Hough Transform

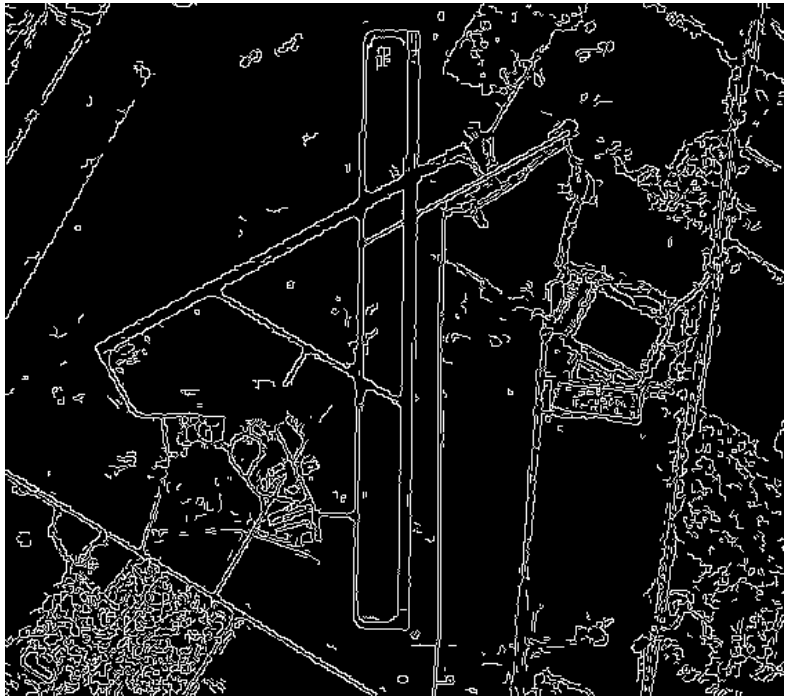
Outline

- Hough transform
  - Hough line
  - Hough circle

What is Hough transform?

- We’ve learnt Canny edge detection method
- However, it does not “characterize” the lines in the image
  - Characterize means give a parametric description (slope, intercept, etc.)
- Hough transform is a feature extraction method for detecting simple shapes in an image, such as lines or circles
  - Simple shapes means shapes that can be represented by only a few parameters, e.g. line is represented by two parameters, circle is represented by three parameters (center (x,y) and radius r)

Hough Line

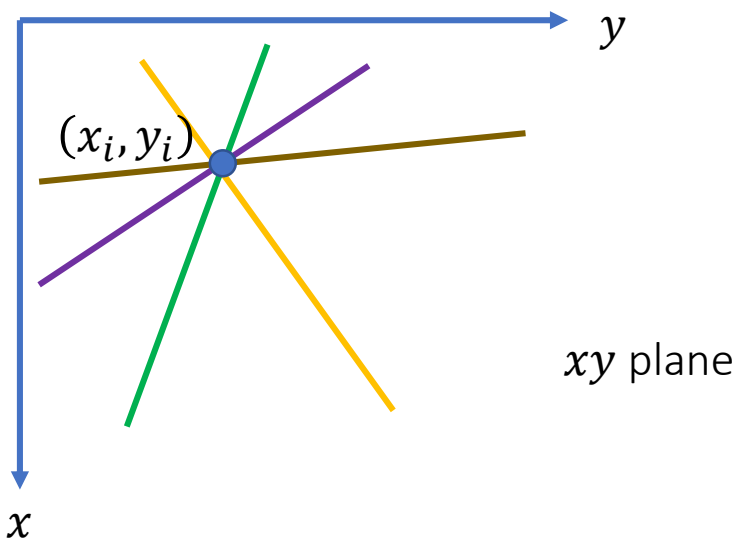


Hough line

- Given n points in an image, we want to find subsets of these points that lie on straight lines.
- Possible solution
  1. Find all lines determined by every pair of points:
     - $n(n-1) \sim n^2$ lines
  2. Find all subsets of points that are close to particular lines
     - $n \cdot n(n-1) \sim n^3$ comparisons of every point to all lines.
  3. Computationally prohibitive

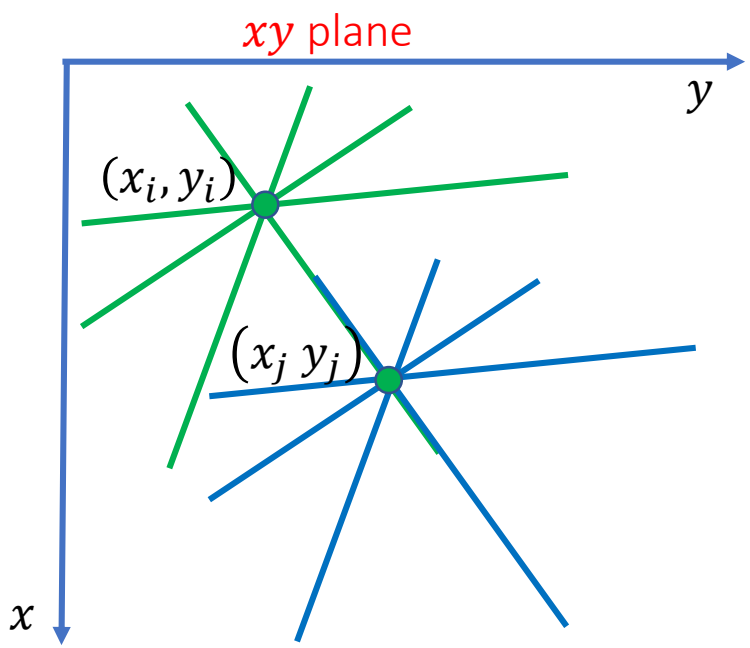
Hough Line

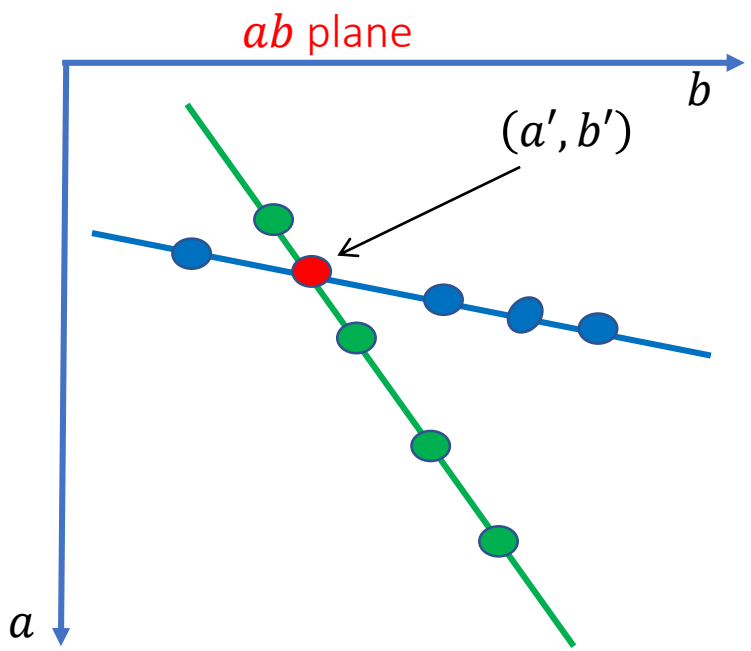
- Hough proposed an alternative approach (1962)
- Given a point (x_i, y_i), a line that pass through (x_i, y_i) has an equation $y_i = ax_i + b$.
- Infinitely many lines pass through (x_i, y_i), but all them satisfy the equation $y_i = ax_i + b$ for varying values a, b.



Hough Line

- $y_i = ax_i + b$.
- We can write the equation as $b = -ax_i + y_i$ and consider ab plane
  - ab plane is also called a parameter space
  - A line in xy plane is represented as a point in ab plane





Hough Line

- **The idea**

  for all values of a in parameter plane

  for all values of b in parameter plane

  count how many lines pass through (a,b)

  if the value of counter at point $(a, b\ )$ > Threshold

  (a, b) represents a line in xy-plane

What is the problem?

The value of a (slope) approaches infinity as the line approaches vertical direction.

So we need infinite memory to be able to store the ab-plane.

Hough line

- One way to overcome this is to *use the normal representation* of a line:

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

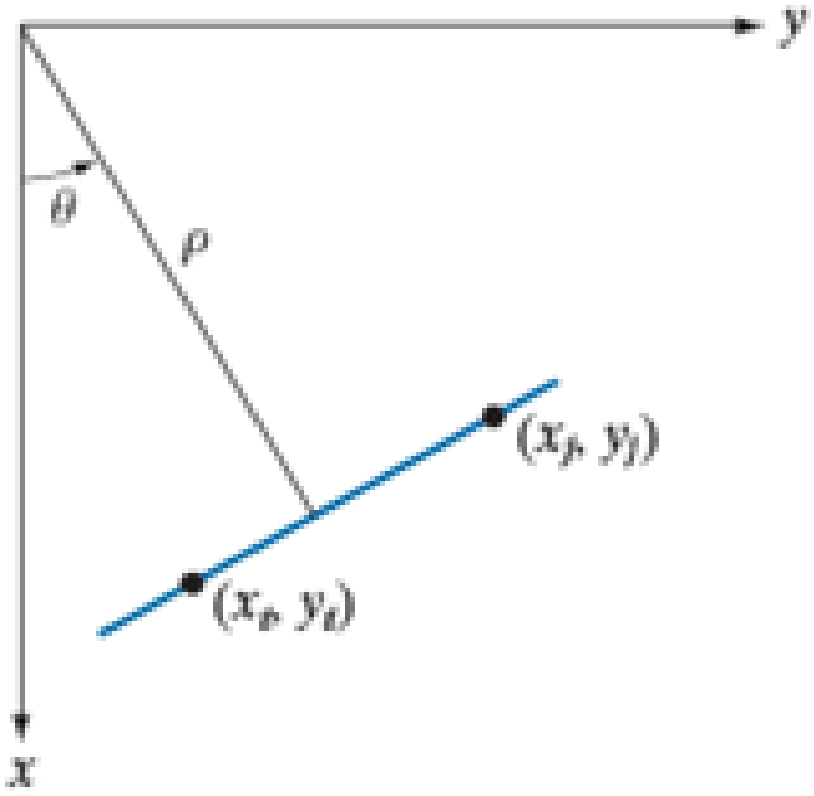
where:

ρ (rho) = distance from origin to the line.

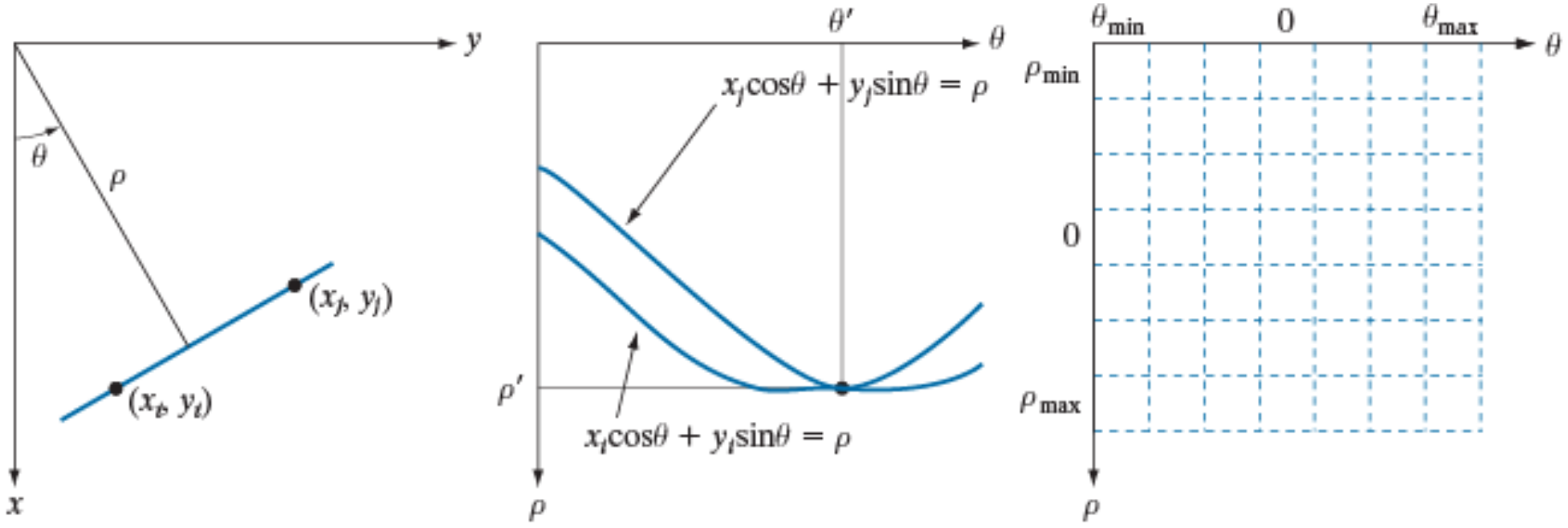
[-D to D].

D is the diagonal length of the image.

θ = angle from origin to the line. [-90° to 90°]



Hough line - Accumulator



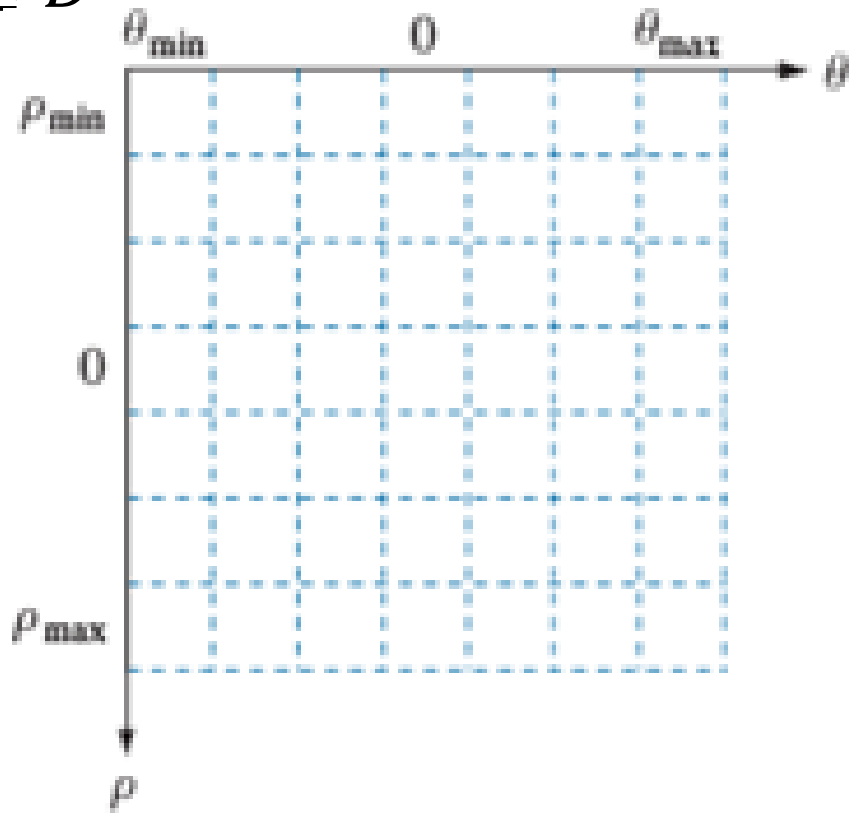
a b c

FIGURE 10.29 (a) (ρ,θ) parameterization of a 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.

Hough Transform – the algorithm

1. Subdivide the $\rho\theta$ parameter space into accumulator cells $A(i,j)$, where $-90° \leq \theta \leq 90°, -D \leq \rho \leq D$
   - D is the diagonal length of the image
   - The cell (i,j) is associated with parameters (ρ_i, θ_i)

2. Initialization: $A(i,j) = 0$ for all i,j



Hough Transform – the algorithm

3. **for** every non-background point (x_k, y_k)

    for every value of θ

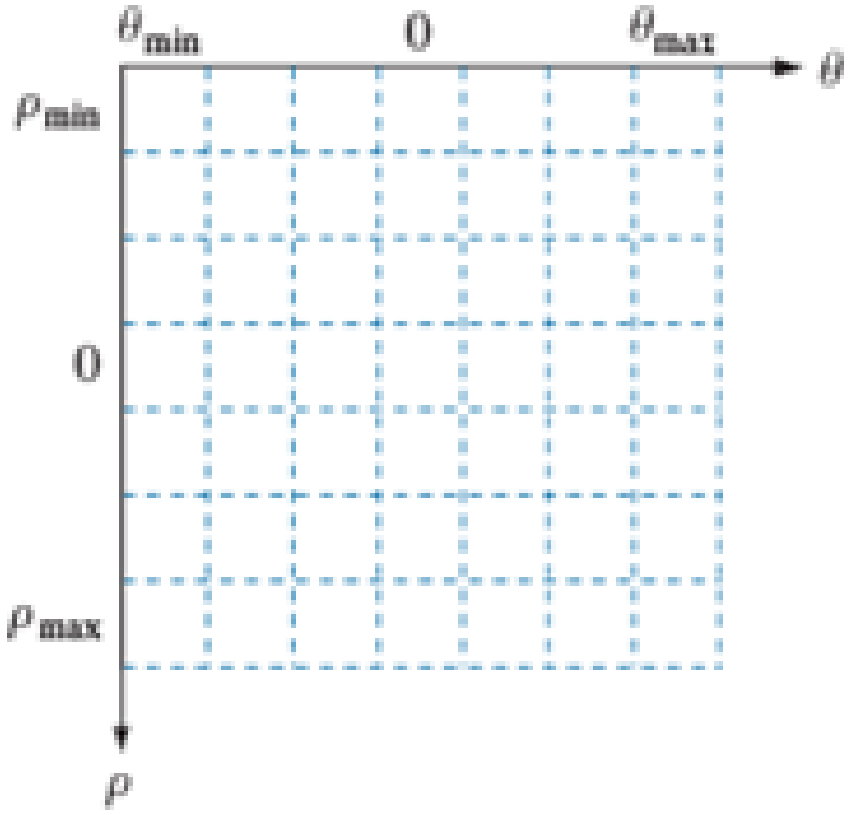
        calculate $\rho = x_k cos\theta + y_k\ sin\theta$

        $A(\rho, \theta) = A(\rho, \theta) + 1$

4. **if** the value of $A(\rho, \theta)$ > Threshold

    (ρ, θ)represents a line in xy-plane

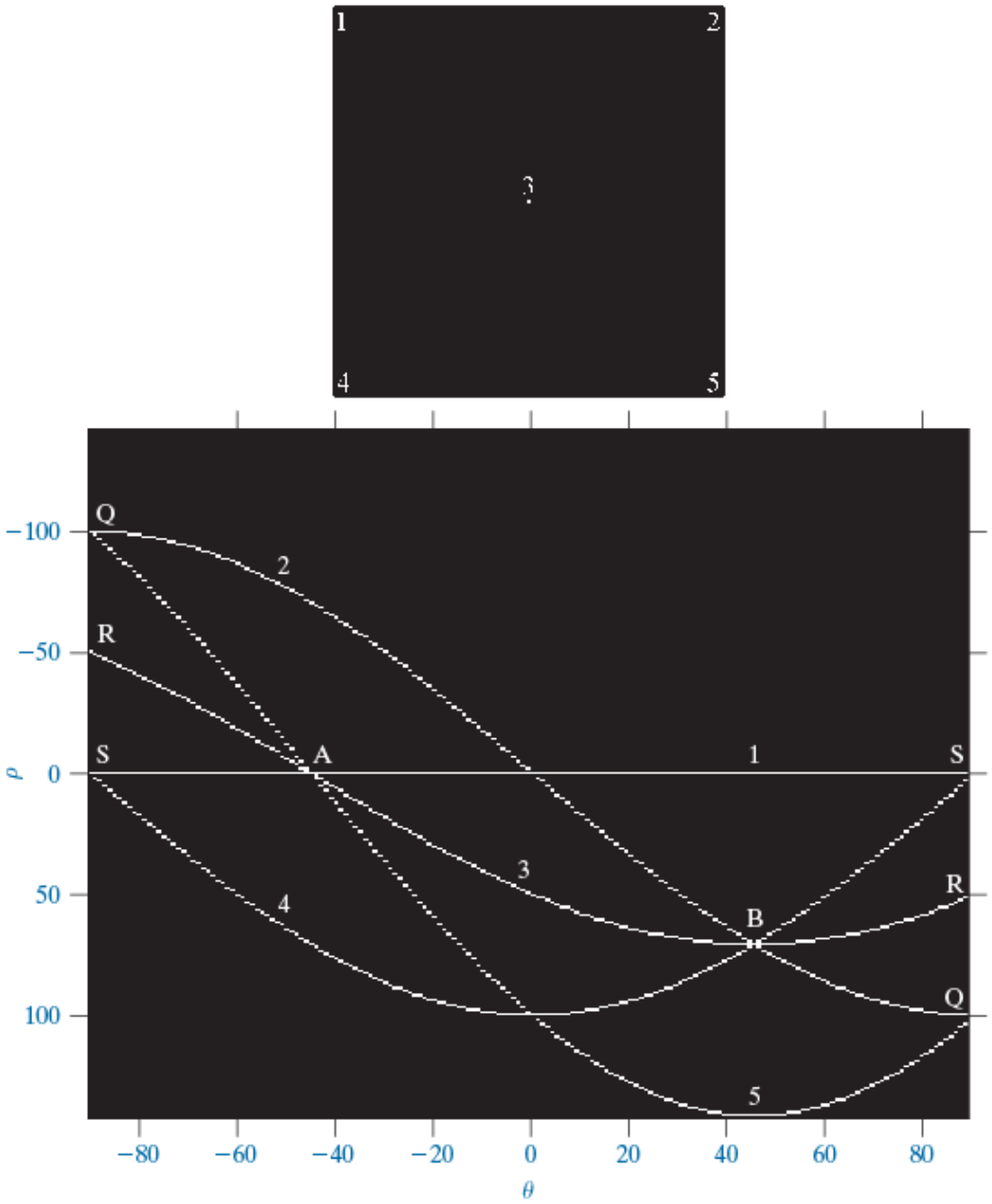
- **Note:** $A(\rho, \theta)$ means K points lie on the line with parameters (ρ, θ)



Example

a
b

FIGURE 10.30
(a) Image of size 101×101 pixels, containing five white points (four in the corners and one in the center). (b) Corresponding parameter space.



Hough transform – OpenCV implementation

OpenCV implements two kinds of Hough Line Transforms:

- **The Standard Hough Transform** HoughLines()
  - It returns as result a vector of couples (θ, ρ)
- **The Probabilistic Hough Line Transform** HoughLinesP()
  - A more efficient implementation of the Hough Line Transform.
  - It returns as output the extremes of the detected lines $(x0, y0, x1, y1)$

Hough transform – OpenCV implementation

- lines=**cv2.HoughLines**(image, rho, theta, threshold[, lines[, srn[, stn[, min_theta[, max_theta]]]]])

image	8-bit, single-channel binary source image. The image may be modified by the function.
lines	Output vector of lines. Each line is represented by a 2 or 3 element vector (ρ,θ) or (ρ,θ,votes) . ρ is the distance from the coordinate origin (0,0) (top-left corner of the image). θ is the line rotation angle in radians (0~vertical line,π/2~horizontal line). votes is the value of accumulator.
rho	Distance resolution of the accumulator in pixels.
theta	Angle resolution of the accumulator in radians.
threshold	Accumulator threshold parameter. Only those lines are returned that get enough votes (>threshold).
srn	For the multi-scale Hough transform, it is a divisor for the distance resolution rho . The coarse accumulator distance resolution is rho and the accurate accumulator resolution is rho/srn . If both srn=0 and stn=0 , the classical Hough transform is used. Otherwise, both these parameters should be positive.
stn	For the multi-scale Hough transform, it is a divisor for the distance resolution theta.
min_theta	For standard and multi-scale Hough transform, minimum angle to check for lines. Must fall between 0 and max_theta.
max_theta	For standard and multi-scale Hough transform, maximum angle to check for lines. Must fall between min_theta and CV_PI.

Hough transform – OpenCV implementation

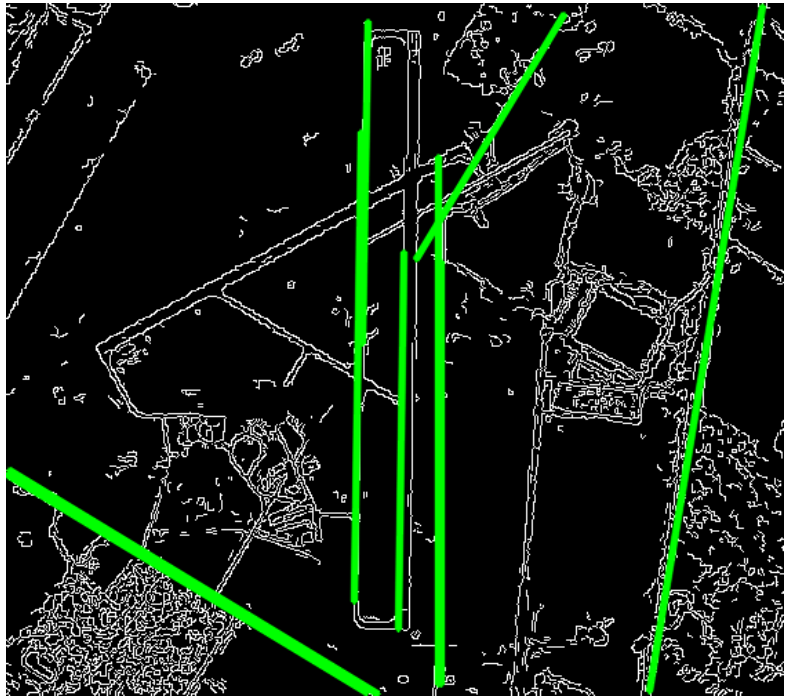
- lines=**cv2.HoughLinesP**(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])

image	**8-bit, single-channel binary source image. The image may be modified by the function.**
lines	Output vector of lines. Each line is represented by a 4-element vector (x1,y1,x2,y2) , where (x1,y1) and (x2,y2) are the ending points of each detected line segment.
rho	Distance resolution of the accumulator in pixels.
theta	Angle resolution of the accumulator in radians.
threshold	Accumulator threshold parameter. Only those lines are returned that get enough votes (>threshold).
minLineLength	Minimum line length. Line segments shorter than that are rejected.
maxLineGap	Maximum allowed gap between points on the same line to link them.

Hough transform – OpenCV implementation

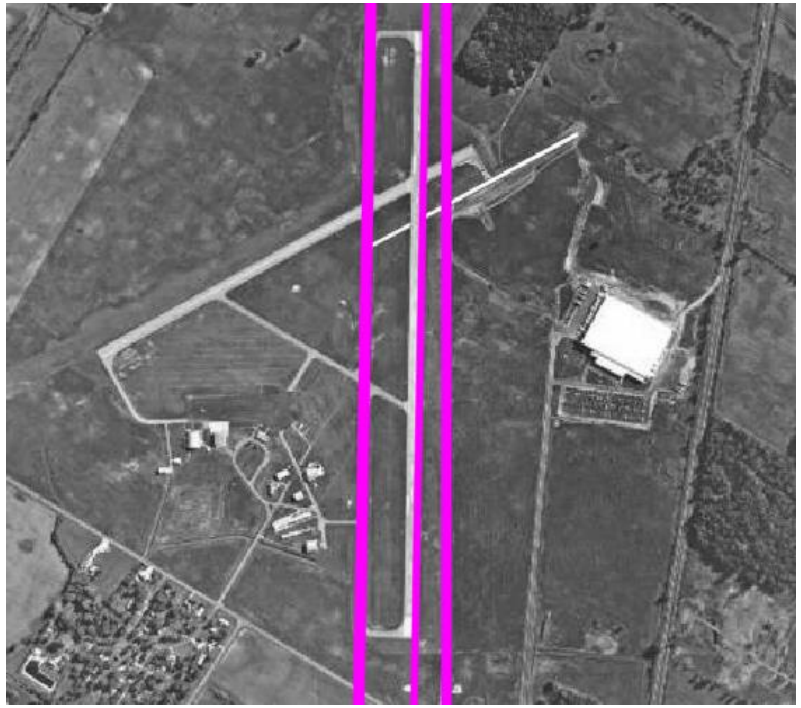
```
src = cv2.imread(path, cv2.IMREAD_GRAYSCALE)

rows, cols = src.shape
# finding edges
edges = cv2.Canny(src,120, 200, None, 3)

minLineLength = np.min([rows, cols]) * 0.35
maxLineGap = 0.2 * minLineLength
threshold = int(np.min([rows, cols]) * 0.35)
# apply Hough transform to find lines
linesP = cv2.HoughLinesP(edges, 1, np.pi / 180, threshold, None,  minLineLength, maxLineGap)

dst = np.copy(src)
dst = cv2.cvtColor(dst, cv2.COLOR_GRAY2BGR)
# overlay lines on image in green color
if linesP is not None:
    for i in range(0, len(linesP)):
        l = linesP[i][0]
        cv2.line(dst, (l[0], l[1]), (l[2], l[3]), (0, 255, 0), 3)
```

Hough Line - Example



Hough Line Example

- Show only principal runway (horizontal lines)



Hough Circle

- To detect circles, we require three parameters
  - Coordinates of the center of the circle (x, y)
  - Radius r
  - The equation of the circle $(x - x_0)^2 + (y - y_0)^2 = r^2$
- We need a 3D accumulator matrix – one for each parameter

Hough Circle

1. Initialization: $A(i, j, k) = 0$ for all i, j, k
2. **for** every non-background point (x_k, y_k)

    for every value of x_0

    for every value of y_0

    calculate $r = \sqrt{(x_k - x_0)^2 + (y_k - y_0)^2}$

    $A(x_0, y_0, r) = A(x_0, y_0, r) + 1$

3. **if** the value of $A(x_0, y_0, r)$ > Threshold

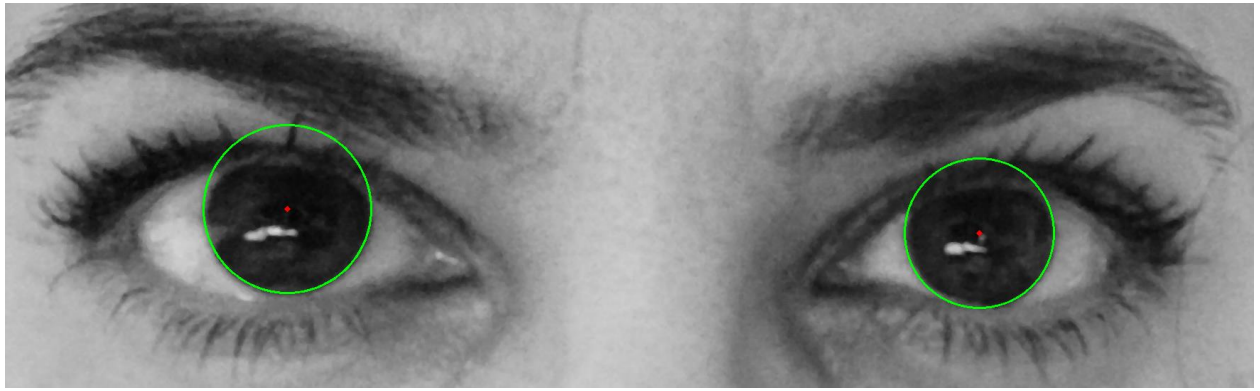
    $A(x_0, y_0, r)$represents a circle in xy-plane

Hough Circle – OpenCV implementation

circles=cv2.HoughCircles(image, method, dp, minDist[, circles[, param1[, param2[, minRadius[, maxRadius]]]]])

image	**8-bit, single-channel, grayscale input image.**
circles	Output vector of found circles. Each vector is encoded as 3 or 4 element floating-point vector (x,y,radius) or (x,y,radius,votes) .
method	Detection method, see HoughModes. The available methods are HOUGH_GRADIENT and HOUGH_GRADIENT_ALT.
dp	Inverse ratio of the accumulator resolution to the image resolution. For example, if dp=1 , the accumulator has the same resolution as the input image. If dp=2 , the accumulator has half as big width and height. For HOUGH_GRADIENT_ALT the recommended value is dp=1.5, unless some small very circles need to be detected.
minDist	Minimum distance between the centers of the detected circles. If the parameter is too small, multiple neighbor circles may be falsely detected in addition to a true one. If it is too large, some circles may be missed.
param1	First method-specific parameter. In case of HOUGH_GRADIENT and HOUGH_GRADIENT_ALT, it is the higher threshold of the two passed to the Canny edge detector (the lower one is twice smaller). Note that HOUGH_GRADIENT_ALT uses Scharr algorithm to compute image derivatives, so the threshold value shough normally be higher, such as 300 or normally exposed and contrasty images.
param2	Second method-specific parameter. In case of HOUGH_GRADIENT, it is the accumulator threshold for the circle centers at the detection stage. The smaller it is, the more false circles may be detected. Circles, corresponding to the larger accumulator values, will be returned first. In the case of HOUGH_GRADIENT_ALT algorithm, this is the circle "perfectness" measure. The closer it to 1, the better shaped circles algorithm selects. In most cases 0.9 should be fine. If you want get better detection of small circles, you may decrease it to 0.85, 0.8 or even less. But then also try to limit the search range [minRadius, maxRadius] to avoid many false circles.
minRadius	Minimum circle radius.
maxRadius	Maximum circle radius. If <= 0, uses the maximum image dimension. If < 0, HOUGH_GRADIENT returns centers without finding the radius. HOUGH_GRADIENT_ALT always computes circle radiuses.

```
src = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
src = cv2.medianBlur(src, 5)
csrc = cv2.cvtColor(src, cv2.COLOR_GRAY2BGR)


# Detect circles using HoughCircles transform
circles = cv2.HoughCircles(src, cv2.HOUGH_GRADIENT, 1, csrc.shape[1] / 3,
param1=50, param2=20, minRadius=80,
                        maxRadius=120)

# If at least 1 circle is detected
if circles is not None:
    circles = np.uint16(np.around(circles))
    for i in circles[0, :]:
        # Draw the circle
        cv2.circle(csrc, (i[0], i[1]), i[2], (0, 255, 0), 2)
        # Draw the center of the circle
        cv2.circle(csrc, (i[0], i[1]), 2, (0, 0, 255), 3)
```



Practice

1. Find the main lanes in the image

2. Apply Hough transform to find the coins in the image