pixels in the region.

To each region, therefore, we can construct an N-tuple or vector of its measurement properties. One way to recognize different objects, object defects, or characters is to distinguish between the regions on the basis of their measurement properties. This is the role of statistical pattern recognition, which is discussed in Chapter 4. This section examines connected components labeling algorithms, which in essence group together all pixels belonging to the same region and give them the same label. Software for performing connected components labeling can be found in Ronse and Divijver (1984) and in Cunningham (1981). An APL-based strategy for the extraction of binary image structures is given in Mussio and Padula (1985).

2.3.1 Connected Components Operators

Once a gray level image has been processed to remove noise and thresholded to produce a binary image, a connected components labeling operator can be employed to group the binary-1 pixels into maximal connected regions. These regions are called the *connected components* of the binary image, and the associated operator is called the *connected components operator*. Its input is a binary image and its output is a symbolic image in which the label assigned to each pixel is an integer uniquely identifying the connected component to which that pixel belongs. Figure 2.13 illustrates the connected components operator as applied to the 1-pixels of a binary image. The same operator can, of course, be applied to the 0-pixels.

Two 1-pixels p and q belong to the same connected component C if there is a sequence of 1-pixels $(p_0, p_1, \ldots, p_n)$ of C where $p_0 = p$, $p_n = q$, and p_i is a neighbor of p_{i-1} for $i = 1, \ldots, n$. Thus the definition of a connected component depends on the definition of neighbor. When only the north, south, east, and west neighbors of a pixel are considered part of its neighborhood, then the resulting regions are called *4-connected*. When the north, south, east, west, northeast, northwest, southeast, and southwest neighbors of a pixel are considered part of its neighborhood, the resulting regions are called *8-connected*. This is illustrated in Fig. 2.14. Whichever definition is used, the neighbors of a pixel are said to be *adjacent* to that pixel. The *border* of a connected component of 1-pixels is the subset of pixels belonging to the component that are also adjacent to 0-pixels. Similarly,

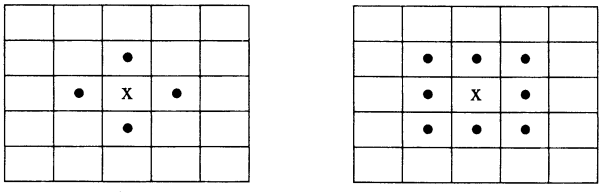


Figure 2.13 (a) Function of the connected components operator on a binary image; (b) Symbolic image produced from (a) by the connected components operator.

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

(a)

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

(b)

Figure 2.14 (a) Pixels, •, that are 4-connected to the center pixel x; (b) pixels, •, that are 8-connected to the center pixel x.

the border of a connected component of 0-pixels is the subset of pixels of that component that are also adjacent to 1-pixels.

Rosenfeld (1970) has shown that if C is a component of 1s and D is an adjacent component of 0s, and if 4-connectedness is used for 1-pixels and 8-connectedness is used for 0-pixels, then either C surrounds D (D is a hole in C) or D surrounds C (C is a hole in D). This is also true when 8-connectedness is used for 1-pixels and 4-connectedness for 0-pixels, but not when 4-connectedness is used for both 1-pixels and 0-pixels and not when 8-connectedness is used for both 1-pixels and 0-pixels. Figure 2.15 illustrates this phenomenon. The surroundedness property is desirable because it allows borders to be treated as closed curves. Because of this, it is common to use one type of connectedness for 1-pixels and the other for 0-pixels.

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

(a)

a	a	a	a	a
a	a	1	a	a
a	2	b	3	a
a	a	4	a	a
a	a	a	a	a

(b)

a	a	a	a	a
a	a	1	a	a
a	1	a	1	a
a	a	1	a	a
a	a	a	a	a

(c)

a	a	a	a	a
a	a	1	a	a
a	1	b	1	a
a	a	1	a	a
a	a	a	a	a

(d)

Figure 2.15 Phenomenon associated with using 4- and 8-adjacency in connected components analyses. Numeric labels are used for components of 1-pixels and letter labels for 0-pixels. (a) Binary image; (b) connected components labeling with 4-adjacency used for both 1-pixels and 0-pixels; (c) connected components labeling with 8-adjacency used for both 1-pixels and 0-pixels; and (d) connected components labeling with 8-adjacency used for 1-pixels and 4-adjacency used for 0-pixels.

2.3.3 An Iterative Algorithm

The iterative algorithm (Haralick, 1981) uses no auxillary storage to produce the labeled image from the binary image. It would be useful in environments whose storage is severely limited or on SIMD hardware. It consists of an initialization step plus a sequence of top-down label propagation followed by bottom-up label propagation iterated until no label changes occur. Figure 2.17 illustrates the iterative algorithm on a simple image.

This algorithm and the others will be expressed in pseudocode for an NLINES by NPIXELS binary image I and label image LABEL. The function NEWLABEL generates a new integer label each time it is called. The function NEIGHBORS returns the set of already-labeled neighbors of a given pixel on its own line or the previous line. The function LABELS, when given such a set of already-labeled pixels, returns the set of their labels. Finally, the function MIN, when given a set of labels, returns the minimum label.

```
procedure Iterate;
"Initialization of each 1-pixel to a unique label"
for L := 1 to NLINES do
  for P := 1 to NPIXELS do
    if I(L,P) = 1
    then LABEL (L,P) := NEWLABEL( )
    else LABEL(L,P) := 0
  end for
end for;
"Iteration of top-down followed by bottom-up passes"
repeat
```

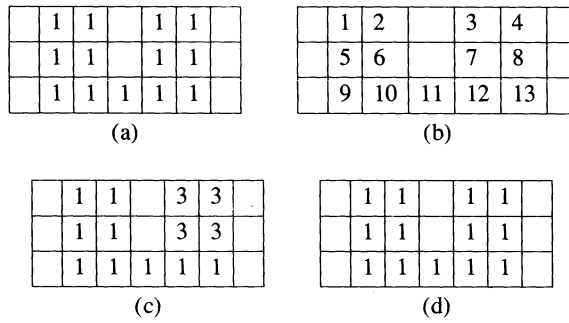


Figure 2.17 Iterative algorithm for connected components labeling. Part (a) shows the original binary image; (b) the results after initialization of each 1-pixel to a unique label; (c) the results after the first top-down pass, in which the value of each nonzero pixel is replaced by the minimum value of its nonzero neighbors in a recursive manner going from left to right and top to bottom; and (d) the results after the first bottom-up pass.

```
"Top-down pass"
CHANGE := false;
for L := 1 to NLINES do
  for P := 1 to NPIXELS do
    if LABEL(L,P) <> 0 then
      begin
        M := MIN(LABELS(NEIGHBORS((L,P)) ∪ (L,P)));
        if M <> LABEL(L,P)
        then CHANGE := true;
        LABEL(L,P) := M
      end
  end for
end for;

"Bottom-up pass"
for L := NLINES to 1 by -1 do
  for P := NPIXELS to 1 by -1 do
    if LABEL(L,P) <> 0 then
      begin
        M := MIN(LABELS(NEIGHBORS((L,P)) ∪ (L,P)));
        if M <> LABEL(L,P)
        then CHANGE := true;
        LABEL(L,P) := M
      end
  end for
end for
until CHANGE := false
end Iterate
```

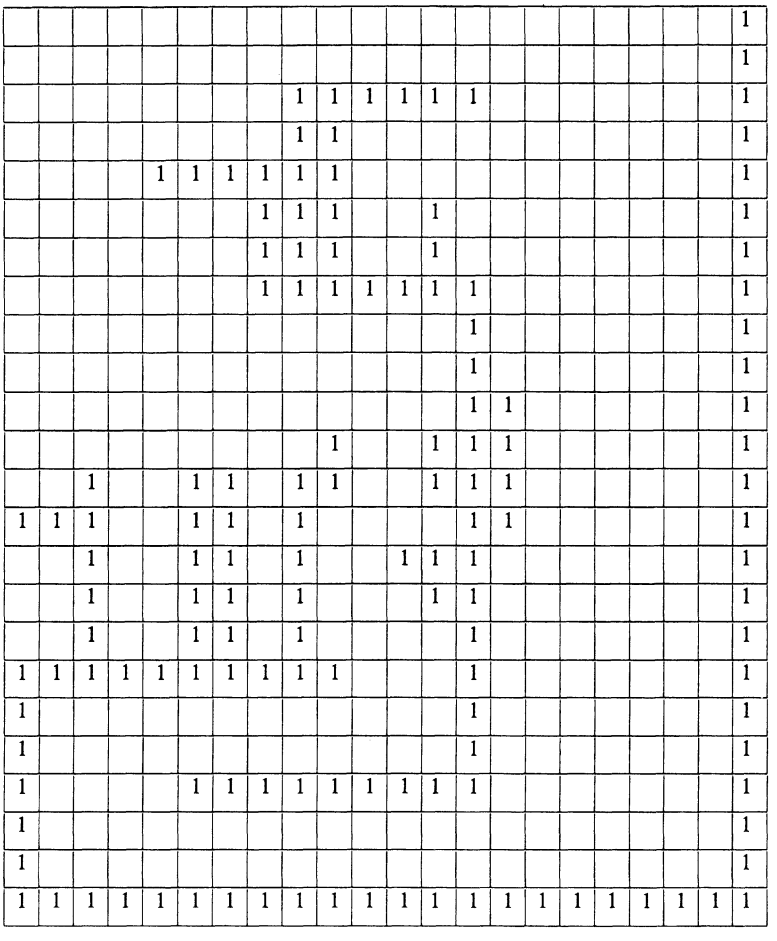
This algorithm selects the minimum label of its neighbors to assign to pixel A. It does not directly keep track of equivalences but instead uses a number of passes through the image to complete the labeling. It alternates top-down, left-to-right passes with bottom-up, right-to-left passes so that labels near the bottom or right margins of the image will propagate sooner than if all passes were top-down, left-to-right. This is an attempt to reduce the number of passes. The algorithm has a natural extension for SIMD parallel processing (Manohar and Ramapriyan, 1989).

2.3.4 The Classical Algorithm

The classical algorithm, deemed so because it is based on the classical connected components algorithm for graphs, was described in Rosenfeld and Pfaltz (1966). This algorithm makes only two passes through the image but requires a large global table for recording equivalences. The first pass performs label propagation, much as described above. Whenever a situation arises in which two different labels can propagate to the same pixel, the smaller label propagates and each such equivalence

found is entered in an equivalence table. Each entry in the equivalence table consists of an ordered pair, the values of its components being the labels found to be equivalent. After the first pass, the equivalence classes are found by taking the transitive closure of the set of equivalences recorded in the equivalence table. In the algorithm we call this the "Resolve" function. It is a standard algorithm discussed in many books on algorithms, such as Aho, Hopcroft, and Ullman (1983). Each equivalence class is assigned a unique label, usually the minimum (or oldest) label in the class. Finally, a second pass through the image performs a translation, assigning to each pixel the label of the equivalence class of its pass-1 label. This process is illustrated in Fig. 2.18, and the algorithm is given below.

```
procedure Classical
"Initialize global equivalence table."
EQTABLE := CREATE( );
"Top-down pass 1"
for L := 1 to NLINES do
  "Initialize all labels on line L to zero."
  for P := 1 to NPIXELS do
    LABEL(L,P) := 0
  end for;
  "Process the line."
  for P := 1 to NPIXELS do
    if F(L,P) := 1 then
      begin
        A := NEIGHBORS((L,P));
        if ISEMPTY(A)
        then M := NEWLABEL( )
        else M := MIN(LABELS(A));
        LABEL(L,P) := M;
        for X in LABELS(A) and X <> M
          ADD(X, M, EQTABLE)
        end for;
      end
  end for
end for;
"Find equivalence classes."
EQCLASSES := Resolve(EQTABLE);
for E in EQCLASSES
  EQLABEL(E) := min(LABELS(E))
end for;
"Top-down pass 2"
for L := 1 to NLINES do
  for P := 1 to NPIXELS do
```



(a)

Figure 2.18 Classical connected components labeling algorithm: Part (a) shows the initial binary image, and (b) the labeling after the first top-down pass of the algorithm. The equivalence classes found are 1: { 1,12,7,8,9,10,5 } and 2: { 2,3,4,6,11,13 }.

```
    if I(L,P) = 1
    then LABEL(L,P) := EQLABEL(CLASS(LABEL(L,P)))
  end for
end for
end Classical
```

The algorithm referred to as RESOLVE is simply the algorithm for finding the connected components of the graph structure defined by the set of equivalences (EQTABLE) defined in pass 1. The nodes of the graph are region labels, and the edges are pairs of labels that have been declared equivalent. The procedure, which

																1	1	1	1	
																			1	
							2	2	2	2	2	2							1	
							2	2											1	
				3	3	3	3	2	2										1	
							3	2	2			4							1	
							3	2	2			4							1	
							3	2	2	2	2	2	2						1	
													2						1	
													2						1	
													2	2					1	
									5			6	2	2					1	
		7			8	8		9	5			6	2	2					1	
10	10	7			8	8		9					2	2					1	
		7			8	8		9			11	11	2						1	
		7			8	8		9				11	2						1	
		7			8	8		9					2						1	
12	12	7	7	7	7	7	7	7	7				2						1	
12													2						1	
12													2						1	
12					13	13	13	13	13	13	13	13	2						1	
12																			1	
12																			1	
12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	1

(b)

Figure 2.18 *Continued.*

uses a standard depth-first search algorithm, can be stated as follows:

```
procedure RESOLVE(EQTABLE);
  list_of_components := nil;
  for each unmarked node N in EQTABLE
    current_component := DFS(N,EQTABLE);
    add_to_list(list_of_components,current_component)
  end for
end RESOLVE
```

In this procedure, *list_of_components* is a list that will contain the final resultant equivalence classes. The function DFS performs a depth-first search of the graph beginning at the given node N and returns a list of all the nodes it has visited in the process. It also marks each node as it is visited. A standard depth-first search

algorithm is given in Horowitz and Sahni (1982) and in most other data structures texts.

The main problem with the classical algorithm is the global equivalence table. For large images with many regions, the equivalence table can become very large. On some machines there is not enough memory to hold the table. On other machines that use paging, the table gets paged in and out of memory frequently. For example, on a VAX 11/780 system, the classical algorithm ran (including I/O) in 8.4 seconds with 1791 page faults on one 6000-pixel image but took 5021.0 seconds with 23,674 page faults on one 920,000-pixel image. This motivates algorithms that avoid the use of the large global equivalence table for computers employing virtual memory.

2.3.5 A Space-Efficient Two-Pass Algorithm That Uses a Local Equivalence Table

One solution to the space problem is the use of a small local equivalence table that stores only the equivalences detected from the current line of the image and the line that precedes it. Thus the maximum number of equivalences is the number of pixels per line. These equivalences are then used in the propagation step to the next line. In this case not all the equivalencing is done by the end of the first top-down pass, and a second pass is required for both the remainder of the equivalence finding and for assigning the final labels. The algorithm is illustrated in Fig. 2.19. As in the iterative algorithm, the second pass is bottom-up. This is not required but is done this way for consistency with other algorithms that do require a bottom-up pass, and will be discussed in a later chapter. We will state the general algorithm (Lumia, Shapiro, and Zuniga, 1983) in the same pseudocode we have been using and then describe, in more detail, an efficient run-length implementation.

```
procedure Local_Table_Method
"Top-down pass"
for L := 1 to NLINES do
  begin
    "Initialize local equivalence table for line L."
    EQTABLE := CREATE( );
    "Initialize all labels on line L to zero."
    for P := 1 to NPIXELS do
      LABEL(L,P) := 0
    end for;
    "Process the line."
    for P := 1 to NPIXELS do
      if I(L,P) := 1 then
        begin
          A := NEIGHBORS((L,P));
          if ISEMPTY(A)
          then M := NEWLABEL( )
```

																	1	1	1	1
																				1
							2	2	2	2	2	2								1
							2	2												1
				2	2	2	2	2												1
							2	2	2			4								1
							2	2	2			4								1
							2	2	2	2	2	2	2							1
													2							1
													2							1
													2	2						1
									5			2	2	2						1
		7			8	8		5	5			2	2	2						1
7	7	7			8	8		5					2	2						1
		7			8	8		5			2	2	2							1
		7			8	8		5				2	2							1
		7			8	8		5					2							1
5	5	5	5	5	5	5	5	5	5				2							1
5													2							1
5													2							1
5					2	2	2	2	2	2	2	2	2							1
5																				1
5																				1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Figure 2.19 Results after the top-down pass of the local table method on the binary image of Fig. 2.18(a). Note that on the lines where equivalences were detected, the pixels have different labels from those they had after pass 1 of the classical algorithm. For example, on line 5 the four leading 3s were changed to 2s on the second scan of that line, after the equivalence of labels 2 and 3 was detected. The bottom up pass will now propagate the label 1 to all pixels of the single connected component.

```
        else M := MIN(LABELS(A) );
        LABEL(L,P) := M;
        for X in LABELS(A) and X <> M
          ADD (X,M, EQTABLE)
        end for
      end
end for;

"Find equivalence classes detected on this line."

EQCLASSES := Resolve(EQTABLE);
```

```
      for E in EQCLASSES do
        EQLABEL(E) := MIN(LABELS(E))
      end for;
      "Relabel the parts of line L with their equivalence class labels."
      for P := 1 to NPIXELS do
        if I(L,P) := 1
        then LABEL(L,P) := EQLABEL(CLASS(LABEL(L,P)))
      end for
    end
  end for;
  "Bottom-up pass"
  for L := NLINES to 1 by -1 do
    begin
      "Initialize local equivalence table for line L."
      EQTABLE := CREATE( );
      "Process the line."
      for P := 1 to NPIXELS do
        if LABEL(L,P) <>0 then
          begin
            LA := LABELS(NEIGHBORS(L,P));
            for X in LA and X <> LABEL(L,P)
              ADD (X,LABEL(L,P), EQTABLE)
            end for
          end
      end for
    end
  end for
  "Find equivalence classes."
  EQCLASSES := Resolve(EQTABLE);
  for E in EQCLASSES do
    EQLABEL(E) := MIN(LABELS(E))
  end for ;
  "Relabel the pixels of line L one last time."
  for P := 1 to NPIXELS do
    if LABEL(L,P) <> 0
    then LABEL(L,P) := EQLABEL(CLASS(LABEL(L,P)))
  end for
  end Local_Table_Method
```

In comparison with the classical algorithm, the local table method took 8.8 seconds with 1763 page faults on the 6000-pixel image, but only 626.83 seconds with 15,391 page faults on the 920,000-pixel image, which is 8 times faster. For an

even larger 5,120,000-pixel image, the local table method ran 31 times faster than the classical method.

2.3.6 An Efficient Run-Length Implementation of the Local Table Method

In many industrial applications the image used is from a television camera and thus is roughly 512 × 512 pixels, or 260K, in size. On an image half this size, the local table method as implemented on the VAX 11/780 took 116 seconds to execute, including I/O time. But industrial applications often require times of less than one second. To achieve this kind of efficiency, the algorithm can be implemented on a machine with some special hardware capabilities. The hardware is used to rapidly extract a run-length encoding of the image, and the software implementation can then work on the more compact run-length data. Ronse and Devijver (1984) advocate this approach.

A *run-length encoding* of a binary image is a list of contiguous typically horizontal runs of 1-pixels. For each run, the location of the starting pixel of the run and either its length or the location of its ending pixel must be recorded. Figure 2.20 shows the run-length data structure used in our implementation. Each run in the image is encoded by its starting- and ending-pixel locations. (ROW, START_COL) is the location of the starting pixel, and (ROW, END_COL) is the location of the ending pixel, PERM_LABEL is the field in which the label of the connected component to which this run belongs will be stored. It is initialized to zero and assigned temporary values in pass 1 of the algorithm. At the end of pass 2, the PERM_LABEL field contains the final, permanent label of the run. This structure can then be used to output the labels back to the corresponding pixels of the output image.

Consider a run P of 1-pixels. During pass 1, when the run has not yet been fully processed, PERM_LABEL(P) will be zero. After run P has been processed and determined to be adjacent to some other run Q on the previous row, it will be assigned the current label of Q, PERM_LABEL(Q). If it is determined to be adjacent to other runs $Q_1, Q_2, \ldots, Q_K$ also on the previous row, then the equivalence of PERM_LABEL(Q), PERM_LABEL(Q_1), PERM_LABEL(Q_2), . . . , PERM_LABEL(Q_K) must be recorded. The data structures used for recording the equivalences are shown in Fig. 2.21. For a given run P, PERM_LABEL(P) may be zero or nonzero. If it is nonzero, then LABEL(PERM_LABEL(P)) may be zero or nonzero. If it is zero, then PERM_LABEL(P) is the current label of the run and there is no equivalence class. If it is nonzero, then there is an equivalence class and the value of LABEL(PERM_LABEL(P)) is the label assigned to that class. All the labels that have been merged to form this class will have the same class label; that is, if run P and run P′ are in the same class, LABEL(PERM_LABEL(P)) = LABEL(PERM_LABEL(P′)). When such an equivalence is determined, if each run was already a member of a class and the two classes were different, the two classes are merged. This is accomplished by linking together each prior label belonging

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

(a)

	ROW_START	ROW_END
1	1	2
2	3	4
3	5	6
4	0	0
5	7	7

(b)

	ROW	START_COL	END_COL	PERM_LABEL
1	1	1	2	0
2	1	4	5	0
3	2	1	2	0
4	2	5	5	0
5	3	1	3	0
6	3	5	5	0
7	5	2	5	0

(c)

Figure 2.20 Binary image (a) and its run-length encoding (b) and (c). Each run of 1-pixels is encoded by its row (ROW) and the columns of its starting and ending pixels (START_COL and END_COL). In addition, for each row of the image, ROW_START points to the first run of the row and ROW_END points to the last run of the row. The PERM_LABEL field will hold the component label of the run; it is initialized to zero.

to a single class in a linked list pointed to by EQ_CLASS(L) for class label L and linked together using the NEXT field of the LABEL/NEXT structure. To merge two classes, the last cell of one is made to point to the first cell of the second, and the LABEL field of each cell of the second is changed to reflect the new label of the class.

In this implementation the value of the minimum label does not always become the label of the equivalence class. Instead, a single-member class is always merged into and assigned the label of a multimember class. This allows the algorithm to avoid traversing the linked list of the larger class to change all its labels when only one element is being added. Only when both classes are multimember is one of the labels selected. In this case the first label of the two being merged is selected as the equivalence class field. The equivalencing procedure, make_equivalent, is given below.

procedure make_equivalent (I1, I2);

"I1 is the value of PERM_LABEL(R1) and I2 is the value of PERM_LABEL(R2) for two different runs R1 and R2. They have been detected to be

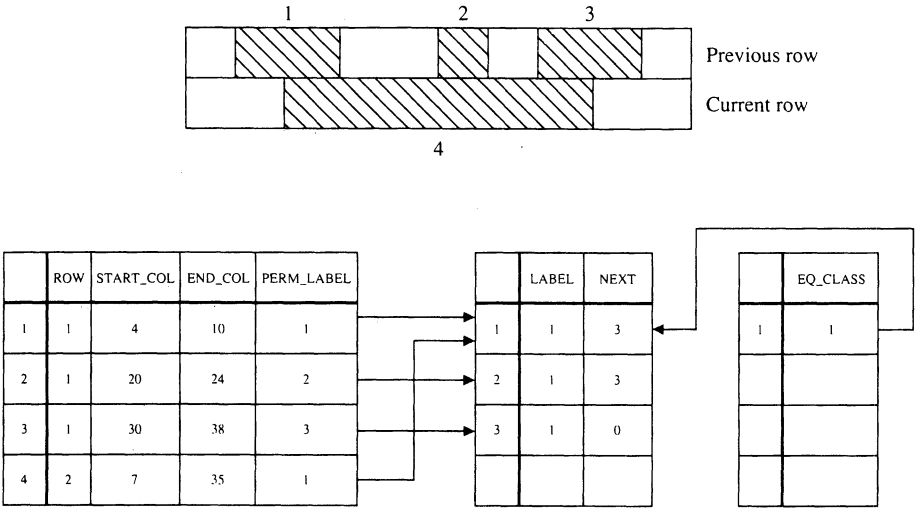


Figure 2.21 Data structures used for keeping track of equivalence classes. In this example, run 4 has PERM_LABEL 1, which is an index into the LABEL array, that gives the equivalence class label for each possible PERM_LABEL value. In the example, PERM_LABELS 1,2, and 3 have all been determined to be equivalent, so LABEL(1), LABEL(2), and LABEL(3) all contain the equivalence class label, which is 1. Furthermore, the equivalence class label is an index into the EQ_CLASS array that contains pointers to the beginnings of the equivalence classes that are linked lists in the LABEL/NEXT structure. In this example there is only one equivalence class, class 1, and three elements of the LABEL/NEXT array are linked together to form this class.

equivalent. The purpose of this routine is to make them equivalent in the data structures."

case

LABEL(I1) = 0 and LABEL(I2) = 0:

"Both classes have only one member. Create a new class with I1 as the label."

begin
LABEL(I1) := I1;
LABEL(I2) := I1;
NEXT(I1) := I2;
NEXT(I2) := 0;
EQ_CLASS(I1) := I1
end;

LABEL(I1) = LABEL(I2):

"Both labels already belong to the same class."

return;

LABEL(I1) <> 0 and LABEL(I2) = 0:

```
    "There is more than one member in the class with label I1,
    but only one in the class with label I2. So add the
    smaller class to the larger."
      begin
        BEGINNING := LABEL(I1);
        LABEL(I2) := BEGINNING;
        NEXT(I2) := EQ_CLASS(BEGINNING);
        EQ_CLASS(BEGINNING) := I2
      end;
    LABEL(I1) = 0 and LABEL(I2) <> 0:
    "There is more than one member in the class with label I2,
    but only one in the class with label I1. Add the smaller class
    to the larger."
      begin
        BEGINNING := LABEL(I2);
        LABEL(I1) := BEGINNING;
        NEXT(I1) := EQ_CLASS(BEGINNING);
        EQ_CLASS(BEGINNING) := I1
      end;
    LABEL (I1) <> 0 and LABEL (I2) <> 0:
    "Both classes are multimember. Merge them by linking the first
    onto the end of the second, and assign label I1."
      begin
        BEGINNING := LABEL(I2);
        MEMBER := EQ_CLASS(BEGINNING);
        EQ_LABEL := LABEL(I1);
        while NEXT(MEMBER) <> 0 do
          LABEL(MEMBER) := EQ_LABEL;
          MEMBER := NEXT(MEMBER)
        end while;
        LABEL(MEMBER) := EQ_LABEL;
        NEXT(MEMBER) := EQ_CLASS(EQ_LABEL);
        EQ_CLASS(EQ_LABEL) := EQ_CLASS(BEGINNING);
        EQ_CLASS(BEGINNING) := 0
      end
end case;
return
end make_equivalent
```

With this procedure, the run length implementation is as follows:

```
procedure Run_Length_Implementation
"Initialize PERM_LABEL array."
for R := 1 to NRUNS do
```

```
    PERM_LABEL(R) := 0
  end for;
  "Top-down pass"
  for L := 1 to NLINES do
    begin
      P := ROW_START(L);
      PLAST := ROW_END(L);
      if L = 1
      then begin Q := 0; QLAST := 0 end
      else begin Q := ROW_START(L-1); QLAST := ROW_END(L-1) end;
      if P <> 0 and Q <> 0
      then INITIALIZE_EQUIV( );
      "SCAN 1"
      "Either a given run is connected to a run on the previous row or
      it is not. If it is, assign it the label of the first run to which it
      is connected. For each subsequent run of the previous
      row to which it is connected and whose label is different from its
      own, equivalence its label with that run's label."
      while P<= PLAST and Q <= QLAST do
        "Check whether runs P and Q overlap."
        case
          END_COL(P) < START_COL(Q):
          "Current run ends before start of run on previous row."
            P := P +1;
          END_COL(Q) < START_COL(P):
          "Current run begins after end of run on previous row."
            Q := Q + 1;
          else :
          "There is some overlap between run P and run Q."
            begin
              PLABEL := PERM_LABEL(P);
              case
                PLABEL = 0:
                "There is no permanent label yet; assign Q's label."
                  PERM_LABEL(P) := PERM_LABEL(Q);
                PLABEL <> 0 and PERM_LABEL(Q) <> PLABEL;
                "There is a permanent label that is different from the
                label of run Q; make them equivalent."
                  make_equivalent(LABEL, PERM_LABEL(Q));
              end case;
              "Increment P or Q or both as necessary."
```

```
            case
              END_COL(P) > END_COL(Q):
                Q := Q+1;
              END_COL(Q) > END_COL(P);
                P := P + 1;
              END_COL(Q) = END_COL(P):
                begin Q := Q+1; P := P+1 end;
            end case
          end
      end case
    end while;
    "SCAN 2"
    "Make a second scan through the runs of the current row.
    Assign new labels to isolated runs and the labels of their
    equivalence classes to all the rest."
    P := ROWSTART(L);
    while P <= PLAST do
      begin
        PLABEL := PERM_LABEL(P);
        case
          PLABEL = 0:
          "No permanent label exists yet, so assign one."
            PERM_LABEL(P) := NEW_LABEL( );
          PLABEL <> 0 and LABEL(PLABEL) <> 0:
          "Permanent label and equivalence class;
          assign the equivalence class label."
            PERM_LABEL(P):=LABEL(PLABEL);
        end case;
        P := P + 1
      end
    end while
"Bottom-up pass"
for L := NLINES to 1 by -1 do
  begin
    P := ROW_START(L);
    PLAST := ROW_END(L);
    if L = NLINES
    then begin Q := 0; QLAST := 0 end
    else begin Q := ROW_START(L+1); QLAST := ROW_END(L+1) end
    if P <> 0 and Q <> 0
    then INITIALIZE_EQUIV( );
    "SCAN 1"
    while P ≤ PLAST and Q ≤ QLAST do
```

```
case
  END_COL(P) < START_COL(Q):
    P := P+1;
  END_COL(Q) < START_COL(P):
    Q := Q+1
  else :
  "There is some overlap; if the two adjacent runs have different labels,
  then assign Q's label to run P."
    begin
      if PERM_LABEL(P) <> PERM_LABEL(Q) then
        begin
          LABEL(PERM_LABEL(P)) := PERM_LABEL(Q);
          PERM_LABEL(P) := PERM_LABEL(Q)
        end;
      "Increment P or Q or both as necessary."
      case
        END_COL(P) > END_COL(Q):
          Q := Q + 1
        END_COL(Q) > END_COL(P):
          P := P+1 ,
        END_COL(Q) = END_COL(P):
          begin Q := Q+1; P := P+1 end
      end case;
    end
  end case
end while
"SCAN 2"
P := ROW_START(L);
while P ≤ PLAST do
  "Replace P's label by its class label."
  if LABEL(PERM_LABEL(P)) <> 0
  then PERM_LABEL(P) := LABEL(PERM_LABEL(P));
end while
end
end Run_Length_Implementation
```

There is one other significant difference between procedure Run_Length_Implementation and procedure Local_Table_Method besides the data structures. Procedure Local_Table_Method computed equivalence classes using Resolve both in the top-down pass and in the bottom-up pass. Procedure Run_Length_Implementation updates equivalence classes in the top-down pass using make_equivalent, but it only propagates and replaces labels in the bottom-up pass. This gives correct results not only in procedure Run_Length_Implementation but also back in pro cedure Local_Table_Method. To prove this, note that the last row of the image is labeled correctly by the two algorithms. That is, after the top-down pass, each pixel

in the last row has its final label. (This was proved in Lumia, Shapiro, and Zuniga, 1983.)

Now consider the bottom-up pass. The bottom row is already correctly labeled. Suppose k rows have been correctly labeled and the algorithm proceeds with row $k+1$ (from the bottom). Further suppose there are two pixels p and q on row $k+1$ that are part of the same connected component but have different labels from the top-down pass. Then the bridge between p and q that connects them must certainly be below them, since if it were above them, they would have the same label from the propagation and equivalencing efforts of the top-down pass. Since the bridge is below, there must be pixels on row k that form part of the bridge and whose labels will propagate to p and q. But row k is correctly labeled. So these bridge pixels all have the same final label, and that one label will become the label of both p and q. Therefore only propagation is necessary on the bottom-up pass in both algorithms to take care of this situation.

A second situation occurs when row $k+1$ has two pixels p and q that are part of the same connected component and, although they have the same label from the top-down pass, it is not the final label because of some bridging that occurs for the component segment from below. In this case at least one of the pixels p or q must bridge to a correctly labeled neighboring pixel on row k from the bottom. Then one

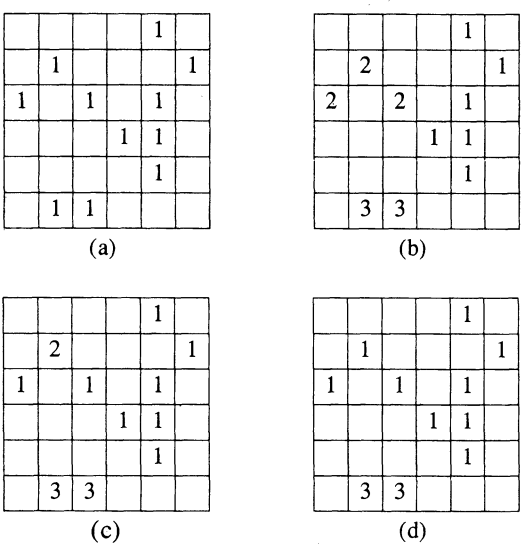


Figure 2.22 Diagram showing how the bottom-up pass needs only to propagate and replace the label 1 to the two pixels having label 2 on the fourth row from bottom. Note that one of these label-2 pixels is not connected to a label-1 pixel. This means that the neighborhood propagate only will not work for the bottom-up pass. After the 1-labeled pixel of row 4, column 4, propagates its label to the 2-labeled pixel of row 3, column 3, the replace operation must replace the label of the 1-labeled pixel of row 3, column 1, to label 1. (a) Binary image; (b) labeling after top-down pass; (c) labeling after processing the first four bottom rows; (d) final labeling after bottom-up pass.

pixel, say p, will get a correct label from the neighborhood propagate operation. In order for q to get the correct label, the replace operation must be performed in which each pixel on row $k + 1$ from the bottom that has the same label as q gets its label replaced by the label that the neighborhood propagate operation gave pixel p. This situation is illustrated in Fig. 2.22.

2.4 Signature Segmentation and Analysis

Signature analysis, like connected components analysis, performs a unit change from the pixel to the region or segment. Signature segmentation consists of taking one or more projections of a binary image or a subimage of a binary image, segmenting each projection to form the new higher-level unit, and taking property measurements of each projection segment. Signature analysis consists of generating the binary image by a thresholding technique, taking projections, and finally making decisions about the objects on the basis of properties of the projections. Projections can be vertical, horizontal, diagonal, circular, radial, spiral, or general projections. Signature analysis was first used in character recognition. It is important for binary vision because it can easily be computed in real time. Sanz, Hinkle, and Dinstein (1985) and Sanz and Dinstein (1987) discuss one kind of a pipeline architecture to compute projections and projection-based geometric features.

A general projection of a given binary image can be produced by masking a projection index image with the given binary image. Each pixel of the resulting image has a value of 0 if the corresponding pixel on the binary image is a 0. And if the pixel of the binary image is a 1, then the pixel will take the value of the corresponding pixel of the projection index image. The *signature*, which is a projection, is the histogram of the nonzero pixels of the resulting masked image. To produce a vertical projection, the projection index image contains the value c in every pixel having column coordinate c. Thus a vertical projection is a one-dimensional function that for each column has a value given by the number of pixels in the binary image in the column having value binary 1. Figure 2.23 illustrates a 45° diagonal projection.

In signature analysis the binary image must be processed in such a way that the clutter of all noninteresting object entities is eliminated before projections are taken. This could be done partly by the gray scale processing preceding the thresholding operation, partly by binary morphological operations (discussed in Chapter 5) after thresholding, and partly by simple masking.

If the clutter has been successfully eliminated and the objects are known to be separated horizontally, then a vertical projection will be most useful. If the objects are known to be separated vertically, then a horizontal projection will be most useful. If a window, called a bounding box, can be found that is guaranteed to contain one and only one object, then all projections relative to this window would have potential use. O'Gorman and Sanderson (1986) discuss a converging squares projection technique to locate centroids and bounding boxes of image regions.

The next stage of signature segmentation is a projection segmentation. The segmentation accomplishes a transformation of the pixel as a unit to the projection segment as a unit. Successive mutually exclusive segments can be determined by

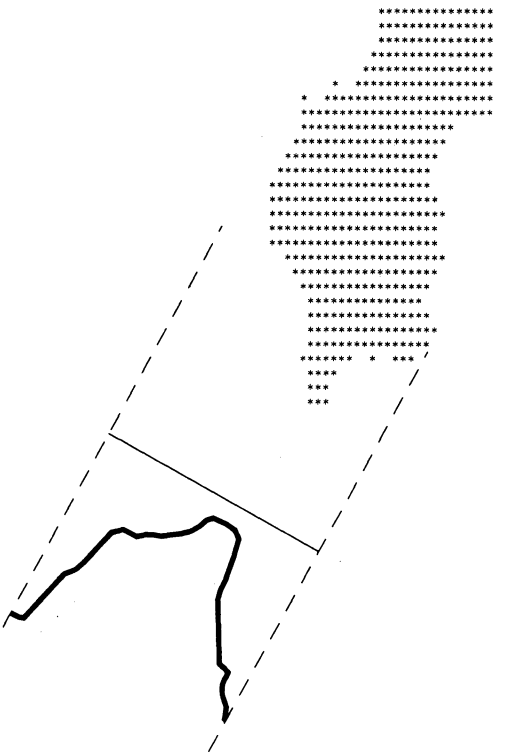


Figure 2.23 Diagonal projection of a shape. The direction of the projection is 45° counterclockwise from the column axis.

locating places where the projection values are relatively small. For example, those locations having a vertical projection with sufficiently or relatively small values are indicative of columns that have a very small number of pixels with binary 1. Hence they are likely to arise from the projection of object ends. A similar statement can be made about horizontal, diagonal, or arbitrary projections. These locations constitute endpoints of intervals that mark the projected boundary of objects of interest. Thus the segmentation process produces projection segments bounded by zero or low-valued projection counts and whose projection counts within the segment tend to be unimodal. These projection segments constitute the new units for the next step of the analysis.

Projection segmentation induces a segmentation of the image in the following way. Suppose that one segment determined from the vertical projection is given by $\{c | s \leq c \leq t\}$ and that one segment determined from the horizontal projection is given by $\{r | u \leq r \leq v\}$. These vertical and horizontal segments naturally define a segment R of the image by

$$R = \{(r, c) \mid u \leq r \leq v \text{ and } s \leq c \leq t\}$$

If there are N_H segments of the horizontal projection and N_V segments of the vertical projection, then $N_H N_V$ mutually exclusive segments will be induced on the image, each segment being a rectangular subimage.

This induction of a segmentation on the image from its vertical and horizontal projection segmentations leads to an iterative way of refining the initial segmentation of the image. The iterative refinement technique works as follows: Segment the vertical and horizontal projections of the image. Induce a segmentation on the image from these segmentations. Now treat each rectangular subimage as the image was treated. Determine its vertical and horizontal projections, segment the projections, and induce the segmentation back onto the subimage. The refinement process can continue in this way until each resulting rectangular subimage has a vertical and horizontal projection that consists of precisely one segment and therefore cannot be further divided.

EXAMPLE 2.1

In the example we consider the binary image shown in Fig. 2.24, which also shows the vertical and horizontal projections of the image. The horizontal projection mask image associated with the horizontal projection is shown in Fig. 2.25. The binary image masked by the horizontal projection mask is shown in Fig. 2.26. The horizontal projection shown in Fig. 2.24 is the histogram of

1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10	10	10	10	10	10	10
11	11	11	11	11	11	11	11	11	11	11	11	11	11
12	12	12	12	12	12	12	12	12	12	12	12	12	12
13	13	13	13	13	13	13	13	13	13	13	13	13	13

Figure 2.24 Binary image and its vertical and horizontal projections. The horizontal projection is the histogram of the binary image masked by the horizontal projection mask of Fig. 2.25. The masked image is shown in Fig. 2.26.

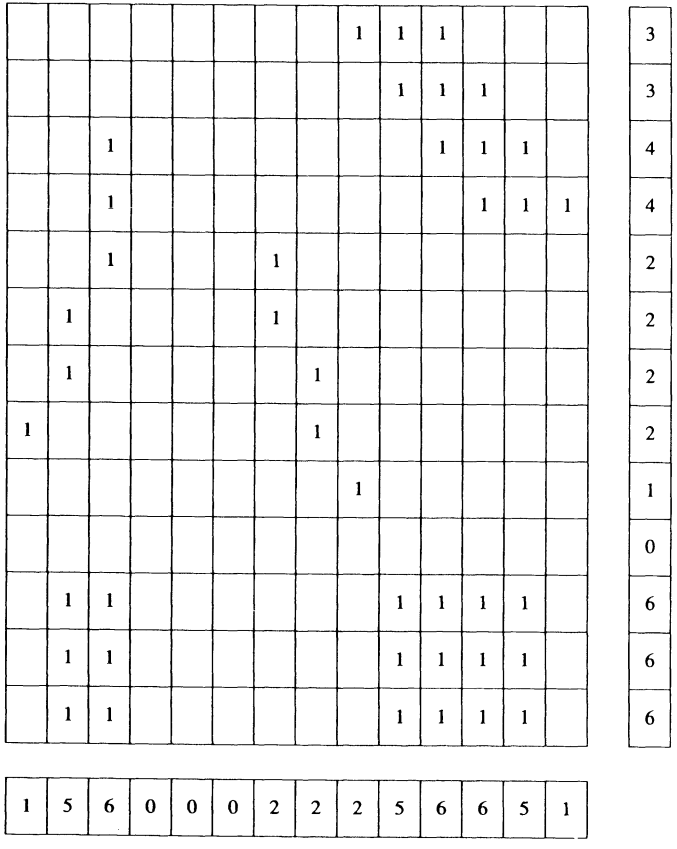
This induction of a segmentation on the image from its vertical and horizontal projection segmentations leads to an iterative way of refining the initial segmentation of the image. The iterative refinement technique works as follows: Segment the vertical and horizontal projections of the image. Induce a segmentation on the image from these segmentations. Now treat each rectangular subimage as the image was treated. Determine its vertical and horizontal projections, segment the projections, and induce the segmentation back onto the subimage. The refinement process can continue in this way until each resulting rectangular subimage has a vertical and horizontal projection that consists of precisely one segment and therefore cannot be further divided.

EXAMPLE 2.1

In the example we consider the binary image shown in Fig. 2.24, which also shows the vertical and horizontal projections of the image. The horizontal projection mask image associated with the horizontal projection is shown in Fig. 2.25. The binary image masked by the horizontal projection mask is shown in Fig. 2.26. The horizontal projection shown in Fig. 2.24 is the histogram of

1	1	1	1	1	1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9	9	9	9	9	9	9
10	10	10	10	10	10	10	10	10	10	10	10	10	10
11	11	11	11	11	11	11	11	11	11	11	11	11	11
12	12	12	12	12	12	12	12	12	12	12	12	12	12
13	13	13	13	13	13	13	13	13	13	13	13	13	13

Figure 2.24 Binary image and its vertical and horizontal projections. The horizontal projection is the histogram of the binary image masked by the horizontal projection mask of Fig. 2.25. The masked image is shown in Fig. 2.26.

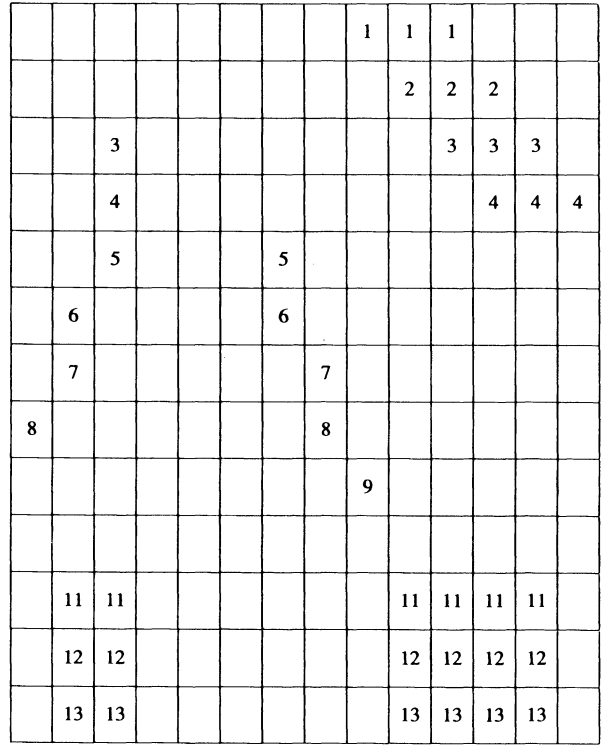


								1	1	1			
									1	1	1		
		1								1	1	1	
		1									1	1	1
		1				1							
	1					1							
	1						1						
1							1						
								1					
	1	1							1	1	1	1	
	1	1							1	1	1	1	
	1	1							1	1	1	1	

3
3
4
4
2
2
2
2
1
0
6
6
6

1	5	6	0	0	0	2	2	2	5	6	6	5	1

Figure 2.25 Horizontal projection mask.



								1	1	1			
									2	2	2		
		3								3	3	3	
		4									4	4	4
		5				5							
	6					6							
	7						7						
8							8						
								9					
	11	11							11	11	11	11	
	12	12							12	12	12	12	
	13	13							13	13	13	13	

Figure 2.26 Binary image of Fig. 2.24 masked by the horizontal projection mask shown in Fig. 2.25.

the nonzero pixels of the horizontally masked image. The vertical projection can be understood in a similar manner.

Each projection of Fig. 2.24 has one segment with zero value surrounded by nonzero-value segments. This permits the image to be segmented once vertically and once horizontally. This segmentation is shown in Fig. 2.27.

Figure 2.27 also shows the vertical and horizontal projections for each of the four image regions arising from the initial segmentation. Only the horizon-

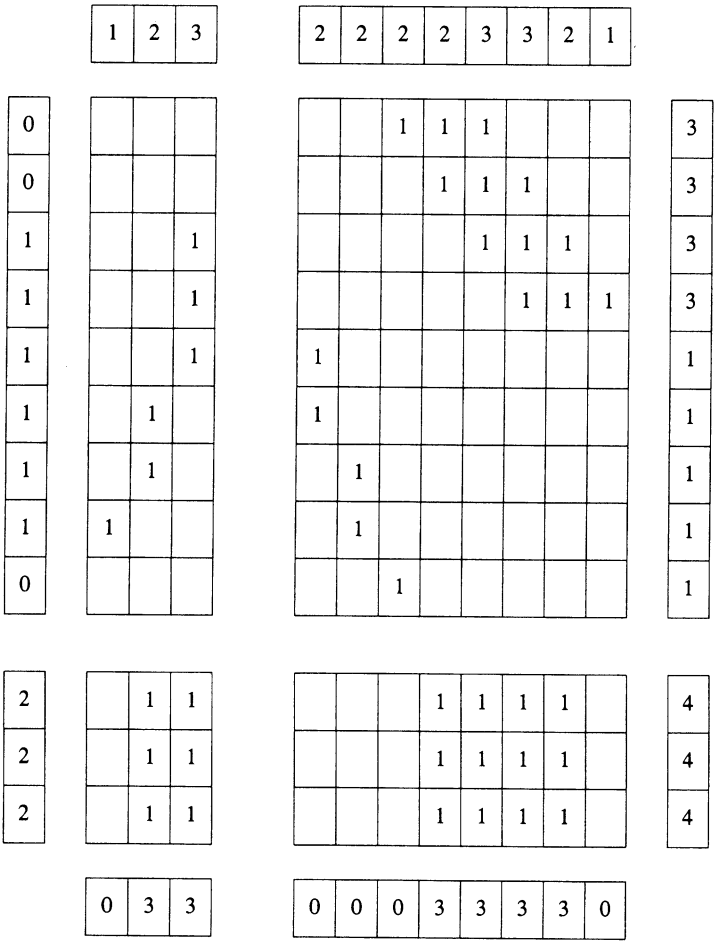


Figure 2.27 Binary image segmented into regions on the basis of the segmentation of the initial vertical and horizontal projections. Also shown are the vertical and horizontal projections of each region.

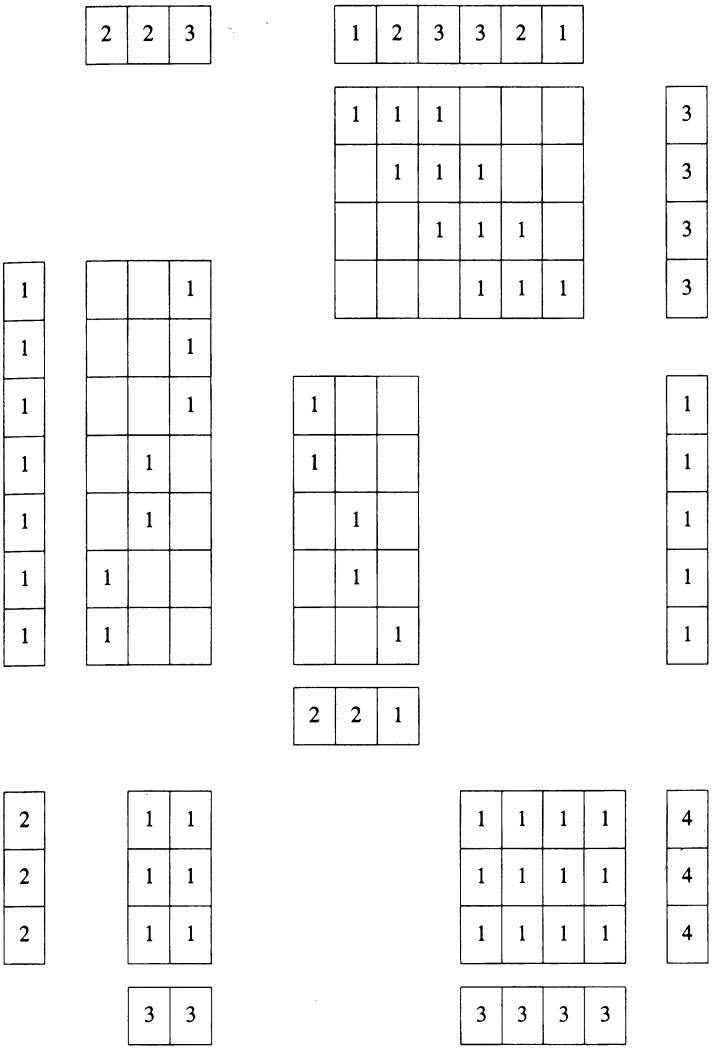


Figure 2.28 Binary image segmented into regions on the basis of the segmentation of Fig. 2.27. Also shown are the vertical and horizontal projections of each region.

tal projection for the upper right region has a zero value area surrounded by nonzero values. This suggests that the upper right region can be further divided horizontally into two pieces. But all the other regions cannot be divided any further. The final segmentation and its vertical and horizontal projections are shown in Fig. 2.28.

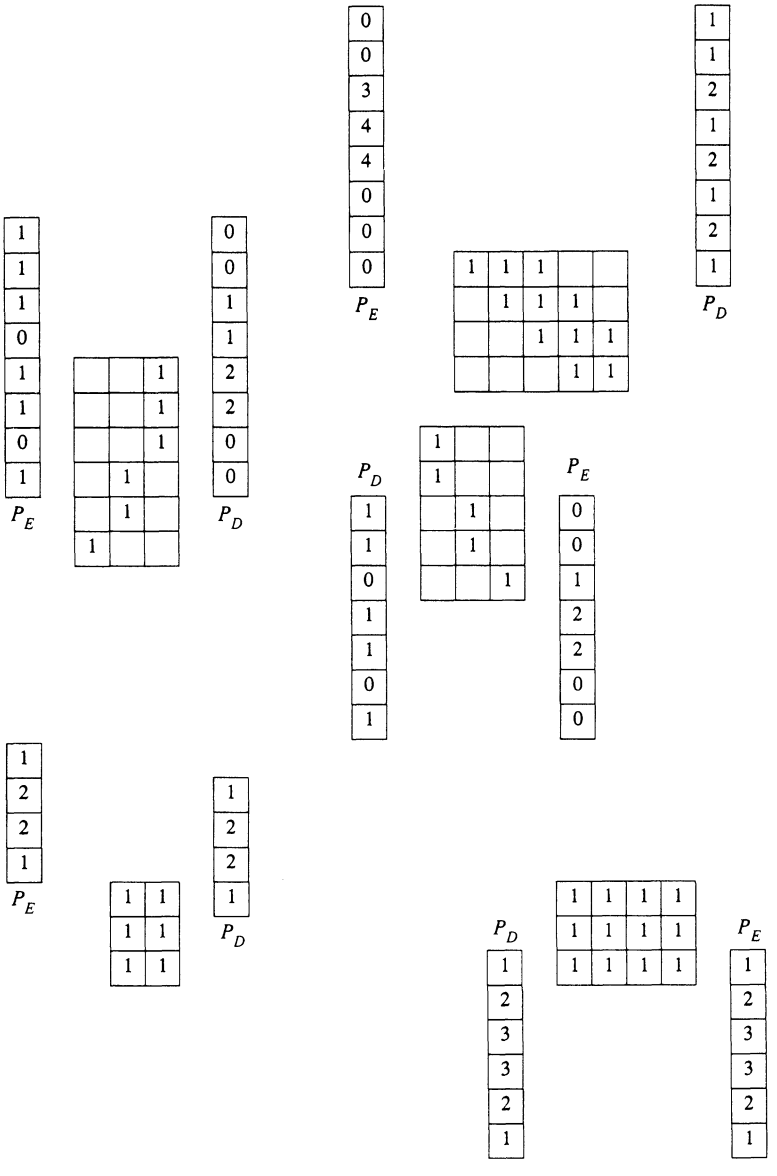


Figure 2.29 Diagonal projections P_D and P_E for each of the five image regions of Fig. 2.28. P_D is the diagonal projection taken 45° clockwise from the horizontal. P_E is the diagonal projection taken 135° clockwise from the horizontal.

Figure 2.29 shows the diagonal projections for each of the five image regions. From the vertical, horizontal, and diagonal projections, the centroids, second central moments, and bounding rectangle can be easily computed. Such computations are described in Chapter 3.

The final phase of signature analysis measures features of each of the projection segments. One feature can consist of the sum of all the projection values in the segment. This feature gives object area. Another is the weighted sum of all projection positions in the segment, weighted by the projection value. This produces a central projected position. Others can include numbers and heights of peaks and numbers and depths of valleys. If the shape of the projection is known to be sufficiently simple, a fit of a suitable functional form to the segment projection values can be made. A feature vector can be constructed from the computed parameters of the fit. Finally, the projection segment can be normalized to have a prespecified length, and the normalized projection values can constitute a projection measurement vector. Computation of signature properties is described in Chapter 3.

Thus after signature analysis, to each projection segment we have an N-tuple or vector of its measurement properties. To recognize different objects, object defects, or characters is to distinguish between their projected segments on the basis of their measurement properties. This is the role of statistical pattern recognition, which is discussed in Chapter 4.

2.5 Summary

When the component segmentation is done in a simple situation—simple meaning that the components are spaced away from each other and there are relatively few components—then signature segmentation is the technique of choice because it has a faster implementation than the connected components analysis techniques, both in special pipeline hardware and in standard computer hardware. However, when there are many components and they are near one another, with protrusions that wiggle into another component's bays, signature segmentation will not work. In this case connected component analysis must be used.

Exercises

2.1. Use a pseudorandom-number generator to generate 1000 numbers from a Gaussian distribution having mean 100 and standard deviation 10. Then generate another 1000 numbers from a Gaussian distribution having mean 150 and standard deviation 10. Round the resulting 2000 numbers to integers. Determine their histogram.

2.2. Write a program that inputs a histogram and evaluates the location of a threshold by using the Otsu technique. Execute this program on the histogram generated in Exercise 2.1.

2.3. Write a program that inputs a histogram and outputs the location of a threshold by using the Kittler-Illingworth technique. Execute this program on the histogram generated in Exercise 2.1.

2.4. Use a pseudorandom-number generator to generate 1000 numbers from a Gaussian distribution having mean 100 and standard deviation 10. Then generate 5000 numbers from a Gaussian distribution having mean 150 and standard deviation 10. Round the resulting 6000 numbers to integers. Determine their histogram.

2.5. Compare the locations where the Otsu and the Kittler-Illingworth techniques locate the threshold. Which produces the better threshold value, and why is it better?

2.6. Write a program to determine the connected components of a binary image by using the classical algorithm. Perform some experiments to determine how long the algorithm takes to execute as a function of image size and the fraction of binary-1 pixels the image has.

2.7. Write a program to determine the connected component of a binary image by using the algorithm of Section 2.3.5. Determine how long the algorithm takes to execute as a function of image size and the fraction of binary-1 pixels the image has.

2.8. Write a program to determine the connected components of a binary image by first run-length encoding the binary image and then using the run-length algorithm of Section 2.3.5. Determine how long the algorithm takes to execute as a function of image size and the fraction of binary-1 pixels the image has.

2.9. Write a program that can take the horizontal, vertical, and diagonal projections of a binary image.

2.10. Write a program that uses the horizontal, vertical, and diagonal projections to perform a signature segmentation. Apply this segmentation procedure to binary images that you create having rectangular, square, and circular objects. How many and how large do the objects have to be before the signature segmentation procedure is not able to fully segment the image?

Bibliography

Abutaleb, A. S., "Automatic Thresholding of Gray-Level Pictures Using Two-Dimensional Entropy," *Computer Vision, Graphics, and Image Processing,* Vol. 47, 1989, pp. 22–32.

Agin, G., "Computer Vision System for Industrial Inspection and Assembly," *IEEE Transactions on Computers,* Vol. 13, 1980, pp. 11–20.

Aho, A. V., J. E. Hopcroft, and J.D. Ullman, *Data Structures and Algorithms,* Addison-Wesley, Reading, MA, 1983.

Ahuja, N., and A. Rosenfeld, "A Note on the Use of Second-Order Gray-Level Statistics for Threshold Selection," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-8, 1978, pp. 895–898.

Bartneck, N., "A General Data Structure for Image Analysis Based on a Description of Connected Components," *Computing,* Vol. 42, 1989, pp. 17–34.

Carlotto, M. J., "Histogram Analysis Using a Scale-Space Approach," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-9, 1987, pp. 121–129.

Cho, S., R. M. Haralick, and S. Yi, "Improvement of Kittler and Illingworth's Minimum Error Thresholding," *Pattern Recognition,* Vol. 22, 1989, pp. 609–617.

Cunningham, R., "Segmenting Binary Images," *Robotics Age,* July 1981, pp. 4–19.

Danielsson, P. E., "An Improved Segmentation and Coding Algorithm for Binary and Non-Binary Images," *IBM Journal of Research and Development,* Vol. 26, 1982, pp. 698–707.

Derin, H., "Estimating Components of Univariate Gaussian Mixtures using Prony's Method," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-9, 1987, pp. 142–148.

Dunn, S. M., D. Harwood, and L. S. Davis, "Local Estimation of the Uniform Error Threshold," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-6, 1984, pp. 742–747.

Gleason, G. J., and G. J. Agin, "A Modular Vision System for Sensor Controlled Manipulation and Inspection," *Proceedings of the Ninth International Symposium on Industrial Robots,* Washington, DC, 1979, pp. 57–70.

Haralick, R. M., "Some Neighborhood Operators," *Real-Time Parallel Computing Image Analysis,* M. Onoe, K. Preston, Jr., and A. Rosenfeld (eds.), Plenum, New York, 1981.

Horowitz, E., and Sahni, *Fundamentals of Data Structures,* Computer Science Press, Rockville, MD, 1982.

Kapur, J. N., P. K. Sahoo, and A.K.C. Wong, "A New Method for Gray-Level Picture Thresholding Using the Entropy of the Histogram," *Computer Vision, Graphics, and Image Processing,* Vol. 29, 1985, pp. 273–285.

Kirby, R. L., and A. Rosenfeld, "A Note on the use of (Gray Level, Local Average Gray Level) Space as an Aid in Threshold Selection," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-9, 1979, pp. 860–864.

Kittler, J., and J. Illingworth, "On Threshold Selection Using Clustering Criteria," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-15, 1985, pp. 652–655.

Kittler, J., and J. Illingworth, "Minimum Error Thresholding," *Pattern Recognition,* Vol. 19, 1986, pp. 41–47.

Kittler, J., J. Illingworth, and J. Föglein, "Threshold Selection Based on a Simple Image Statistic," *Computer Vision, Graphics, and Image Processing,* Vol. 30, 1985, pp. 125–147.

Kohler, R., "A Segmentation System Based on Thresholding," *Computer Graphics and Image Processing,* Vol. 15, 1981, pp. 319–338.

Kullback, S., *Information Theory and Statistics,* Wiley, New York, 1959.

Lee, J. S., and M. C. K. Yang, "Threshold Selection Using Estimates from Truncated Normal Distribution," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-19, 1989, pp. 422–429.

Lumia, R., L. G. Shapiro, and O. Zuniga, "A New Connected Components Algorithm for Virtual Memory Computers," *Computer Vision, Graphics, and Image Processing,* Vol. 22, 1983, pp. 287–300.

Manohar, M., and H. K. Ramapriyan, "Connected Component Labeling of Binary Images on a Mesh Connected Massively Parallel Processor," *Computer Vision, Graphics, and Image Processing,* Vol. 45, 1989, pp. 133–149.

Mardia, K. V., and T. J. Hainsworth, "A Spatial Thresholding Method for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-10, 1988, pp. 919–927.

Mason, S. J. and J. K. Clemens, "Character Recognition in an Experimental Reading Machine for the Blind," *Recognizing Patterns,* S. Kolers and M. Eden (eds.), MIT Press, Cambridge, MA, 1968, pp. 156–167.

Milgram, D. L., and M. Herman, "Clustering Edge Values for Threshold Selection," *Computer Graphics and Image Processing,* Vol. 10, 1979, pp. 272–280.

Mussio, P., and M. Padula, "An Approach to the Definition, Description, and Extraction of Structures in Binary Digital Images," *Computer Vision, Graphics, and Image Processing,* Vol. 31, 1985, pp. 19–49.

O'Gorman, L., and A. C. Sanderson, "Some Extensions of the Converging Squares Algorithm for Image Feature Analysis," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-8, 1986, pp. 520–524.

Otsu, N., "A Threshold Selection Method from Gray-Level Histograms," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-9, 1979, pp. 62–66.

Pérez, A., and R. C. Gonzalez, "An Iterative Thresholding Algorithm for Image Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-9, 1987, pp. 742–751.

Pun, T., "Entropic Thresholding, A New Approach," *Computer Vision, Graphics, and Image Processing,* Vol. 16, 1981, pp. 210–239.

Ronse, C., and P. A. Devijver, *Connected Components in Binary Images: The Detection Problem,* Research Studies, Letchworth, Herts, England, 1984.

Rosenfeld, A., "Connectivity in Digital Pictures," *Journal of the Association for Computing Machinery,* Vol. 17, 1970, pp. 146–160.

Rosenfeld, A., and J. L. Pfaltz, "Sequential Operations in Digital Picture Processing," *Journal of the Association for Computing Machinery,* Vol. 13, 1966, pp. 471–494.

Sahoo, P. K., *et al.,* "A Survey of Thresholding Techniques," *Computer Vision, Graphics, and Image Processing,* Vol. 41, 1988, pp. 233–260.

Sanz, J., and I. Dinstein, "Projection-Based Geometrical Feature Extraction for Computer Vision: Algorithms in Pipeline Architectures," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-9, 1987, pp. 160–168.

Sanz, J., E. B. Hinkle, and I. Dinstein, "Computing Geometric Features of Digital Objects in General Purpose Image Processing Pipeline Architectures," *Proceedings of the Conference on Computer Vision and Pattern Recognition,* San Francisco, 1985, pp. 265–270.

Tsai, W. H., "Moment-Preserving Thresholding: A New Approach," *Computer Vision, Graphics, and Image Processing,* Vol. 29, 1985, pp. 377–393.

Weszka, J. S., "A Survey of Threshold Selection Techniques," *Computer Graphics and Image Processing,* Vol. 7, 1978, pp. 259–265.

Weszka, J. S., R. N. Nagel, and A. Rosenfeld, "A Threshold Selection Technique," *IEEE Transactions on Computers,* December, 1974, pp. 1322–26.

Weszka, J. S., and A. Rosenfeld, "Threshold Evaluation Techniques," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-8, 1978, pp. 622–628.

——, "Histogram Modification for Threshold Selection," *IEEE Transactions on Systems, Man, and Cybernetics,* Vol. SMC-9, 1979, pp. 38–52.

Wu, A. Y., T. H. Hong, and A. Rosenfeld, "Threshold Selection Using Quadtrees," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* Vol. PAMI-4, 1982, pp. 90–94.