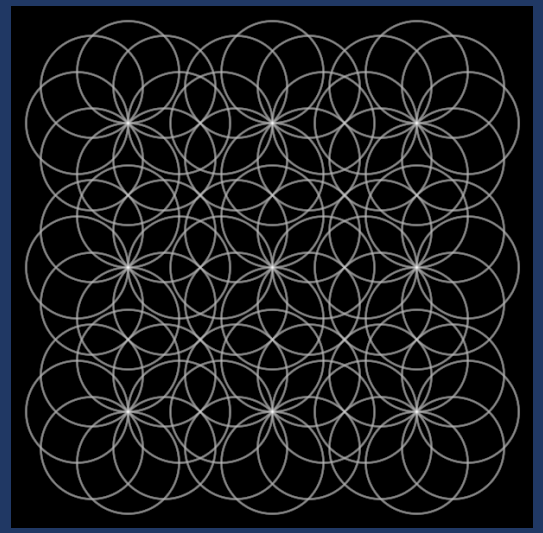
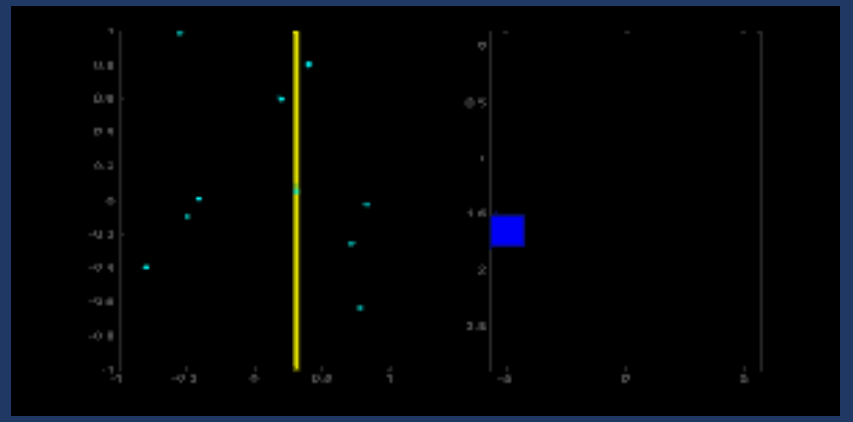
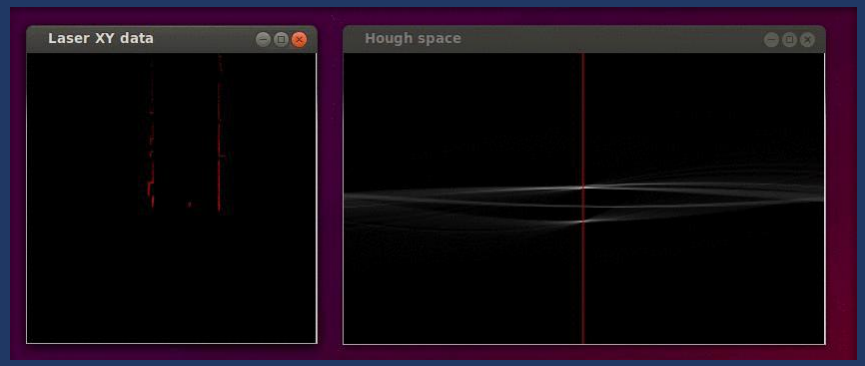
Hough Transform

February 25th, 2020

Presented by Dr. Sander Ali Khowaja



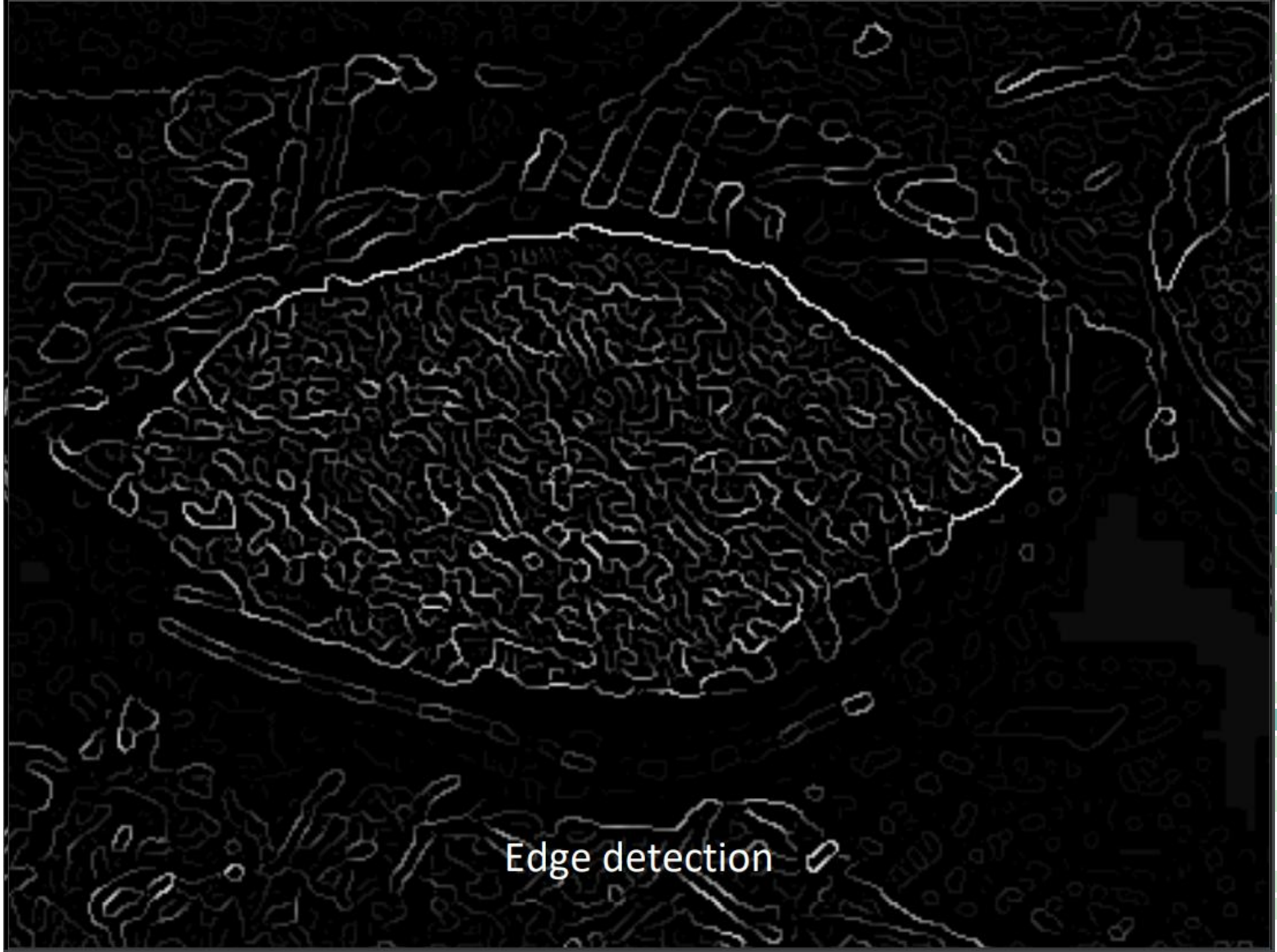


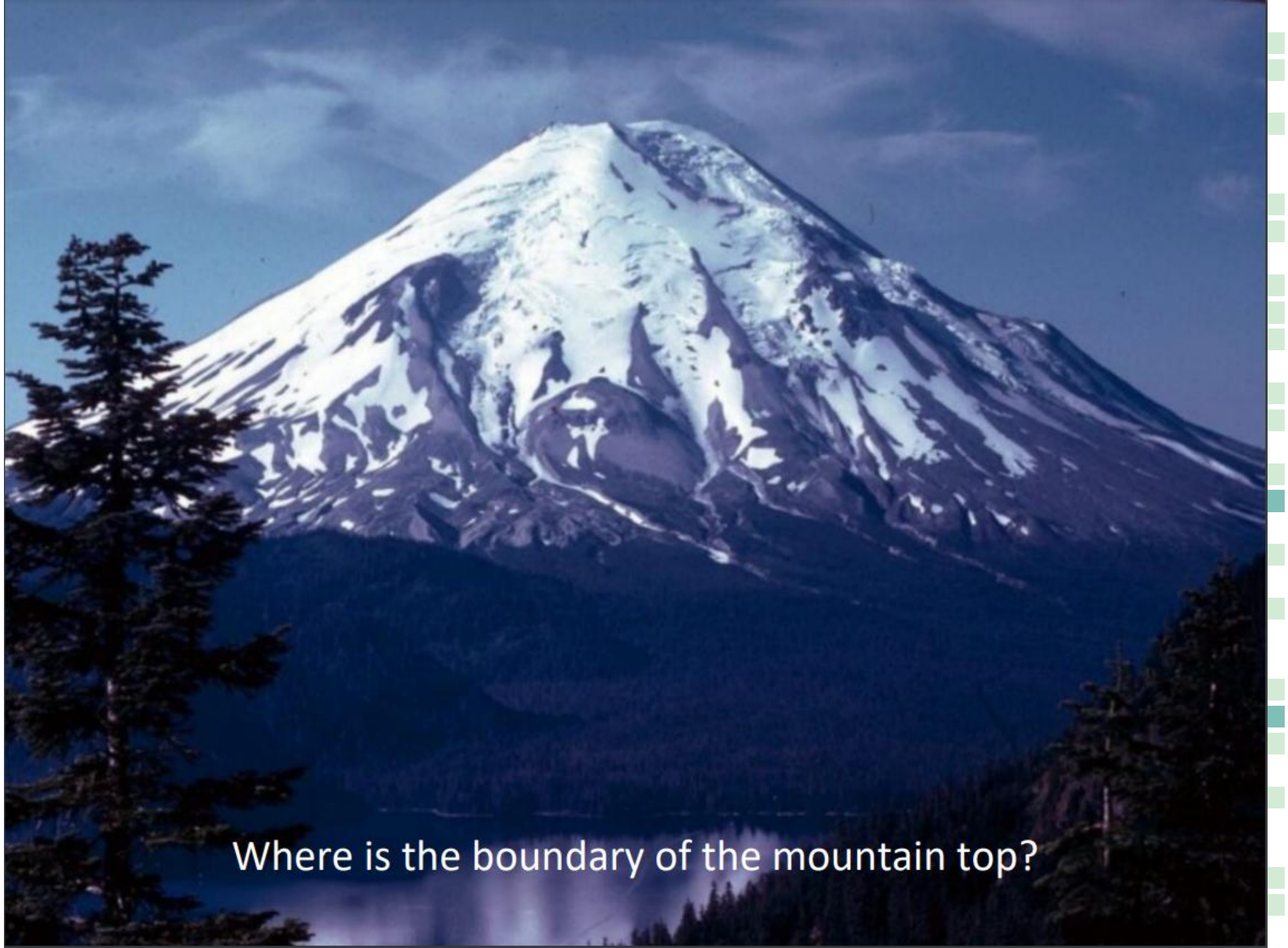


Introduction to Hough Transform

- The Hough transform (HT) can be used to detect lines.
- It was introduced in 1962 (Hough 1962) and first used to find lines in images a decade later (Duda 1972).
- The goal is to find the location of lines in images.
- **Caveat**: Hough transform can detect lines, circles and other structures ONLY if their parametric equation is known.
- It can give robust detection under noise and partial occlusion

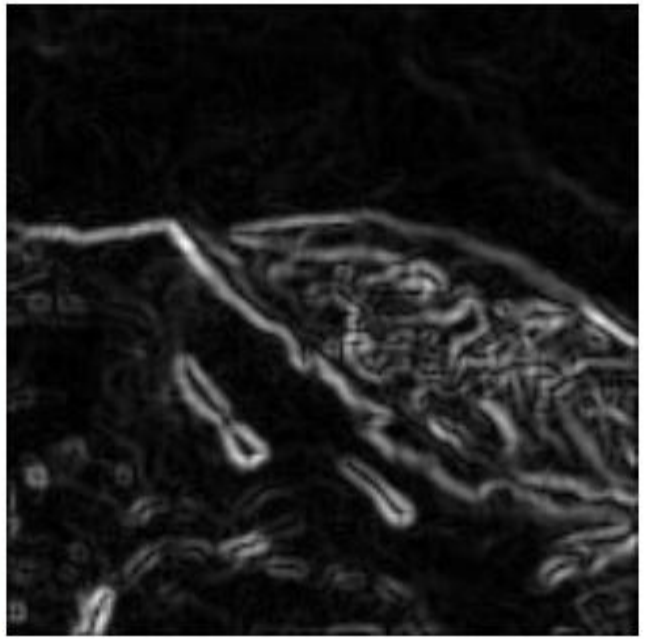

Edge strength does not necessarily
correspond to our perception of boundaries


Edge detection

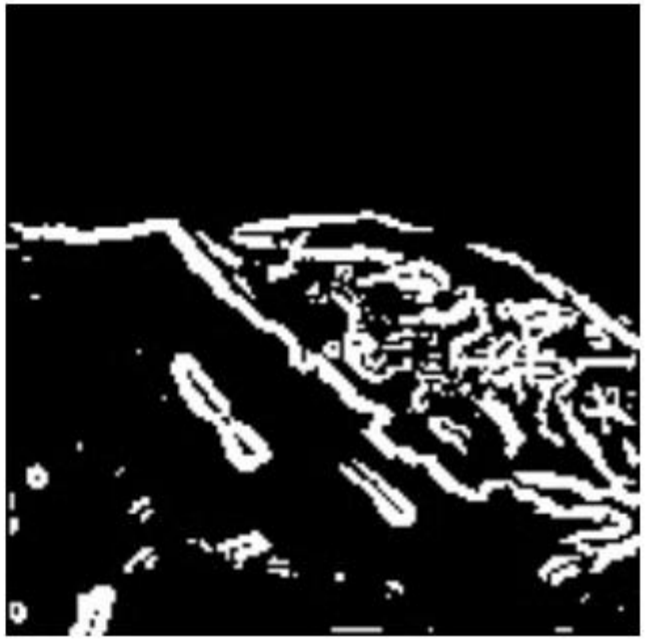

Where is the boundary of the mountain top?

Lines are Hard to Find

Original image



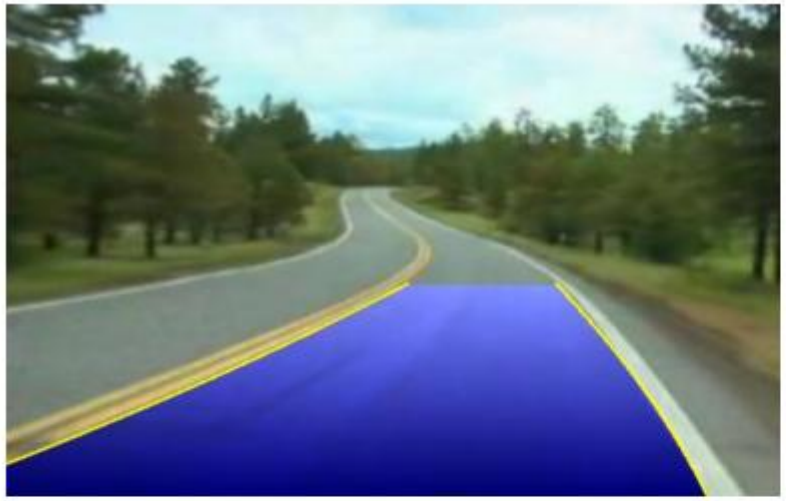
Edge detection



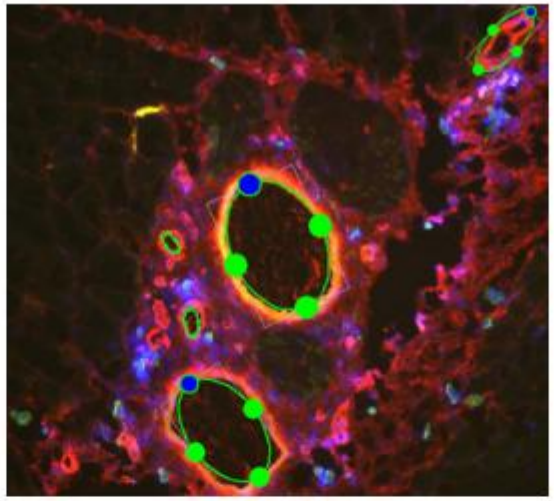
Thresholding

Noisy edge image
Incomplete boundaries

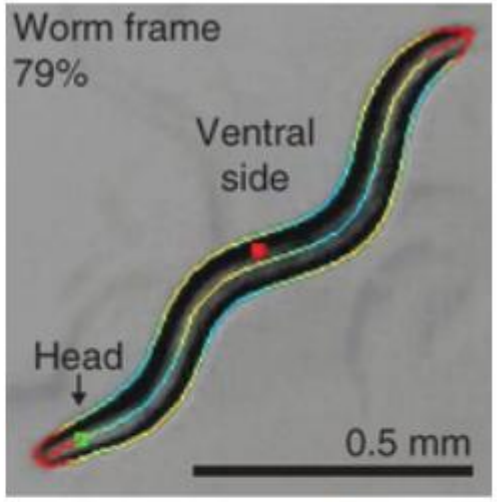
Applications



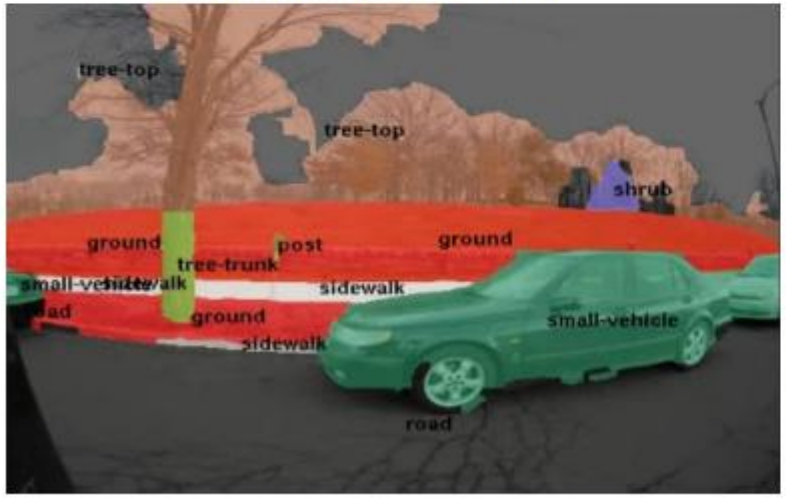
Autonomous Vehicles
(lane line detection)



tissue engineering
(blood vessel counting)



behavioral genetics
(earthworm contours)



Autonomous Vehicles
(semantic scene segmentation)

Computational Photography
(image inpainting)

Prior to Hough Transform

- Assume that we have performed some edge detection, and a thresholding of the edge magnitude image.
- Thus, we have some pixels that may partially describe the boundary of some objects.



Edges vs Boundaries

- Edges
  - Local Intensities Discontinuities
  - Points
  - Not dependent on Models
- Boundaries
  - Extensive
  - Composed of many points
  - May be dependent on models
- Typically our goal is to reconstruct the boundary from local edge elements

Local edge point or edge element

Object boundary

To group edge points, we may need a domain or object model

Detecting the lines using Hough Transform

- We wish to find sets of pixels that make up straight lines.
- Consider a point of known coordinates $(x_i;y_i)$
  - There are many lines passing through the point (x_i, y_i).
- Straight lines that pass that point have the form $y_i = m*x_i + c$
  - Common to them is that they satisfy the equation for some set of parameters (m, c)

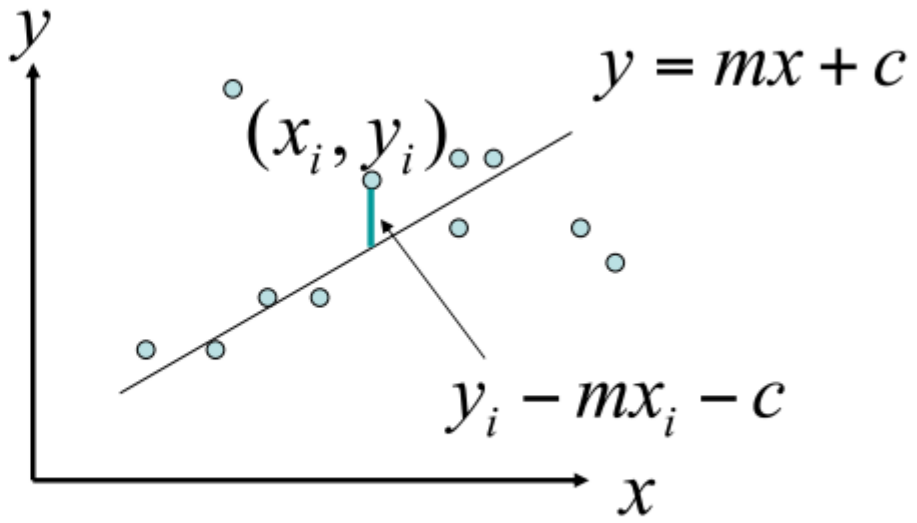
Line Fitting

Given: Many (x_i, y_i) pairs

Find: Parameters (m, c)

Minimize: Average square distance:

$$E = \sum_i \frac{(y_i - mx_i - c)^2}{N}$$



How can we solve this minimization?

Line fitting

Given: Many (x_i, y_i) pairs

Find: Parameters (m, c)

Minimize: Average square distance:

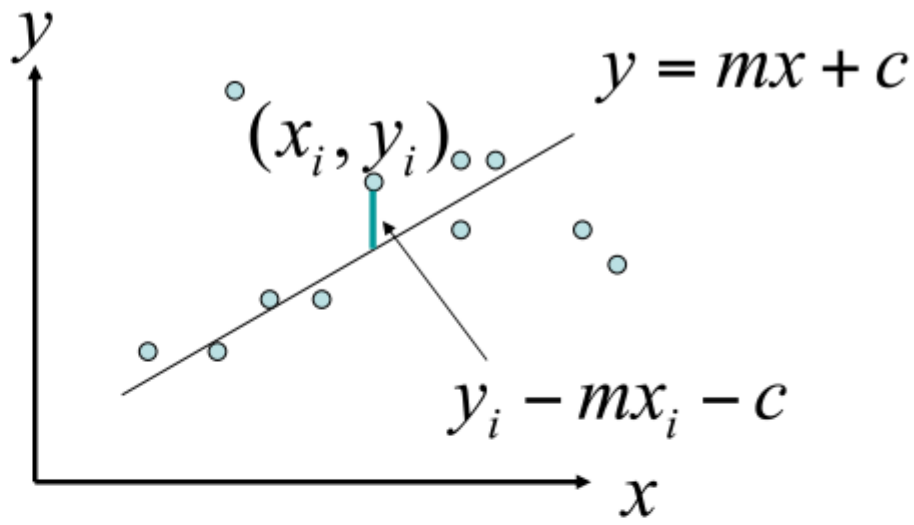
$$E = \sum_i \frac{(y_i - mx_i - c)^2}{N}$$

Using:

$$\frac{\partial E}{\partial m} = 0 \quad \& \quad \frac{\partial E}{\partial c} = 0$$

Note:

$$\bar{y} = \frac{\sum_i y_i}{N} \qquad \bar{x} = \frac{\sum_i x_i}{N}$$



$$c = \bar{y} - m\,\bar{x}$$

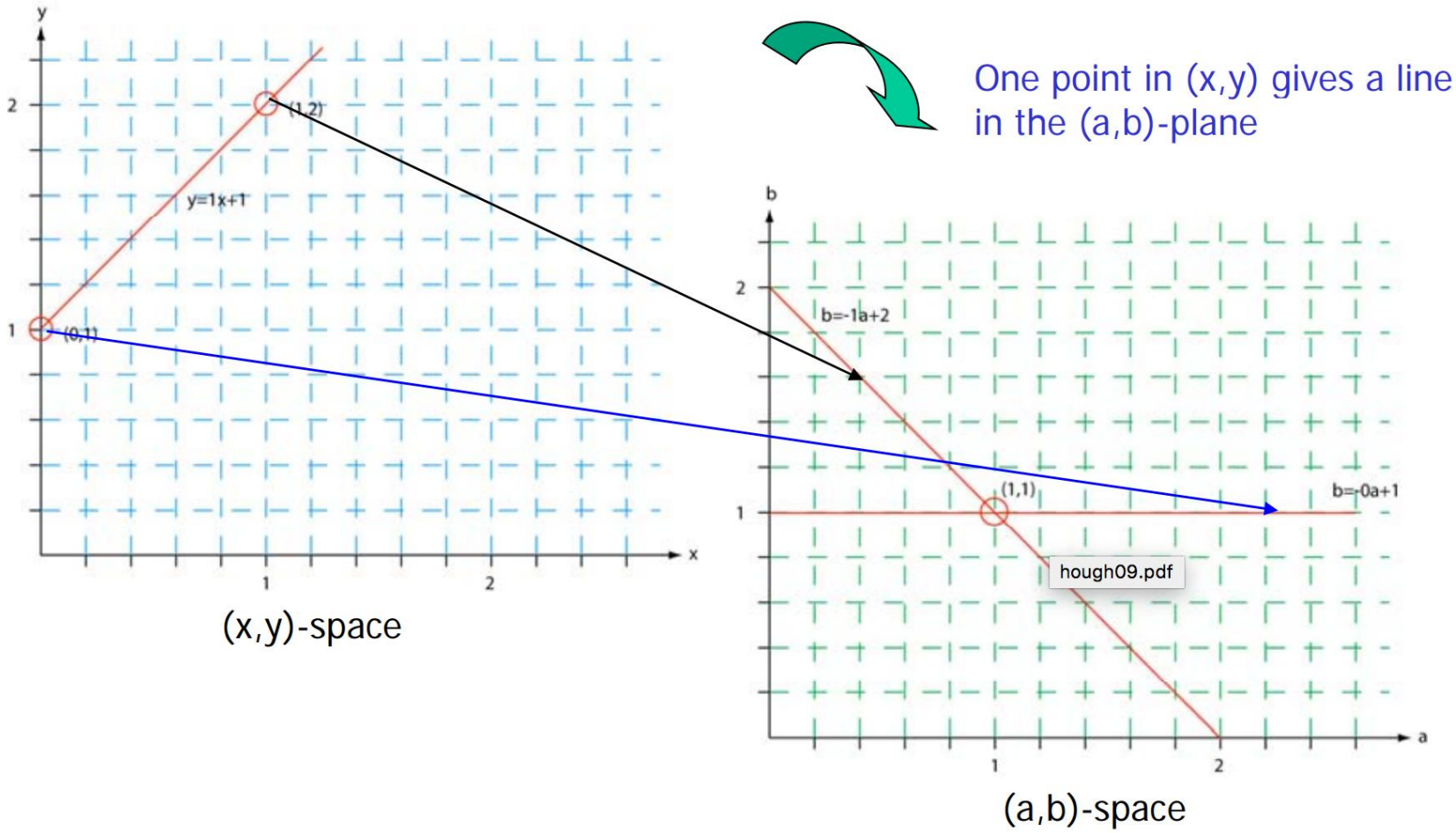
$$m = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sum_i (x_i - \bar{x})^2}$$

What are some problems with the approach?

Detecting lines using Hough Transform

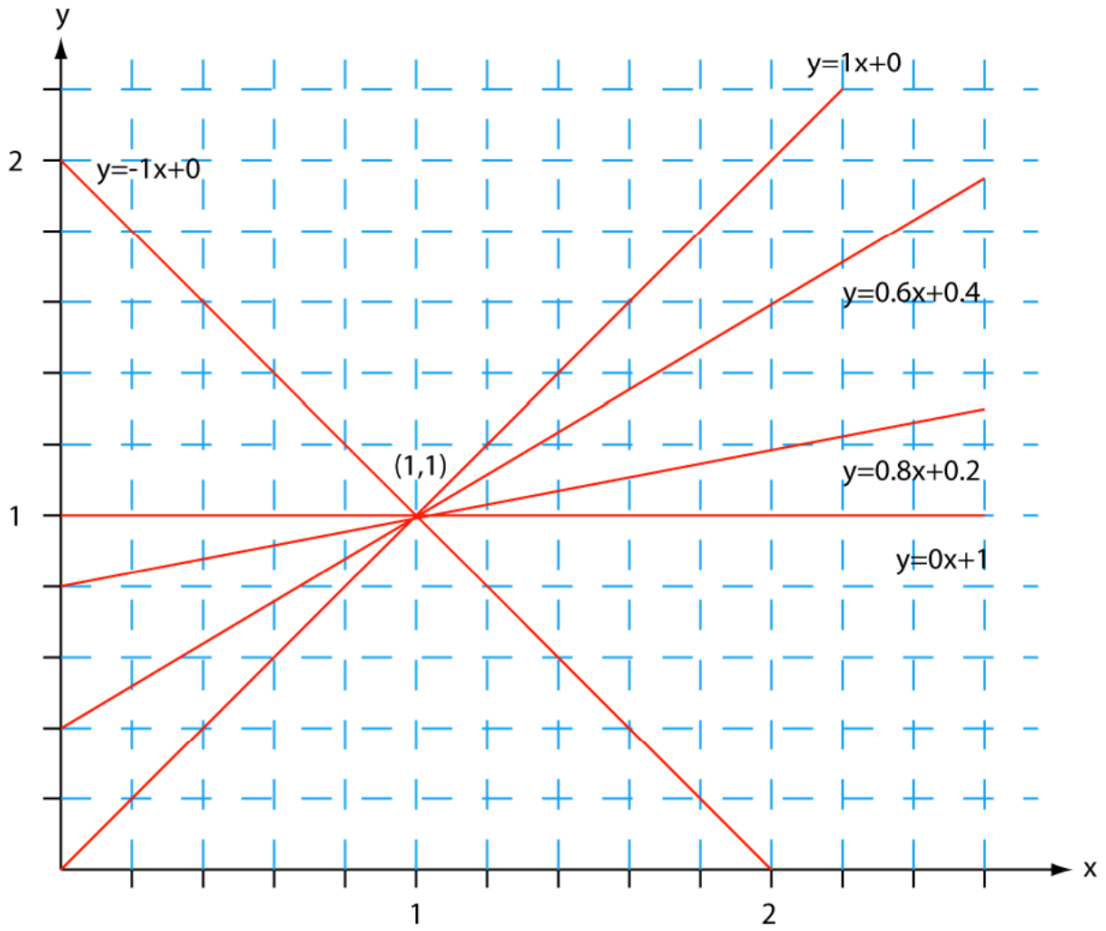
- This equation can obviously be rewritten as follows:
  - $c = -m*x_i + y_i$
  - We can now consider x and y as parameters
  - m and c as variables.
- This is a line in (m, c) space parameterized by x and y.
  - So: a single point in x_1,y_1-space gives a line in (m,c) space.
  - Another point (x_2, y_2) will give rise to another line (m,c) space.

Detecting Lines using Hough Transform



What if we map all points from the red line into ab space?

Detecting lines using Hough Transform

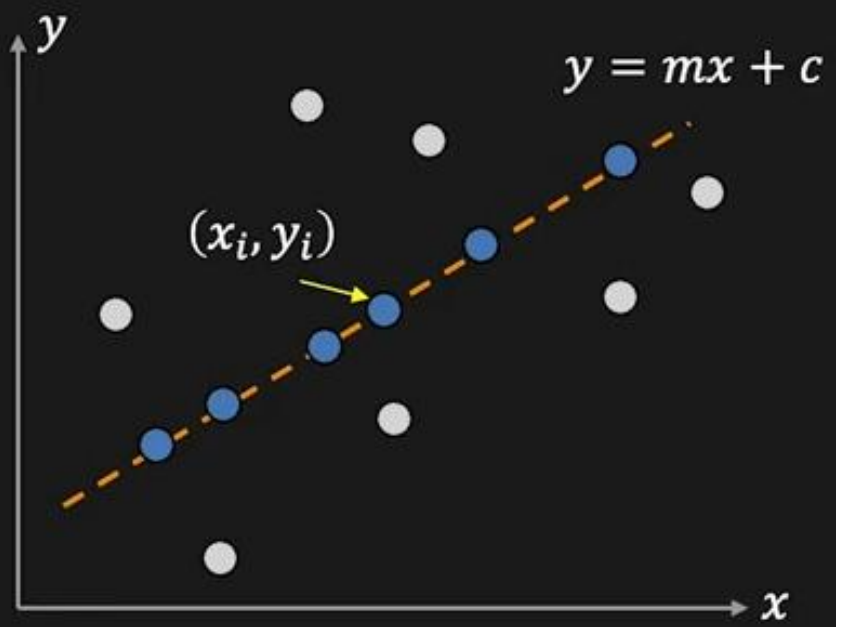


Step by Step Example

Given: Edge Points (x_i, y_i)

Task: Detect line

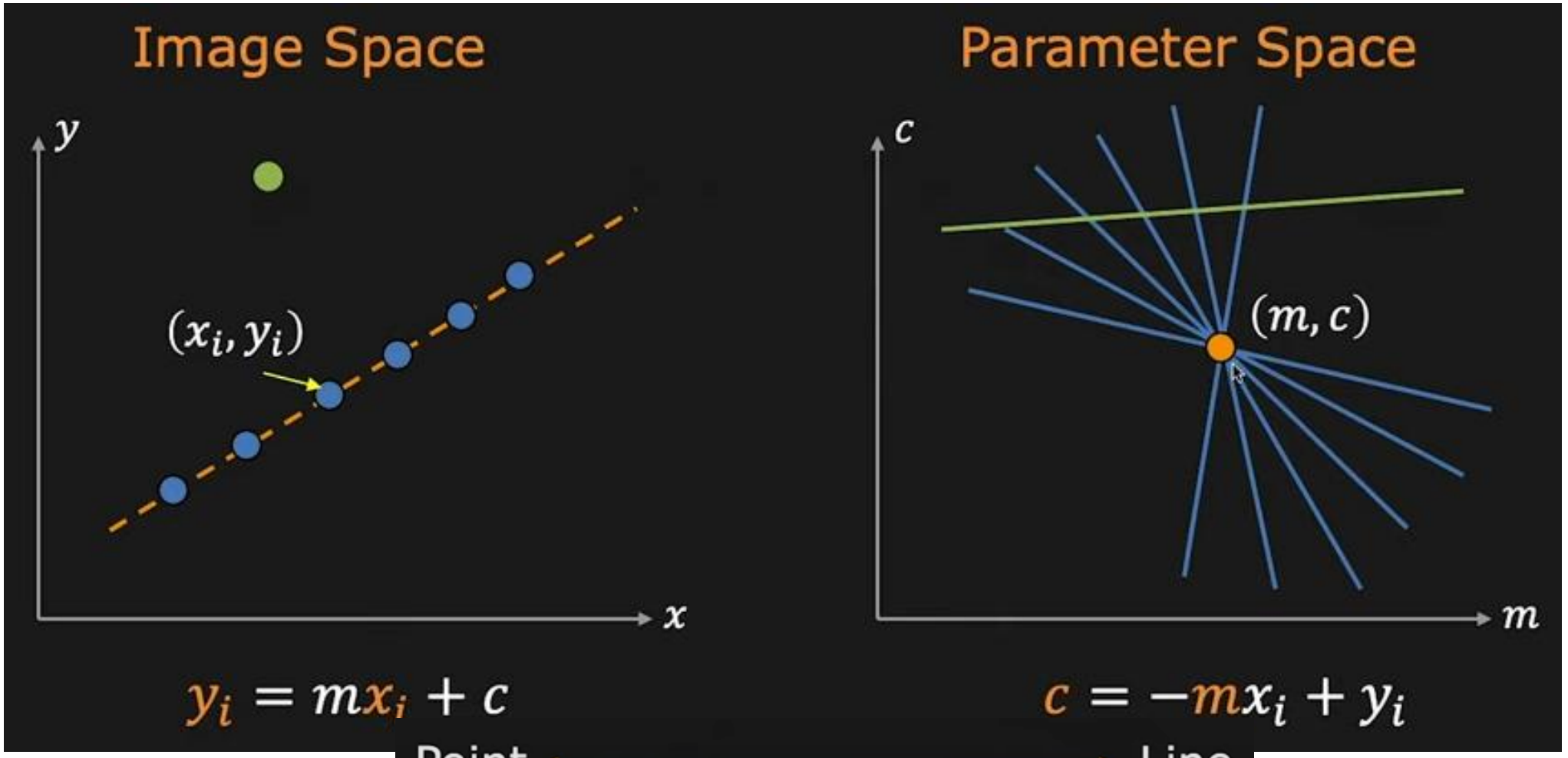
$$y = mx + c$$



Consider point (x_i, y_i)

$$y_i = mx_i + c \iff c = -mx_i + y_i$$

Step by Step Example

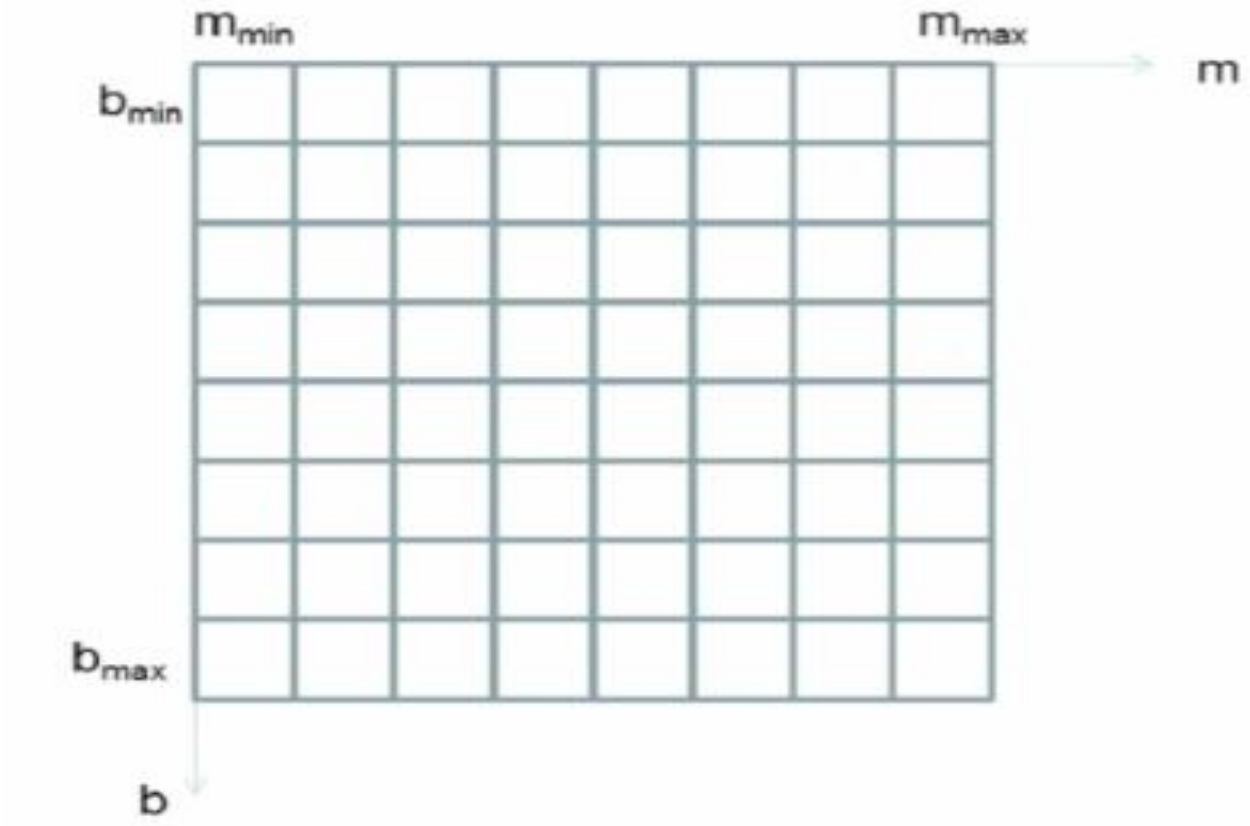


Algorithm for Hough Transform

- Quantize the parameter space (m c) by dividing it into cells
- This quantized space is often referred to as the accumulator cells.
- Count the number of times a line intersects a given cell.
  - For each pair of points (x_1, y_1) and (x_2, y_2) detected as an edge, find the intersection (m',c') in (m, c)space.
  - Increase the value of a cell in the range $[[m_{min}, m_{max}],[c_{min},c_{max}]]$ that (m', c') belongs to.
  - Cells receiving more than a certain number of counts (also called 'votes') are assumed to correspond to lines in (x,y) space.

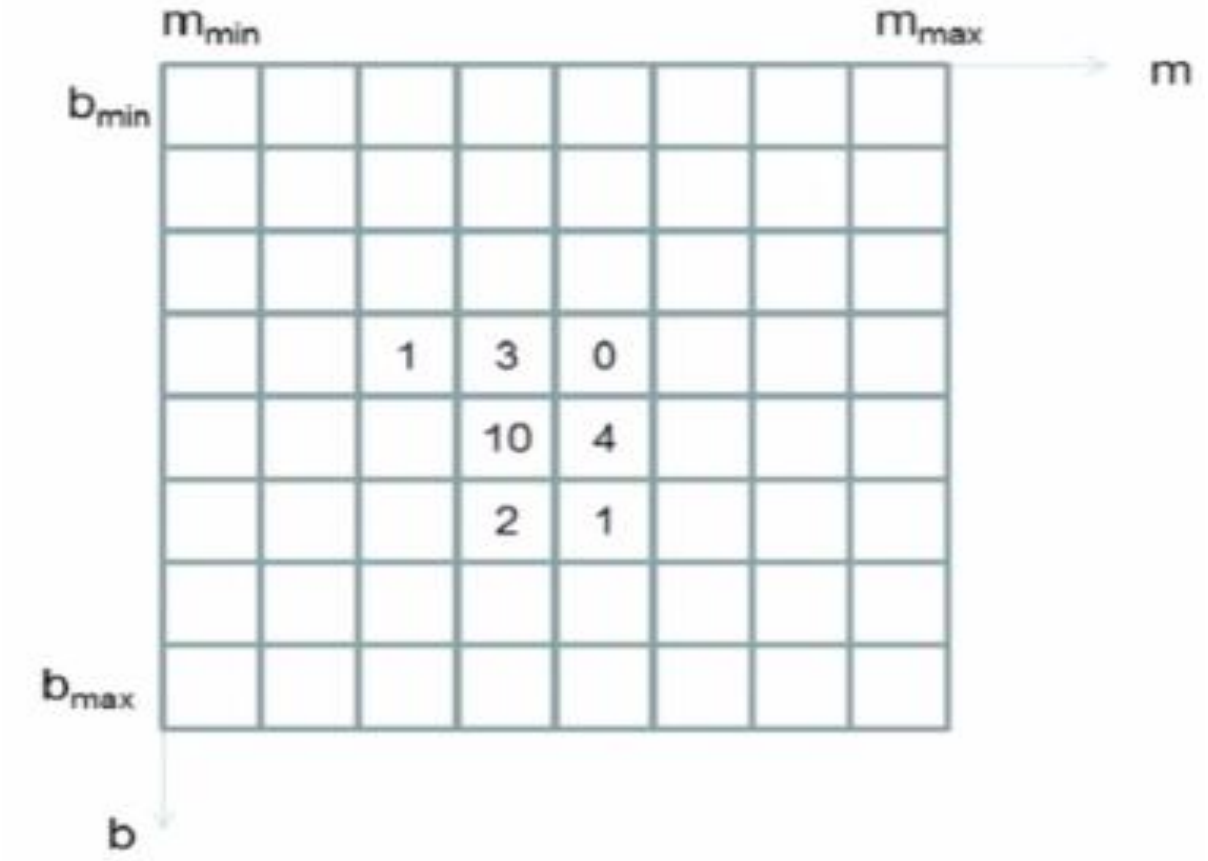
Hough Transform Algorithm

- Initialize an accumulator array A(m,b) to zero



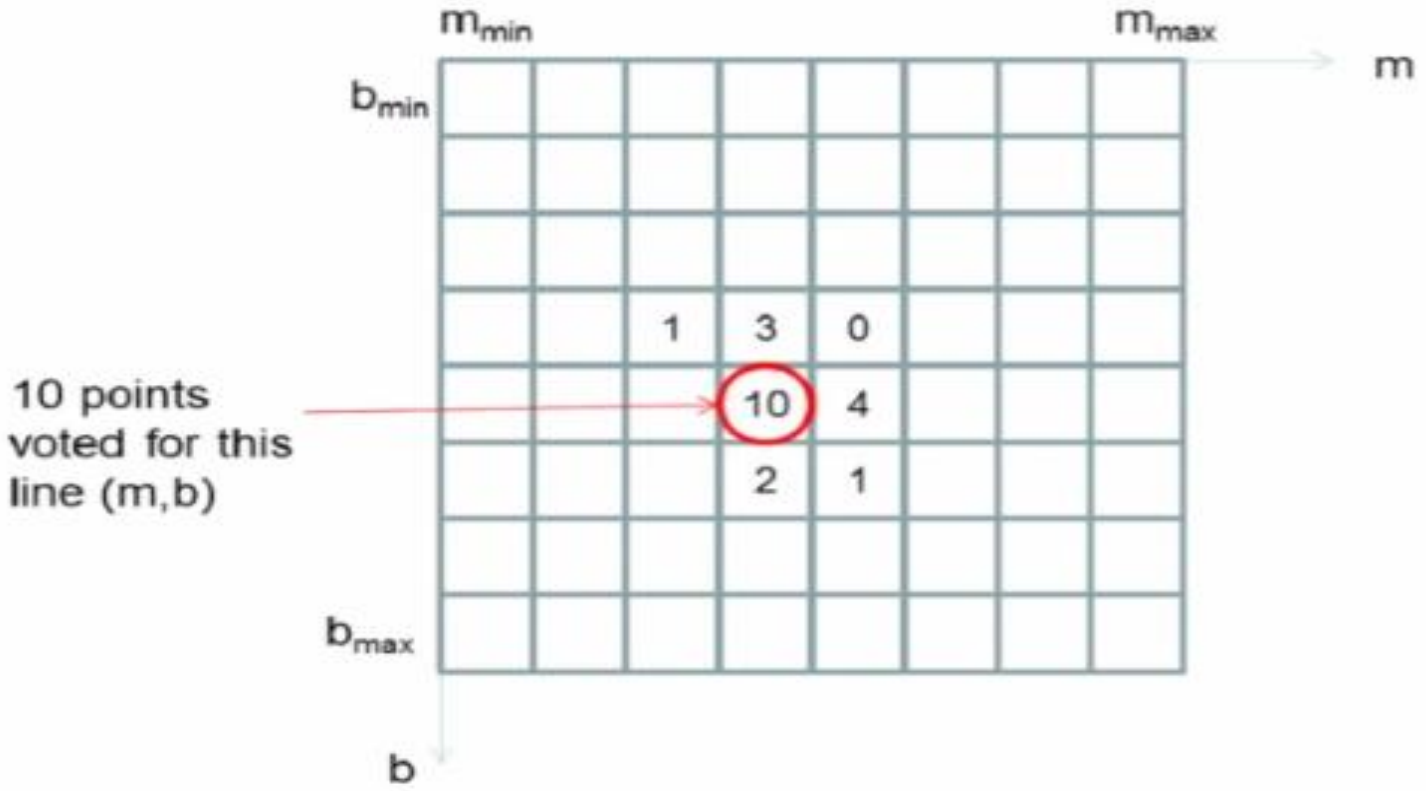
Hough Transform Algorithm

- Initialize an accumulator array A(m,b) to zero
- For each edge element (x,y), increment all cells that satisfy c = -xm + y



Hough Transform Algorithm

- Initialize an accumulator array A(m,b) to zero
- For each edge element (x,y), increment all cells that satisfy b = -xm + y
- Look for local maxima in A(m,b)corresponds to line



Line Detection Algorithm

Step 1. Quantize parameter space (m, c)

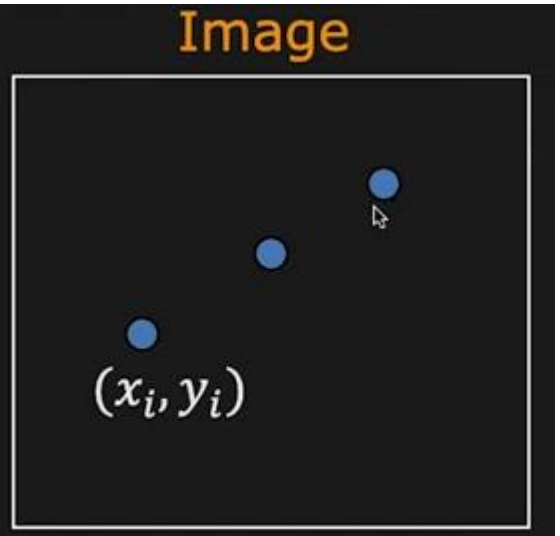
Step 2. Create accumulator array $A(m, c)$

Step 3. Set $A(m, c) = 0$ for all (m, c)

Step 4. For each edge point (x_i, y_i),

$$A(m, c) = A(m, c) + 1$$

if (m, c) lies on the line: $c = -mx_i + y_i$



Step 5. Find local maxima in $A(m, c)$

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

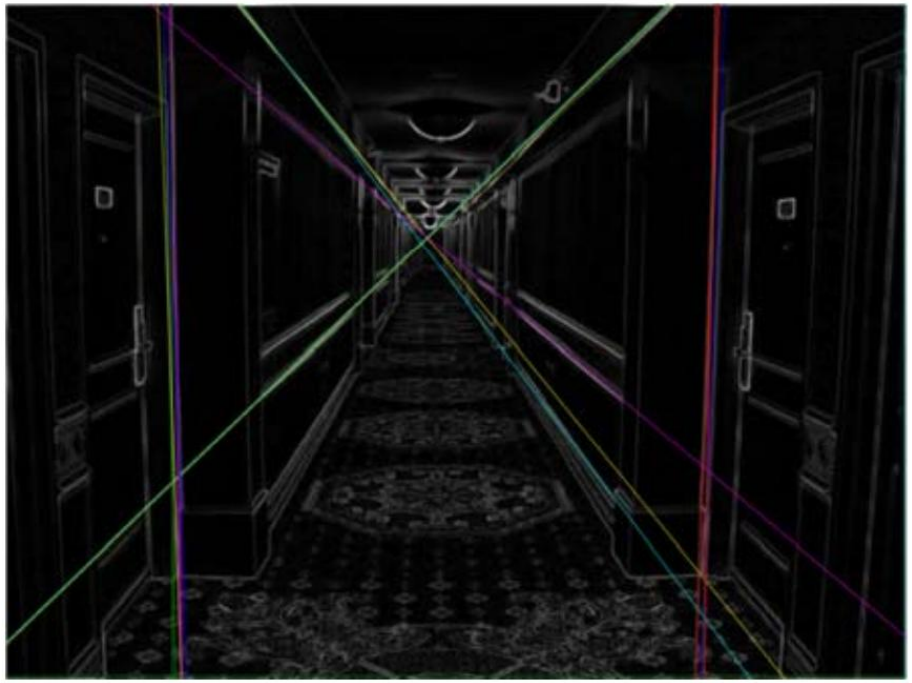
1	0	0	0	0
0	1	0	0	0
0	0	1	0	0
0	0	0	1	0
0	0	0	0	1

1	0	0	0	1
0	1	0	1	0
0	0	2	0	0
0	1	0	1	0
1	0	0	0	1

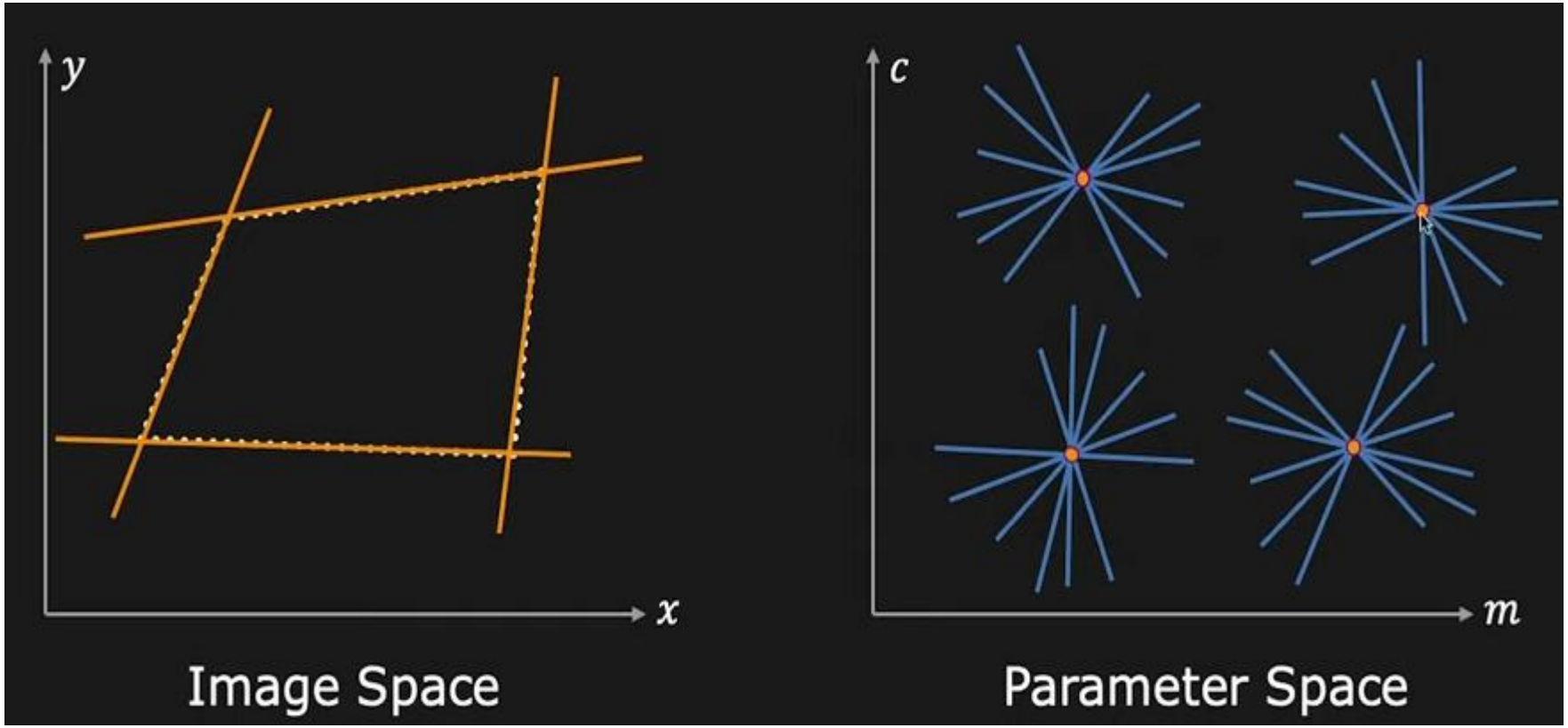
1	0	0	0	1
0	1	0	1	0
1	1	3	1	1
0	1	0	1	0
1	0	0	0	1

Output of Hough Transform

- Here are the top 20 most voted lines in the image:

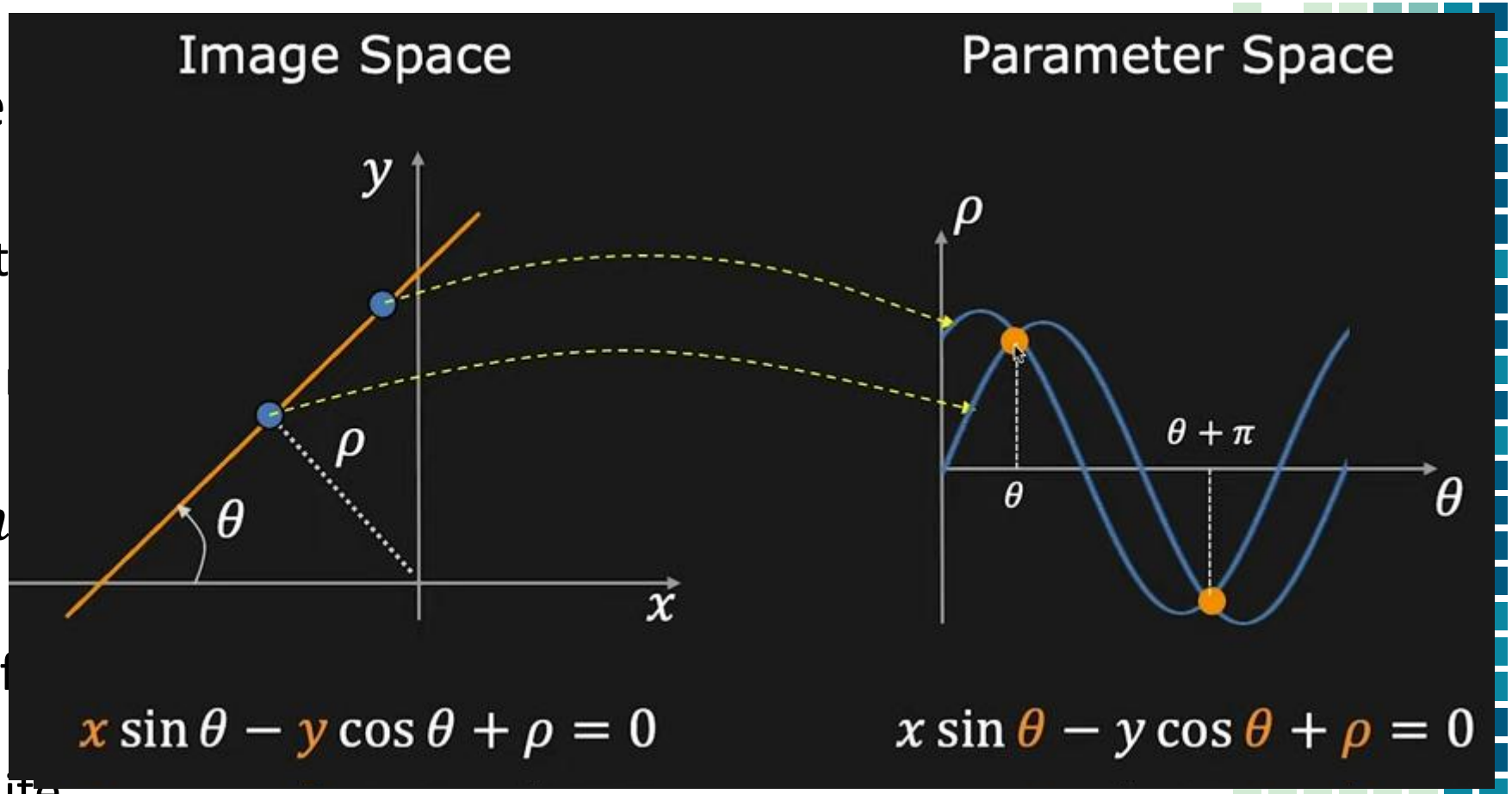


Multiple Line Detection



Better Parameterization

- Issue: Slope of the
  - Large Accumulat
  - More Memory a
- Solution: Use $xsin$
  - Orientation θ is f
  - Distance ρ is finite.



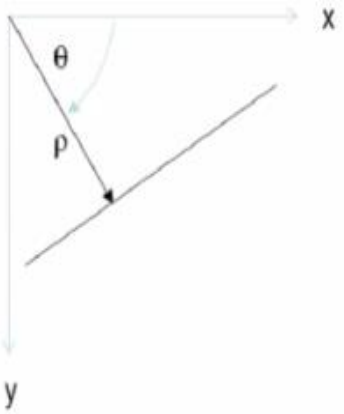
Other Hough Transformations

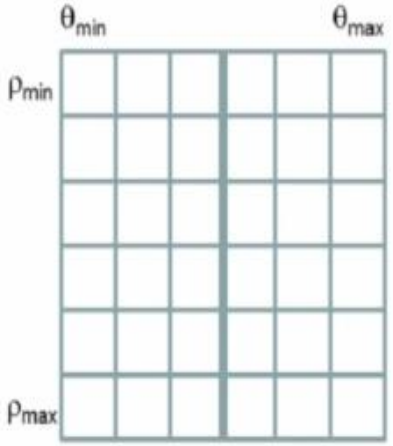
- We can represent lines as polar coordinates instead of y = m*x + c
- Polar coordinate representation:
  - x*cosθ + y*sinθ = ρ
- Can you figure out the relationship between
  - (x y) and (ρ θ)?

Hough Transform Polar Coordinates

- $\boldsymbol{\rho} = \mathrm{x}\cos\boldsymbol{\theta} + \mathrm{y}\sin\boldsymbol{\theta}$
  - Avoids infinite slope
  - Constant resolution

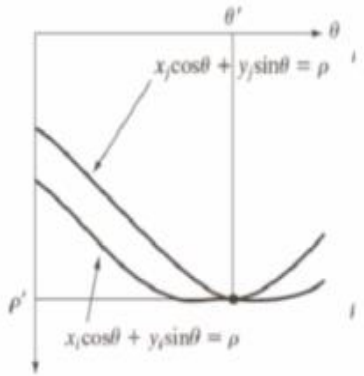
- Slope line have some disadvantages as it can't represent infinite slopes and vertical lines
- Rho is the vector perpendicular to the line from the origin
- Theta is the angle of that vector from x-axis





A(ρ,θ)

The parameter space transform of a point is a sinusoidal curve

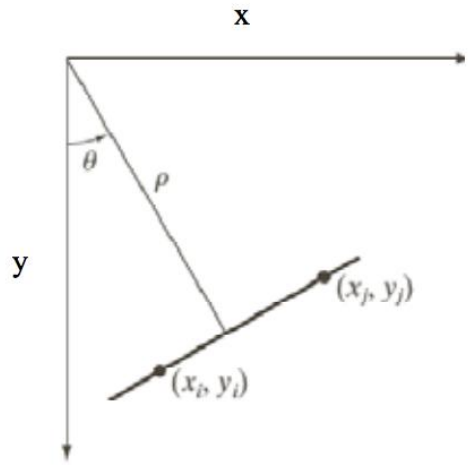


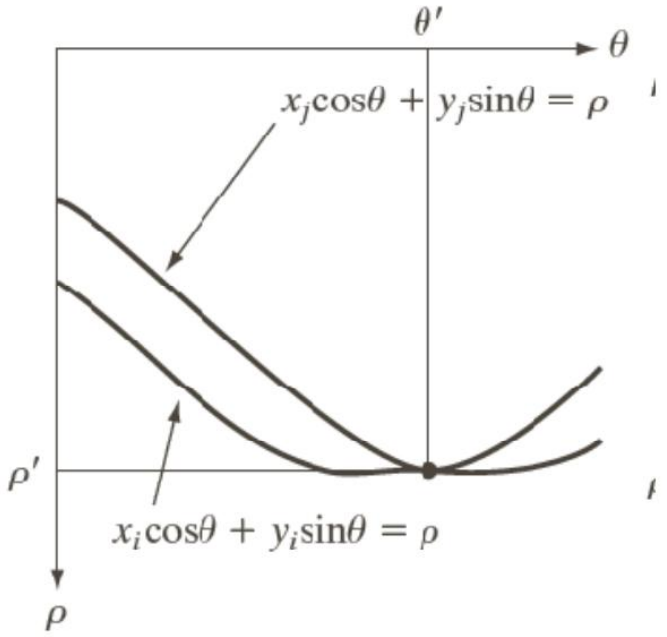
Hough Transform Polar Coordinates

- Note that lines in (x y) space are not lines in
(ρ θ) space, unlike (m c) space.

- A horizontal line will have θ=0 and ρ equal to the intercept with the y-axis.

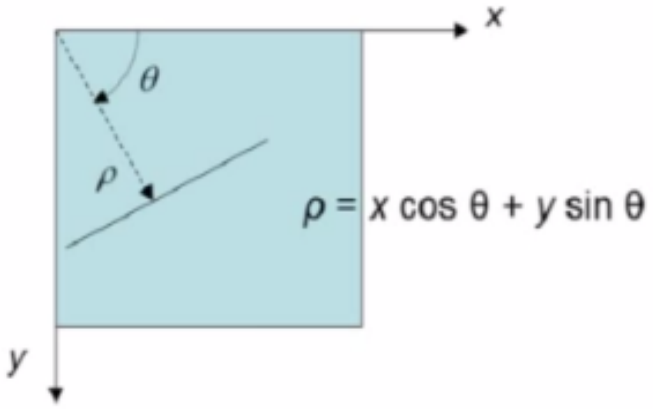
- A vertical line will have θ=90 and ρ equal to the intercept with the x-axis.

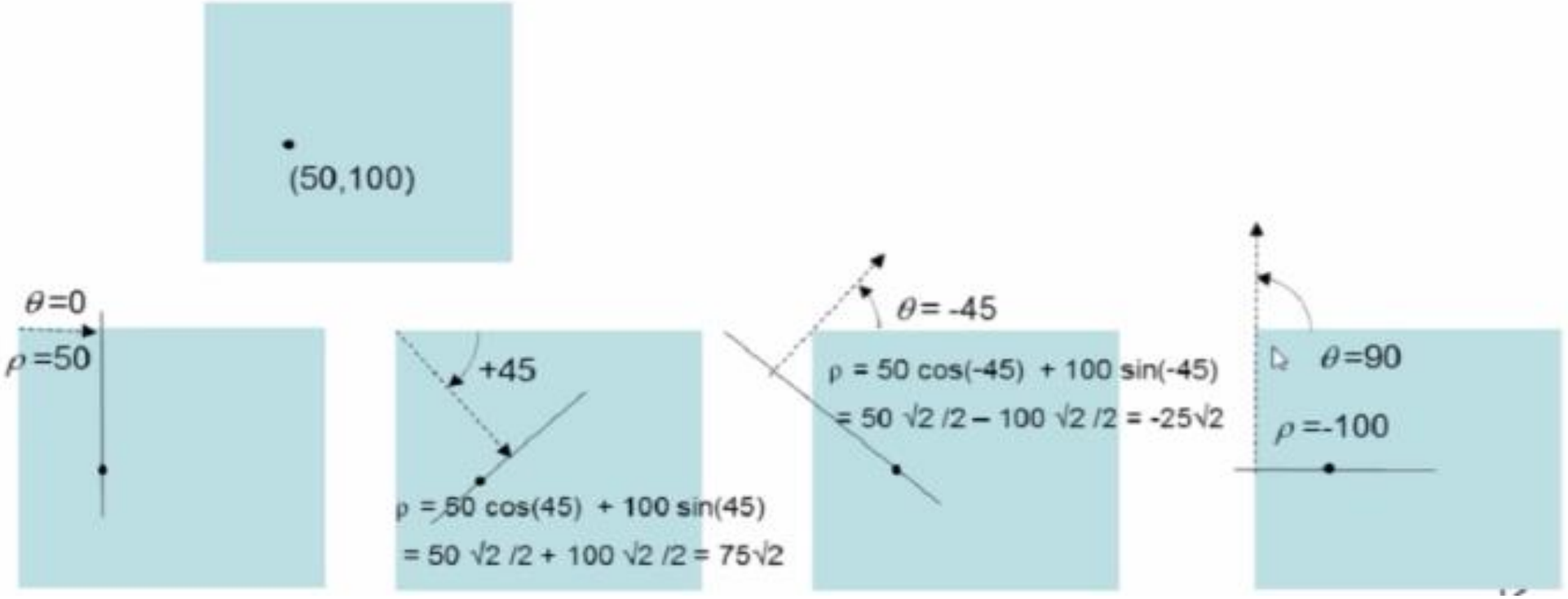




Hough Transform with Polar Coordinates

- Angle, axis conventions
  - Angle range is -90° … +89°
  - Rho range is –dmax … +dmax
    - dmax is the largest possible distance





Hough Transform Example

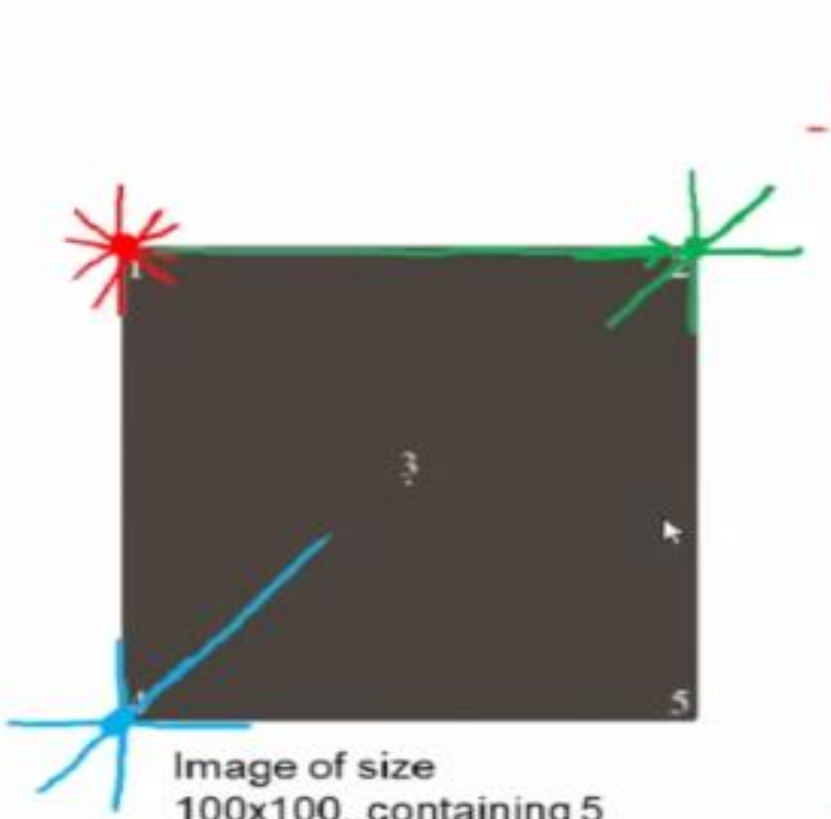
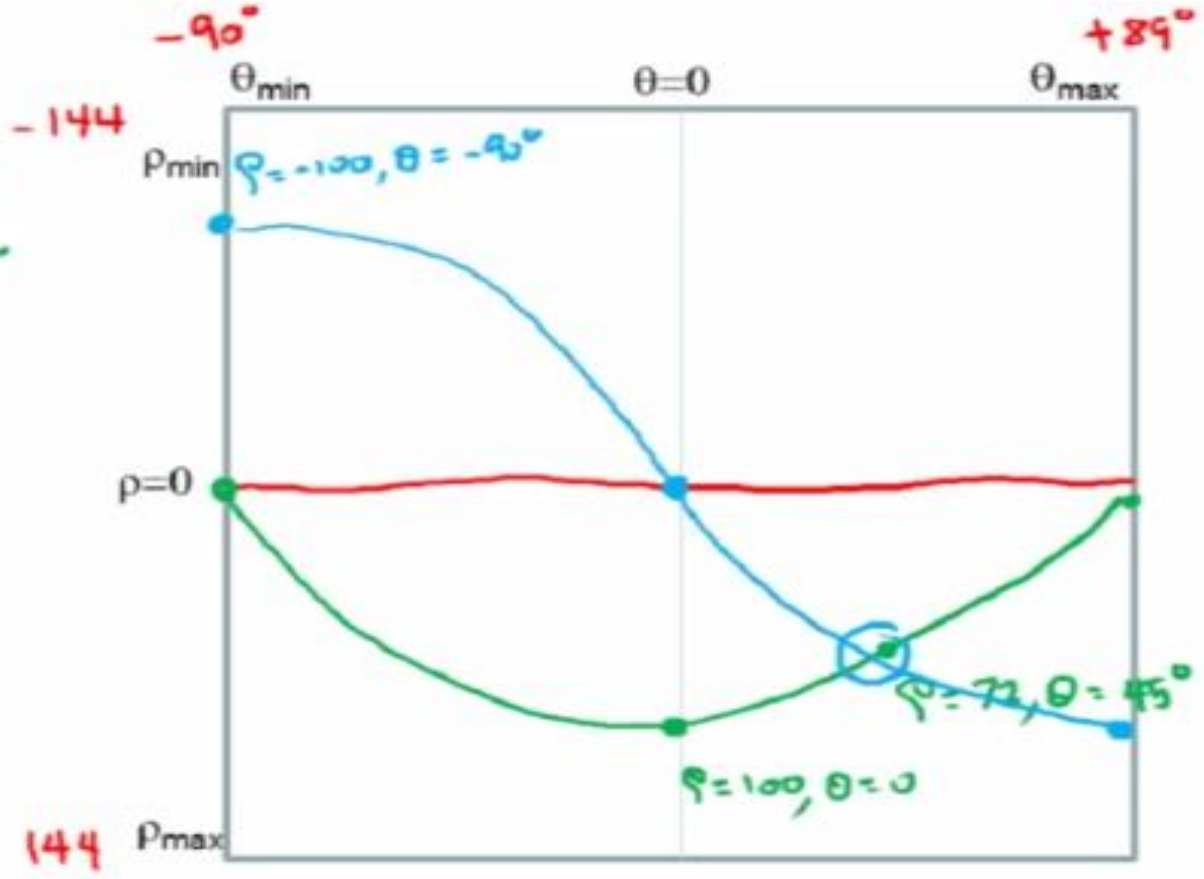
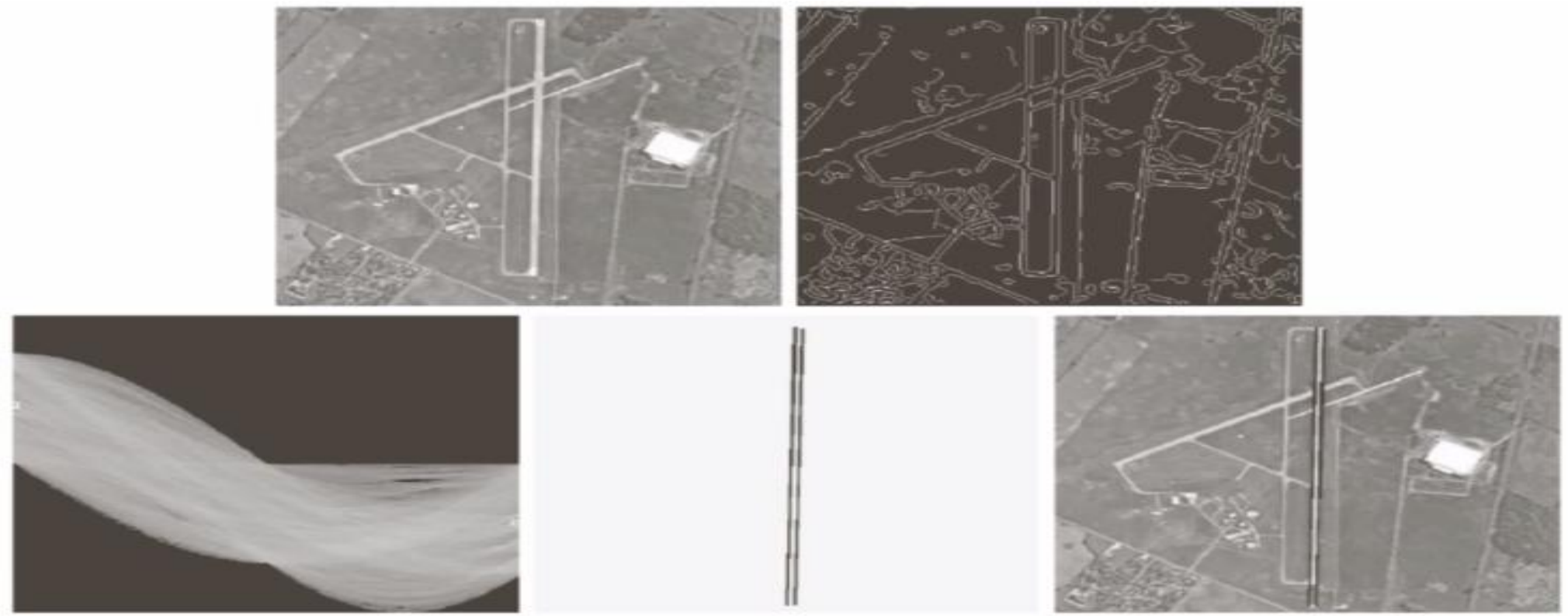


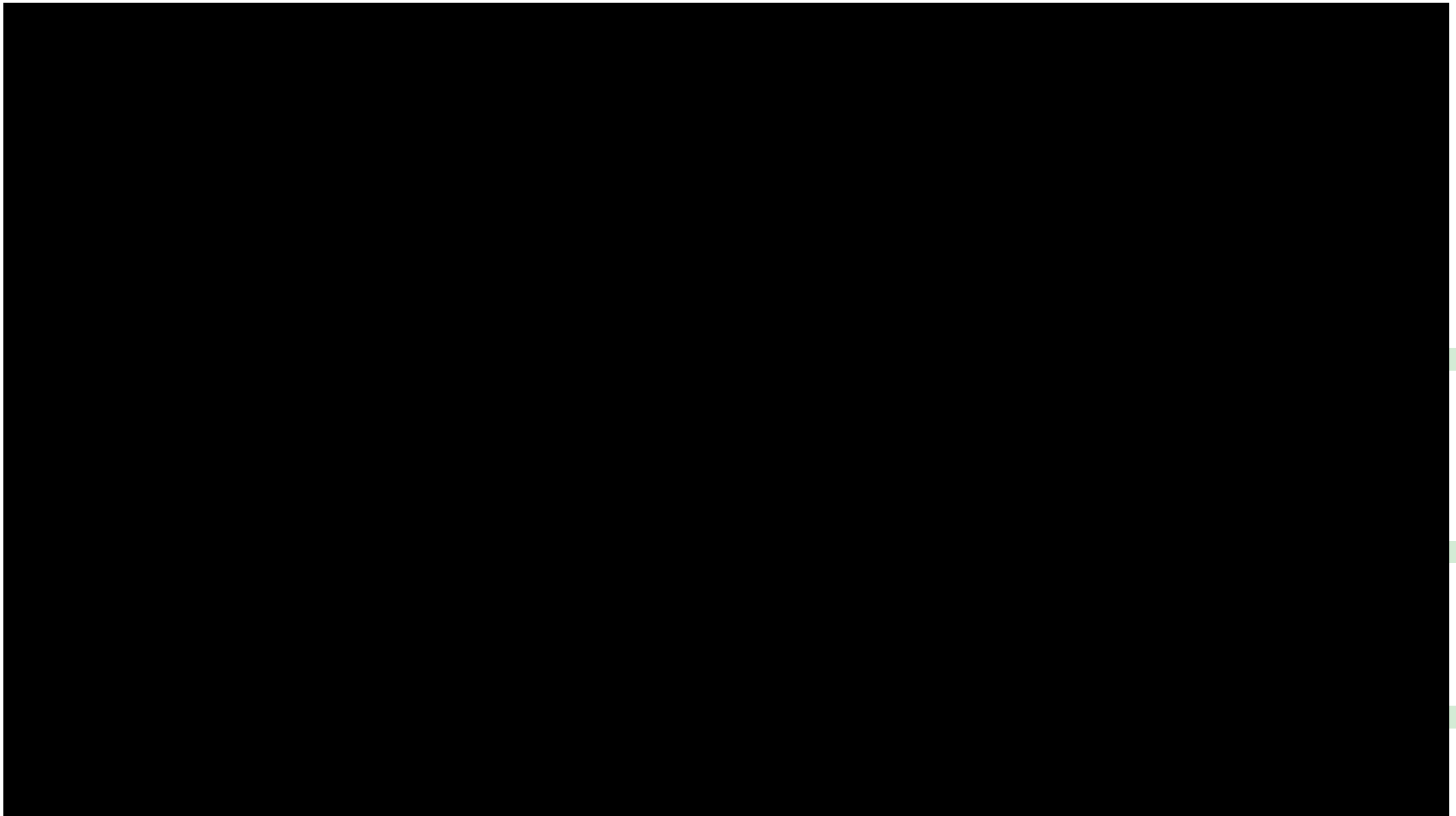
Image of size 100x100, containing 5 points (at the corners)



Hough Transform Example



How Hough Transform Works (Animation)



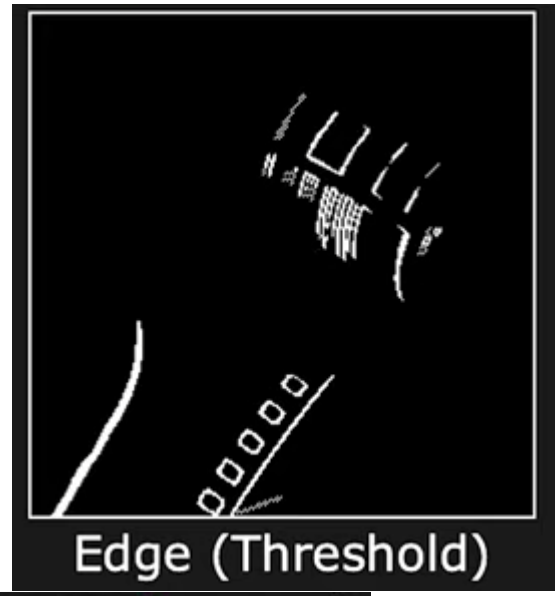
Hough Transform Mechanics

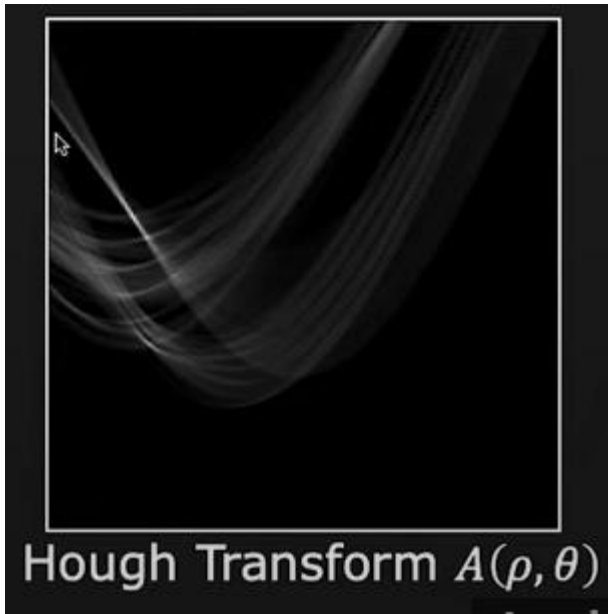
- How big should the accumulator cells be?
  - Too big, and different lines may be merged
  - Too small, and noise causes lines to be missed.
- How many lines?
  - Count the peaks in the accumulator array
- Handling inaccurate edge locations:
  - Increment patch in accumulator rather than single point

Line Detection Results


Original Image

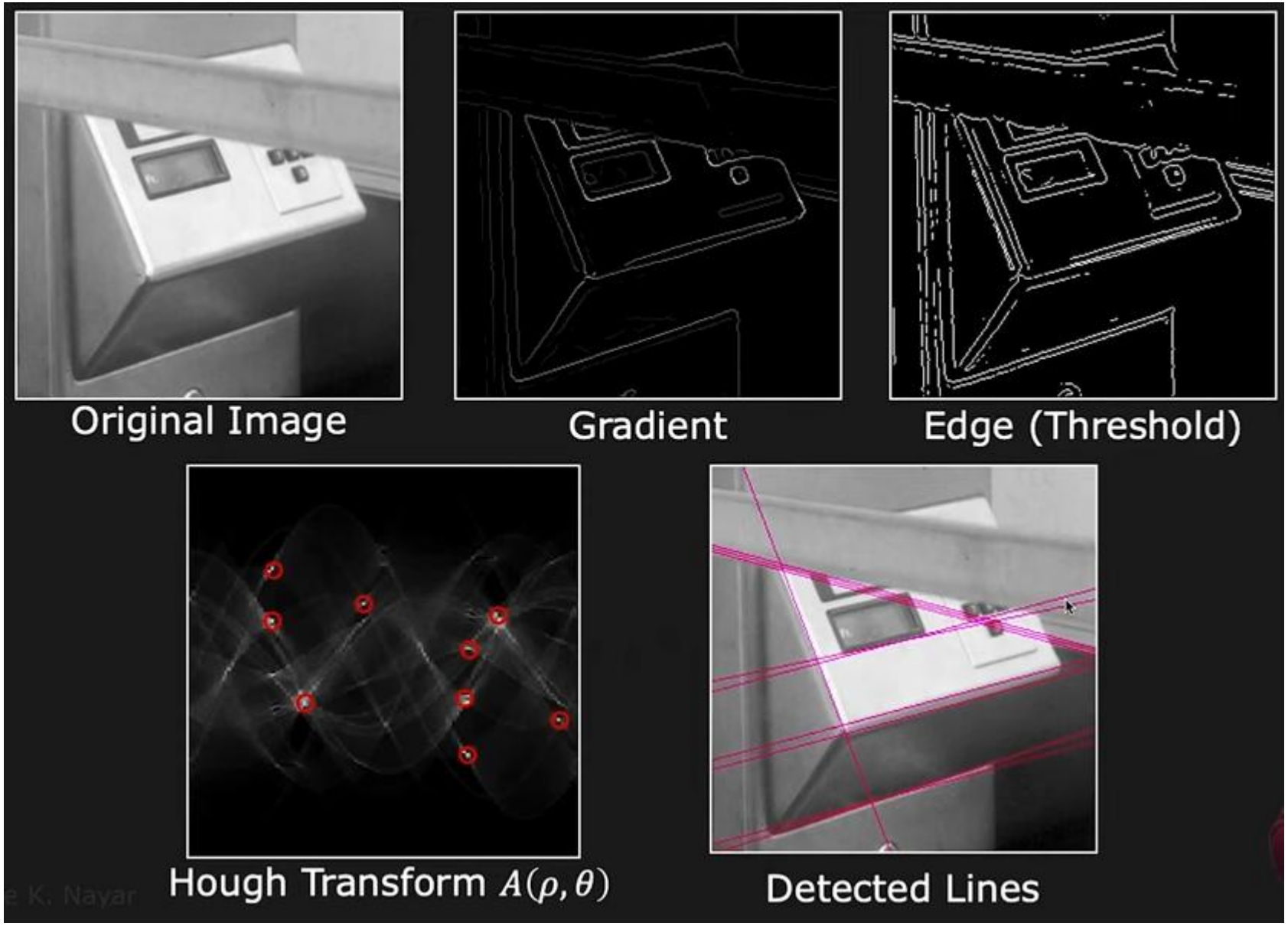

Gradient


Edge (Threshold)

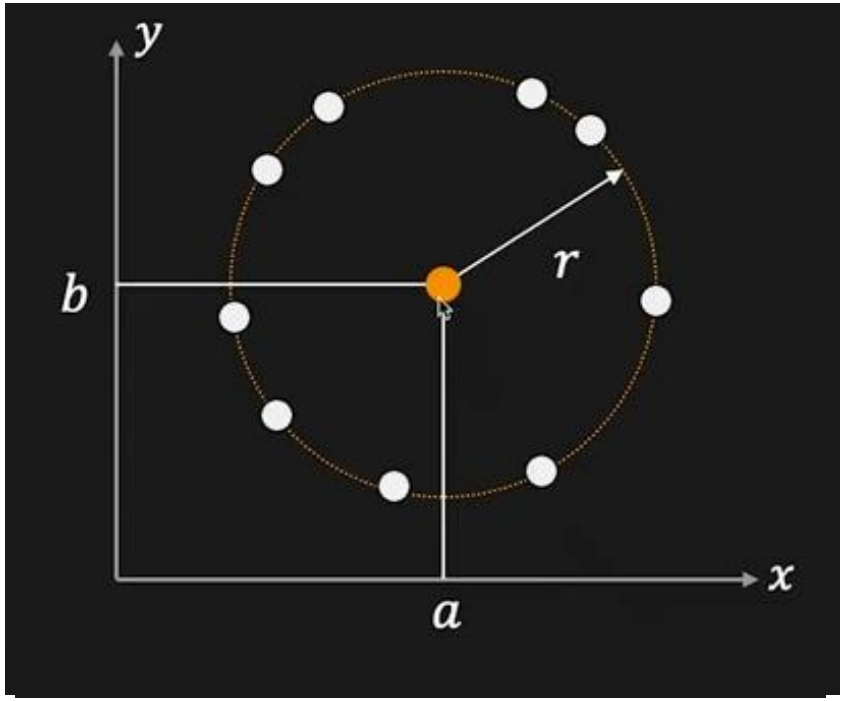

Hough Transform $A(\rho, \theta)$


Detected Lines

Line Detection Results

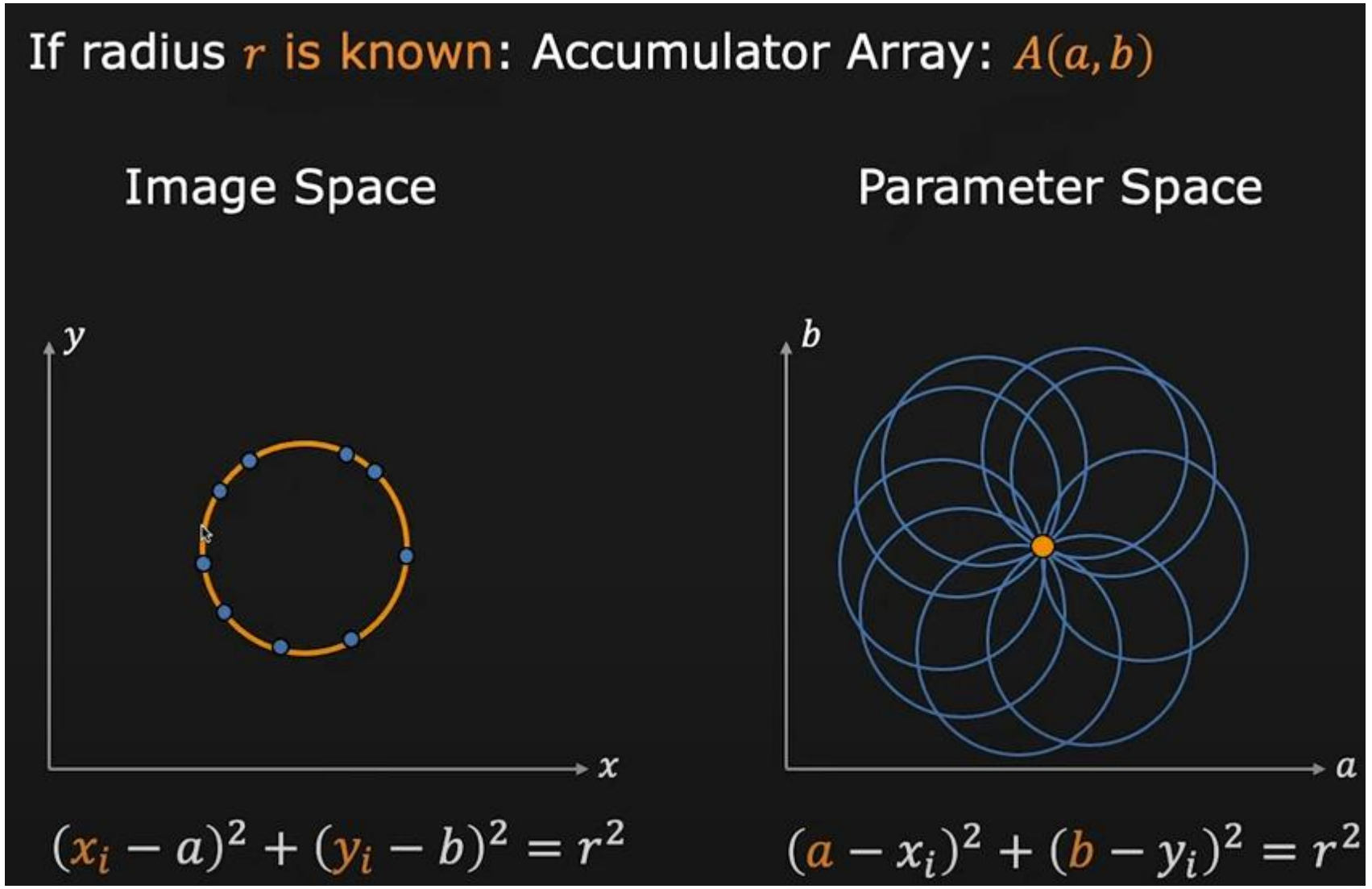


Hough Transform: Circle Detection



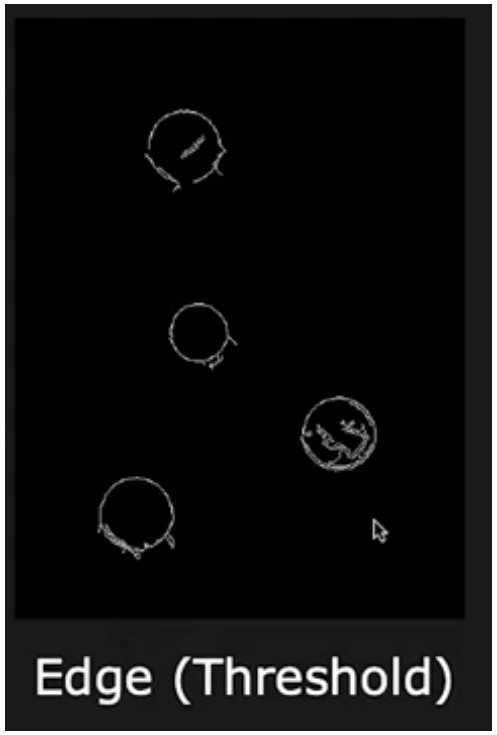
Equation of Circle: $(x_i - a)^2 + (y_i - b)^2 = r^2$

Hough Transform: Circle Detection



Circle Detection Results

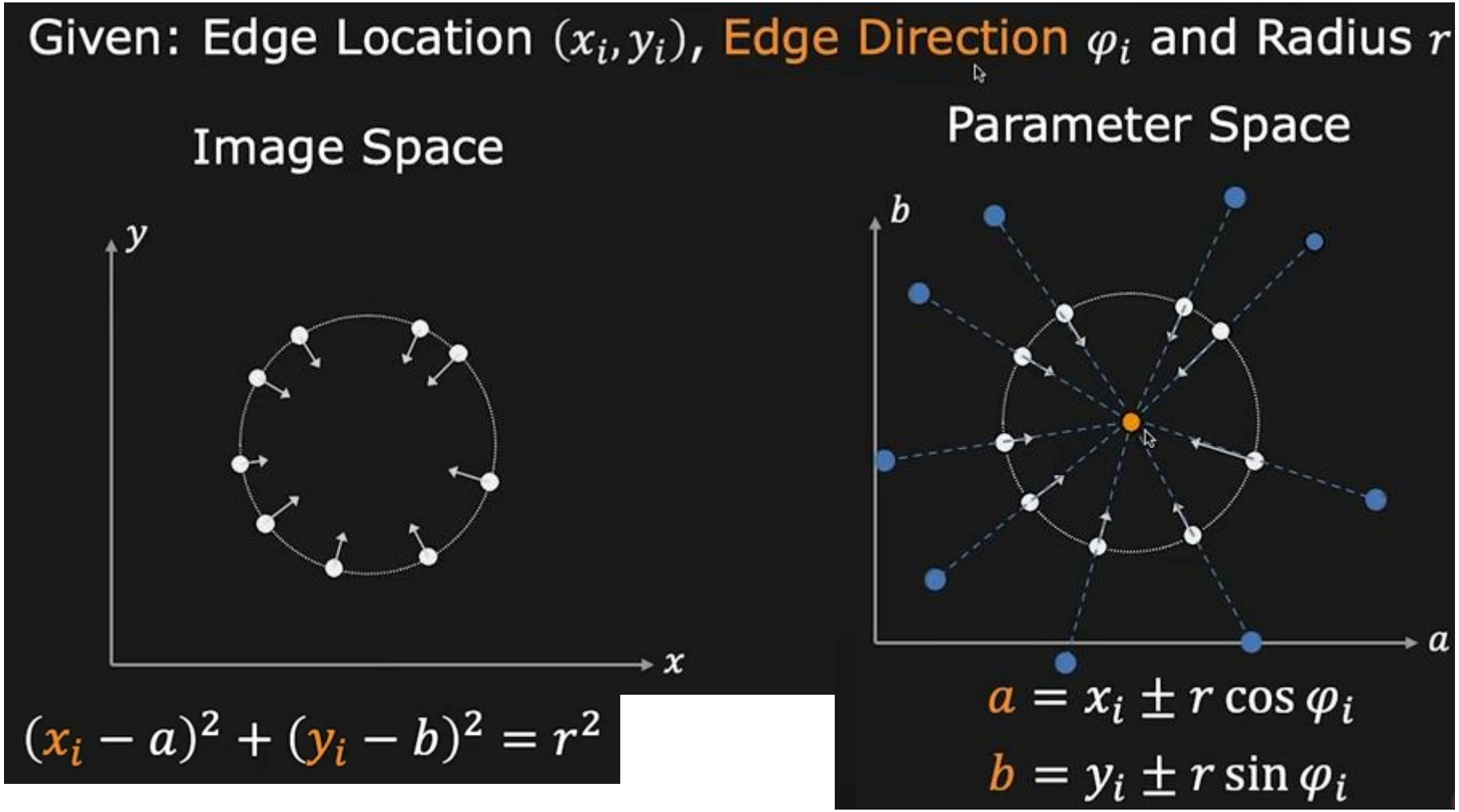
Original Image



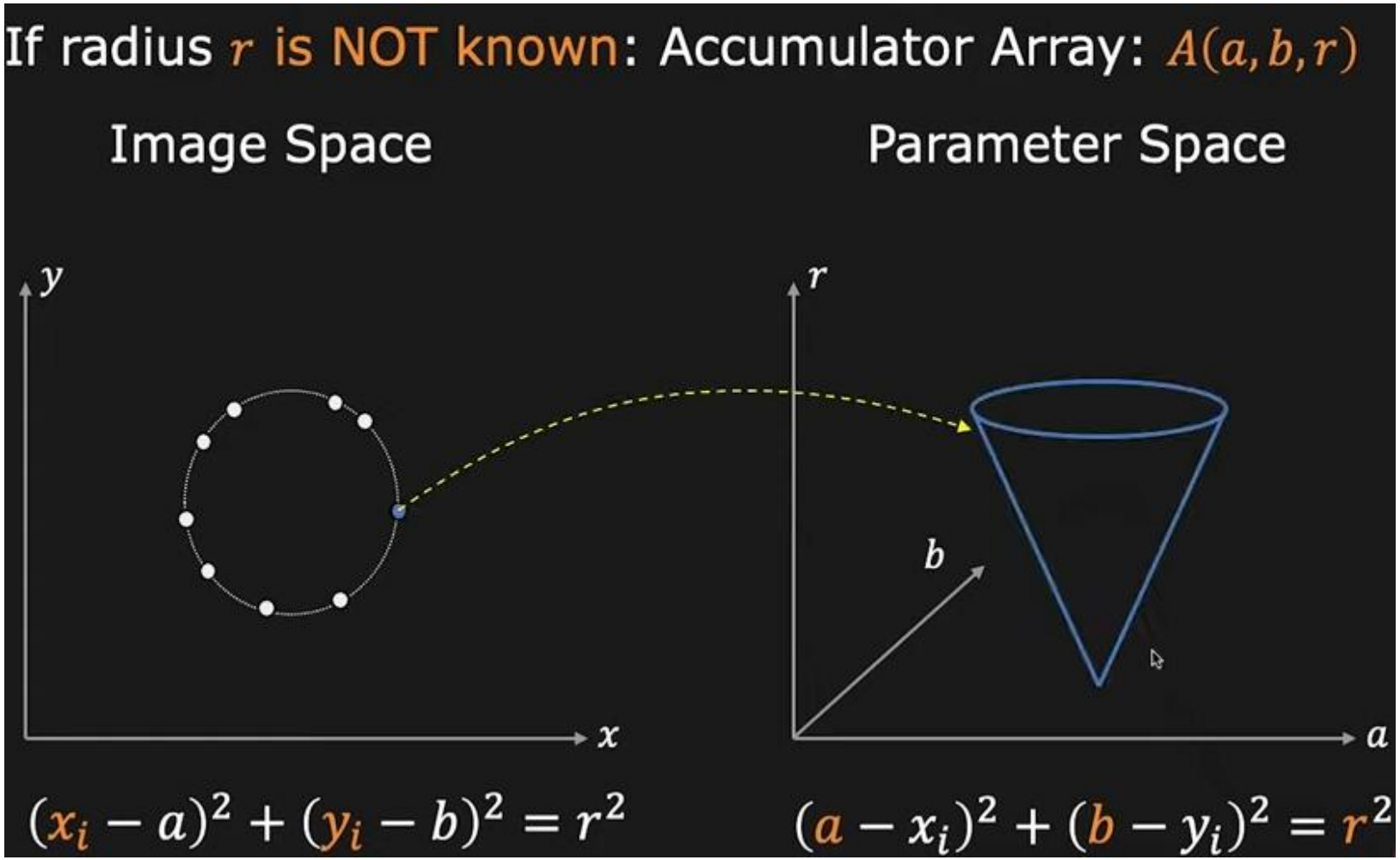
Edge (Threshold)

Hough Transform
$A_1(a, b)$

Using Gradient Information



Hough Transform: Circle Detection



In short, Hough Transform

Deals with occlusion well?

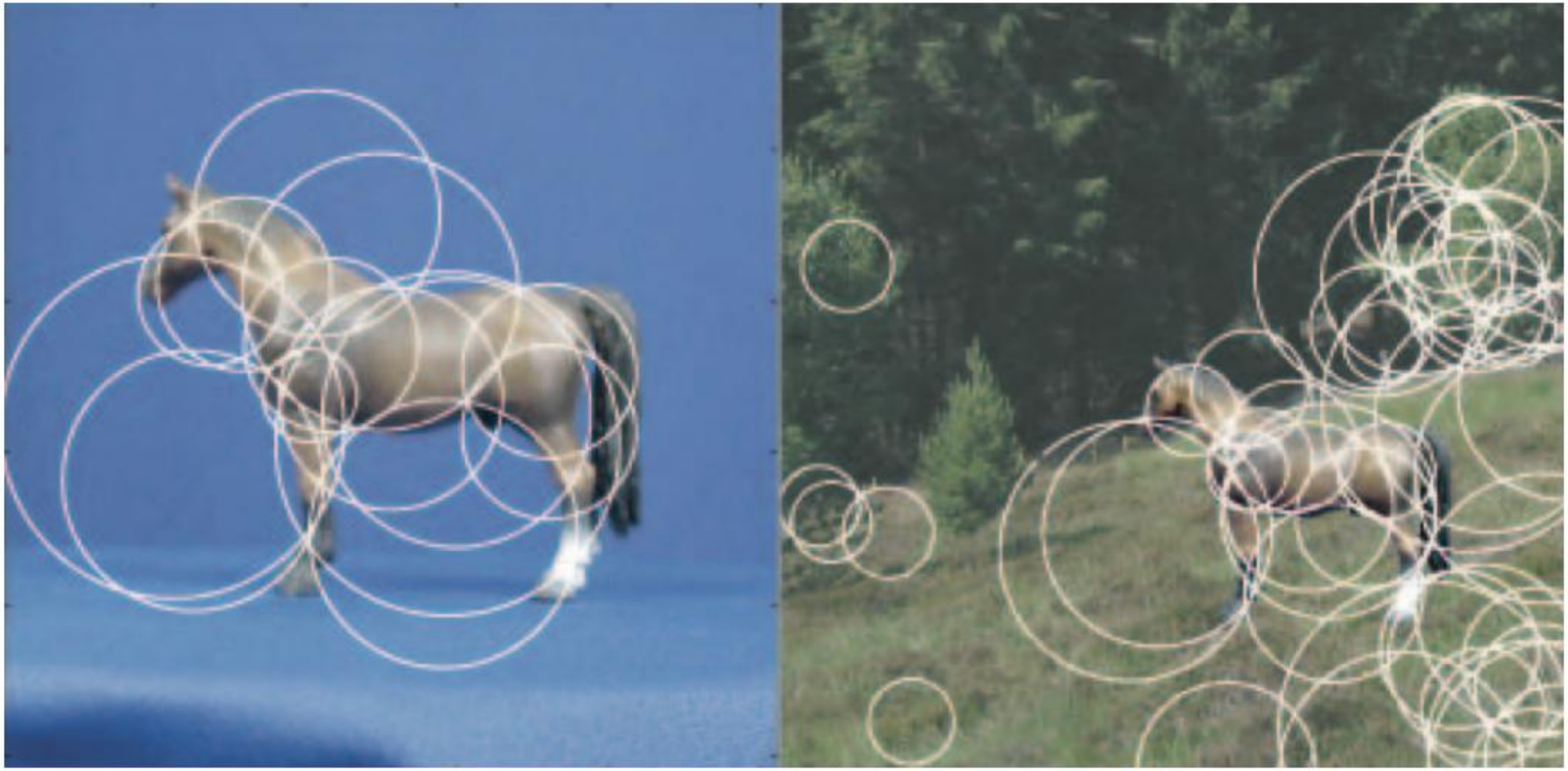
Detects multiple instances?

Robust to noise?

Good computational complexity?

Easy to set parameters?

Detecting Shape Features



Robustness to scale and clutter

Pseudocode for Hough Transform

- For all x
  - For all y
    - If edge point at (x,y)
      - For all theta
        - Rho = x * cos (theta) + y * sin (theta)
        - Increment (add 1 to) the cell in H corresponding to (theta, rho)
      - End
    - End
  - End
- End

Hough Transform in MATLAB

- If we find an edge point at (ix, iy) we loop through all possible values of theta
  - For itheta = 1:length(theta)
  - t = theta (itheta)

theta:	-90	-89	-88	...	0	...	88	89
iTheta	1	2	3	...			179	180

- For each value of t, we compute r using
  - r = ix*cos(t) + iy*sin(t);
- We next find the closest value to r in rho table:
  - We substract r from each value in the rho table
  - The location of the minimum in the difference table is where the closest value of rho was
- We can find this using “min”
  - [d, iRho] = min(abs(rho – r));

Example:

rho:	-144	-143	-142	...	0	...	143	144

If r = -142 then

rho-r:	-2	-1	0	1	2	...	285	286

then the result is
d=0 (minimum distance), iRho=3 (index)

MATLAB function for Hough Transform

- [H, theta, rho] = hough(bw);
- Peaks = houghpeaks (H, numpeaks, ‘Threshold’,thresh, ‘NHoodSize’,[M,N]);

The (rho, theta) values for the ith peak are

```
r = rho(peaks(i,1));
t = theta(peaks(i,2));
```

- Lines = houghlines(bw,theta,rho,peaks)

Hough Transform Python

- import numpy as np
- import cv2
- img = cv2.imread('image path')
- gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
- edges = cv2.Canny(gray,50,150,apertureSize = 3)
- lines = cv2.HoughLines(edges,1,np.pi/180,200)
- for rho,theta in lines[0]:
  - a = np.cos(theta)
  - b = np.sin(theta)
  - x0 = a*rho
  - y0 = b*rho
  - x1 = int(x0 + 1000*(-b))
  - y1 = int(y0 + 1000*(a))
  - x2 = int(x0 - 1000*(-b))
  - y2 = int(y0 – 1000*(a))
  - cv2.line(img, (x1,y1), (x2,y2), (0,0,255), 2)
- cv2.imshow("Hough Lines",img)
- cv2.waitKey(0)

- *Questions*
  - *Feel Free to ask*