Wilhelm Burger · Mark J. Burge

Digital Image Processing

An Algorithmic Introduction

Third Edition

ERRATA

(**Errors** and **Notes**)

Springer
Berlin Heidelberg New York
Hong Kong London
Milano Paris Tokyo

the corresponding "goodness" estimate, as defined in Eqn. (9.23). The graph underneath each image shows the original histogram (gray) overlaid with the variance within the background σ_0^2 (blue), the variance within the foreground σ_1^2 (green), and the between-class variance σ_b^2 (red) for varying threshold values q. The dashed vertical line marks the position of the optimal threshold $q_{\max}$.

Due to the pre-calculation of the mean values, Otsu's method requires only three passes over the histogram and is thus very fast ($\mathcal{O}(K)$), in contrast to opposite accounts in the literature. The method is frequently quoted and performs well in comparison to other approaches [241], despite its long history and its simplicity. In general, the results are very similar to the ones produced by the iterative threshold selection ("isodata") algorithm described in Sec. 9.1.3.

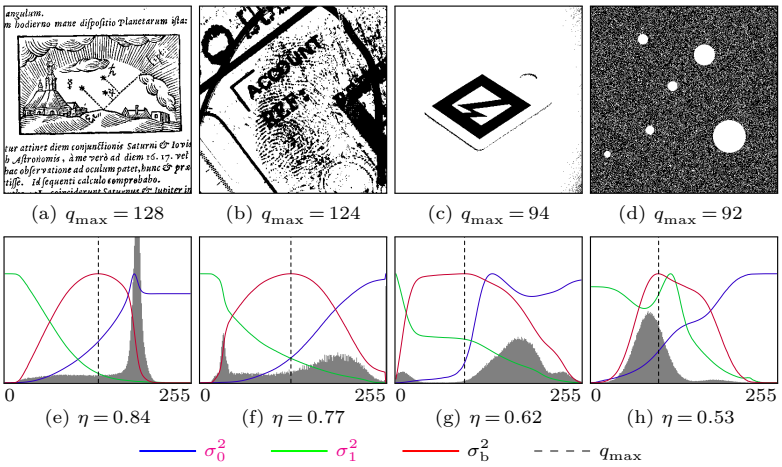

Fig. 9.4 Results of thresholding with Otsu's method. Calculated threshold values q and resulting binary images (a–d). Graphs in (e–h) show the corresponding within-background variance σ_0^2 (*blue*), the within-foreground variance σ_1^2 (*green*), and the between-class variance σ_b^2 (*red*), for varying threshold values $q = 0, \ldots, 255$. The optimal threshold $q_{\max}$ (dashed vertical line) is positioned at the maximum of σ_b^2. The value η denotes the "goodness" estimate for the thresholding, as defined in Eqn. (9.23).

9.1.5 Maximum Entropy Thresholding

Entropy is an important concept in information theory and particularly in data compression. It is a statistical measure that quantifies the average amount of information contained in the "messages" generated by a stochastic data source [118, 120]. For example, the MN pixels in an image I can be interpreted as a message of MN symbols, each taken independently from a finite alphabet of K (e.g., 256) different intensity values. Every pixel is assumed to be statically independent. Knowing the probability of each intensity value g to occur, entropy measures how likely it is to observe a particular image, or, in other words, how much we should be surprised to see such an image. Before going into further details, we briefly review the notion of probabilities in the context of images and histograms (see also Sec. 3.6.1).

For modeling the image generation as a random process, we first need to define an "alphabet", that is, a set of symbols

$$Z = \{0, 1, \ldots, K-1\}, \tag{9.24}$$

which in this case is simply the set of possible intensity values $g = 0, \ldots, K-1$, together with the probability $p(g)$ that a particular intensity value g occurs. These probabilities are supposed to be

$$H_0(q) = -\sum_{i=0}^{q} \frac{\mathsf{p}(i)}{\mathsf{P}_0(q)} \cdot \big[\log\big(\mathsf{p}(i)\big) - \log\big(\mathsf{P}_0(q)\big)\big] \tag{9.37}$$

$$= -\frac{1}{\mathsf{P}_0(q)} \cdot \sum_{i=0}^{q} \mathsf{p}(i) \cdot \big[\log\big(\mathsf{p}(i)\big) - \log\big(\mathsf{P}_0(q)\big)\big] \tag{9.38}$$

$$= -\frac{1}{\mathsf{P}_0(q)} \cdot \underbrace{\sum_{i=0}^{q} \mathsf{p}(i) \cdot \log\big(\mathsf{p}(i)\big)}_{S_0(q)} + \frac{1}{\mathsf{P}_0(q)} \cdot \underbrace{\sum_{i=0}^{q} \mathsf{p}(i)}_{=\mathsf{P}_0(q)} \cdot \underbrace{\log\big(\mathsf{P}_0(q)\big)}_{\text{indep. of } i}$$

$$= -\frac{1}{\mathsf{P}_0(q)} \cdot S_0(q) + \log\big(\mathsf{P}_0(q)\big). \tag{9.39}$$

Similarly, $H_1(q)$ in Eqn. (9.35) becomes

$$H_1(q) = -\sum_{i=q+1}^{K-1} \frac{\mathsf{p}(i)}{\mathsf{P}_1(q)} \cdot \big[\log\big(\mathsf{p}(i)\big) - \log\big(\mathsf{P}_1(q)\big)\big] \tag{9.40}$$

$$= -\frac{1}{1-\mathsf{P}_0(q)} \cdot S_1(q) + \log\big(1-\mathsf{P}_0(q)\big). \tag{9.41}$$

Given the estimated probability distribution $\mathsf{p}(i)$, the cumulative probability P_0 and the sums S_0, S_1 (see Eqns. (9.39)–(9.41)) can be calculated efficiently using the recurrence relations

$$\begin{aligned}
\mathsf{P}_0(q) &= \begin{cases} \mathsf{p}(0) & \text{for } q = 0, \\ \mathsf{P}_0(q-1) + \mathsf{p}(q) & \text{for } 0 < q < K, \end{cases} \\
S_0(q) &= \begin{cases} \mathsf{p}(0) \cdot \log\big(\mathsf{p}(0)\big) & \text{for } q = 0, \\ S_0(q-1) + \mathsf{p}(q) \cdot \log\big(\mathsf{p}(q)\big) & \text{for } 0 < q < K, \end{cases} \\
S_1(q) &= \begin{cases} 0 & \text{for } q = K-1, \\ S_1(q+1) + \mathsf{p}(q) \cdot \log\big(\mathsf{p}(q)\big) & \text{for } 0 \le q < K-1. \end{cases}
\end{aligned} \tag{9.42}$$

The complete procedure is summarized in Alg. 9.5, where the quantities $S_0(q), S_1(q)$ are obtained from the precalculated tables $\mathsf{S}_0, \mathsf{S}_1$, respectively. The algorithm performs three passes over the histogram of length K (two for filling the tables $\mathsf{S}_0, \mathsf{S}_1$ and one in the main loop), so its time complexity is $\mathcal{O}(K)$, like the algorithms described before.

Results obtained with this technique are shown in Fig. 9.5. The technique described in this section is simple and efficient, because it again relies entirely on the image's histogram. More advanced entropy-based thresholding techniques exist that, among other improvements, take into account the spatial structure of the original image. An extensive review of entropy-based methods can be found in [55].

9.1.6 Minimum Error Thresholding

The goal of minimum error thresholding is to optimally fit a combination (mixture) of Gaussian distributions to the image's histogram. Before we proceed, we briefly look at some additional concepts from statistics. Note, however, that the following material is only intended as a superficial outline to explain the elementary concepts. For a

```
1: MaximumEntropyThreshold(h)
     Input: h: [0,K−1] ↦ ℕ, a grayscale histogram. Returns the
     optimal threshold value or −1 if no threshold is found.
2:   K ← size(h)                                  ▷ number of intensity levels
3:   p ← Normalize(h)                             ▷ normalize histogram
4:   (S0, S1) ← MakeTables(p, K)                  ▷ tables for S0(q), S1(q)
5:   P0 ← 0                                       ▷ P0 ∈ [0, 1]
6:   qmax ← −1
7:   Hmax ← −∞                                    ▷ maximum joint entropy
8:   for q ← 0, ···, K−2 do                       ▷ check all possible threshold values q
9:     P0 ← P0 + p(q)
10:    P1 ← 1 − P0                                ▷ P1 ∈ [0, 1]
11:    if P0 = 0 then                             ▷ empty histogram so far, nothing to do
12:      continue                                 ▷ end current iteration
13:    if P1 = 0 then                             ▷ no more histogram entries, done
14:      break                                    ▷ terminate for-loop
15:    H0 ← −(1/P0)·S0(q) + log(P0)               ▷ background entropy H0(q)
16:    H1 ← −(1/P1)·S1(q) + log(P1)               ▷ foreground entropy H1(q)
17:    H01 = H0 + H1                              ▷ overall entropy for threshold q
18:    if H01 > Hmax then                         ▷ maximize H01(q)
19:      Hmax ← H01
20:      qmax ← q
21:  return qmax
22: MakeTables(p, K)
23:   Create maps S0, S1 : [0, K−1] ↦ ℝ
24:   s0 ← 0                                      ▷ S0(q) = Σ_{i=0}^{q} p(i)·log(p(i))
25:   for q ← 0, ···, K−1 do
26:     if p(q) > 0 then
27:       s0 ← s0 + p(q) · log(p(q))
28:     S0(q) ← s0
29:   s1 ← 0                                      ▷ S1(q) = Σ_{i=q+1}^{K−1} p(i)·log(p(i))
30:   for q ← K−1, ···, 0 do
31:     S1(q) ← s1
32:     if p(q) > 0 then
33:       s1 ← s1 + p(q) · log(p(q))
34:   return (S0, S1)
```

Alg. 9.5 Maximum entropy threshold selection [156]. Initially (outside the **for**-loop), the threshold q is assumed to be -1, which corresponds to the background class being empty ($n_0 = 0$) and all pixels assigned to the foreground class ($n_1 = MN$). The **for**-loop (lines 8–20) examines each possible threshold $q = 0, \ldots, K-2$. The optimal threshold value $(0, \ldots, K-2)$ is returned, or -1 if no valid threshold was found. The two checks in lines 11–14 are needed for the algorithm to handle strictly binary images as well. Note that the remaining body of the for-loop (i.e., the actual entropy calculation) is only executed if $\mathsf{P}_0 > 0$ and $\mathsf{P}_1 > 0$ and terminates as soon as $\mathsf{P}_1 = 0$.

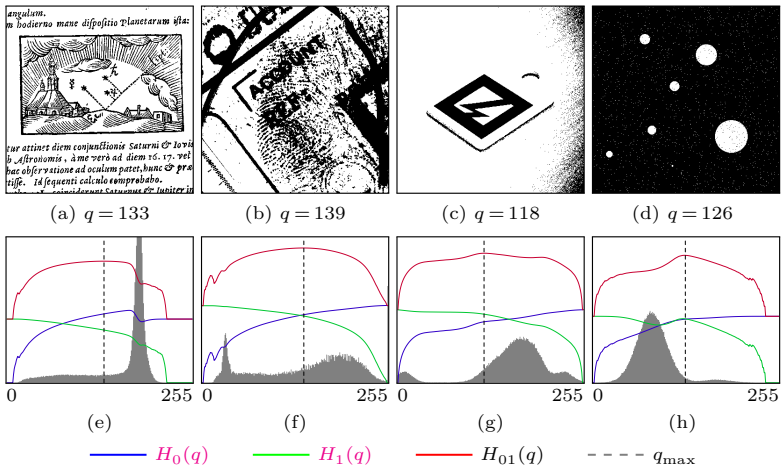

Fig. 9.5 Thresholding with the maximum-entropy method. Calculated threshold values q and resulting binary images (a–d). Graphs in (e–h) show the *background* entropy $H_0(q)$ (*blue*), *foreground* entropy $H_1(q)$ (*green*) and *overall* entropy $H_{01}(q) = H_0(q) + H_1(q)$ (*red*), for varying threshold values q. The optimal threshold $q_{\max}$ is found at the maximum of H_{01} (dashed vertical line).

Alg. 11.10
Geometric ellipse fitting ("distance-based").

1: FitEllipseGeometricDist$(X, \boldsymbol{p}_0)$
Input: $X = (\boldsymbol{x}_i)$, a collection of $n \geq 6$ 2D sample points $\boldsymbol{x}_i = (x_i, y_i)$; $\boldsymbol{p}_0 = (r_a, r_b, x_c, y_c, \theta)$, initial geometric ellipse parameters (typ. obtained by algebraic fitting). Returns the geometric parameters for the optimal ellipse.
2: $n \leftarrow |X|$ ▷ X, n are global
3: $\boldsymbol{z} \leftarrow \mathbf{0}_n$ ▷ "target" vector $(2n)$
4: $\boldsymbol{p}_{\text{opt}} \leftarrow$ SolveNLS(V, J, $\boldsymbol{z}, \boldsymbol{p}_0$) ▷ run the NLS solver
5: **return** $\boldsymbol{p}_{\text{opt}}$ ▷ ellipse param. $\boldsymbol{p}_{\text{opt}} = (r_a, r_b, x_c, y_c, \theta)$

6: V$(\boldsymbol{p})$ ▷ value function, to be called by SolveNLS()
Returns the n-vector $\boldsymbol{v} = (v_0, \ldots, v_{n-1})$ of model *values* for the current parameter "point" $\boldsymbol{p}$ (see Eqn. (11.108)).
7: $\boldsymbol{v} \leftarrow$ new vector $\in \mathbb{R}^n$ ▷ vector of "model" values
8: **for** $i \leftarrow 0, \ldots, n-1$ **do**
9: $\boldsymbol{x}_i \leftarrow X(i)$
10: $\breve{\boldsymbol{x}}_i \leftarrow$ ClosestEllipsePoint$(\boldsymbol{p}, \boldsymbol{x}_i)$ ▷ see Alg. 11.9
11: $d_i \leftarrow \|\breve{\boldsymbol{x}}_i - \boldsymbol{x}_i\|$ ▷ distance of $\boldsymbol{x}_i$ from ellipse
12: $\boldsymbol{v}(i) \leftarrow d_i$
13: **return** $\boldsymbol{v}$ ▷ $\boldsymbol{v} = (v_0, v_1, \ldots, v_{n-1})$

14: J$(\boldsymbol{p})$ ▷ Jacobian function, to be called by SolveNLS()
Builds and returns the $n \times 5$ Jacobian matrix $\mathbf{J}$ for the current parameter "point" $\boldsymbol{p} = (r_a, r_b, x_c, y_c, \theta)$ (see Eqn. (11.109)).
15: $\mathbf{R} \leftarrow \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$
16: **for** $i \leftarrow 0, \ldots, n-1$ **do**
17: $\boldsymbol{x}_i = (x, y)^\mathsf{T} \leftarrow X(i)$ ▷ $= (x_i, x_i)^\mathsf{T}$
18: $\breve{\boldsymbol{x}}_i = (\breve{x}, \breve{y})^\mathsf{T} \leftarrow$ ClosestEllipsePoint$(\boldsymbol{p}, \boldsymbol{x}_i)$ ▷ $= (\breve{x}_i, \breve{y}_i)^\mathsf{T}$
19: $d \leftarrow \|\breve{\boldsymbol{x}}_i - \boldsymbol{x}_i\|$ ▷ distance of $\boldsymbol{x}_i$ from ellipse
20: $\boldsymbol{u} = (u, v)^\mathsf{T} \leftarrow \mathbf{R}^\mathsf{T} \cdot (\boldsymbol{x}_i - \boldsymbol{x}_c)$ ▷ $= (u_i, v_i)^\mathsf{T}$
21: $\breve{\boldsymbol{u}} = (\breve{u}, \breve{v})^\mathsf{T} \leftarrow \mathbf{R}^\mathsf{T} \cdot (\breve{\boldsymbol{x}}_i - \boldsymbol{x}_c)$ ▷ $= (\breve{u}_i, \breve{v}_i)^\mathsf{T}$
22: $g \leftarrow \operatorname{sgn}\left[\frac{(u-\breve{u})\cdot\breve{u}}{r_a^2} + \frac{(v-\breve{v})\cdot\breve{v}}{r_b^2}\right] \Big/ \sqrt{\left(\frac{\breve{u}}{r_a^2}\right)^2 + \left(\frac{\breve{v}}{r_b^2}\right)^2}$
23: $\partial_{r_a} \leftarrow -g \cdot \frac{\breve{u}^2}{r_a^3}$
24: $\partial_{r_b} \leftarrow -g \cdot \frac{\breve{v}^2}{r_b^3}$
25: $\partial_{x_c} \leftarrow \frac{\breve{x}-x}{d}$
26: $\partial_{y_c} \leftarrow \frac{\breve{y}-y}{d}$
27: $\partial_\theta \leftarrow \frac{(y-\breve{y})(x_c-\breve{x}) - (x-\breve{x})(y_c-\breve{y})}{d}$
28: $\mathbf{J}(i, *) \leftarrow \left(\partial_{r_a}, \partial_{r_b}, \partial_{x_c}, \partial_{y_c}, \partial_\theta\right)$ ▷ i^{th} row of $\mathbf{J}$
29: **return** $\mathbf{J}$

dinates $\breve{\boldsymbol{x}}_i = (\breve{x}_i, \breve{y}_i)$ are the current "model" values and the coordinates of the data points $\boldsymbol{x}_i = (x_i, y_i)$ are the observed "target" values. Given n data points, the resulting vectors are of length $m = 2n$, i.e. (analogous to Eqn. (11.54)),

$$V(\boldsymbol{p}) = \begin{pmatrix} v_0 \\ v_1 \\ \vdots \\ v_{2i} \\ v_{2i+1} \\ \vdots \\ v_{2n-2} \\ v_{2n-1} \end{pmatrix} = \begin{pmatrix} \breve{x}_0(\boldsymbol{p}) \\ \breve{y}_0(\boldsymbol{p}) \\ \vdots \\ \breve{x}_i(\boldsymbol{p}) \\ \breve{y}_i(\boldsymbol{p}) \\ \vdots \\ \breve{x}_{n-1}(\boldsymbol{p}) \\ \breve{y}_{n-1}(\boldsymbol{p}) \end{pmatrix}, \quad \boldsymbol{z} = \begin{pmatrix} z_0 \\ z_0 \\ \vdots \\ z_{2i} \\ z_{2i+1} \\ \vdots \\ z_{2n-2} \\ z_{2n-1} \end{pmatrix} = \begin{pmatrix} x_0 \\ y_0 \\ \vdots \\ x_1 \\ y_1 \\ \vdots \\ x_{n-1} \\ y_{n-1} \end{pmatrix}. \tag{11.114}$$

Alg. 11.11 Geometric ellipse fit ("coordinate-based").

1: FitEllipseGeometricCoord($X, \boldsymbol{p}_0$)
Input: $X = (\boldsymbol{x}_i)$, a collection of $n \geq 6$ 2D sample points $\boldsymbol{x}_i = (x_i, y_i)$; $\boldsymbol{p}_0 = (r_a, r_b, x_c, y_c, \theta)$, initial geometric ellipse parameters (typ. obtained by algebraic fitting). Returns the geometric parameters for the optimal ellipse.
2: $n \leftarrow |X|$ ▷ X, n are global
3: $\boldsymbol{z} \leftarrow (x_0, y_0, \ldots, x_i, y_i, \ldots, x_{n-1}, y_{n-1})^\mathsf{T}$ ▷ "target" vector $(2n)$
4: $\boldsymbol{p}_{\text{opt}} \leftarrow$ SolveNLS(V, J, $\boldsymbol{z}, \boldsymbol{p}_0$) ▷ run the NLS solver
5: **return** $\boldsymbol{p}_{\text{opt}}$ ▷ ellipse param. $\boldsymbol{p}_{\text{opt}} = (r_a, r_b, x_c, y_c, \theta)$

6: V($\boldsymbol{p}$) ▷ value function, to be called by SolveNLS()
Returns the $2n$-vector $\boldsymbol{v} = (v_0, \ldots, v_{2n-1})$ of model *values* for the current parameter "point" $\boldsymbol{p}$ (see Eqn. (11.114)).
7: $\boldsymbol{v} \leftarrow$ new vector $\in \mathbb{R}^{2n}$ ▷ vector of "model" values
8: **for** $i \leftarrow 0, \ldots, n-1$ **do**
9: $\boldsymbol{x}_i \leftarrow X(i)$
10: $(\breve{x}_i, \breve{y}_i) \leftarrow$ ClosestEllipsePoint($\boldsymbol{p}, \boldsymbol{x}_i$) ▷ see Alg. 11.9
11: $\boldsymbol{v}(2i) \leftarrow \breve{x}_i$
12: $\boldsymbol{v}(2i+1) \leftarrow \breve{y}_i$
13: **return** $\boldsymbol{v}$ ▷ $\boldsymbol{v} = (\breve{x}_0, \breve{y}_0, \ldots \breve{x}_{n-1}, \breve{y}_{n-1})$

14: J($\boldsymbol{p}$) ▷ Jacobian function, to be called by SolveNLS()
Builds and returns the $2n \times 5$ Jacobian matrix $\mathbf{J}$ for the current parameter "point" $\boldsymbol{p}$ (see Eqn. (11.116)).
15: $\mathbf{J} \leftarrow$ new matrix $\in \mathbb{R}^{2n \times 5}$ ▷ Jacobian matrix of size $2n \times 5$
16: **for** $i \leftarrow 0, \ldots, n-1$ **do**
17: $\boldsymbol{x}_i \leftarrow X(i)$
18: $\mathbf{J}_i \leftarrow$ GetJacobian($\boldsymbol{p}, \boldsymbol{x}_i$) ▷ $= \mathbf{J}_i(p)$, see Eqn. (11.115)
19: $\mathbf{J}(2i, *) \leftarrow \mathbf{J}_i(0, *)$ ▷ copy row 0 of $\mathbf{J}_i$
20: $\mathbf{J}(2i+1, *) \leftarrow \mathbf{J}_i(1, *)$ ▷ copy row 1 of $\mathbf{J}_i$
21: **return** $\mathbf{J}$

22: GetJacobian($\boldsymbol{p}, \boldsymbol{x}_i$)
Builds and returns the 2×5 Jacobian $\mathbf{J}_i(\boldsymbol{p})$ for ellipse parameters $\boldsymbol{p} = (r_a, r_b, x_c, y_c, \theta)$ and a specific sample point $\boldsymbol{x}_i$ (see Eqn. (11.115)).
23: $\mathbf{R} \leftarrow \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}$
24: $\breve{\boldsymbol{x}}_i = (\breve{x}, \breve{y})^\mathsf{T} \leftarrow$ ClosestEllipsePoint($\boldsymbol{p}, \boldsymbol{x}_i$)
25: $\boldsymbol{u} = (u, v)^\mathsf{T} \leftarrow \mathbf{R}^\mathsf{T} \cdot (\boldsymbol{x}_i - \boldsymbol{x}_c)$
26: $\breve{\boldsymbol{u}} = (\breve{u}, \breve{v})^\mathsf{T} \leftarrow \mathbf{R}^\mathsf{T} \cdot (\breve{\boldsymbol{x}}_i - \boldsymbol{x}_c)$
27: $\mathbf{Q} \leftarrow \begin{pmatrix} 0 & 0 \\ v - \breve{v} & \breve{u} - u \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{r_a^2} & 0 \\ 0 & \frac{1}{r_b^2} \end{pmatrix} + \begin{pmatrix} \frac{\breve{u}}{r_a^2} & \frac{\breve{v}}{r_b^2} \\ \frac{\breve{v}}{r_b^2} & -\frac{\breve{u}}{r_a^2} \end{pmatrix}$
28: $\mathbf{T} \leftarrow \begin{pmatrix} 0 & 0 \\ -\frac{\breve{v}}{r_b^2} & \frac{\breve{u}}{r_a^2} \end{pmatrix}$
29: $\mathbf{U} \leftarrow \begin{pmatrix} 0 & 0 & -\cos(\theta) & -\sin(\theta) & v \\ 0 & 0 & \sin(\theta) & -\cos(\theta) & -u \end{pmatrix}$
30: $\mathbf{V} \leftarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & v - \breve{v} & \breve{u} - u \end{pmatrix} \cdot \begin{pmatrix} -\frac{\breve{u}^2}{r_a^3} & -\frac{\breve{v}^2}{r_b^3} & 0 & 0 & 0 \\ -\frac{2\breve{u}}{r_a^3} & 0 & 0 & 0 & 0 \\ 0 & -\frac{2\breve{v}}{r_b^3} & 0 & 0 & 0 \end{pmatrix}$
31: $\mathbf{W} \leftarrow \begin{pmatrix} 0 & 0 & 1 & 0 & y_c - \breve{y} \\ 0 & 0 & 0 & 1 & \breve{x} - x_c \end{pmatrix}$
32: $\mathbf{J}_i \leftarrow -\mathbf{R} \cdot \mathbf{Q}^{-1} \cdot (\mathbf{T} \cdot \mathbf{U} + \mathbf{V}) + \mathbf{W}$ ▷ 2×5 matrix
33: **return** $\mathbf{J}_i$

Alg. 12.3 RANSAC circle detection (Part 1). Procedure RansacFindBestCircle() (line 9) is called repeatedly until no more acceptable line is found. In each iteration the points inside the line's consensus set S are removed from the data set X (line 6).

```
1: RansacFindMultipleCircles(X, M, δ, s_min)
       Input: X = (x_0, ..., x_{n-1}), a collection of n 2D sample points
       x_i = (x_i, y_i); M, the number of iterations required; δ, the max.
       point-to-line distance for inliers; s_min, the min. "concensus" (num-
       ber of supporting inliers). Returns the set of detected circles (C).
2:     C ← { }                                          ▷ set of detected circles
3:     (C, S) ← RansacFindBestCircle(X, M, δ, s_min)    ▷ see Alg. 12.1
4:     while C ≠ nil do
5:         C ← C ∪ {C}                                  ▷ collect this circle
6:         X ← X \ S                                    ▷ remove all inliers from X
7:         (C, S) ← RansacFindBestCircle(X, M, δ, s_min)   ▷ S ⊆ X
8:     return L
9: RansacFindBestCircle(X, M, δ, s_min)
       Input: see above. Returns the circle C = ⟨x_c, y_c, r⟩ with the max-
       imum support (or nil if no such circle could be found) and the set
       of associated inliers S.
10:    (C_max, s_max) ← (nil, −1)                       ▷ strongest circle/score so far
11:    for i ← 1, ..., M do
12:        do                                           ▷ randomly pick 3 different points
13:            x_0 ← RandomPick(X)
14:            x_1 ← RandomPick(X)
15:            x_2 ← RandomPick(X)
16:        while (x_0 = x_1 ∨ x_0 = x_2 ∨ x_1 = x_2)
17:        C ← FitCircle3(x_0, x_1, x_2)                ▷ see Alg. 12.4
18:        if C ≠ nil then                              ▷ x_0, x_1, x_2 are not collinear
19:            S ← CollectInliers(X, C, δ)              ▷ count inliers only
20:            if |S| ≥ s_min ∧ |S| > s_max then
21:                (C_max, s_max, ) ← (C, |S|)
22:    if C_max = nil then
23:        return (nil, ∅)                              ▷ no acceptable circle found
24:    else
25:        S ← CollectInliers(X, C_max, δ)
26:        C′ ← FitCircleN(S)                           ▷ see Algs. 10.1–10.3
27:        return (C′, S)
28: CollectInliers(X, C, δ)                             ▷ collect all points close to circle C
29:    let C = ⟨x_c, y_c, r⟩:
30:    S ← { }
31:    for all x_i ∈ X do
32:        r_i ← ‖x_i − x_c‖                            ▷ distance of x_i from circle center
33:        if |r_i − r| ≤ δ then
34:            S ← S ∪ {x_i}
35:    return S
```

According to Fig. 12.5, approximately 4000 independent random tries are required for the sample size $k = 3$ to achieve $p = 0.99$ detection probability. Only $M = 1000$ tries were used in this experiment. The minimum number of inliers was set to $s_{\min} = 100$, which resulted in a third (phantom) circle to be detected (with 106 inliers).

Alg. B.2 Solving a generalized symmetric eigenvalue problem by Cholesky decomposition. Function eigen(), to perform ordinary eigendecomposition on matrix $\mathbf{Y}$ by returning eigenvalues $\boldsymbol{\lambda}$ and eigenvectors $\mathbf{V}$ (i.e., $\mathbf{Y}\cdot\boldsymbol{v}_k = \lambda_k\cdot\boldsymbol{v}_k$), is assumed to be available. Note that matrix $\mathbf{B}$ must be *non-singular* for the Cholesky decomposition to work (line 2).

1: GeneralSymmetricEigen($\mathbf{A},\mathbf{B}$)
   Input: $\mathbf{A},\mathbf{B}$, symmetric matrices of size $n\times n$, $\mathbf{B}$ positive definite. Returns the vector of eigenvalues $\boldsymbol{\lambda}=(\lambda_0,...,\lambda_{n-1})$ and a matrix $\mathbf{X}=(\boldsymbol{x}_0|...|\boldsymbol{x}_{n-1})$ whose column vectors are the associated eigenvectors, i.e., $\mathbf{A}\cdot\boldsymbol{x}_k = \lambda_k\cdot\mathbf{B}\cdot\boldsymbol{x}_k$.
2: $\mathbf{L}\leftarrow$ solve($\mathbf{B}=\mathbf{L}\cdot\mathbf{L}^\mathsf{T}$) ▷ by Cholesky decomposition
3: $\mathbf{Q}\leftarrow$ solve($\mathbf{L}\cdot\mathbf{Q}=\mathbf{A}$) ▷ e.g., by LU decomposition
4: $\mathbf{Y}\leftarrow$ solve($\mathbf{L}\cdot\mathbf{Y}=\mathbf{Q}^\mathsf{T}$) ▷ alternatively $\mathbf{Y}=\mathbf{L}^{-1}\cdot\mathbf{A}\cdot(\mathbf{L}^{-1})^\mathsf{T}$
5: $(\boldsymbol{\lambda},\mathbf{V})\leftarrow$ eigen($\mathbf{Y}$) ▷ $\boldsymbol{\lambda}=(\lambda_0,...,\lambda_{n-1})$, $\mathbf{V}=(\boldsymbol{v}_0\mid...\mid\boldsymbol{v}_{n-1})$
6: $\mathbf{X}\leftarrow$ solve($\mathbf{L}^\mathsf{T}\cdot\mathbf{X}=\mathbf{V}$) ▷ eigenvectors $\mathbf{X}=(\boldsymbol{x}_0\mid...\mid\boldsymbol{x}_{n-1})$
7: **return** $(\boldsymbol{\lambda},\mathbf{X})$

$$\mathbf{Y}=\mathbf{L}^{-1}\cdot\mathbf{A}\cdot(\mathbf{L}^{-1})^\mathsf{T} \tag{B.54}$$

has the *same* eigenvalues as the system in Eqn. (B.53) but different eigenvectors $\boldsymbol{y}_k$, from which the system's eigenvectors can be found as $\boldsymbol{x}_k=(\mathbf{L}^{-1})^\mathsf{T}\cdot\boldsymbol{y}_k$ [214, Sec. 11.0.5]. Thus, given symmetric matrices $\mathbf{A}$ and $\mathbf{B}$ of size $n\times n$, solving the problem in Eqn. (B.53) involves the following steps:

1. Calculate $\mathbf{L}$, such that $\mathbf{L}\cdot\mathbf{L}^\mathsf{T}=\mathbf{B}$ (by Cholesky decomposition).
2. Create matrix $\mathbf{Y}=\mathbf{L}^{-1}\cdot\mathbf{A}\cdot(\mathbf{L}^{-1})^\mathsf{T}$.
3. Get the eigenvalues $\boldsymbol{\lambda}=(\lambda_0,\ldots,\lambda_{n-1})$ of $\mathbf{Y}$ and the associated matrix of eigenvectors $\mathbf{V}=(\boldsymbol{v}_0|...|\boldsymbol{v}_{n-1})$.
4. Return eigenvalues $\boldsymbol{\lambda}$ and the matrix of transformed eigenvectors $\mathbf{X}=(\mathbf{L}^{-1})^\mathsf{T}\cdot\mathbf{V}$ (alternatively obtained by solving $\mathbf{L}^\mathsf{T}\cdot\mathbf{X}=\mathbf{V}$).

As suggested in [214, Sec. 11.0.5], matrix $\mathbf{Y}$ Step 2) can be efficiently calculated as follows:

2a. find $\mathbf{Q}$ by solving $\mathbf{L}\cdot\mathbf{Q}=\mathbf{A}^\mathsf{T}=\mathbf{A}$ (since $\mathbf{A}$ is symmetric), then
2b. find $\mathbf{Y}$ by solving $\mathbf{L}\cdot\mathbf{Y}=\mathbf{Q}^\mathsf{T}$,

e.g., by using a LU decomposition solver (see Sec. B.8.1). The complete calculation is summarized in Alg. B.2.

B.6 Homogeneous Coordinates

Homogeneous coordinates are an alternative representation of points in multi-dimensional space. They are commonly used in 2D and 3D geometry because they can greatly simplify the description of certain transformations. For example, affine and projective transformations become matrices with homogeneous coordinates and the composition of transformations can be performed by simple matrix multiplication.[8]

To convert a given n-dimensional *Cartesian* point $\boldsymbol{x}=(x_0,\ldots,x_{n-1})^\mathsf{T}$ to *homogeneous* coordinates $\underline{\boldsymbol{x}}$, we use the notation[9]

$$\hom(\boldsymbol{x})=\underline{\boldsymbol{x}}. \tag{B.55}$$

This operation increases the dimensionality of the original vector by one by inserting the additional element $\underline{x}_n=1$, that is,

[8] See Sec. 21.1.2.
[9] The operator hom( ) is introduced here for convenience and clarity.