23 Image Matching and Registration

When we compare two images, we are faced with the following basic question: when are two images the same or similar, and how can this similarity be measured? Of course one could trivially define two images I_1, I_2 as being identical when all pixel values are the same (i.e., the difference $I_1 - I_2$ is zero). Although this kind of definition may be useful in specific applications, such as for detecting changes in successive images under constant lighting and camera conditions, simple pixel differencing is usually too inflexible to be of much practical use. Noise, quantization errors, small changes in lighting, and minute shifts or rotations can all create large numerical pixel differences for pairs of images that would still be perceived as perfectly identical by a human viewer. Obviously, human perception incorporates a much wider concept of similarity and uses cues such as structure and content to recognize similarity between images, even when a direct comparison between individual pixels would not indicate any match. The problem of comparing images at a structural or semantic level is a difficult problem and an interesting research field, for example, in the context of image-based searches on the Internet or database retrieval.

This chapter deals with the much simpler problem of comparing images at the pixel level; in particular, localizing a given subimage—often called a "template"—within some larger image. This task is frequently required, for example, to find matching patches in stereo images, to localize a particular pattern in a scene, or to track a certain pattern through an image sequence. The principal idea behind "template matching" is simple: move the given pattern (template) over the search image, measure the difference against the corresponding subimage at each position, and record those positions where the highest similarity is obtained. But this is not as simple as it may initially sound. After all, what is a suitable distance measure, what total difference is acceptable for a match, and what happens when brightness or contrast changes?

We already touched on this problem of invariance under geometric transformations when we discussed the shape properties of seg-

© Springer-Verlag London 2016
W. Burger, M.J. Burge, *Digital Image Processing*, Texts in Computer Science,
DOI 10.1007/978-1-4471-6684-9_23

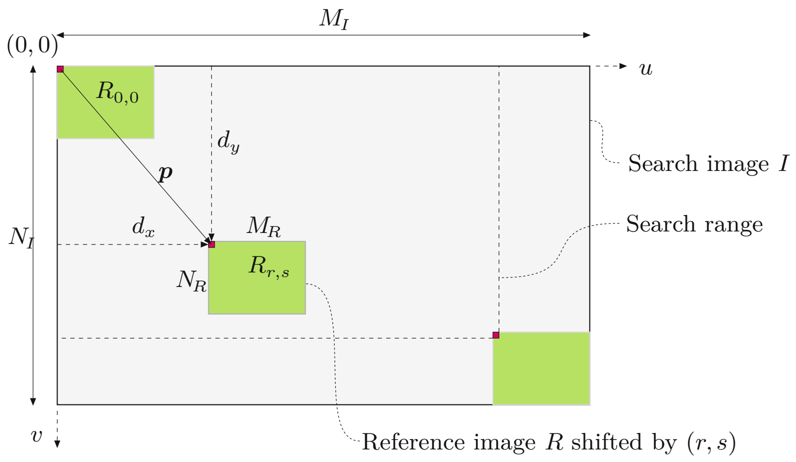


Fig. 23.1
Geometry of template matching. The reference image R is shifted across the search image I by an offset (r, s) using the origins of the two images as the reference points. The dimensions of the search image $(M_I \times N_I)$ and the reference image $(M_R \times N_R)$ determine the maximal search region for this comparison.

mented regions in Chapter 10, Sec. 10.4.2. However, geometric invariance is not our main concern in the remaining part of this chapter, where we describe only the most basic template-matching techniques: correlation-based methods for intensity images and "chamfer-matching" for binary images.

23.1 Template Matching in Intensity Images

First we look at the problem of localizing a given *reference image* (template) R within a larger intensity (grayscale) image I, which we call the *search image*. The task is to find those positions where the contents of the reference image R and the corresponding subimage of I are either the same or most similar. If we denote by

$$R_{r,s}(u,v) = R(u-r, v-s) \tag{23.1}$$

the reference image R *shifted* by the distance (r, s) in the horizontal and vertical directions, respectively, then the matching problem (illustrated in Fig. 23.1) can be summarized as follows:

- Given are the search image I and the reference image R. Find the offset $(r, s) \in \mathbb{Z}^2$ such that the similarity between the shifted reference image $R_{r,s}$ and the corresponding subimage of I is a maximum.

To successfully solve this task, several issues need to be addressed, such as determining a minimum similarity value for accepting a match and developing a good search strategy for finding the optimal displacement. First, and most important, a suitable measure of similarity between subimages must be found that is reasonably tolerant against intensity and contrast variations.

23.1.1 Distance between Image Patterns

To quantify the amount of agreement, we compute a "distance" $\mathrm{d}(r, s)$ between the shifted reference image R and the corresponding subimage of I for each offset position (r, s) (Fig. 23.2). Several distance

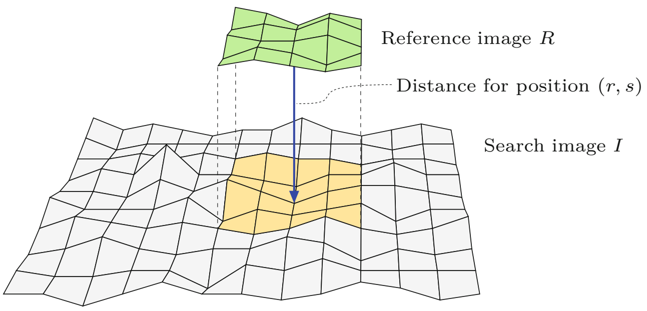


Fig. 23.2
Measuring the distance between 2D image functions. The reference image R is positioned at offset (r, s) on top of the search image I.

measures have been proposed for 2D intensity images, including the following three basic definitions:[1]

Sum of absolute differences:

$$\mathrm{d}_A(r,s) = \sum_{(i,j)\in R} \left| I(r+i, s+j) - R(i,j) \right| . \tag{23.2}$$

Maximum difference:

$$\mathrm{d}_M(r,s) = \max_{(i,j)\in R} \left| I(r+i, s+j) - R(i,j) \right| . \tag{23.3}$$

Sum of squared differences:

$$\mathrm{d}_E(r,s) = \Bigl[\sum_{(i,j)\in R} \bigl(I(r+i, s+j) - R(i,j) \bigr)^2 \Bigr]^{1/2} . \tag{23.4}$$

Note that the expression in Eqn. (23.4) is nothing else but the *Euclidean distance* between two N-dimensional vectors of pixels values. Similarly, the sum of differences in Eqn. (23.2) is equivalent to the L_1 distance, and the maximum difference in Eqn. (23.3) equals the L_∞ distance norm.[2]

Distance and correlation

Because of its formal properties, the N-dimensional distance d_E (Eqn. (23.4)) is of special importance and well-known in statistics and optimization. To find the best-matching position between the reference image R and the search image I, it is sufficient to *minimize the square* of d_E (which is always positive), which can be expanded to

$$\begin{aligned} \mathrm{d}_E^2(r,s) &= \sum_{(i,j)\in R} \bigl(I(r+i, s+j) - R(i,j) \bigr)^2 \qquad (23.5) \\ &= \underbrace{\sum_{(i,j)\in R} I^2(r+i,s+j)}_{A(r,s)} + \underbrace{\sum_{(i,j)\in R} R^2(i,j)}_{B} - 2 \cdot \underbrace{\sum_{(i,j)\in R} I(r+i,s+j)\cdot R(i,j)}_{C(r,s)} . \end{aligned}$$

[1] We use the short notation $(i,j) \in R$ to specify the set of all possible template coordinates, that is, $\{(i,j) \mid 0 \le i < M_R, 0 \le j < N_R\}$.

[2] See also Sec. B.1.2 in the Appendix.

Notice that the term B in Eqn. (23.5) is the sum of the squared pixel values in the reference image R, a constant value (independent of r, s) that can thus be ignored. The term $A(r, s)$ is the sum of the squared values within the subimage of I at the current offset (r, s). $C(r, s)$ is the so-called *linear cross correlation* ($\circledast$) between I and R, which is defined in the general case as

$$(I \circledast R)(r,s) = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} I(r{+}i, s{+}j) \cdot R(i,j), \tag{23.6}$$

which—since R and I are assumed to have zero values outside their boundaries—is, furthermore, equivalent to

$$\sum_{i=0}^{M_R-1} \sum_{j=0}^{N_R-1} I(r{+}i, s{+}j) \cdot R(i,j) = \sum_{(i,j)\in R} I(r{+}i, s{+}j) \cdot R(i,j) \tag{23.7}$$

and thus the same as $C(r, s)$ in Eqn. (23.5). As we can see in Eqn. (23.6), correlation is in principle the same operation as linear *convolution* (see Ch. 5, Eqn. (5.16)), with the only difference being that the convolution kernel ($R(i, j)$ in this case) is implicitly mirrored.

If we assume for a minute that $A(r, s)$—the "signal energy"— in Eqn. (23.5) is constant throughout the image I, then $A(r, s)$ can also be ignored and the position of maximum cross correlation $C(r, s)$ coincides with the best match between R and I. In this case, the minimum of $d_E^2(r, s)$ (Eqn. (23.5)) can be found by computing the maximum value of the correlation $I \circledast R$ only. This could be interesting for practical reasons if we consider that the linear convolution (and thus the correlation) with large kernels can be computed very efficiently in the frequency domain (see also Ch. 19, Sec. 19.5).

Normalized cross correlation

Unfortunately, the assumption made earlier that $A(r, s)$ is constant does not hold for most images, and thus the result of the cross correlation strongly varies with intensity changes in the image I. The *normalized cross correlation* $C_N(r, s)$ compensates for this dependency by taking into account the energy in the reference image and the current subimage:

$$C_N(r,s) = \frac{C(r,s)}{\sqrt{A(r,s) \cdot B}} = \frac{C(r,s)}{\sqrt{A(r,s)} \cdot \sqrt{B}} \tag{23.8}$$

$$= \frac{\sum\limits_{(i,j)\in R} I(r{+}i, s{+}j) \cdot R(i,j)}{\Big[\sum\limits_{(i,j)\in R} I^2(r{+}i, s{+}j)\Big]^{1/2} \cdot \Big[\sum\limits_{(i,j)\in R} R^2(i,j)\Big]^{1/2}} . \tag{23.9}$$

If the values in the search and reference images are all positive (which is usually the case), then the result of $C_N(r, s)$ is always in the range $[0, 1]$, independent of the remaining contents in I and R. In this case, the result $C_N(r, s) = 1$ indicates a maximum match between R and the current subimage of I at the offset (r, s), while $C_N(r, s) =$

0 signals no agreement. Thus the normalized correlation has the additional advantage of delivering a standardized match value that can be used directly (using a suitable threshold between 0 and 1) to decide about the acceptance or rejection of a match position.

In contrast to the "global" cross correlation in Eqn. (23.6), the expression in Eqn. (23.8) is a "local" distance measure. However, it, too, has the problem of measuring the *absolute* distance between the template and the subimage. If, for example, the overall intensity of the image I is altered, then even the result of the normalized cross correlation $C_N(r,s)$ may also change dramatically.

Correlation coefficient

One solution to this problem is to compare not the original function values but the differences with respect to the average value of R and the average of the current subimage of I. This modification turns Eqn. (23.8) into

$$C_L(r,s) = \frac{\sum\limits_{(i,j)\in R} \bigl(I(r{+}i, s{+}j) - \bar{I}_{r,s}\bigr) \cdot \bigl(R(i,j) - \bar{R}\bigr)}{\Bigl[\sum\limits_{(i,j)\in R} \bigl(I(r{+}i, s{+}j) - \bar{I}_{r,s}\bigr)^2\Bigr]^{1/2} \cdot \underbrace{\Bigl[\sum\limits_{(i,j)\in R} \bigl(R(i,j) - \bar{R}\bigr)^2\Bigr]^{1/2}}_{S_R^2 = K\cdot\sigma_R^2}}, \tag{23.10}$$

with the average values $\bar{I}_{r,s}$ and $\bar{R}$ defined as

$$\bar{I}_{r,s} = \frac{1}{K} \cdot \sum_{(i,j)\in R} I(r{+}i, s{+}j) \quad \text{and} \quad \bar{R} = \frac{1}{K} \cdot \sum_{(i,j)\in R} R(i,j), \tag{23.11}$$

respectively, ($K = |R|$ being the size of the reference image R). In statistics, the expression in Eqn. (23.10) is known as the *correlation coefficient*. However, different from the usual application as a global measure in statistics, $C_L(r,s)$ describes a *local*, piecewise correlation between the template R and the current subimage (at offset r,s) of I. The resulting values of $C_L(r,s)$ are in the range $[-1,1]$ regardless of the contents in R and I. Again a value of 1 indicates maximum agreement between the compared image patterns, while -1 corresponds to a maximum mismatch. The term

$$S_R^2 = K \cdot \sigma_R^2 = \sum_{(i,j)\in R} \bigl(R(i,j) - \bar{R}\bigr)^2 \tag{23.12}$$

in the denominator of Eqn. (23.10) is K times the *variance* (σ_R^2) of the values in the template R, which is constant and thus needs to be computed only once. Due to the fact that $\sigma_R^2 = \frac{1}{K} \sum R^2(i,j) - \bar{R}^2$, the expression in Eqn. (23.12) can be reformulated as

$$S_R^2 = \sum_{(i,j)\in R} R^2(i,j) \; - \; K \cdot \bar{R}^2 \tag{23.13}$$

$$= \sum_{(i,j)\in R} R^2(i,j) \; - \; \frac{1}{K} \cdot \Bigl[\sum_{(i,j)\in R} R(i,j)\Bigr]^2. \tag{23.14}$$

By inserting the results from Eqns. (23.11) and (23.14) we can rewrite Eqn. (23.10) as

$$C_L(r,s) = \frac{\sum\limits_{(i,j)\in R} \bigl(I(r{+}i,s{+}j)\cdot R(i,j)\bigr) \;-\; K\cdot \bar{I}_{r,s}\cdot \bar{R}}{\bigl[\sum\limits_{(i,j)\in R} I^2(r{+}i,s{+}j) \;-\; K\cdot \bar{I}^2_{r,s}\bigr]^{1/2}\cdot S_R}, \qquad (23.15)$$

and thereby obtain an efficient way to compute the local correlation coefficient. Since $\bar{R}$ and $S_R = (S_R^2)^{1/2}$ must be calculated only once and the local average of the current subimage $\bar{I}_{r,s}$ is not immediately required for summing up the differences, the whole expression in Eqn. (23.15) can be computed in one common iteration, as shown in Alg. 23.1.

Note that in the calculation of $C_L(r,s)$ in Eqn. (23.15), the denominator becomes zero if any of the two factors is zero. This may happen, for example, if the search image I is locally "flat" and thus has zero variance or if the reference image R is constant. The quantity 1 is added to the denominator in Alg. 23.1 (line 23) to avoid divisions by zero in such cases, which otherwise has no significant effect on the result.

A direct Java implementation of this procedure is shown in Progs. 23.1 and 23.2 in Sec. 23.1.3 (class `CorrCoeffMatcher`).

Examples and discussion

Figure 23.3 compares the performance of the described distance functions in a typical example. The original image (Fig. 23.3(a)) shows a repetitive flower pattern produced under uneven lighting and differences in local brightness. One instance of the repetitive pattern was extracted as the reference image (Fig. 23.3(b)).

- The *sum of absolute differences* (Eqn. (23.2)) in Fig. 23.3(c) shows a distinct peak value at the original template position, as does the *Euclidean distance* (Eqn. (23.4)) in Fig. 23.3(e). Both measures work satisfactorily in this regard but are strongly affected by global intensity changes, as demonstrated in Figs. 23.4 and 23.5.
- The *maximum difference* (Eqn. (23.3)) in Fig. 23.3(d) proves completely useless as a distance measure since it responds more strongly to the lighting changes than to pattern similarity. As expected, the behavior of the *global cross correlation* in Fig. 23.3(f) is also unsatisfactory. Although the result exhibits a *local* maximum at the true template position (hardly visible in the printed image), it is completely dominated by the high-intensity responses in the brighter parts of the image.
- The result from the *normalized cross correlation* in Fig. 23.3(g) appears naturally very similar to the Euclidean distance (Fig. 23.3(e)), because in principle it is the same measure. As expected, the *correlation coefficient* (Eqn. (23.10)) in Fig. 23.3(h) yields the best results. Distinct peaks of similar intensity are produced for all six instances of the template pattern, and the result is unaffected by changing lighting conditions. In this case, the

```
1: CorrelationCoefficient ($I, R$)
       Input: $I(u,v)$, search image; $R(i,j)$, reference image.
       Returns a map $C(r,s)$ containing the values of the correlation
       coefficient between $I$ and $R$ positioned at $(r,s)$.

       Step 1–initialize:
2:     $(M_I, N_I) \leftarrow \mathsf{Size}(I)$
3:     $(M_R, N_R) \leftarrow \mathsf{Size}(R)$
4:     $K \leftarrow M_R \cdot N_R$
5:     $\Sigma_R \leftarrow 0,\ \Sigma_{R2} \leftarrow 0$
6:     for $i \leftarrow 0, \ldots, (M_R - 1)$ do
7:         for $j \leftarrow 0, \ldots, (N_R - 1)$ do
8:             $\Sigma_R \leftarrow \Sigma_R + R(i,j)$
9:             $\Sigma_{R2} \leftarrow \Sigma_{R2} + R^2(i,j)$
10:    $\bar{R} \leftarrow \Sigma_R / K$                                ▷ Eq. 23.11
11:    $S_R \leftarrow (\Sigma_{R2} - K \cdot \bar{R}^2)^{1/2}$          ▷ Eq. 23.14

       Step 2—compute the correlation map:
12:    Create map $C\colon (M_I - M_R + 1) \times (N_I - N_R + 1) \mapsto \mathbb{R}$
13:    for $r \leftarrow 0, \ldots, M_I - M_R$ do              ▷ place $R$ at position $(r,s)$
14:        for $s \leftarrow 0, \ldots, N_I - N_R$ do
               Compute the correlation coefficient for position $(r,s)$:
15:            $\Sigma_I \leftarrow 0,\ \Sigma_{I2} \leftarrow 0,\ \Sigma_{IR} \leftarrow 0$
16:            for $i \leftarrow 0, \ldots, M_R - 1$ do
17:                for $j \leftarrow 0, \ldots, N_R - 1$ do
18:                    $a_I \leftarrow I(r+i, s+j)$
19:                    $a_R \leftarrow R(i,j)$
20:                    $\Sigma_I \leftarrow \Sigma_I + a_I$
21:                    $\Sigma_{I2} \leftarrow \Sigma_{I2} + a_I^2$
22:                    $\Sigma_{IR} \leftarrow \Sigma_{IR} + a_I \cdot a_R$
23:            $C(r,s) \leftarrow \dfrac{\Sigma_{IR} - \Sigma_I \cdot \bar{R}}{1 + \sqrt{\Sigma_{I2} - \Sigma_I^2 / K} \cdot S_R}$
24:    return $C$                                               ▷ $C(r,s) \in [-1,1]$
```

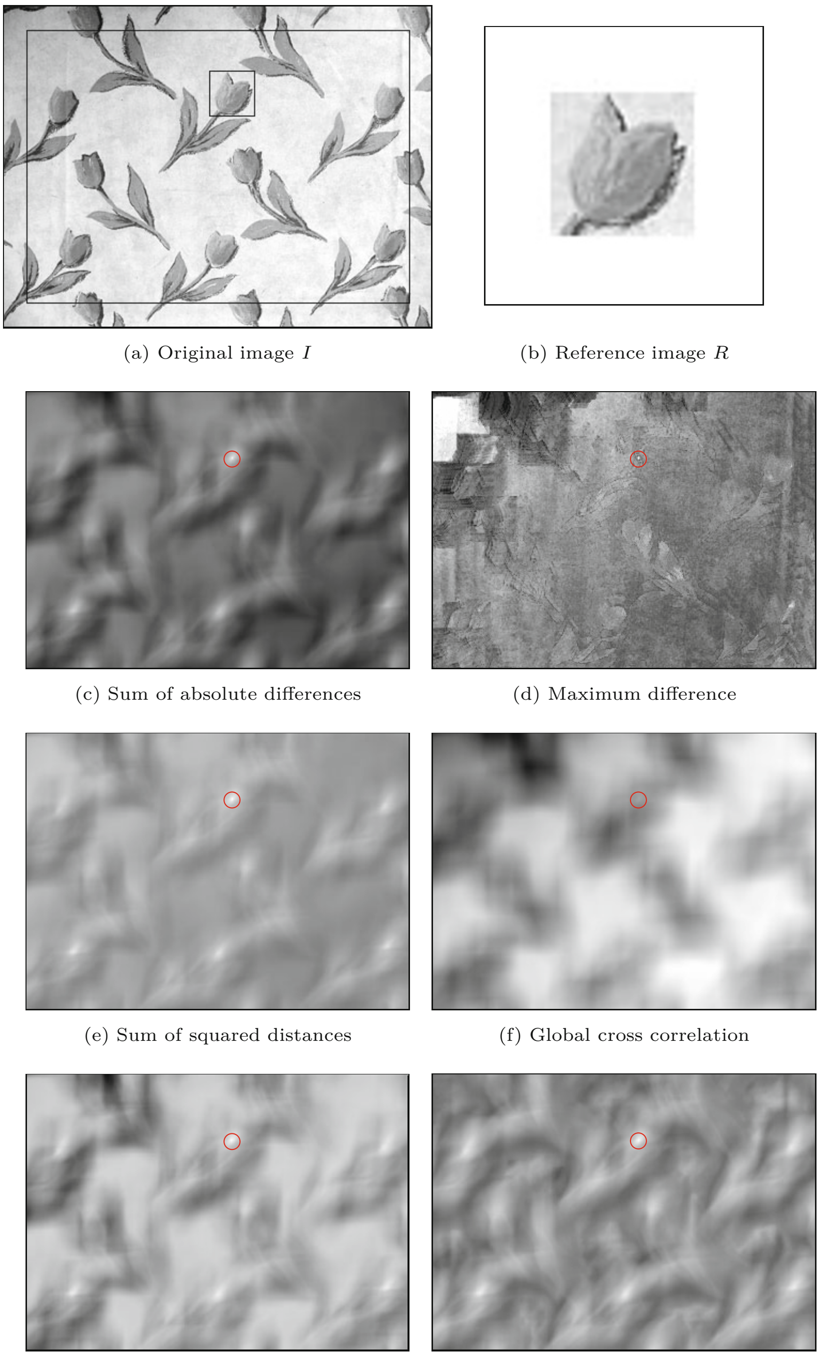
Alg. 23.1
Calculation of the correlation coefficient. Given is the search image I and the reference image (template) R. In Step 1, the template's average $\bar{R}$ and variance term S_R are computed once. In Step 2, the match function is computed for every template position (r,s) as prescribed by Eqn. (23.15). The result is a map of correlation values $C(r,s) \in [-1,1]$ that is returned. In line 23 (cf. Eqn. (23.15)) the quantity 1 is added to the denominator to avoid division by zero in the case of zero variance.

values range from -1.0 (black) to $+1.0$ (white), and zero values are shown as gray.

Figure 23.4 compares the results of the *Euclidean distance* against the *correlation coefficient* under globally changing intensity. For this purpose, the intensity of the reference image R is raised by 50 units such that the template is different from any subpattern in the original image. As can be seen clearly, the initially distinct peaks disappear under the Euclidean distance (Fig. 23.4(c)), while the correlation coefficient (Fig. 23.4(d)) naturally remains unaffected by this change.

In summary, the correlation coefficient can be recommended as a reliable measure for template matching in intensity images under realistic lighting conditions. This method proves relatively robust against global changes of brightness or contrast and tolerates small deviations from the reference pattern. Since the resulting values are in the fixed range of $[-1,1]$, a simple threshold operation can be used to localize the best match points (Fig. 23.6).

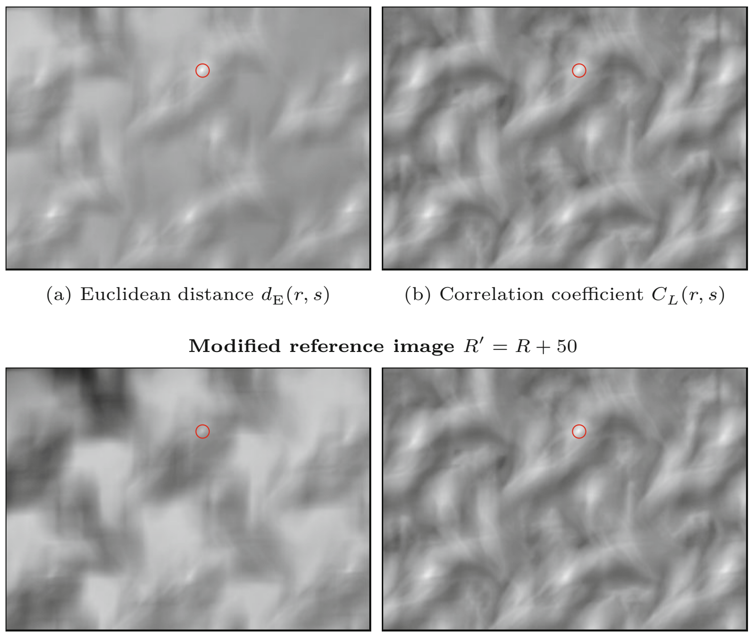
Fig. 23.3
Comparison of various distance functions. From the original image I (a), the marked section is used as the reference image R, shown enlarged in (b). In the resulting difference images (c–h), brightness corresponds to the amount of agreement (white equals minimum distance). The position of the true reference point is marked by a red circle.


(a) Original image I
(b) Reference image R
(c) Sum of absolute differences
(d) Maximum difference
(e) Sum of squared distances
(f) Global cross correlation
(g) Normalized cross correlation
(h) Correlation coefficient

Shape of the template

The shape of the reference image does not need to be rectangular as in the previous examples, although it is convenient for the processing. In some applications, circular, elliptical, or custom-shaped templates may be more applicable than a rectangle. In such a case, the template

Original reference image R

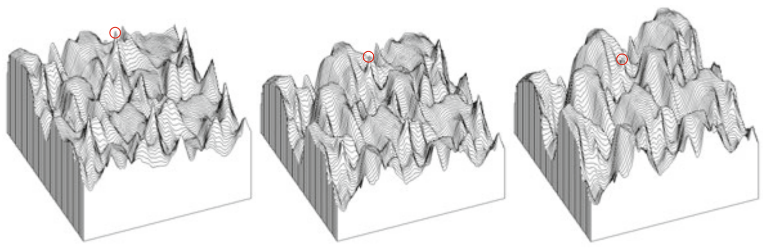


(a) Euclidean distance $d_E(r,s)$ (b) Correlation coefficient $C_L(r,s)$

Modified reference image $R' = R + 50$

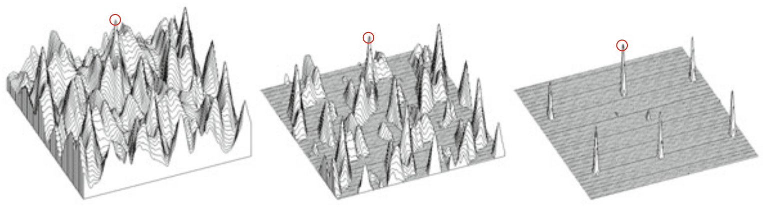
(c) Euclidean distance $d_E(r,s)$ (d) Correlation coefficient $C_L(r,s)$

Fig. 23.4
Effects of changing global brightness. Original reference image R: the results from both the Euclidean distance (a) and the correlation coefficient (b) show distinct peaks at the positions of maximum agreement. Modified reference image $R' = R + 50$: the peak values disappear in the Euclidean distance (c), while the correlation coefficient (d) remains unaffected.



R $R + 25$ $R + 50$

Fig. 23.5
Euclidean distance under global intensity changes. Distance function for the original template R (left), with the template intensity increased by 25 units (center) and 50 units (right). Notice that the local peaks disappear as the template intensity (and thus the total distance between the image and the template) is increased.



(a) (b) (c)

Fig. 23.6
Detection of match points by simple thresholding: correlation coefficient (a), positive values only (b), and values greater than 0.5 (c). The remaining peaks indicate the positions of the six similar (but not identical) tulip patterns in the original image (Fig. 23.3(a)).

may still be stored in a rectangular array, but the relevant pixels must somehow be marked (e.g., using a binary mask).

Even more general is the option to assign individual continuous weights to the template elements such that, for example, the center of a template can be given higher significance in the match than the peripheral regions. Implementing such a “windowed matching” technique should be straightforward and require only minor modifications to the standard approach.

23.1.2 Matching Under Rotation and Scaling

Correlation-based matching methods applied in the way described in this section cannot handle significant rotation or scale differences between the search image and the template. One obvious way to overcome rotation is to match using multiple rotated versions of the template, of course at the price of additional computation time. Similarly, one could try to match using several scaled versions of the template to achieve scale independence to some extent. Although this could be combined by using a set of rotated *and* scaled template patterns, the combinatorially growing number of required matching steps could soon become prohibitive for a practical implementation.

An interesting technique is matching in *logarithmic-polar* space, where rotation and scaling map to translations and can thus be handled with correlation-type methods [267]. However, this requires an initial "anchor point", which again needs to be detected in a rotation and scale invariant way [152, 209, 238]. Another alternative is the popular Lucas-Kanade technique for elastic local matching, which is described at detail in Chapter 24. In principle, given an approximate starting solution, this method cannot only handle rotation and scaling, but arbitrary image transformations or distortions.

23.1.3 Java Implementation

Implementations of most methods described in this chapter are openly available as part of the `imagingbook` library.[3] As an example, the code listed in Progs. 23.1 and 23.2 demonstrates the use of the `CorrCoeffMatcher` class for template matching based on the local correlation coefficient (Eqn. (23.10)). The application assumes that the search image (`I`) and the reference image (`R`) are already available as objects of type `FloatProcessor`. They are used to create a new instance of class `CorrCoeffMatcher`, as shown in the following code segment:

```java
FloatProcessor I = ...       // search image
FloatProcessor R = ...       // reference image
CorrCoeffMatcher matcher = new CorrCoeffMatcher(I);
float[][] C = matcher.getMatch(R);
```

The correlation coefficient is computed by the method `getMatch()` and returned as a 2D `float`-array (`C`).

23.2 Matching Binary Images

As became evident in the previous section, the comparison of intensity images based on correlation may not be an optimal solution but is sufficiently reliable and efficient under certain restrictions. If we compare binary images in the same way, by counting the number of identical pixels in the search image and the template, the total difference will only be small when most pixels are in exact agreement.

[3] Package `imagingbook.pub.matching`.

```
 1 package imagingbook.pub.matching;
 2
 3 import ij.process.FloatProcessor;
 4
 5 class CorrCoeffMatcher {
 6
 7   private final FloatProcessor I; // search image
 8   private final int MI, NI;          // width/height of search image
 9
10   private FloatProcessor R;       // reference image
11   private int MR, NR;             // width/height of reference image
12   private int K;
13   private double meanR;          // mean value of reference ($\bar{R}$)
14   private double varR;           // square root of reference variance
        ($\sigma_R$)
15
16   public CorrCoeffMatcher(FloatProcessor I) { // constructor
17     this.I = I;
18     this.MI = this.I.getWidth();
19     this.NI = this.I.getHeight();
20     }
21
22   public float[][] getMatch(FloatProcessor R) {
23     this.R = R;
24     this.MR = R.getWidth();
25     this.NR = R.getHeight();
26     this.K = MR * NR;
27
28     // calculate the mean ($\bar{R}$) and variance term ($S_R$) of the template:
29     double sumR = 0;        // $\Sigma_R = \sum R(i,j)$
30     double sumR2 = 0;       // $\Sigma_{R2} = \sum R^2(i,j)$
31     for (int j = 0; j < NR; j++) {
32       for (int i = 0; i < MR; i++) {
33         float aR = R.getf(i,j);
34         sumR  += aR;
35         sumR2 += aR * aR;
36       }
37     }
38
39     this.meanR = sumR / K;      // $\bar{R} = [\sum R(i,j)]/K$
40     this.varR =                 // $S_R = [\sum R^2(i,j) - K \cdot \bar{R}^2]^{1/2}$
41         Math.sqrt(sumR2 - K * meanR * meanR);
42
43     float[][] C = new float[MI - MR + 1][NI - NR + 1];
44     for (int r = 0; r <= MI - MR; r++) {
45       for (int s = 0; s <= NI - NR; s++) {
46         float d = (float) getMatchValue(r, s);
47         C[r][s] = d;
48       }
49     }
50     return C;
51   }
52
53   // continued...
```

Prog. 23.1
Implementation of class `CorrCoeffMatcher` (part 1/2). The constructor method (lines 16–20) calculates the mean $\bar{R}$ = `meanR` (Eqn. (23.11)) and the variance S_R = `varR` (Eqn. (23.14)) of the reference image R. The method `getMatch(R)` (lines 22–51) determines the match values between the search image I and the reference image R f for all positions (r, s).

Prog. 23.2
Implementation of class `CorrCoeffMatcher` (part 2/2). The local match value $C(r, s)$ (see Eqn. (23.15)) at the individual position (r, s) is calculated by method `getMatchValue(r,s)` (lines 54–72).

```java
54   private double getMatchValue(int r, int s) {
55     double sumI = 0;     // $\Sigma_I = \sum I(r{+}i, s{+}j)$
56     double sumI2 = 0;    // $\Sigma_{I2} = \sum (I(r{+}i, s{+}j))^2$
57     double sumIR = 0;    // $\Sigma_{IR} = \sum I(r{+}i, s{+}j) \cdot R(i, j)$
58
59     for (int j = 0; j < NR; j++) {
60       for (int i = 0; i < MR; i++) {
61         float aI = I.getf(r + i, s + j);
62         float aR = R.getf(i, j);
63         sumI  += aI;
64         sumI2 += aI * aI;
65         sumIR += aI * aR;
66       }
67     }
68
69     double meanI = sumI / K;  // $\bar{I}_{r,s} = \Sigma_I / K$
70     return (sumIR - K * meanI * meanR) /
71          (1 + Math.sqrt(sumI2 - K * meanI * meanI) * varR);
72   }
73
74 } // end of class CorrCoeffMatcher
```

Since there is no continuous transition between pixel values, the distribution produced by a simple distance function will generally be ill-behaved (i.e., highly discontinuous with many local extrema; see Fig. 23.7).

23.2.1 Direct Comparison of Binary Images

The problem with directly comparing binary images is that even the smallest deviations between image patterns, such as those caused by a small shift, rotation, or distortion, can create very high distance values. Shifting a thin line drawing by only a single pixel, for example, may be sufficient to switch from full agreement to no agreement at all (i.e., from zero difference to maximum difference). Thus a simple distance function gives no indication how far away and in which direction to search for a better match position.

An interesting question is how matching of binary images can be made more tolerant against small differences of the compared patterns. Thus the goal is not only to detect the single image position, where most foreground pixels in the two images match up, but also (if possible) to obtain a measure indicating how far (in terms of geometry) we are away from this position.

23.2.2 The Distance Transform

A first step in this direction is to record the distance to the closest foreground pixel for every position (u, v) in the search image I. This gives us the minimum distance (though not the direction) for shifting a particular pixel onto a foreground pixel. Starting from a binary image $I(u, v) = I(\boldsymbol{u})$, we denote

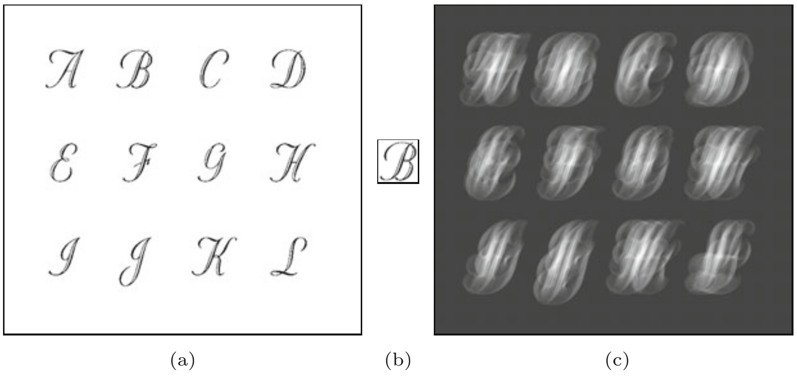


Fig. 23.7 Direct comparison of binary images. Given are a binary search image (a) and a binary reference image (b). The local similarity value for any template position corresponds to the relative number of matching (black) foreground pixels. High similarity values are shown as bright spots in the result (c). While the maximum similarity is naturally found at the correct position (at the center of the glyph *B*) the match function behaves wildly, with many local maxima.

$$FG(I) = \{\boldsymbol{u} \mid I(\boldsymbol{u}) = 1\}, \tag{23.16}$$

$$BG(I) = \{\boldsymbol{u} \mid I(\boldsymbol{u}) = 0\}, \tag{23.17}$$

as the set of coordinates of the foreground and background pixels, respectively. The so-called distance transform of I, $\mathsf{D}(\boldsymbol{u}) \in \mathbb{R}$, is defined as

$$\mathsf{D}(\boldsymbol{u}) := \min_{\boldsymbol{u}' \in FG(I)} \operatorname{dist}(\boldsymbol{u}, \boldsymbol{u}'), \tag{23.18}$$

for all $\boldsymbol{u} = (u, v)$, where $u = 0, \ldots, M-1$, $v = 0, \ldots, N-1$ (for image size $M \times N$). The value D at a given position $\boldsymbol{u}$ thus equals the distance between $\boldsymbol{u}$ and the nearest foreground pixel in I. If $I(\boldsymbol{u})$ is a foreground pixel itself (i.e., $x \in FG$), then the distance $D(\boldsymbol{u}) = 0$ since no shift is necessary for moving this pixel onto a foreground pixel.

The function $\operatorname{dist}(\boldsymbol{u}, \boldsymbol{u}')$ in Eqn. (23.18) measures the geometric distance between the two coordinate points $\boldsymbol{u} = (u, v)$ and $\boldsymbol{u}' = (u', v')$. Examples of suitable distance functions are the Euclidean distance (L_2 norm)

$$\mathrm{d}_E(\boldsymbol{u}, \boldsymbol{u}') = \|\boldsymbol{u} - \boldsymbol{u}'\| = \sqrt{(u - u')^2 + (v - v')^2} \quad \in \mathbb{R}^+ \tag{23.19}$$

and the *Manhattan distance*[4] (L_1 norm)

$$\mathrm{d}_M(\boldsymbol{u}, \boldsymbol{u}') = |u - u'| + |v - v'| \quad \in \mathbb{N}_0. \tag{23.20}$$

Figure 23.8 shows a simple example of a distance transform using the Manhattan distance $\mathrm{d}_M()$.

The direct calculation of the distance transform (following the definition in Eqn. (23.18)) is computationally expensive, because the closest foreground pixel must be found for each pixel position $\boldsymbol{p}$ (unless $I(\boldsymbol{p})$ is a foreground pixel itself).[5]

Chamfer algorithm

The so-called *chamfer* algorithm [30] is an efficient method for computing the distance transform. Similar to the sequential region labeling algorithm (see Ch. 10, Alg. 10.2), the chamfer algorithm traverses

[4] Also called "city block distance".

[5] A simple (brute force) algorithm for the distance transform would perform a full scan over the entire image for each processed pixel, resulting in $\mathcal{O}(N^2 \cdot N^2) = \mathcal{O}(N^4)$ steps for an image of size $N \times N$.

Binary image

0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	**1**	0	0	0	0	0	0	0
0	0	**1**	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	**1**	**1**	0	0	0
0	0	0	0	0	0	0	**1**	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0

$\rightarrow$

Distance transform

5	4	3	3	2	3	4	5	6	7	8	9
4	3	2	2	1	2	3	4	5	6	7	8
3	2	1	1	**0**	1	2	3	4	5	6	7
2	1	**0**	1	1	2	3	3	3	4	5	6
3	2	1	2	2	3	3	2	2	3	4	5
4	3	2	3	3	3	2	1	1	2	3	4
5	4	3	4	3	2	1	**0**	**0**	1	2	3
6	5	4	4	3	2	1	**0**	1	2	3	4
7	6	5	5	4	3	2	1	2	3	4	5
8	7	6	6	5	4	3	2	3	4	5	6

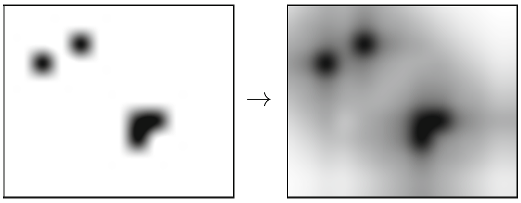


Fig. 23.8
Example of a distance transform of a binary image using the Manhattan distance $\mathrm{d}_M()$. Foreground pixels in the binary image have value 1 (shown inverted).

the image twice by propagating the computed values across the image like a wave. The first traversal starts at the upper left corner of the image and propagates the distance values downward in a diagonal direction. The second traversal proceeds in the opposite direction from the bottom to the top. For each traversal, a "distance mask" is used for the propagation of the distance values; that is,

$$M^L = \begin{bmatrix} m_2 & m_1 & m_2 \\ m_1 & \times & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad M^R = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \times & m_1 \\ m_2 & m_1 & m_2 \end{bmatrix} \tag{23.21}$$

for the first and second traversals, respectively. The values in M^L and M^R describe the geometric distance between the current pixel (marked $\times$) and the relevant neighboring pixels. They depend upon the distance function $\mathrm{dist}(\boldsymbol{x}, \boldsymbol{x}')$ used. Algorithm 23.2 outlines the chamfer method for computing the distance transform $D(u, v)$ for a binary image $I(u, v)$ using the above distance masks.

For the Manhattan distance, the chamfer algorithm computes the distance transform (Eqn. (23.20)) *exactly* using the masks

$$M^L_M = \begin{bmatrix} 2 & 1 & 2 \\ 1 & \times & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad M^R_M = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \times & 1 \\ 2 & 1 & 2 \end{bmatrix}. \tag{23.22}$$

Similarly for the Euclidean distance (Eqn. (23.19)) can be calculated with the masks

$$M^L_E = \begin{bmatrix} \sqrt{2} & 1 & \sqrt{2} \\ 1 & \times & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad M^R_E = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \times & 1 \\ \sqrt{2} & 1 & \sqrt{2} \end{bmatrix}. \tag{23.23}$$

Note that the result obtained with these masks is only an *approximation* of the Euclidean distance to the nearest foreground pixel, which is nevertheless more accurate than the estimate produced by the Manhattan distance. As demonstrated by the examples in Fig. 23.9, the distances obtained with the Euclidean masks are exact along the coordinate axes and the diagonals but are overestimated (i.e., too

```
1: DistanceTransform($I, norm$)
       Input: $I$, a, binary image; $norm \in \{\mathrm{L}_1, \mathrm{L}_2\}$, distance function.
       Returns the distance transform of $I$.

   Step 1: initialize
2:     $(m_1, m_2) \leftarrow \begin{cases} (1,2) & \text{for } norm = \mathrm{L}_1 \\ (1,\sqrt{2}) & \text{for } norm = \mathrm{L}_2 \end{cases}$
3:     $(M, N) \leftarrow \mathsf{Size}(I)$
4:     Create map $\mathsf{D}\colon M \times N \mapsto \mathbb{R}$
5:     for all $(u,v) \in M \times N$ do
6:         $\mathsf{D}(u,v) \leftarrow \begin{cases} 0 & \text{for } I(u,v) > 0 \\ \infty & \text{otherwise} \end{cases}$

       Step 2: L→R pass
7:     for $v \leftarrow 0,\ldots,N-1$ do                          ▷ top → bottom
8:         for $u \leftarrow 0,\ldots,M-1$ do                      ▷ left → right
9:             if $\mathsf{D}(u,v) > 0$ then
10:                $d_1, d_2, d_3, d_4 \leftarrow \infty$
11:                if $u > 0$ then
12:                    $d_1 \leftarrow m_1 + \mathsf{D}(u-1, v)$
13:                    if $v > 0$ then
14:                        $d_2 \leftarrow m_2 + \mathsf{D}(u-1, v-1)$
15:                if $v > 0$ then
16:                    $d_3 \leftarrow m_1 + \mathsf{D}(u, v-1)$
17:                    if $u < M-1$ then
18:                        $d_4 \leftarrow m_2 + \mathsf{D}(u+1, v-1)$
19:                $\mathsf{D}(u,v) \leftarrow \min(\mathsf{D}(u,v), d_1, d_2, d_3, d_4)$

       Step 3: R→L pass
20:    for $v \leftarrow N-1,\ldots,0$ do                          ▷ bottom → top
21:        for $u \leftarrow M-1,\ldots,0$ do                      ▷ right → left
22:            if $\mathsf{D}(u,v) > 0$ then
23:                $d_1, d_2, d_3, d_4 \leftarrow \infty$
24:                if $u < M-1$ then
25:                    $d_1 \leftarrow m_1 + \mathsf{D}(u+1, v)$
26:                    if $v < N-1$ then
27:                        $d_2 \leftarrow m_2 + \mathsf{D}(u+1, v+1)$
28:                if $v < N-1$ then
29:                    $d_3 \leftarrow m_1 + \mathsf{D}(u, v+1)$
30:                    if $u > 0$ then
31:                        $d_4 \leftarrow m_2 + \mathsf{D}(u-1, v+1)$
32:                $\mathsf{D}(u,v) \leftarrow \min(\mathsf{D}(u,v), d_1, d_2, d_3, d_4)$

33:    return D
```

Alg. 23.2
Chamfer algorithm for computing the distance transform. From the binary image I, the distance transform D (Eqn. (23.18)) is computed using a pair of distance masks (Eqn. (23.21)) for the first and second passes. Notice that the image borders require special treatment.

high) for all other directions. A more precise approximation can be obtained with distance masks of greater size (e.g., 5×5 pixels; see Exercise 23.3), which include the exact distances to pixels in a larger neighborhood [30]. Furthermore, floating point-operations can be avoided by using distance masks with scaled integer values, such as the masks

$$M^L_{E'} = \begin{bmatrix} 4 & 3 & 4 \\ 3 & \times & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \quad \text{and} \quad M^R_{E'} = \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \times & 3 \\ 4 & 3 & 4 \end{bmatrix} \tag{23.24}$$

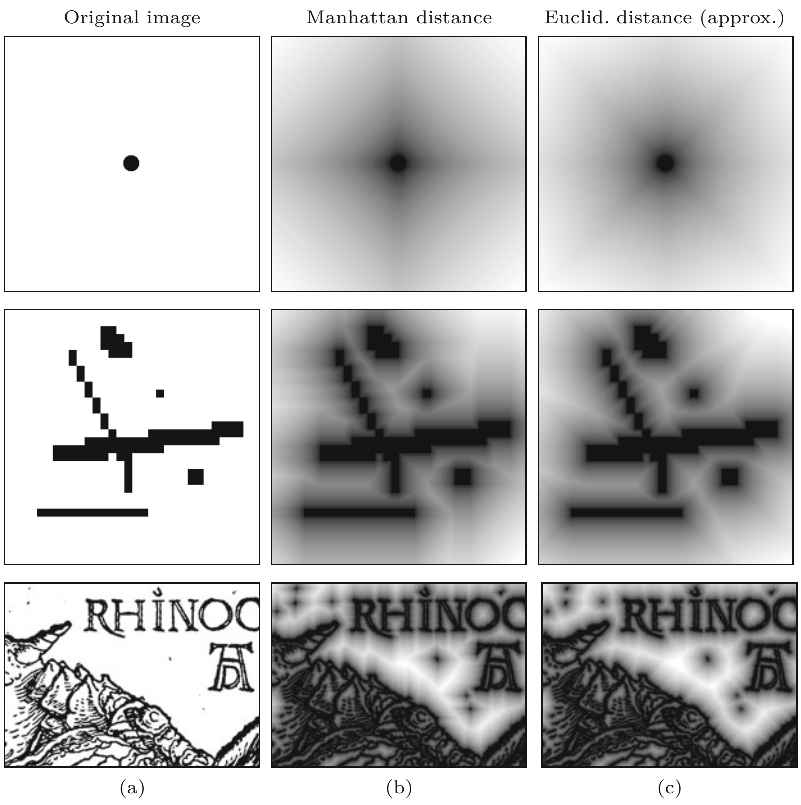


Fig. 23.9
Distance transform with the chamfer algorithm: original image with black foreground pixels (a), and results of distance transforms using the Manhattan distance (b) and the Euclidean distance (c). The brightness (scaled to maximum contrast) corresponds to the estimated distance to the nearest foreground pixel.

for the Euclidean distance. Compared with the original masks (Eqn. (23.23)), the resulting distance values are scaled by about the factor 3.

23.2.3 Chamfer Matching

The chamfer algorithm offers an efficient way to approximate the distance transform for a binary image of arbitrary size. The next step is to use the distance transform for matching binary images. *Chamfer matching* (first described in [19]) uses the distance transform to localize the points of maximum agreement between a binary search image I and a binary reference image (template) R. Instead of counting the overlapping foreground pixels as in the direct approach (see Sec. 23.2.1), chamfer matching uses the accumulated values of the distance transform as the match score Q. At each position (r, s) of the template R, the distance values corresponding to all foreground pixels in R are accumulated, that is,

$$Q(r,s) = \frac{1}{|FG(R)|} \cdot \sum_{\substack{(i,j)\in\\ FG(R)}} D(r+i,\, s+j)\,, \tag{23.25}$$

where $K = |FG(R)|$ denotes the number of foreground pixels in the template R.

The complete procedure for computing the match score Q is summarized in Alg. 23.3. If at some position each foreground pixel in the

```
1: ChamferMatch $(I, R)$
       Input: $I$, binary search image; $R$, binary reference image.
       Returns a 2D map of match scores.

       Step 1 – initialize:
2:     $(M_I, N_I) \leftarrow \mathsf{Size}(I)$
3:     $(M_R, N_R) \leftarrow \mathsf{Size}(R)$
4:     $\mathsf{D} \leftarrow \mathsf{DistanceTransform}(I)$                ▷ Alg. 23.2
5:     Create map $Q\colon (M_I - M_R + 1) \times (N_I - N_R + 1) \mapsto \mathbb{R}$

       Step 2 – compute match function:
6:     for $r \leftarrow 0, \ldots, M_I - M_R$ do                ▷ place $R$ at $(r, s)$
7:         for $s \leftarrow 0, \ldots, N_I - N_R$ do
               Get match score for $R$ placed at $(r, s)$
8:             $q \leftarrow 0$
9:             $n \leftarrow 0$                  ▷ number of foreground pixels in $R$
10:            for $i \leftarrow 0, \ldots, M_R - 1$ do
11:                for $j \leftarrow 0, \ldots, N_R - 1$ do
12:                    if $R(i, j) > 0$ then          ▷ foreground pixel in $R$
13:                        $q \leftarrow q + \mathsf{D}(r + i, s + j)$
14:                        $n \leftarrow n + 1$
15:            $Q(r, s) \leftarrow q / n$

16:    return $Q$
```

Alg. 23.3
Chamfer matching (calculation of the match function). Given is a binary search image I and a binary reference image (template) R. In step 1, the distance transform D is computed for the image I using the chamfer algorithm (Alg. 23.2). In step 2, the sum of distance values is accumulated for all foreground pixels in template R for each template position (r, s). The resulting scores are stored in the 2D match map Q, which is returned.

template R coincides with a foreground pixel in the image I, the sum of the distance values is zero, which indicates a perfect match. The more foreground pixels of the template fall onto distance values greater than zero, the larger is the resulting score value Q (sum of distances). The best match is found at the global minimum of Q, that is,

$$\boldsymbol{x}_{\mathrm{opt}} = (r_{\mathrm{opt}}, s_{\mathrm{opt}}) = \underset{(r,s)}{\operatorname{argmin}}(Q(r, s)). \tag{23.26}$$

The example in Fig. 23.10 demonstrates the difference between direct pixel comparison and chamfer matching using the binary image shown in Fig. 23.7. Obviously the match score produced by the chamfer method is considerably smoother and exhibits only a few distinct local maxima. This is of great advantage because it facilitates the detection of optimal match points using simple local search methods. Figure 23.11 shows another example with circles and squares. The circles have different diameters and the medium-sized circle is used as the template. As this example illustrates, chamfer matching is tolerant against small-scale changes between the search image and the template and even in this case yields a smooth score function with distinct peaks.

While chamfer matching is not a “silver bullet”, it is efficient and works sufficiently well if the applications and conditions are suitable. It is most suited for matching line or edge images where the percentage of foreground pixels is small, such as for registering aerial images or aligning wide-baseline stereo images. The method tolerates deviations between the image and the template to a small extent but is of course not generally invariant under scaling, rotation, and deformation. The quality of the results deteriorates quickly when images contain random noise (“clutter”) or large foreground regions, because

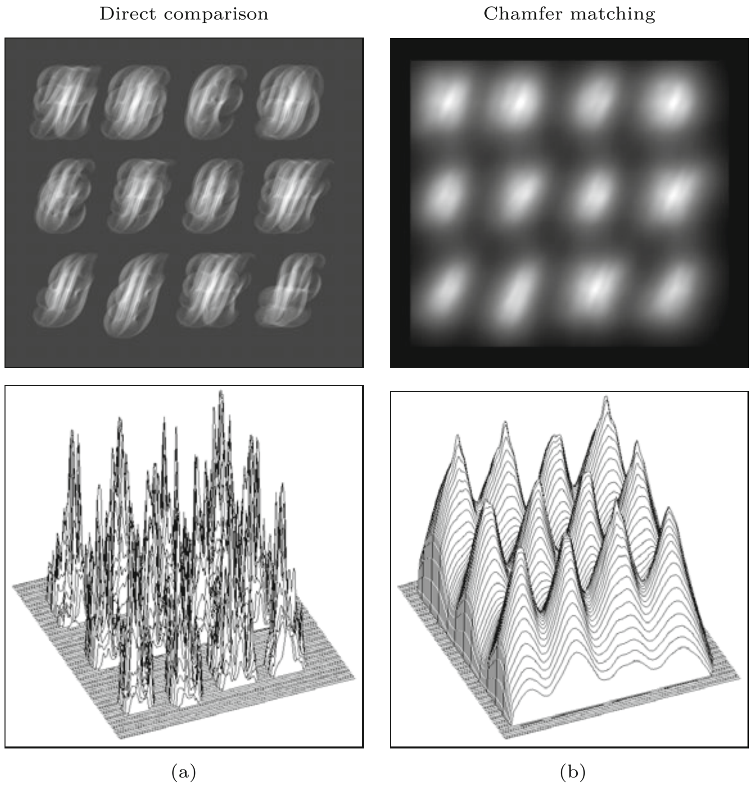


Fig. 23.10
Direct pixel comparison vs. chamfer matching (see original images in Fig. 23.7). Unlike the results of the direct pixel comparison (a), the chamfer match score Q (b) is much smoother. It shows distinct peak values in places of high agreement that are easy to track down with local search methods. The match score Q (Eqn. (23.25)) in (b) is shown inverted for easy comparison.

the method is based on minimizing the distances to foreground pixels. One way to reduce the probability of false matches is not to use a *linear* summation (as in Eqn. (23.25)) but add up the *squared* distances, that is,

$$Q_{rms}(r,s) = \Big[\frac{1}{K} \cdot \sum_{\substack{(i,j)\in \\ FG(R)}} \big(D(r+i, s+i)\big)^2\Big]^{1/2} \tag{23.27}$$

("root mean square" of the distances) as the match score between the template R and the current subimage, as suggested in [30]. Also, hierarchical variants of the chamfer method have been proposed to reduce the search effort as well as to increase robustness [31].

23.2.4 Java Implementation

The calculation of the distance transform, as described in Alg. 23.2, is implemented by the class `DistanceTransform`.[6] Program 23.3 shows the complete code for the class `ChamferMatcher` for comparing binary images with the distance transform, which is a direct implementation of Alg. 23.3. Additional examples (ImageJ plugins) can be found in the on-line code repository.

[6] Package `imagingbook.pub.matching`.

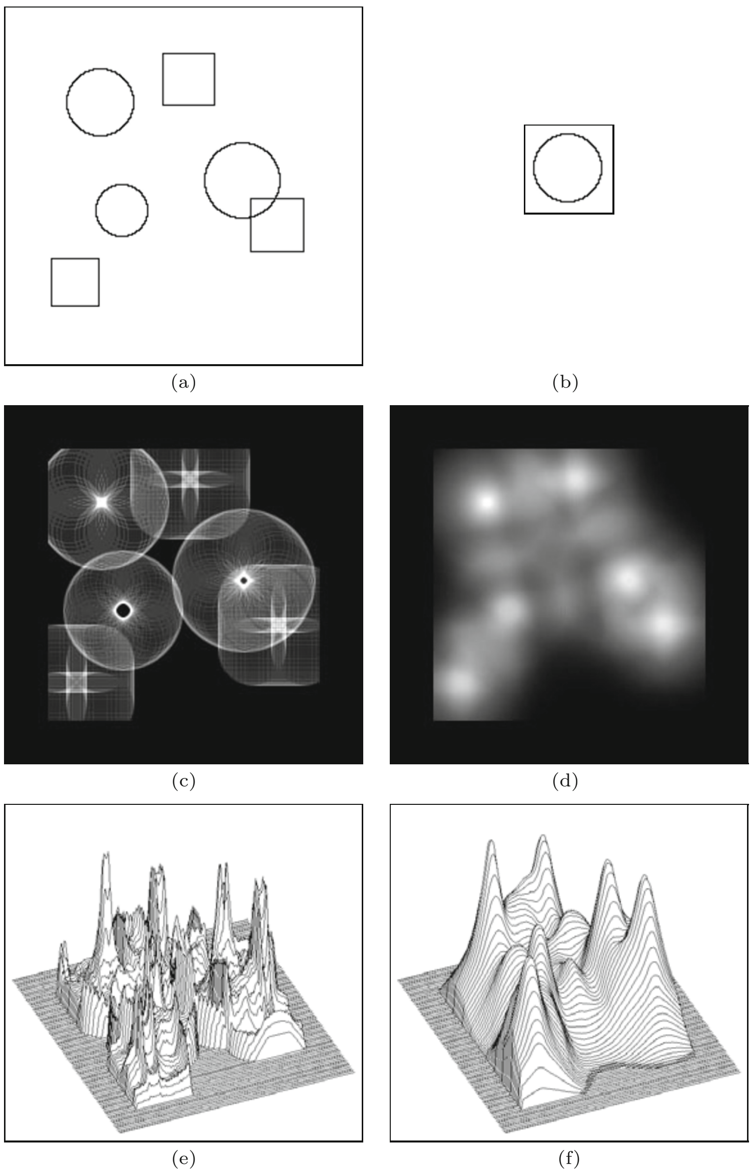


Fig. 23.11
Chamfer matching under varying scales. Binary search image with three circles of different diameters and three identical squares (a). The medium-sized circle at the top is used as the template (b). The result from a direct pixel comparison (c, e) and the result from chamfer matching (d, f). Again the chamfer match produces a much smoother score, which is most notable in the 3D plots shown in the bottom row (e, f). Notice that the three circles and the squares produce high match scores with similar absolute values (f).

23.3 Exercises

Exercise 23.1. Implement the chamfer-matching method (Alg. 23.2) for binary images using the Euclidean distance and the Manhattan distance.

Exercise 23.2. Implement the *exact* Euclidean distance transform using a "brute-force" search for each closest foreground pixel (this may take a while to compute). Compare your results with the approximation obtained with the chamfer method (Alg. 23.2), and compute the maximum deviation (as percentage of the real distance).

Exercise 23.3. Modify the chamfer algorithm for computing the distance transform (Alg. 23.2) by replacing the 3×3 pixel Euclidean distance masks (Eqn. (23.23)) with the following masks of size 5×5:

$$M^{L} = \begin{bmatrix} \cdot & 2.236 & \cdot & 2.236 & \cdot \\ 2.236 & 1.414 & 1.000 & 1.414 & 2.236 \\ \cdot & 1.000 & \times & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{bmatrix}, \qquad (23.28)$$

$$M^{R} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \cdot & \times & 1.000 & \cdot \\ 2.236 & 1.414 & 1.000 & 1.414 & 2.236 \\ \cdot & 2.236 & \cdot & 2.236 & \cdot \end{bmatrix}. \qquad (23.29)$$

Compare the results with those obtained with the standard masks. Why are no additional mask elements required along the coordinate axes and the diagonals?

Exercise 23.4. Implement the chamfer-matching technique using (a) the linear summation of distances (Eqn. (23.25)) and (b) the summation of squared distances (Eqn. (23.27)) for computing the match score. Select suitable test images to find out if version (b) is really more robust in terms of reducing the number of false matches.

Exercise 23.5. Adapt the template-matching method described in Sec. 23.1 for the comparison of RGB color images.

```
 1 package imagingbook.pub.matching;
 2 import ij.process.ByteProcessor;
 3 import imagingbook.pub.matching.DistanceTransform.Norm;
 4
 5 public class ChamferMatcher {
 6   private final ByteProcessor I;
 7   private final int MI, NI;
 8   private final float[][] D;        // distance transform of I
 9
10   public ChamferMatcher(ByteProcessor I) {
11     this(I, Norm.L2);
12   }
13
14   public ChamferMatcher(ByteProcessor I, Norm norm) {
15     this.I = I;
16     this.MI = this.I.getWidth();
17     this.NI = this.I.getHeight();
18     this.D = (new DistanceTransform(I, norm)).
       getDistanceMap();
19   }
20
21   public float[][] getMatch(ByteProcessor R) {
22     final int MR = R.getWidth();
23     final int NR = R.getHeight();
24     final int[][] Ra = R.getIntArray();
25     float[][] Q = new float[MI - MR + 1][NI - NR + 1];
26     for (int r = 0; r <= MI - MR; r++) {
27       for (int s = 0; s <= NI - NR; s++) {
28         float q = getMatchValue(Ra, r, s);
29         Q[r][s] = q;
30       }
31     }
32     return Q;
33   }
34
35   private float getMatchValue(int[][] R, int r, int s) {
36     float q = 0.0f;
37     for (int i = 0; i < R.length; i++) {
38       for (int j = 0; j < R[i].length; j++) {
39         if (R[i][j] > 0) {  // foreground pixel in reference image
40           q = q + D[r + i][s + j];
41         }
42       }
43     }
44     return q;
45   }
46 }
```

Prog. 23.3
Java implementation of Alg. 23.3 (class `ChamferMatcher`). The distance transform of the binary search image `I` is calculated in the constructor method by an instance of class `DistanceTransform` and stored as a 2D `float` array (line 18). The method `getMatch(R)` in lines 21–45 computes the 2D match function `Q` (again as a `float` array) for the reference image `R`.