Fourier descriptors

An elegant approach to describing contours are so-called Fourier descriptors, which interpret the two-dimensional contour $\boldsymbol{c}_{\mathcal{R}} = [\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots \boldsymbol{x}_{M-1}]$ with $\boldsymbol{x}_i = (u_i, v_i)$ as a sequence of values $[z_0, z_1 \ldots z_{M-1}]$ in the complex plane, where

$$z_i = (u_i + \mathrm{i} \cdot v_i) \in \mathbb{C}. \tag{2.6}$$

From this sequence, one obtains (using a suitable method of interpolation in case of an 8-connected contour), a discrete, one-dimensional periodic function $f(s) \in \mathbb{C}$ with a constant sampling interval over s, the path length around the contour. The coefficients of the one-dimensional *Fourier spectrum* (see Sec. 7.3) of this function $f(s)$ provide a shape description of the contour in frequency space, where the lower spectral coefficients deliver a gross description of the shape. The details of this classical method can be found for example in [28, 30, 46, 47, 69].

2.4 Properties of Binary Regions

Imagine that you have to describe the contents of a digital image to another person over the telephone. One possibility would be to call out the value of each pixel in some agreed upon order. A much simpler way of course would be to describe the image on the basis of its properties—for example, "a red rectangle on a blue background", or at an even higher level such as "a sunset at the beach with two dogs playing in the sand". While using such a description is simple and natural for us, it is not (yet) possible for a computer to generate these types of descriptions without human intervention. For computers, it is of course simpler to calculate the mathematical properties of an image or region and to use these as the basis for further classification. Using features to classify, be they images or other items, is a fundamental part of the field of pattern recognition, a research area with many applications in image processing and computer vision [21, 55, 72].

2.4.1 Shape Features

The comparison and classification of binary regions is widely used, for example, in optical character recognition (OCR) and for automating processes ranging from blood cell counting to quality control inspection of manufactured products on assembly lines. The analysis of binary regions turns out to be one of the simpler tasks for which many efficient algorithms have been developed and used to implement reliable applications that are in use every day.

By a *feature* of a region, we mean a specific numerical or qualitative measure that is computable from the values and coordinates of the pixels that make up

the region. As an example, one of the simplest features is its *size* or *area*; that is the number of pixels that make up a region. In order to describe a region in a compact form, different features are often combined into a *feature vector*. This vector is then used as a sort of "signature" for the region that can be used for classification or comparison with other regions. The best features are those that are simple to calculate and are not easily influenced (robust) by irrelevant changes, particularly translation, rotation, and scaling.

2.4.2 Geometric Features

A region $\mathcal{R}$ of a binary image can be interpreted as a two-dimensional distribution of foreground points $\boldsymbol{x}_i = (u_i, v_i)$ on the discrete plane $\mathbb{Z}^2$,

$$\mathcal{R} = \{\boldsymbol{x}_0, \boldsymbol{x}_1 \dots \boldsymbol{x}_{N-1}\} = \{(u_0, v_0), (u_1, v_1) \dots (u_{N-1}, v_{N-1})\}.$$

Most geometric properties are defined in such a way that a region is considered to be a set of pixels that, in contrast to the definition in Sec. 2.1, does not necessarily have to be connected.

Perimeter

The perimeter (or circumference) of a region $\mathcal{R}$ is defined as the length of its outer contour, where $\mathcal{R}$ must be connected. As illustrated in Fig. 2.14, the type of neighborhood relation must be taken into account for this calculation. When using a 4-neighborhood, the measured length of the contour (except when that length is 1) will be larger than its actual length. In the case of 8-neighborhoods, a good approximation is reached by weighing the horizontal and vertical segments with 1 and diagonal segments with $\sqrt{2}$. Given an 8-connected chain code $\boldsymbol{c}'_{\mathcal{R}} = [c'_0, c'_1, \dots c'_{M-1}]$, the perimeter of the region is arrived at by

$$\mathsf{Perimeter}(\mathcal{R}) = \sum_{i=0}^{M-1} \mathrm{length}(c'_i), \tag{2.7}$$

$$\text{with} \quad \mathrm{length}(c) = \begin{cases} 1 & \text{for } c = 0, 2, 4, 6, \\ \sqrt{2} & \text{for } c = 1, 3, 5, 7. \end{cases}$$

However, with this conventional method of calculation,the *real* perimeter ($P(\mathcal{R})$) is systematically overestimated. As a simple remedy, an empirical correction factor of 0.95 works satisfactory even for relatively small regions:

$$P(\mathcal{R}) \approx \mathsf{Perimeter}_{\mathrm{corr}}(\mathcal{R}) = 0.95 \cdot \mathsf{Perimeter}(\mathcal{R}). \tag{2.8}$$

Area

The area of a binary region $\mathcal{R}$ can be found by simply counting the image pixels that make up the region,

$$A(\mathcal{R}) = |\mathcal{R}| = N. \tag{2.9}$$

The area of a connected region without holes can also be approximated from its closed contour, defined by M coordinate points $(\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots \boldsymbol{x}_{M-1})$, where $\boldsymbol{x}_i = (u_i, v_i)$, using the Gaussian area formula for polygons:

$$A(\mathcal{R}) \approx \frac{1}{2} \cdot \left| \sum_{i=0}^{M-1} \big(u_i \cdot v_{(i+1) \bmod M} \; - \; u_{(i+1) \bmod M} \cdot v_i\big) \right|. \tag{2.10}$$

When the contour is already encoded as a chain code $\boldsymbol{c}'_{\mathcal{R}} = [c'_0, c'_1, \ldots c'_{M-1}]$, then the region's area can be computed using Eqn. (2.10) by expanding $\boldsymbol{c}'_{\mathcal{R}}$ into a sequence of contour points, using an arbitrary starting point (e. g., $(0, 0)$).

While simple region properties such as area and perimeter are not influenced (except for quantization errors) by translation and rotation of the region, they are definitely affected by changes in size; for example, when the object to which the region corresponds is imaged from different distances. However, as described below, it is possible to specify combined features that are *invariant* to translation, rotation, *and* scaling as well.

Compactness and roundness

Compactness is understood as the relation between a region's area and its perimeter. We can use the fact that a region's perimeter P increases linearly with the enlargement factor while the area A increases quadratically to see that, for a particular shape, the ratio A/P^2 should be the same at any scale. This ratio can thus be used as a feature that is invariant under translation, rotation, and scaling. When applied to a circular region of any diameter, this ratio has a value of $\frac{1}{4\pi}$, so by normalizing it against a filled circle, we create a feature that is sensitive to the *roundness* or *circularity* of a region,

$$\mathsf{Circularity}(\mathcal{R}) = \; 4\pi \cdot \frac{A(\mathcal{R})}{P^2(\mathcal{R})}, \tag{2.11}$$

which results in a maximum value of 1 for a perfectly round region $\mathcal{R}$ and a value in the range $[0, 1)$ for all other shapes (Fig. 2.15). If an absolute value for a region's roundness is required, the corrected perimeter estimate (Eqn. (2.8)) should be employed:

$$\mathsf{Circularity}(\mathcal{R}) \approx \; 4\pi \cdot \frac{A(\mathcal{R})}{\mathsf{Perimeter}^2_{\mathrm{corr}}(\mathcal{R})}. \tag{2.12}$$

Figure 2.15 shows the circularity values of different regions as computed with the formulation in Eqn. (2.12).

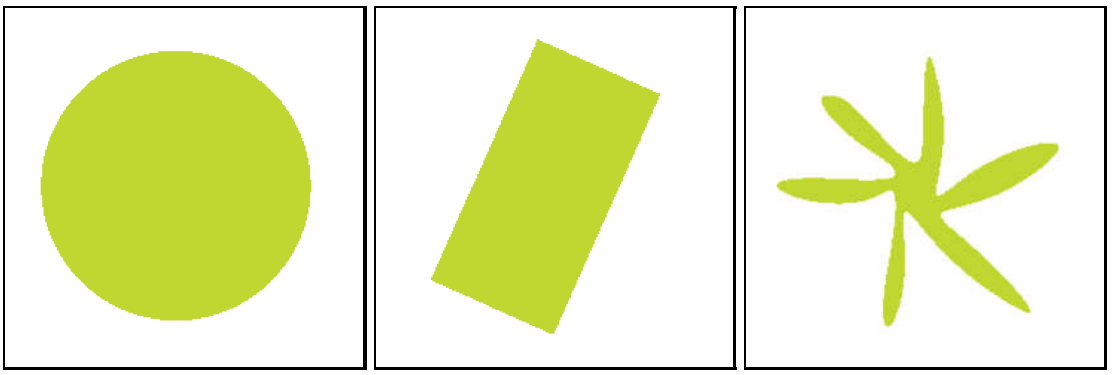


Figure 2.15 Circularity values for different shapes. Shown are the corresponding estimates for $\mathsf{Circularity}(\mathcal{R})$ as defined in Eqn. (2.12).

Bounding box

The bounding box of a region $\mathcal{R}$ is the minimal axis-parallel rectangle that encloses all points of $\mathcal{R}$,

$$\mathsf{BoundingBox}(\mathcal{R}) = \langle u_{\min}, u_{\max}, v_{\min}, v_{\max} \rangle, \tag{2.13}$$

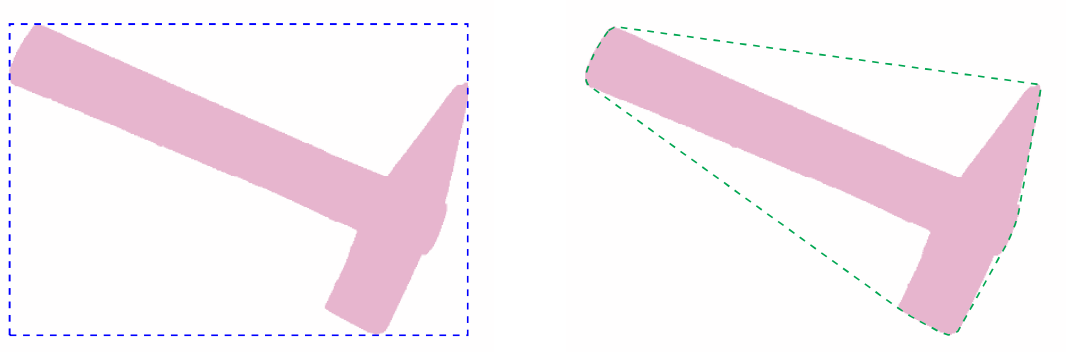
where $u_{\min}, u_{\max}$ and $v_{\min}, v_{\max}$ are the minimal and maximal coordinate values of all points $(u_i, v_i) \in \mathcal{R}$ in the x and y directions, respectively (Fig. 2.16 (a)).

Convex hull

The convex hull is the smallest convex polygon that contains all points of the region $\mathcal{R}$. A physical analogy is a board in which nails stick out in correspondence to each of the points in the region. If you were to place an elastic band around *all* the nails, then, when you release it, it will contract into a convex hull around the nails (Fig. 2.16 (b)). The convex hull can be computed for N contour points in time $\mathcal{O}(N \log V)$, where V is the number vertices in the polygon of the resulting convex hull [3].[9]

The convex hull is useful, for example, for determining the convexity or the *density* of a region. The *convexity* is defined as the relationship between the length of the convex hull and the original perimeter of the region. *Density* is then defined as the ratio between the area of the region and the area of its convex hull. The *diameter*, on the other hand, is the maximal distance between any two nodes on the convex hull.

[9] For $\mathcal{O}()$ complexity notation, see Vol. 1 [14, Appendix A.3].



(a) (b)

Figure 2.16 Example bounding box (a) and convex hull (b) of a binary image region.

2.4.3 Statistical Shape Properties

When computing statistical shape properties, we consider a region $\mathcal{R}$ to be a collection of coordinate points distributed within a two-dimensional space. Since statistical properties can be computed for point distributions that do not form a connected region, they can be applied before segmentation. An important concept in this context are the *central moments* of the region's point distribution, which measure characteristic properties with respect to its midpoint or *centroid*.

Centroid

The centroid or center of gravity of a connected region can be easily visualized. Imagine drawing the region on a piece of cardboard or tin and then cutting it out and attempting to balance it on the tip of your finger. The location on the region where you must place your finger in order for the region to balance is the *centroid* of the region.[10]

The centroid $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y})$ of a binary (not necessarily connected) region is the arithmetic mean of the coordinates in the x and y directions,

$$\bar{x} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(u,v)\in\mathcal{R}} u \qquad \text{and} \qquad \bar{y} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(u,v)\in\mathcal{R}} v \,. \tag{2.14}$$

[10] Assuming you did not imagine a region where the centroid lies outside of the region or within a hole in the region, which is of course possible.

Moments

The formulation of the region's centroid in Eqn. (2.14) is only a special case of the more general statistical concept of a *moment*. Specifically, the expression

$$m_{pq} = \sum_{(u,v)\in\mathcal{R}} I(u,v) \cdot u^p v^q \tag{2.15}$$

describes the (ordinary) moment of the order p, q for a discrete (image) function $I(u,v) \in \mathbb{R}$; for example, a grayscale image. All the following definitions are also generally applicable to regions in grayscale images. The moments of connected binary regions can also be computed directly from the coordinates of the contour points [64, p. 148].

In the special case of a binary image $I(u,v) \in \{0,1\}$, only the foreground pixels with $I(u,v) = 1$ in the region $\mathcal{R}$ need to be considered, and therefore Eqn. (2.15) can be simplified to

$$m_{pq} = \sum_{(u,v)\in\mathcal{R}} u^p v^q. \tag{2.16}$$

In this way, the *area* of a binary region can be expressed as its *zero*-order moment,

$$A(\mathcal{R}) = |\mathcal{R}| = \sum_{(u,v)\in\mathcal{R}} 1 = \sum_{(u,v)\in\mathcal{R}} u^0 v^0 = m_{00}(\mathcal{R}), \tag{2.17}$$

and similarly the *centroid* $\bar{\boldsymbol{x}}$ Eqn. (2.14) as

$$\bar{x} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(u,v)\in\mathcal{R}} u^1 v^0 = \frac{m_{10}(\mathcal{R})}{m_{00}(\mathcal{R})}, \tag{2.18}$$

$$\bar{y} = \frac{1}{|\mathcal{R}|} \cdot \sum_{(u,v)\in\mathcal{R}} u^0 v^1 = \frac{m_{01}(\mathcal{R})}{m_{00}(\mathcal{R})}. \tag{2.19}$$

These moments thus represent concrete physical properties of a region. Specifically, the area m_{00} is in practice an important basis for characterizing regions, and the centroid $(\bar{x}, \bar{y})$ permits the reliable and (within a fraction of a pixel) exact specification of a region's position.

Central moments

To compute position-independent (translation-invariant) region features, the region's centroid, which can be determined precisely in any situation, can be

used as a reference point. In other words, we can shift the origin of the coordinate system to the region's centroid $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y})$ to obtain the *central* moments of order p, q:

$$\mu_{pq}(\mathcal{R}) = \sum_{(u,v)\in\mathcal{R}} I(u,v) \cdot (u-\bar{x})^p \cdot (v-\bar{y})^q. \tag{2.20}$$

For a binary image (with $I(u,v) = 1$ within the region $\mathcal{R}$), Eqn. (2.20) can be simplified to

$$\mu_{pq}(\mathcal{R}) = \sum_{(u,v)\in\mathcal{R}} (u-\bar{x})^p \cdot (v-\bar{y})^q. \tag{2.21}$$

Normalized central moments

Central moment values of course depend on the absolute size of the region since the value depends directly on the distance of all region points to its centroid. So, if a 2D shape is scaled uniformly by some factor $s \in \mathbb{R}$, its central moments multiply by the factor

$$s^{(p+q+2)}. \tag{2.22}$$

Thus size-invariant "normalized" moments are obtained by scaling with the reciprocal of the area $\mu_{00} = m_{00}$ raised to the required power in the form

$$\bar{\mu}_{pq}(\mathcal{R}) = \mu_{pq}(\mathcal{R}) \cdot \Big(\frac{1}{\mu_{00}(\mathcal{R})}\Big)^{(p+q+2)/2} \tag{2.23}$$

for $(p+q) \geq 2$ [46, p. 529].

Program 2.3 gives a direct (brute force) Java implementation for computing the ordinary, central, and normalized central moments for binary images (`BACKGROUND = 0`). This implementation is only meant to clarify the computation, and naturally much more efficient implementations are possible (see, for example, [48]).

2.4.4 Moment-Based Geometrical Properties

While normalized moments can be directly applied for classifying regions, further interesting and geometrically relevant features can be elegantly derived from moments.

Orientation

Orientation describes the direction of the major axis, that is the axis that runs through the centroid and along the widest part of the region (Fig. 2.18 (a)). Since rotating the region around the major axis requires less effort (smaller moment of inertia) than spinning it around any other axis, it is sometimes referred to as the major axis of rotation. As an example, when you hold a

```
1  import ij.process.ImageProcessor;
2
3  public class Moments {
4    static final int BACKGROUND = 0;
5
6    static double moment(ImageProcessor ip,int p,int q) {
7      double Mpq = 0.0;
8      for (int v = 0; v < ip.getHeight(); v++) {
9        for (int u = 0; u < ip.getWidth(); u++) {
10         if (ip.getPixel(u,v) != BACKGROUND) {
11           Mpq += Math.pow(u, p) * Math.pow(v, q);
12         }
13       }
14     }
15     return Mpq;
16   }
17   static double centralMoment(ImageProcessor ip,int p,int q)
18     {
19     double m00 = moment(ip, 0, 0); // region area
20     double xCtr = moment(ip, 1, 0) / m00;
21     double yCtr = moment(ip, 0, 1) / m00;
22     double cMpq = 0.0;
23     for (int v = 0; v < ip.getHeight(); v++) {
24       for (int u = 0; u < ip.getWidth(); u++) {
25         if (ip.getPixel(u,v) != BACKGROUND) {
26           cMpq +=
27             Math.pow(u - xCtr, p) *
28             Math.pow(v - yCtr, q);
29         }
30       }
31     }
32     return cMpq;
33   }
34   static double normalCentralMoment
35                         (ImageProcessor ip,int p,int q) {
36     double m00 = moment(ip, 0, 0);
37     double norm = Math.pow(m00, (double)(p + q + 2) / 2);
38     return centralMoment(ip, p, q) / norm;
39   }
40
41 } // end of class Moments
```

Program 2.3 Example of directly computing moments in Java. The methods `moment()`, `centralMoment()`, and `normalCentralMoment()` compute for a binary image the moments m_{pq}, μ_{pq}, and $\bar{\mu}_{pq}$ (Eqns. (2.16), (2.21), and (2.23)).

pencil between your hands and twist it around its major axis (that is, around the lead), the pencil exhibits the least mass inertia (Fig. 2.17). As long as a region exhibits an orientation at all ($\mu_{20}(\mathcal{R}) \neq \mu_{02}(\mathcal{R})$), the direction $\theta_{\mathcal{R}}$ of the major axis can be found directly from the central moments μ_{pq} as

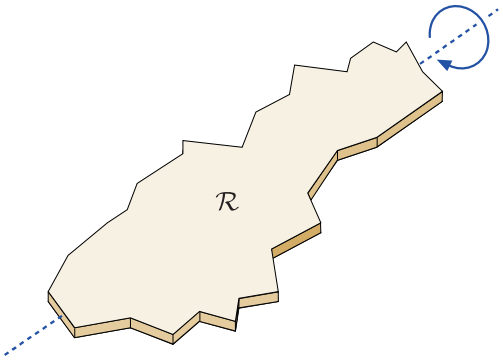


Figure 2.17 Major axis of a region. Rotating an elongated region $\mathcal{R}$, interpreted as a physical body, around its major axis requires less effort (least moment of inertia) than rotating it around any other axis.

$$\tan(2\,\theta_{\mathcal{R}}) = \frac{2 \cdot \mu_{11}(\mathcal{R})}{\mu_{20}(\mathcal{R}) - \mu_{02}(\mathcal{R})} \tag{2.24}$$

and therefore

$$\theta_{\mathcal{R}} = \frac{1}{2}\tan^{-1}\left(\frac{2 \cdot \mu_{11}(\mathcal{R})}{\mu_{20}(\mathcal{R}) - \mu_{02}(\mathcal{R})}\right) \tag{2.25}$$

$$= \frac{\mathrm{Arctan}\big(2 \cdot \mu_{11}(\mathcal{R}),\ \mu_{20}(\mathcal{R}) - \mu_{02}(\mathcal{R})\big)}{2}. \tag{2.26}$$

The resulting angle $\theta_{\mathcal{R}}$ is in the range $[-\frac{\pi}{2}, \frac{\pi}{2}]$.[11] Orientation measurements based on region moments are very accurate in general.

Computing orientation vectors. When visualizing region properties, a frequent task is to plot the region's orientation as a line or arrow, that are usually anchored at the center of gravity $\bar{\boldsymbol{x}} = (\bar{x}, \bar{y})$; for example, by a parametric line of the form

$$\boldsymbol{x} = \bar{\boldsymbol{x}} + \lambda \cdot \boldsymbol{x}_d = \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} + \lambda \cdot \begin{pmatrix} \cos(\theta_{\mathcal{R}}) \\ \sin(\theta_{\mathcal{R}}) \end{pmatrix}, \tag{2.27}$$

for some length $\lambda > 0$. To find the unit orientation vector $\boldsymbol{x}_d = (\cos\theta, \sin\theta)^T$, we could first compute the inverse tangent to get 2θ (Eqn. (2.25)) and then compute the cosine and sine of θ. However, the vector $\boldsymbol{x}_d$ can also be obtained without using trigonometric functions as follows. Rewriting Eqn. (2.24) as

$$\tan(2\theta_{\mathcal{R}}) = \frac{2 \cdot \mu_{11}(\mathcal{R})}{\mu_{20}(\mathcal{R}) - \mu_{02}(\mathcal{R})} = \frac{A}{B} = \frac{\sin(2\theta_{\mathcal{R}})}{\cos(2\theta_{\mathcal{R}})} \tag{2.28}$$

[11] See Appendix A.1 for the computation of angles with the Arctan() (inverse tangent) function and Vol. 1 [14, Appendix B.1.6] for the corresponding Java method `Math.atan2()`.

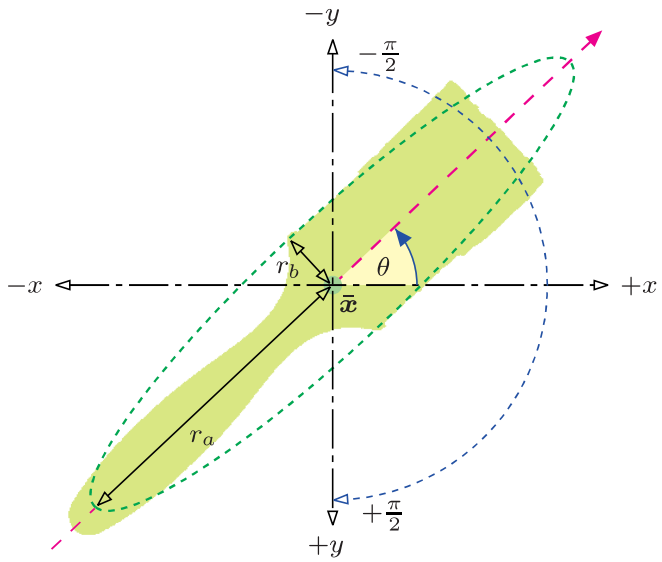


Figure 2.18 Region orientation and eccentricity. The major axis of the region extends through its center of gravity $\bar{\boldsymbol{x}}$ at the orientation θ. Note that angles are in the range $[-\frac{\pi}{2}, +\frac{\pi}{2}]$ and increment in the *clockwise* direction because the y axis of the image coordinate system points downward (in this example, $\theta \approx -0.759 \approx -43.5°$). The eccentricity of the region is defined as the ratio between the lengths of the major axis (r_a) and the minor axis (r_b) of the "equivalent" ellipse.

we get (by Pythagoras' theorem)

$$\sin(2\theta_{\mathcal{R}}) = \frac{A}{\sqrt{A^2+B^2}} \qquad \text{and} \qquad \cos(2\theta_{\mathcal{R}}) = \frac{B}{\sqrt{A^2+B^2}},$$

where $A = 2\mu_{11}(\mathcal{R})$ and $B = \mu_{20}(\mathcal{R}) - \mu_{02}(\mathcal{R})$. Using the relations $\cos^2\alpha = \frac{1}{2}[1 + \cos(2\alpha)]$ and $\sin^2\alpha = \frac{1}{2}[1 - \cos(2\alpha)]$, we can compute the region's orientation vector $\boldsymbol{x}_d = (x_d, y_d)^T$ as

$$x_d = \cos(\theta_{\mathcal{R}}) = \begin{cases} 0 & \text{for } A = B = 0 \\ \left[\frac{1}{2}\left(1 + \frac{B}{\sqrt{A^2+B^2}}\right)\right]^{\frac{1}{2}} & \text{otherwise,} \end{cases} \tag{2.29}$$

$$y_d = \sin(\theta_{\mathcal{R}}) = \begin{cases} 0 & \text{for } A = B = 0 \\ \left[\frac{1}{2}\left(1 - \frac{B}{\sqrt{A^2+B^2}}\right)\right]^{\frac{1}{2}} & \text{for } A \geq 0 \\ -\left[\frac{1}{2}\left(1 - \frac{b}{\sqrt{A^2+B^2}}\right)\right]^{\frac{1}{2}} & \text{for } A < 0, \end{cases} \tag{2.30}$$

straight from the central region moments $\mu_{11}(\mathcal{R})$, $\mu_{20}(\mathcal{R})$, and $\mu_{02}(\mathcal{R})$, as defined in Eqn. (2.28). The horizontal component (x_d) in Eqn. (2.29) is always positive, while the case clause in Eqn. (2.30) corrects the sign of the vertical component (y_d) to map to the same angular range $[-\frac{\pi}{2}, +\frac{\pi}{2}]$ as Eqn. (2.25). The resulting vector $\boldsymbol{x}_d$ is normalized (i. e., $\|(x_d, y_d)\| = 1$) and could be scaled

arbitrarily for display purposes by a suitable length λ, for example, using the region's eccentricity value described below.

Eccentricity

Similar to the region orientation, moments can also be used to determine the "elongatedness" or *eccentricity* of a region. A naive approach for computing the eccentricity could be to rotate the region until we can fit a bounding box (or enclosing ellipse) with a maximum aspect ratio. Of course this process would be computationally intensive simply because of the many rotations required. If we know the orientation of the region (Eqn. (2.25)), then we may fit a bounding box that is parallel to the region's major axis. In general, the proportions of the region's bounding box is not a good eccentricity measure anyway because it does not consider the distribution of pixels inside the box.

Based on region moments, highly accurate and stable measures can be obtained without any iterative search or optimization. Also, moment-based methods do not require knowledge of the boundary length (as required for computing the circularity feature in Sec. 2.4.2), and they can also handle nonconnected regions or point clouds. Several different formulations of region eccentricity can be found in the literature [2,46,47] (see also Exercise 2.11). We adopt the following definition because of its simple geometrical interpretation:

$$\mathsf{Ecc}(\mathcal{R}) = \frac{a_1}{a_2} = \frac{\mu_{20} + \mu_{02} + \sqrt{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2}}{\mu_{20} + \mu_{02} - \sqrt{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}^2}}, \tag{2.31}$$

where $a_1 = 2\lambda_1$, $a_2 = 2\lambda_2$ are multiples of the eigenvalues λ_1, λ_2 of the symmetric 2×2 matrix

$$\boldsymbol{A} = \begin{pmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{pmatrix}$$

formed by the central moments μ_{pq} of the region $\mathcal{R}$. The values of Ecc are in the range $[1, \infty)$, where $\mathsf{Ecc} = 1$ corresponds to a circular disk and elongated regions have values > 1. Ecc itself is invariant to the region's orientation and size. However, the values a_1, a_2 contain information about the spatial extent of the region. Geometrically, the eigenvalues λ_1, λ_2 (and thus a_1, a_2) directly relate to the proportions of the "equivalent" ellipse, positioned at the region's center of gravity $(\bar{x}, \bar{y})$ and oriented at $\theta = \theta_{\mathcal{R}}$ Eqn. (2.25). The lengths of the ellipse's major and minor axes, r_a and r_b, are

$$r_a = 2 \cdot \left(\frac{\lambda_1}{|\mathcal{R}|}\right)^{\frac{1}{2}} = \left(\frac{2\,a_1}{|\mathcal{R}|}\right)^{\frac{1}{2}}, \tag{2.32}$$

$$r_b = 2 \cdot \left(\frac{\lambda_2}{|\mathcal{R}|}\right)^{\frac{1}{2}} = \left(\frac{2\,a_2}{|\mathcal{R}|}\right)^{\frac{1}{2}}, \tag{2.33}$$

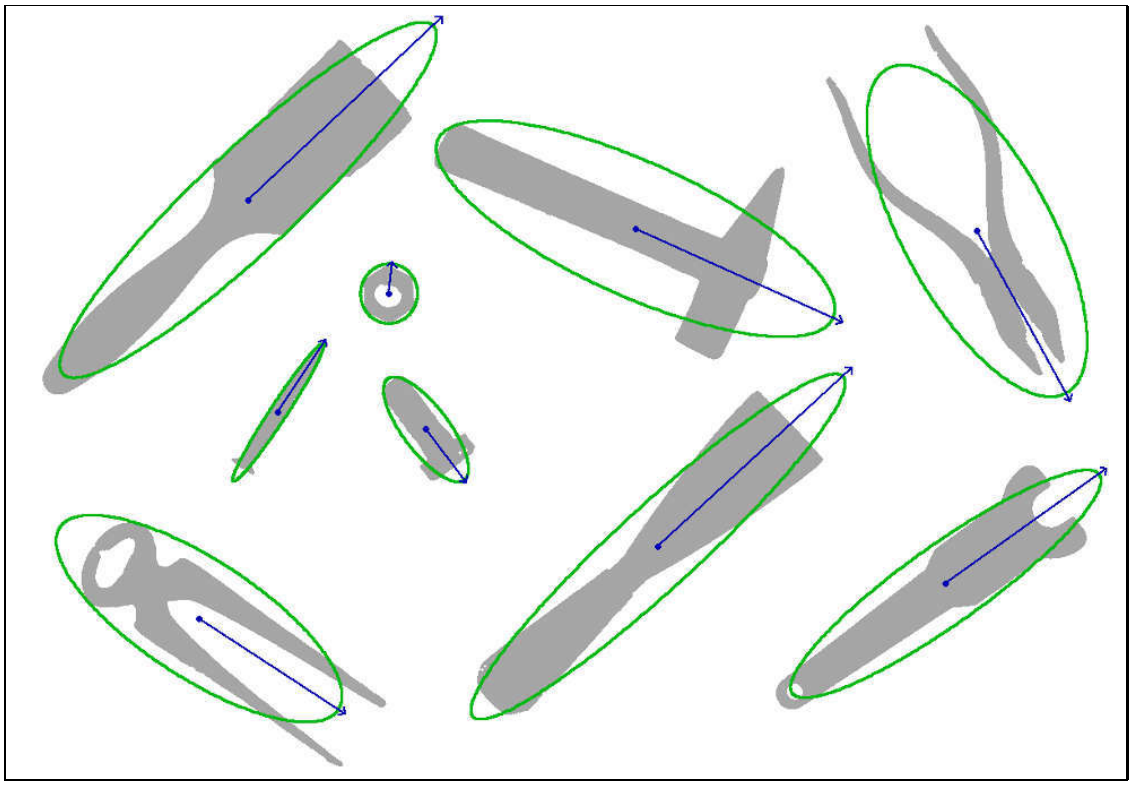


Figure 2.19 Orientation and eccentricity examples. The orientation θ (Eqn. (2.25)) is displayed for each connected region as a vector with the length proportional to the region's eccentricity value $\mathsf{Ecc}(\mathcal{R})$ (Eqn. (2.31)). Also shown are the ellipses (Eqns. (2.32) and (2.33)) corresponding to the orientation and eccentricity parameters.

respectively, with a_1, a_2 as defined in Eqn. (2.31) and $|\mathcal{R}|$ being the number of pixels in the region. The resulting parametric equation of the equivalent ellipse is

$$\begin{aligned}\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} &= \begin{pmatrix} \bar{x} \\ \bar{y} \end{pmatrix} + \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \cdot \begin{pmatrix} r_a \cdot \cos(t) \\ r_b \cdot \sin(t) \end{pmatrix} \\ &= \begin{pmatrix} \bar{x} + \cos(\theta) \cdot r_a \cdot \cos(t) - \sin(\theta) \cdot r_b \cdot \sin(t) \\ \bar{y} + \sin(\theta) \cdot r_a \cdot \cos(t) + \cos(\theta) \cdot r_b \cdot \sin(t) \end{pmatrix} \end{aligned} \tag{2.34}$$

for $0 \leq t < 2\pi$. If entirely *filled*, the region described by this ellipse would have the same (first and second order) central moments as the original region $\mathcal{R}$. Figure 2.19 shows a set of regions with overlaid orientation and eccentricity results.

Invariant moments

Normalized central moments are not affected by the translation or uniform scaling of a region (i. e., the values are invariant), but in general rotating the image will change these values. A classical solution to this problem is a clever

combination of simpler features known as "Hu's Moments" [37]:[12]

$$
\begin{aligned}
H_1 &= \bar{\mu}_{20} + \bar{\mu}_{02}, \\
H_2 &= (\bar{\mu}_{20} - \bar{\mu}_{02})^2 + 4\,\bar{\mu}_{11}^2, \\
H_3 &= (\bar{\mu}_{30} - 3\,\bar{\mu}_{12})^2 + (3\,\bar{\mu}_{21} - \bar{\mu}_{03})^2, \\
H_4 &= (\bar{\mu}_{30} + \bar{\mu}_{12})^2 + (\bar{\mu}_{21} + \bar{\mu}_{03})^2, \\
H_5 &= (\bar{\mu}_{30} - 3\,\bar{\mu}_{12}) \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot \left[(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - 3(\bar{\mu}_{21} + \bar{\mu}_{03})^2\right] \\
&\quad + (3\,\bar{\mu}_{21} - \bar{\mu}_{03}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03}) \cdot \left[3\,(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2\right], \\
H_6 &= (\bar{\mu}_{20} - \bar{\mu}_{02}) \cdot \left[(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2\right] \\
&\quad + 4\,\bar{\mu}_{11} \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03}), \\
H_7 &= (3\,\bar{\mu}_{21} - \bar{\mu}_{03}) \cdot (\bar{\mu}_{30} + \bar{\mu}_{12}) \cdot \left[(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - 3\,(\bar{\mu}_{21} + \bar{\mu}_{03})^2\right] \\
&\quad + (3\,\bar{\mu}_{12} - \bar{\mu}_{30}) \cdot (\bar{\mu}_{21} + \bar{\mu}_{03}) \cdot \left[3\,(\bar{\mu}_{30} + \bar{\mu}_{12})^2 - (\bar{\mu}_{21} + \bar{\mu}_{03})^2\right].
\end{aligned}
\tag{2.35}
$$

In practice, the logarithm of the results (that is, $\log(H_k)$) is used since the raw values can have a very large range. These features are also known as *moment invariants* since they are invariant under translation, rotation, and scaling. While defined here for binary images, they are also applicable to grayscale images; for further information, see [28, p. 517].

2.4.5 Projections

Image projections are one-dimensional representations of the image contents, usually computed parallel to the coordinate axis; in this case, the horizontal, as well as the vertical, projection of an image $I(u, v)$, with $0 \le u < M$, $0 \le v < N$, defined as

$$
P_{\text{hor}}(v_0) = \sum_{u=0}^{M-1} I(u, v_0) \qquad \text{for } 0 \le v_0 < N, \tag{2.36}
$$

$$
P_{\text{ver}}(u_0) = \sum_{v=0}^{N-1} I(u_0, v) \qquad \text{for } 0 \le u_0 < M. \tag{2.37}
$$

The *horizontal* projection $P_{\text{hor}}(v_0)$ (Eqn. (2.36)) is the sum of the pixel values in the image *row* v_0 and has length N corresponding to the height of the image. On the other hand, a *vertical* projection P_{ver} of length M is the sum of all the values in the image *column* u_0 (Eqn. (2.37)). In the case of a binary image with $I(u, v) \in 0, 1$, the projection contains the count of the foreground pixels in the corresponding image row or column.

[12] In order to improve the legibility of Eqn. (2.35) the argument for the region ($\mathcal{R}$) has been dropped; as an example, with the region argument, the first line would read $H_1(\mathcal{R}) = \bar{\mu}_{20}(\mathcal{R}) + \bar{\mu}_{02}(\mathcal{R})$, and so on.

```
1  public void run(ImageProcessor ip) {
2    int M = ip.getWidth();
3    int N = ip.getHeight();
4    int[] horProj = new int[N];
5    int[] verProj = new int[M];
6    for (int v = 0; v < N; v++) {
7      for (int u = 0; u < M; u++) {
8        int p = ip.getPixel(u, v);
9        horProj[v] += p;
10       verProj[u] += p;
11     }
12   }
13   // use projections horProj, verProj now
14   // ...
15 }
```

Program 2.4 Computation of horizontal and vertical projections. The `run()` method for an ImageJ plugin (`ip` is of type `ByteProcessor` or `ShortProcessor`) computes the projections in x and y directions simultaneously in a a single traversal of the image. The projections are represented by the one-dimensional arrays `horProj` and `verProj` with elements of type `int`.

Program Prog. 2.4 gives a direct implementation of the projection calculations as the `run()` method for an ImageJ plugin, where projections in both directions are computed during a single traversal of the image.

Projections in the direction of the coordinate axis are often utilized to quickly analyze the structure of an image and isolate its component parts; for example, in document images it is used to separate graphic elements from text blocks as well as to isolate individual lines (see the example in Fig. 2.20). In practice, especially to account for document skew, projections are often computed along the major axis of an image region Eqn. (2.25). When the projection vectors of a region are computed in reference to the centroid of the region along the major axis, the result is a rotation-invariant vector description (often referred to as a "signature") of the region.

2.4.6 Topological Properties

Topological features do not describe the shape of a region in continuous terms; instead, they capture its structural properties. They are typically invariant even under extreme image transformations. Two simple and robust topological features are the number of regions $N_R(\mathcal{R})$ and the number of holes $N_L(\mathcal{R})$ in those regions. $N_L(\mathcal{R})$ can be easily computed while finding the inner contours of a region, as described in Sec. 2.2.2.

A feature that can be derived from the number of holes is the so-called *Euler number* N_E, which is the difference between the number of connected

Figure 2.20 Example of the horizontal projection $P_{\text{hor}}(v)$ (right) and vertical projection $P_{\text{ver}}(u)$ (bottom) of a binary image.

regions N_R and the number of their holes N_H,

$$N_E(\mathcal{R}) = N_R(\mathcal{R}) - N_H(\mathcal{R}). \tag{2.38}$$

For a single connected region, the above formula simplifies to $1 - N_H$, so, for example, for an image of the number "**8**", $N_E = 1 - 2 = -1$, while for an image of the letter "**D**", $N_E = 1 - 1 = 0$.

Topological features are often used in combination with numerical features for classification, for example in optical character recognition (OCR) [12].

2.5 Exercises

Exercise 2.1

Trace, by hand, the execution of both variations (*depth-first* and *breadth-first*) of the flood-fill algorithm using the image shown in Fig. 2.21 and starting at coordinates $(5, 1)$.

Exercise 2.2

The implementation of the flood-fill algorithm in Prog. 2.1 places all the neighboring pixels of each visited pixel into either the *stack* or the *queue* without ensuring they are foreground pixels and that they lie within the image boundaries. The number of items in the stack or the queue can be reduced by ignoring (not inserting) those neighboring pixels that do not

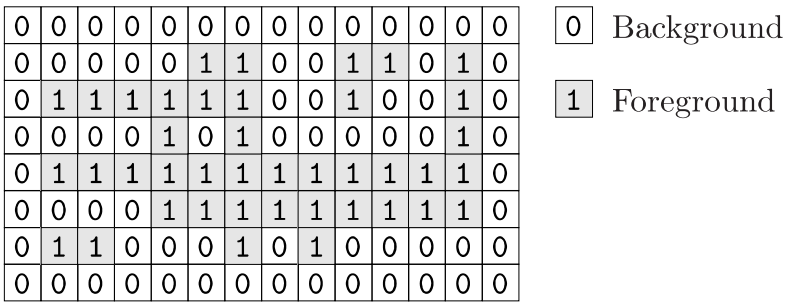


Figure 2.21 Binary image for Exercise 2.1.

meet the two conditions given above. Modify the *depth-first* and *breadth-first* variants given in Prog. 2.1 accordingly and compare the new running times.

Exercise 2.3
Implement an ImageJ plugin that encodes a grayscale image using run length encoding (Sec. 2.3.2) and stores it in a file. Develop a second plugin that reads the file and reconstructs the image.

Exercise 2.4
Calculate the amount of memory required to represent a contour with 1000 points in the following ways: (a) as a sequence of coordinate points stored as pairs of `int` values; (b) as an 8-chain code using Java `byte` elements, and (c) as an 8-chain code using only 3 bits per element.

Exercise 2.5
Implement a Java class for describing a binary image region using chain codes. It is up to you, whether you want to use an absolute or differential chain code. The implementation should be able to encode closed contours as chain codes and also reconstruct the contours given a chain code.

Exercise 2.6
While computing the convex hull of a region, the maximal diameter (maximum distance between two arbitrary points) can also be simply found. Devise an alternative method for computing this feature without using the convex hull. Determine the running time of your algorithm in terms of the number of points in the region.

Exercise 2.7
Implement an algorithm for comparing contours using their shape numbers Eqn. (2.3). For this purpose, develop a metric for measuring the distance between two normalized chain codes. Describe if, and under which conditions, the results will be reliable.

Exercise 2.8
Using Eqn. (2.10) as the basis, develop and implement an algorithm that computes the area of a region from its 8-chain code encoded contour. What type of discrepancy from the region's actual area (the number of pixels it contains) do you expect?

Exercise 2.9
Sketch an example binary region where the centroid lies outside of the region.

Exercise 2.10
Implement the moment features developed by Hu (Eqn. (2.35)) and show that they are invariant under scaling and rotation for both binary and grayscale images.

Exercise 2.11
There are alternative definitions for the eccentricity of a region Eqn. (2.31); for example,

$$\mathsf{Ecc}_2(\mathcal{R}) = \frac{\big(\mu_{20} - \mu_{02}\big)^2 + 4 \cdot \mu_{11}^2}{\big(\mu_{20} + \mu_{02}\big)^2} \qquad \text{[47, p. 394]},$$

$$\mathsf{Ecc}_3(\mathcal{R}) = \frac{(\mu_{20} - \mu_{02})^2 + 4 \cdot \mu_{11}}{m_{00}} \qquad \text{[46, p. 531]},$$

$$\mathsf{Ecc}_4(\mathcal{R}) = \frac{\sqrt{\mu_{20} - \mu_{02}} + 4 \cdot \mu_{11}}{m_{00}} \qquad \text{[2, p. 255]}.$$

Implement all four variations (including the one in Eqn. (2.31)) and contrast the results using suitably designed regions. Determine how these measures work and what their range of values is, and propose a geometrical interpretation for each.

Exercise 2.12
Write an ImageJ plugin that (a) finds (labels) all regions in a binary image, (b) computes the orientation and eccentricity for each region, and (c) shows the results as a direction vector and the equivalent ellipse on top of each region (as exemplified in Fig. 2.19). Hint: Use Eqn. (2.34) to develop a method for drawing ellipses at arbitrary orientations (not available in ImageJ).

Exercise 2.13
The Java method in Prog. 2.4 computes an image's horizontal and vertical projections. For document image processing, projections in the diagonal directions are also useful. Implement these projections and consider what role they play in document image analysis.