10.2.7 Edge linking and boundary detection

Edge detection is always followed by edge linking

Local processing

- **Analyze pixels in small neighbourhood S_{xy} of each edge point**
- **Pixels that are similar are linked**
- **Principal properties used for establishing similarity:**

(1) $M(x,y) = |\nabla f(x,y)|$**: Magnitude of gradient vector**

(2) $\alpha(x,y)$**: Direction of gradient vector**

- **Edge pixel with coordinates** (s,t) **in** S_{xy} **is similar in magnitude to pixel at** (x,y) **if**

$$|M(s,t) - M(x,y)| \leq E$$

- **Edge pixel with coordinates** (s,t) **in** S_{xy} **has an angle similar to pixel at** (x,y) **if**

$$|\alpha(s,t) - \alpha(x,y)| \leq A$$

- **Edge pixel** (s,t) **in** S_{xy} **is linked with** (x,y) **if both criteria are satisfied**

The above strategy is expensive. A record has to be kept of all linked points by, for example, assigning a different label to every set of linked points

Simplification suitable for real-time applications:

(1) Compute $M(x, y)$ and $\alpha(x, y)$ of input image $f(x, y)$

(2) Form binary image

$$g(x,y) = \begin{cases} 1, & \text{if } M(x,y) > T_M \text{ AND } \alpha(x,y) \in [A - T_A, A + T_A] \\ 0, & \text{otherwise} \end{cases}$$

(3) Scan rows of g and fill (set to 1) all gaps (sets of 0s) in each row that do not exceed a specified length K

(4) Rotate g by θ and apply step (3). Rotate result back by $-\theta$.

Image rotation is expensive $\Rightarrow$ when linking in numerous directions is required, steps (3) and (4) are combined into a single, radial scanning procedure.

Example 10.10:

a b c
d e f

FIGURE 10.27 (a) A 534 × 566 image of the rear of a vehicle. (b) Gradient magnitude image. (c) Horizontally connected edge pixels. (d) Vertically connected edge pixels. (e) The logical OR of the two preceding images. (f) Final result obtained using morphological thinning. (Original image courtesy of Perceptics Corporation.)

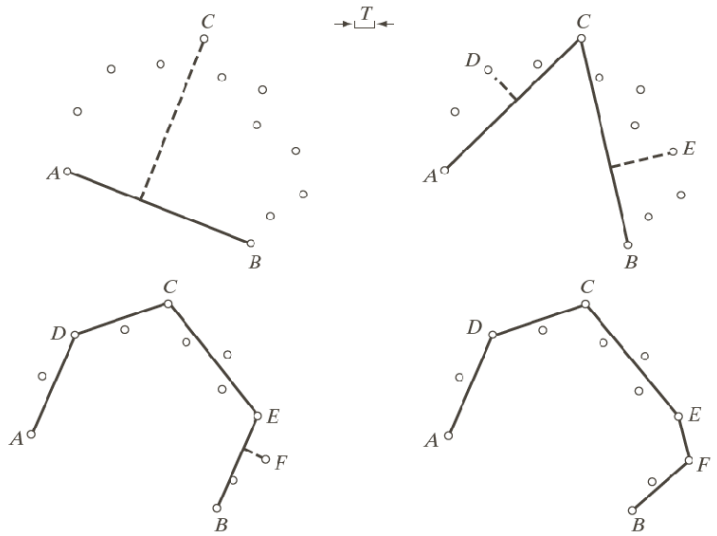
Regional processing (Polygonal approximations)

A conceptual understanding of this idea is sufficient

Requirements: (1) Two starting points must be specified; (2) All the points must be ordered

Large distance between successive points, relative to the distance between other points $\Rightarrow$ boundary segment (open curve) $\Rightarrow$ end points used as starting points

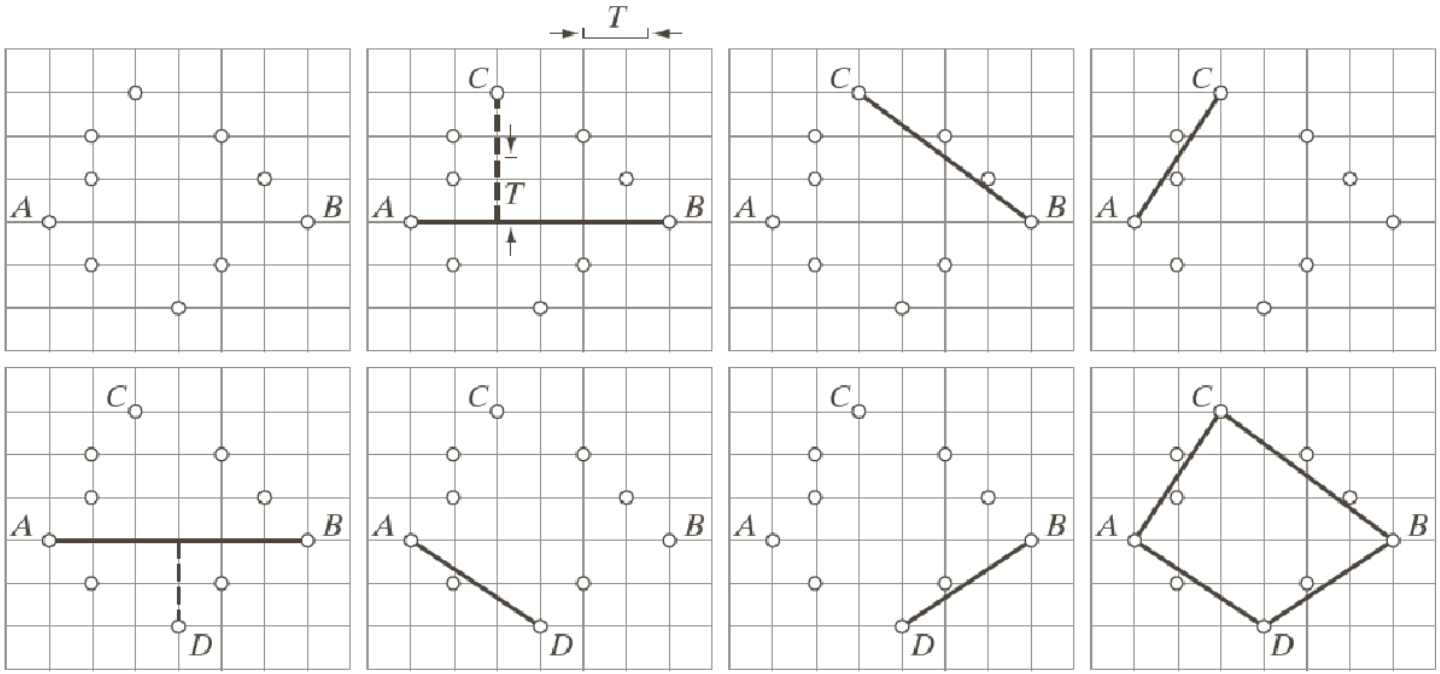
Seperation between points uniform $\Rightarrow$ boundary (closed curve) $\Rightarrow$ extreme points used as starting points



a b
c d

FIGURE 10.28 Illustration of the iterative polygonal fit algorithm.

Example 10.11: Edge linking using polygonal approximation



a b c d
e f g h

FIGURE 10.29 (a) A set of points in a clockwise path (the points labeled A and B were chosen as the starting vertices). (b) The distance from point C to the line passing through A and B is the largest of all the points between A and B and also passed the threshold test, so C is a new vertex. (d)–(g) Various stages of the algorithm. (h) The final vertices, shown connected with straight lines to form a polygon. Table 10.1 shows step-by-step details.

Example 10.12: (READ)

Global Processing using the Hough transform

- **We attempt to link edge pixels that lie on specified curves**

- **Brute force method:**

When the specified curve is a straight line, the line between each pair of edge pixels in the image is considered. The distance between every other edge pixel and the line in question is then calculated.

When the distance is less than a specified threshold, the pixel is considered to be part of the line

Number of calculations for n edge pixels:

Number of possible lines: $\sum_{k=1}^{n-1} k = \frac{n(n-1)}{2} \approx n^2$

Distances per line: n

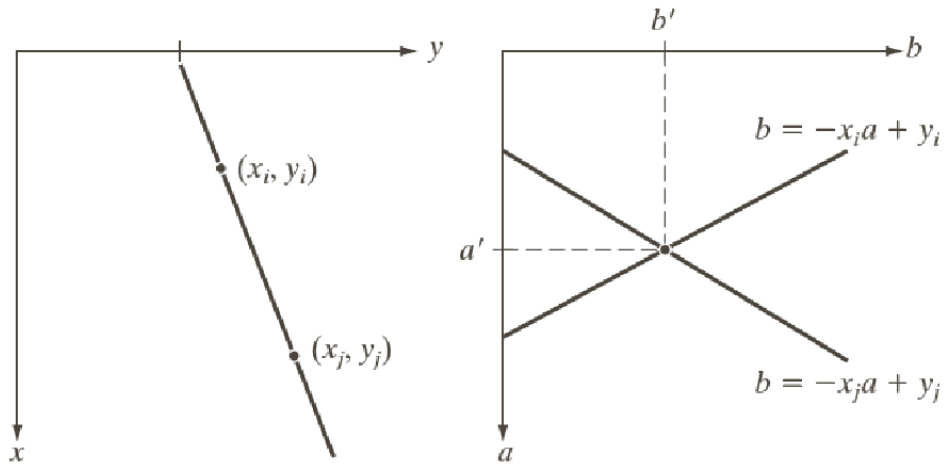
Total number of distances: $\frac{n^2(n-1)}{2} \approx n^3$

When $n = 256^2$ then number of calculations is 1.4×10^{14} !!!

- **Hough transform (1962)**

When different values for a and b are considered, $y_i = ax_i + b$ gives all possible lines through the point (x_i, y_i)

The equation $b = -x_i a + y_i$ gives one line in the ab-plane for a specific (x_i, y_i)



a b

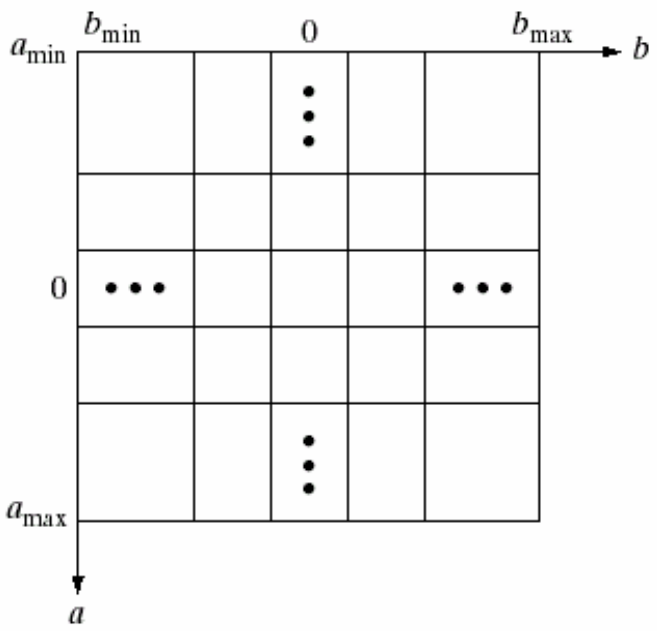
FIGURE 10.31
(a) xy-plane.
(b) Parameter space.

When another point (x_j, y_j) is considered, $b = -x_j a + y_j$ represents another line in the ab-plane

Suppose that these two lines intersect at the point (a', b'), then $y = a'x + b'$ represents the line in the xy-plane on which both (x_i, y_i) and (x_j, y_j) lie

Since a computer can only deal with a finite number of straight lines, we subdivide the parameter space ab into a finite number of accumulator cells...

FIGURE 10.18
Subdivision of the parameter plane for use in the Hough transform.



(Fig from Ed 2) $- - - - - - - - >$

Algorithm:

(1) Set all cells equal to zero

(2) For every (x_k, y_k)

(2.1) Let $a =$ **every subdivision on the** a**-axis**

(2.2) Calculate $b = -x_k a + y_k$

(2.3) Round off b **to the nearest allotted value on the** b**-axis**

(2.4) Increment accumulator cell (a, b) **with** 1

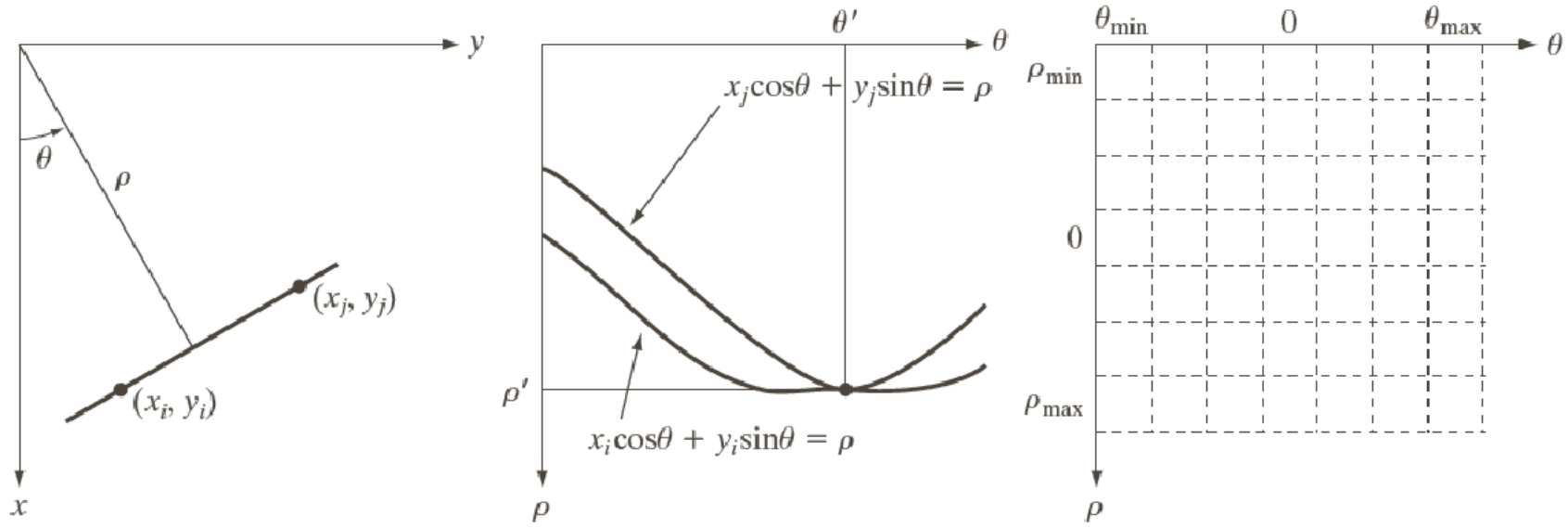
Note: When there are K **subdivisions on the** a**-axis, we need only** nK **calculations, which is linear (recall that we needed** n^3 **calculations for the brute force method)**

We still have a problem though, since $-\infty < a < \infty$ and $-\infty < b < \infty$!

In order to deal with this problem, we now represent a straight line as follows

$$x\cos\theta + y\sin\theta = \rho$$

so that $(a, b) \rightarrow (\rho, \theta)$



a b c

FIGURE 10.32 (a) (ρ, θ) parameterization of line in the xy-plane. (b) Sinusoidal curves in the $\rho\theta$-plane; the point of intersection (ρ', θ') corresponds to the line passing through points (x_i, y_i) and (x_j, y_j) in the xy-plane. (c) Division of the $\rho\theta$-plane into accumulator cells.

Derivation:

$$y = ax + b$$

$$\frac{b}{\rho} = \csc\theta = \frac{1}{\sin\theta}$$

$$b = \rho \csc\theta$$

$$a = -\frac{1}{\tan\theta} \quad \textbf{(negative reciprocal)}$$

$$= -\frac{\cos\theta}{\sin\theta}$$

$$y = -\frac{\cos\theta}{\sin\theta}x + \rho\csc\theta \quad \Rightarrow \quad y\sin\theta = -x\cos\theta + \rho \quad \Rightarrow \quad x\cos\theta + y\sin\theta = \rho$$

Now we have that $\rho \in [-\sqrt{2}D, \sqrt{2}D]$ and $\theta \in [-90^{\mathbf{o}}, 90^{\mathbf{o}}]$, where $\sqrt{2}D$ is the diagonal distance between two opposite corners in the image. Problem solved!

Algorithm:

(1) Set all cells equal to zero

(2) For every (x_k, y_k)

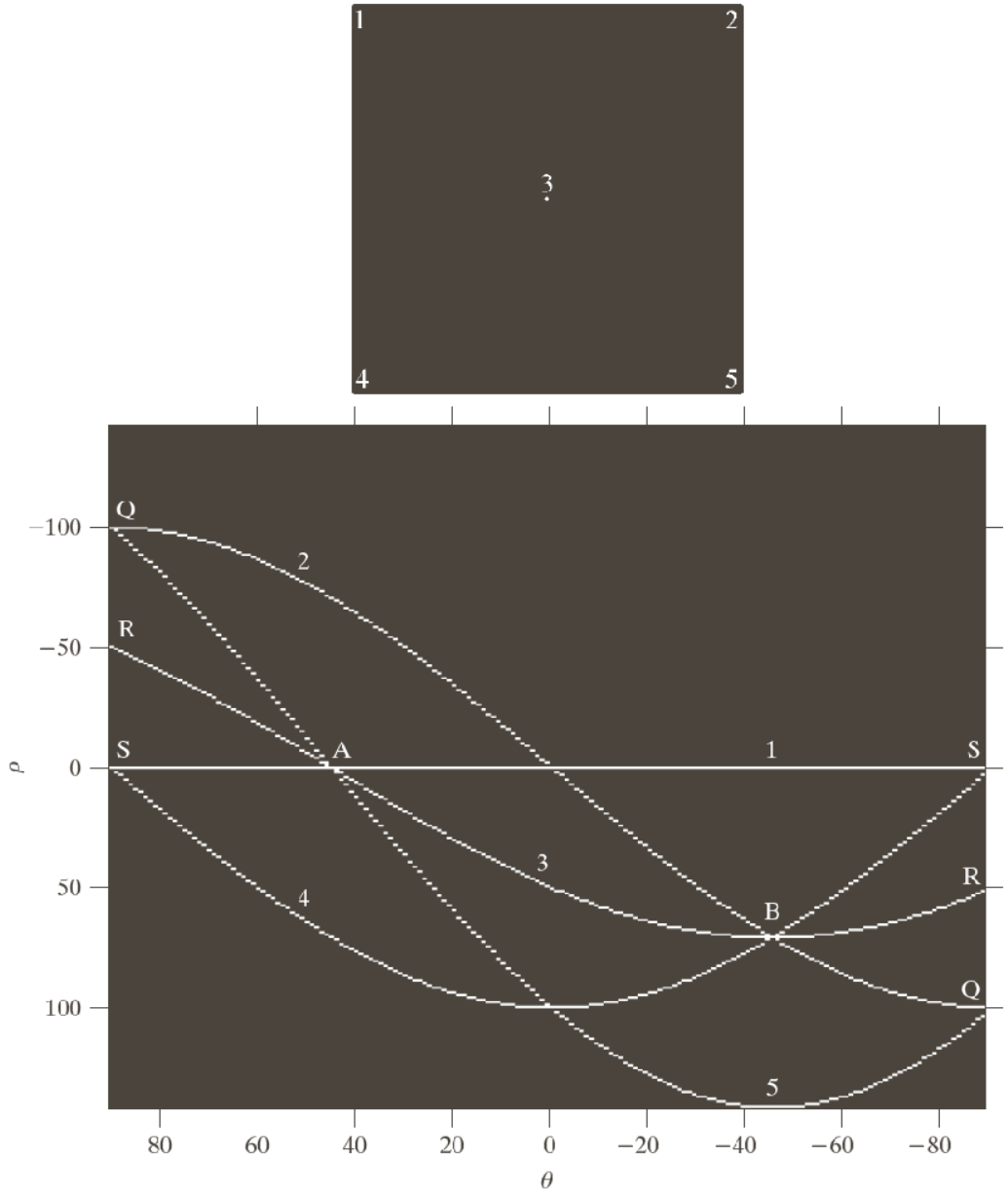
(2.1) Let $\theta =$ every subdivision on the θ-axis

(2.2) Calculate $\rho = x_k \cos\theta + y_k \sin\theta$

(2.3) Round off ρ to the nearest allotted value on the ρ-axis

(2.4) Increment accumulator cell (ρ, θ) with 1

Example 10.13: Illustration of Hough transform properties



a
b

FIGURE 10.33
(a) Image of size 101 × 101 pixels, containing five points.
(b) Corresponding parameter space.
(The points in (a) were enlarged to make them easier to see.)

Algorithm for edge linking:

(1) Compute $|\nabla f|$ and isolate edge pixels through thresholding

(2) Specify subdivisions in the $\rho\theta$-plane

(3) Apply Hough transform to edge pixels

(4) Identify accumulator cells with highest values

(5) Examine continuity of pixels that constitute cell

(6) Link these pixels if gaps are smaller than threshold

Extension to more general graphs

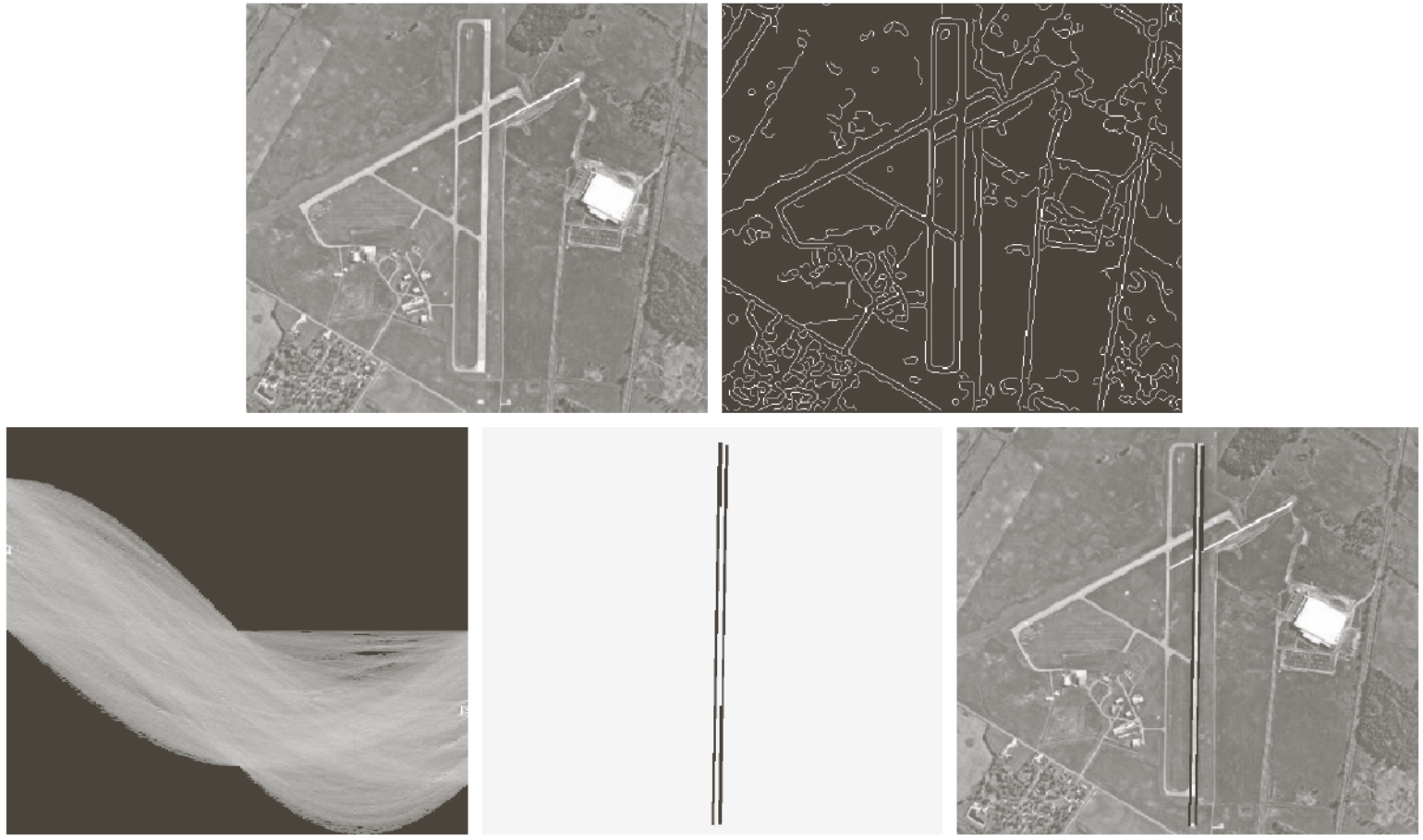
- **Hough transform applicable to any graph $g(\mathbf{v}, \mathbf{c}) = 0$, where $\mathbf{v}$ is vector of coordinates and $\mathbf{c}$ is vector of coefficients**

- **Example: Find the points that lie on a circle**

$$(x - c_1)^2 + (y - c_2)^2 = c_3^2$$

The presence of three parameters (c_1, c_2 and c_3) results in a 3-D parameter space with cubelike cells and accumulators of the form $A(i, j, k)$!

Example 10.14: Using the Hough transform for edge linking



a b
c d e

FIGURE 10.34 (a) A 502 × 564 aerial image of an airport. (b) Edge image obtained using Canny's algorithm.