

---

# Hough Transform

COMP 4900D

Gerhard Roth

Winter 2012

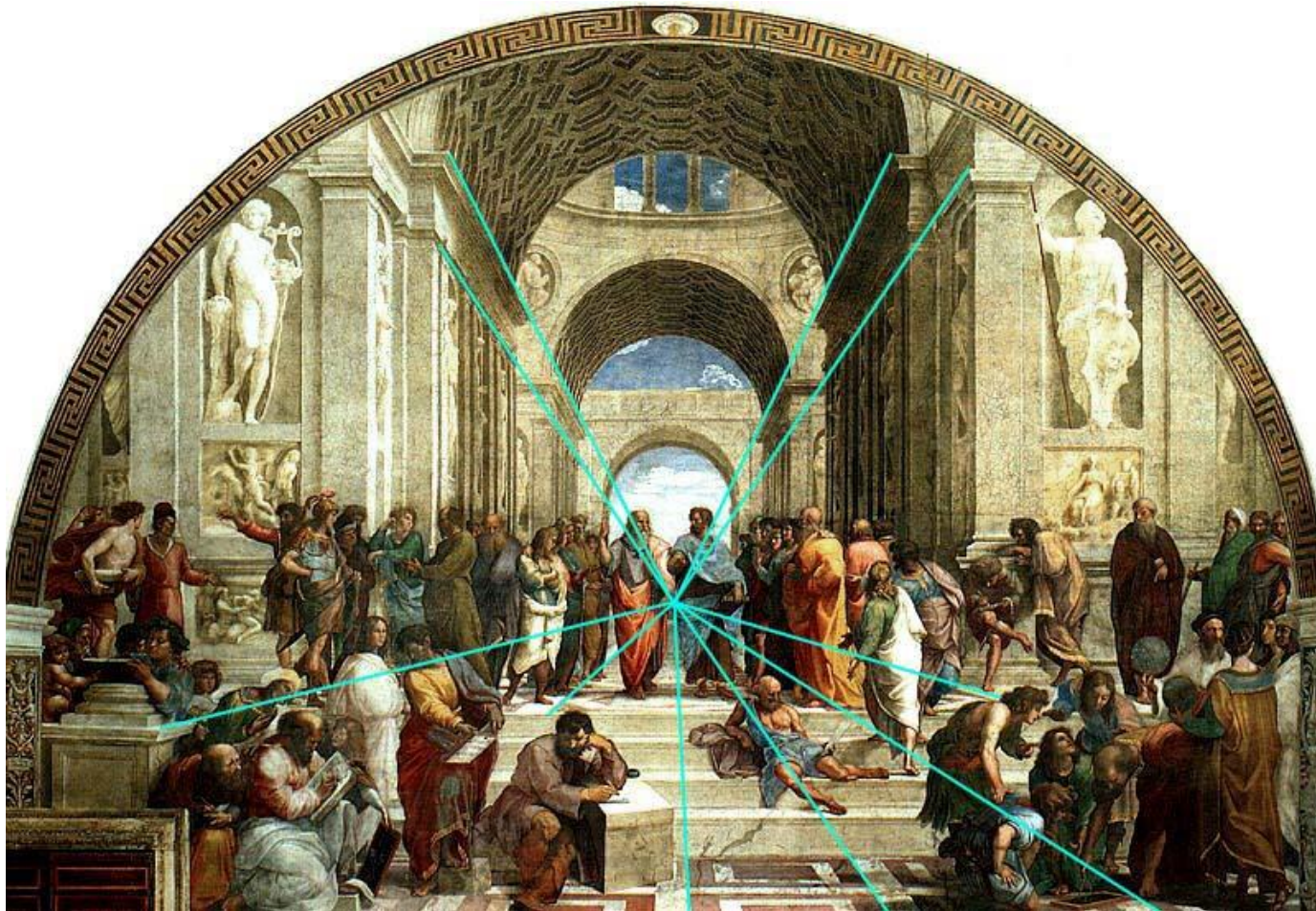
# Lines

---



# Lines

---

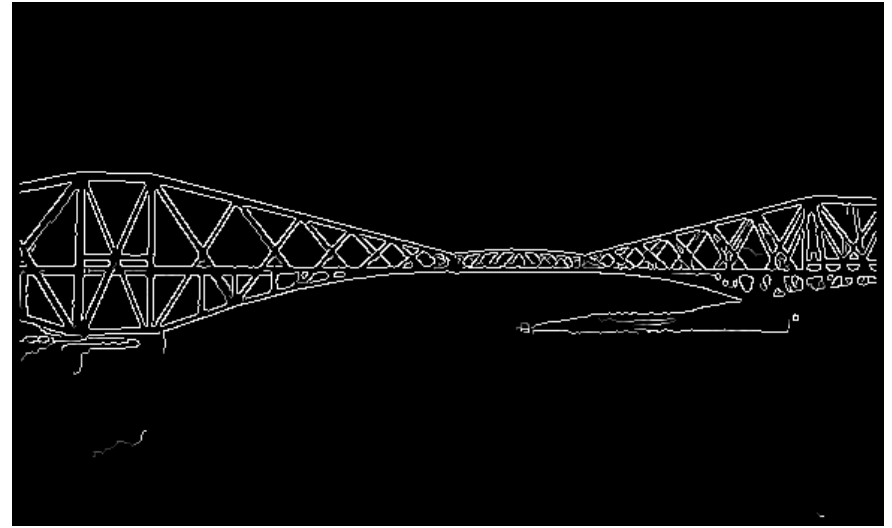


Rafael, The School of Athens (1518)



# Line Detection

---



The problem:

- How many lines?
- Find the lines.

# Equations for Lines

---

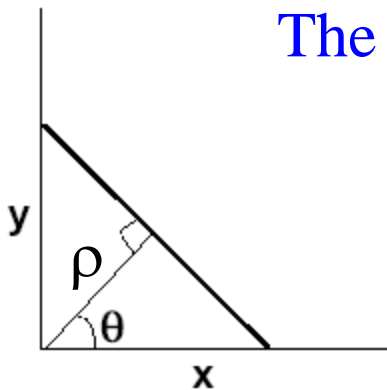
The **slope-intercept** equation of line

$$y = mx + b$$

What happens when the line is vertical? The slope  $a$  goes to infinity.

A better representation – the **polar representation**

The two parameters  $\rho, \theta$  defining line are bounded

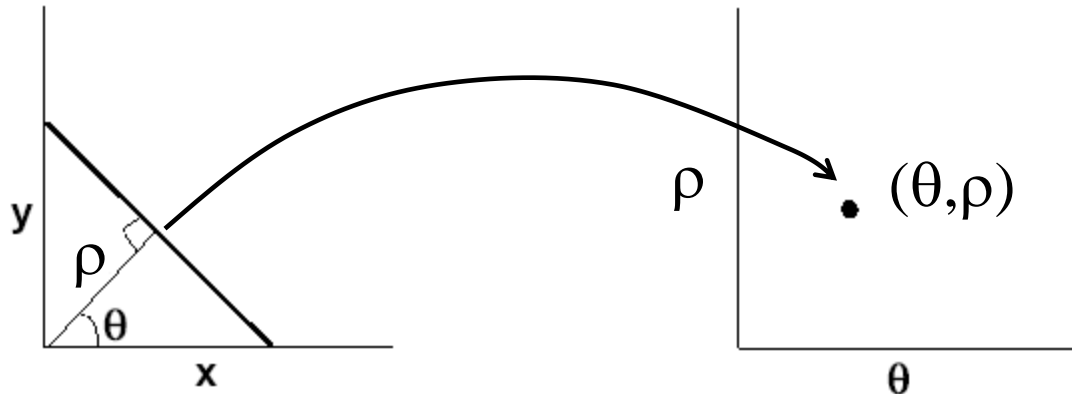


$$\rho = x \cos \theta + y \sin \theta$$

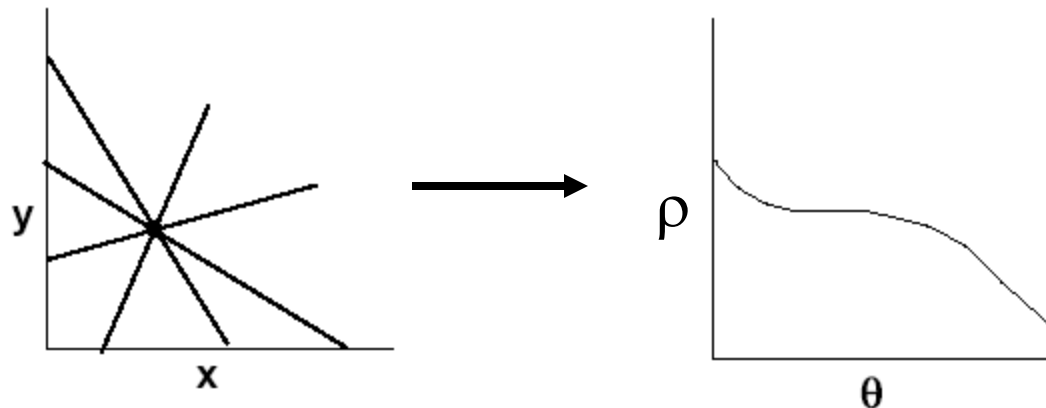
# Hough Transform: line-parameter mapping

---

A line in the plane maps to a point in the  $\theta$ - $\rho$  space.



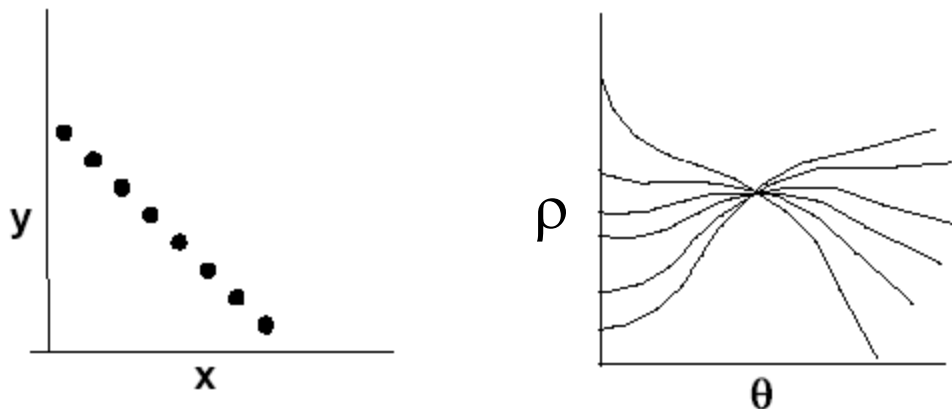
All lines passing through a point map to a sinusoidal curve in the  $\theta$ - $\rho$  (parameter) space.



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

# Mapping of points on a line

---



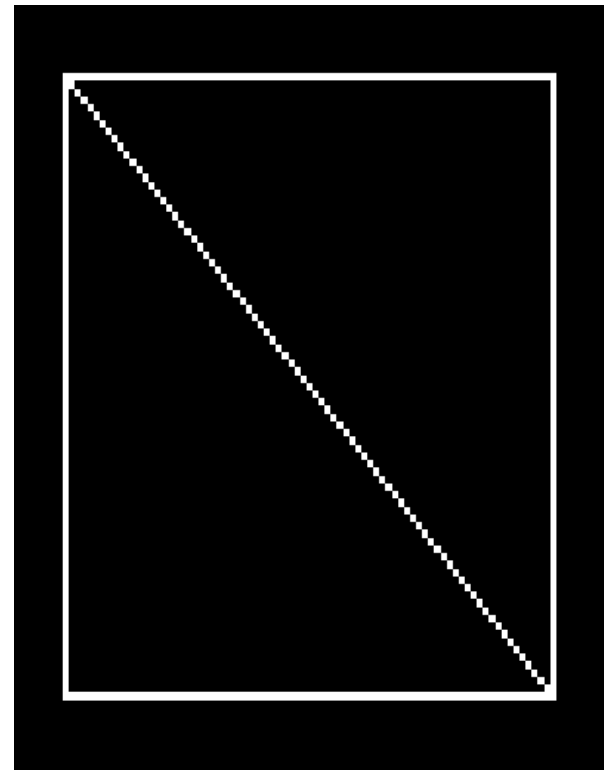
Points on the same line define curves in the parameter space that pass through a single point.

Main idea: transform edge points in  $x$ - $y$  plane to curves in the parameter space. Then find the points in the parameter space that has many curves passing through it.

# Hough Idea

---

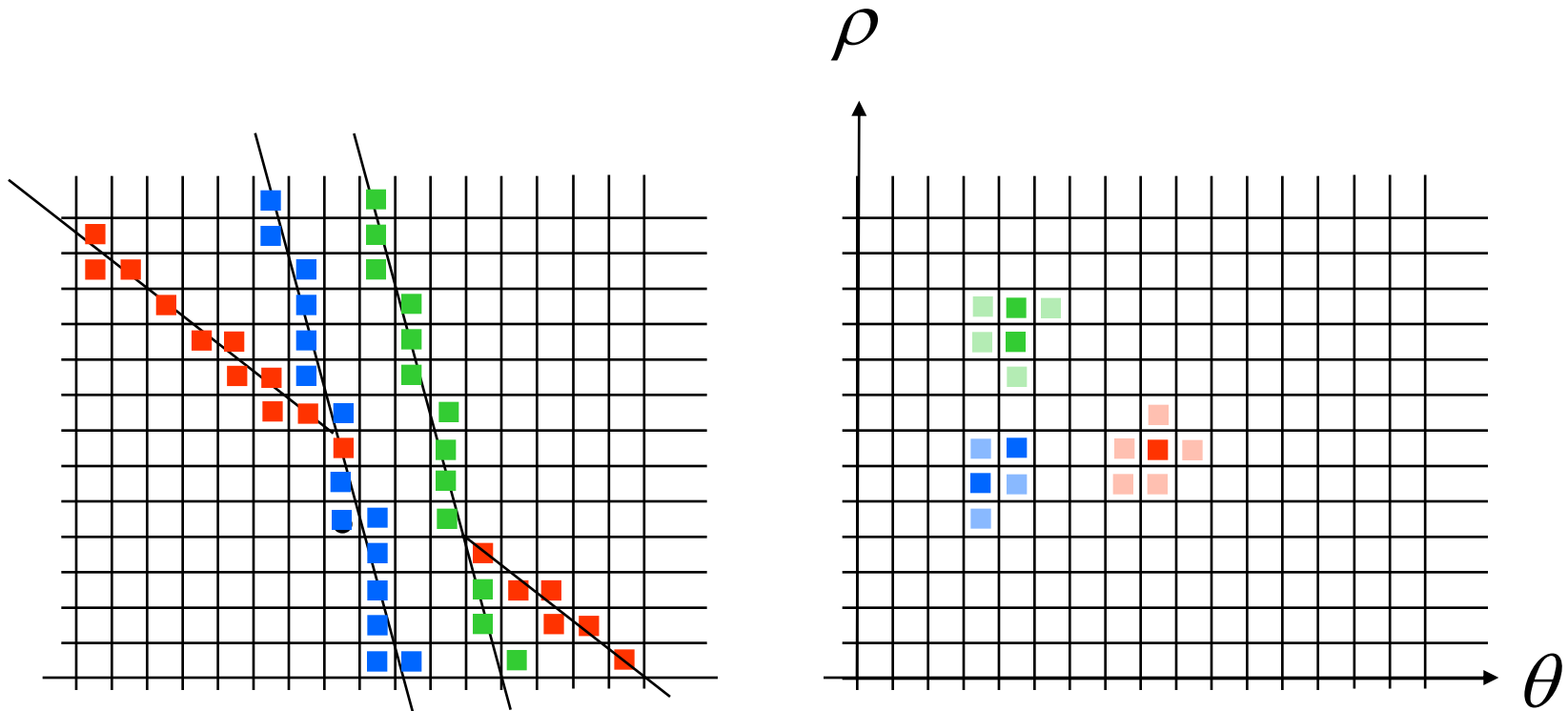
- Each straight line in this image can be described by an equation
- Each white point if considered in isolation could lie on an infinite number of straight lines
- **In the Hough transform each point votes for every line it could be on**
- The lines with the most votes win





# Quantize Parameter Space

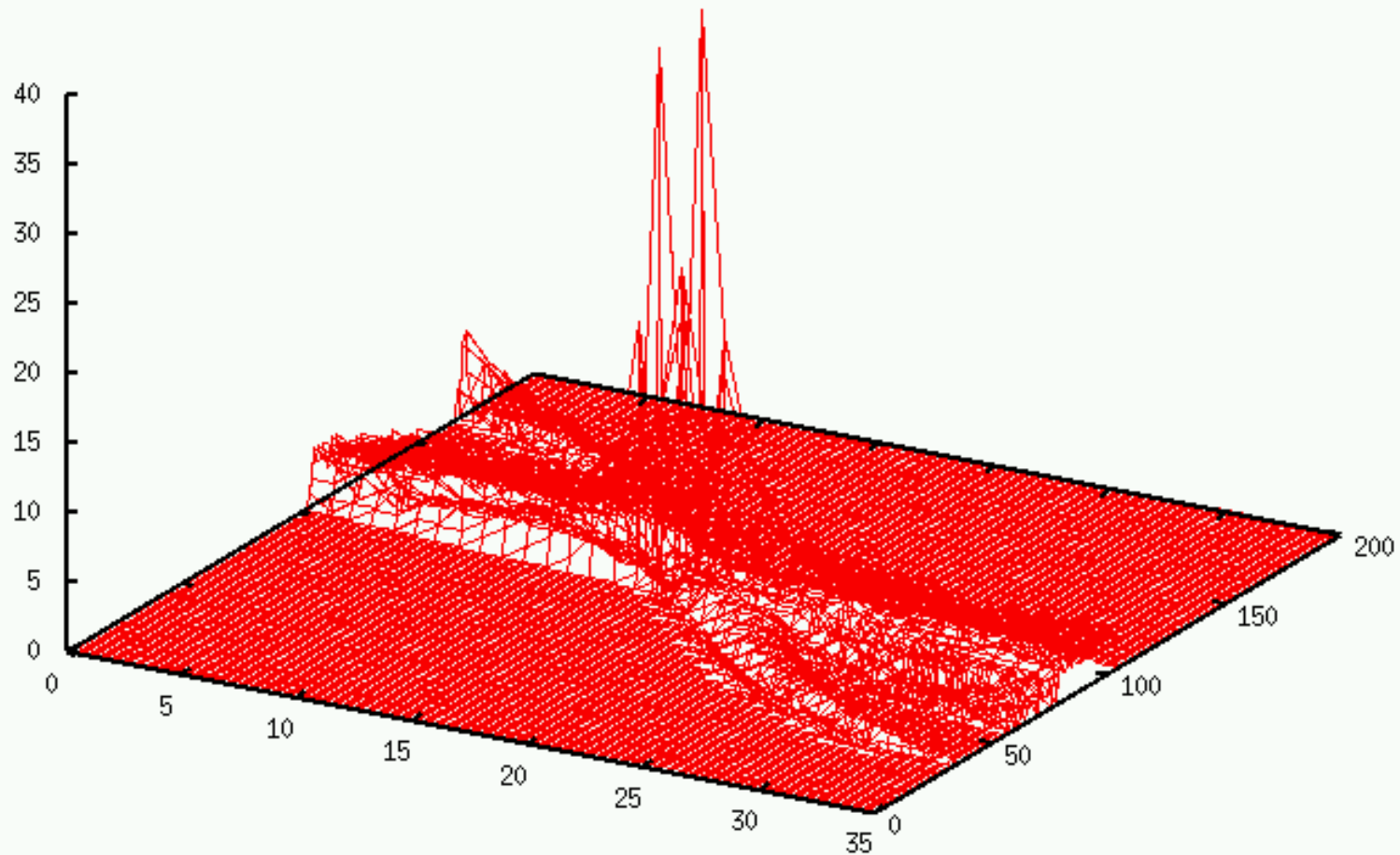
---



Detecting Lines by finding maxima / clustering in parameter space.

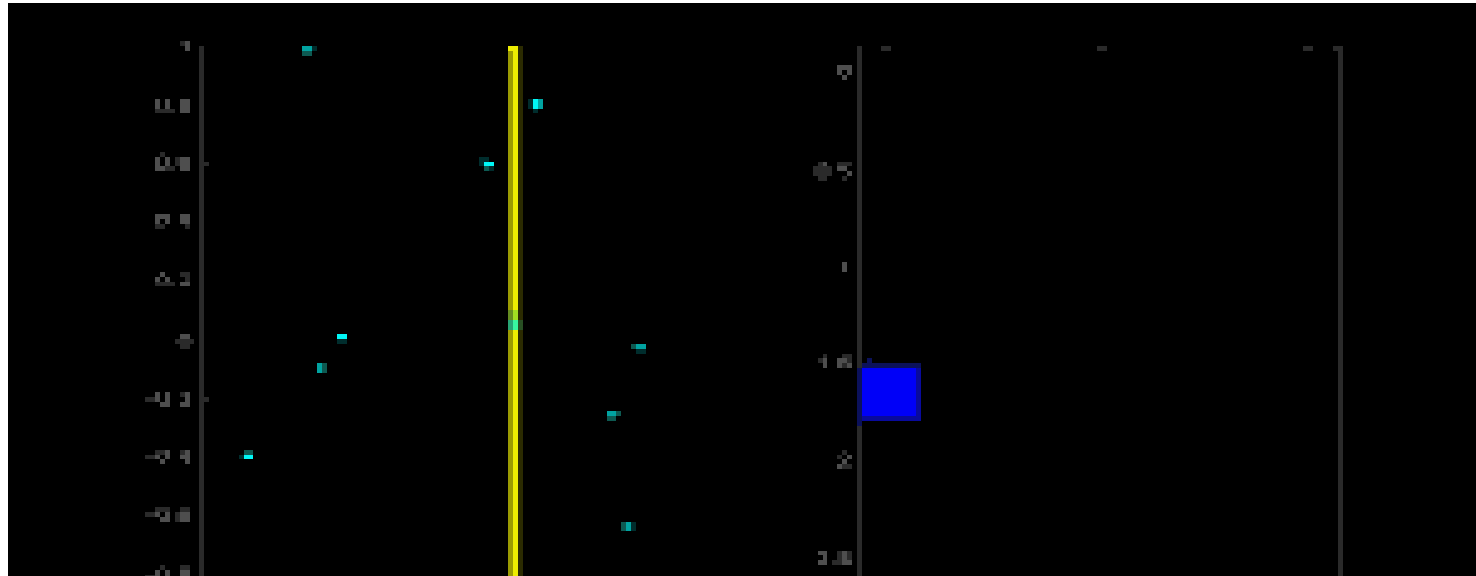
# Parameter space – 3D view

---



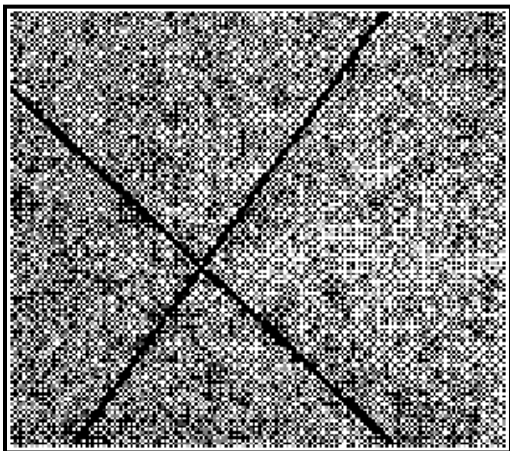
# A Voting Scheme

---



# Hough Processing

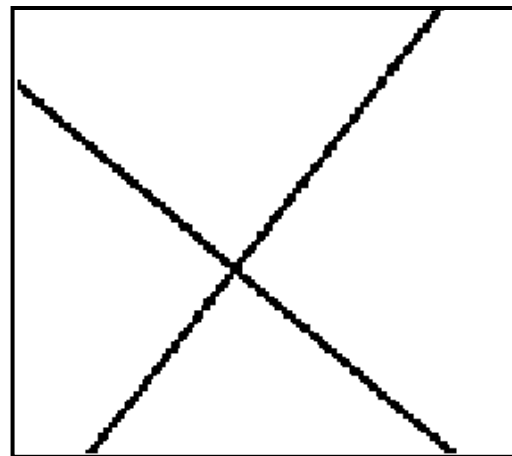
---



**Image**



**Edge detection**



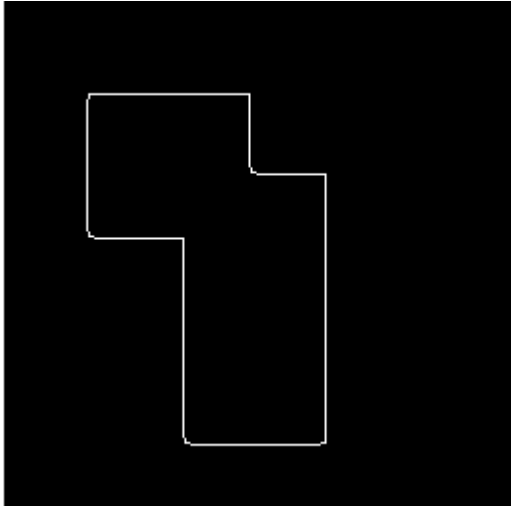
**Hough Transform**

- Find the edges in the image (Canny operator common)
- Use each edge point to vote in the accumulator space
  - Accumulator space also called the Hough Space
- Find the peak(s) in the accumulator space

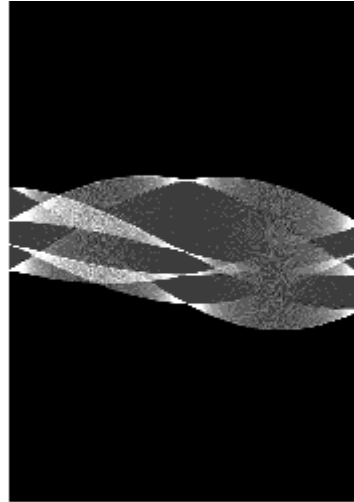
# Examples

---

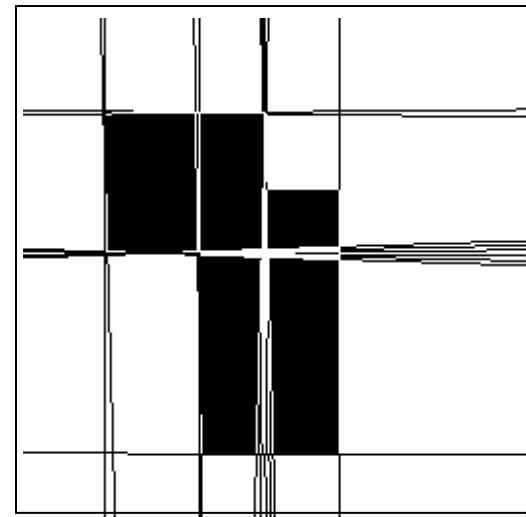
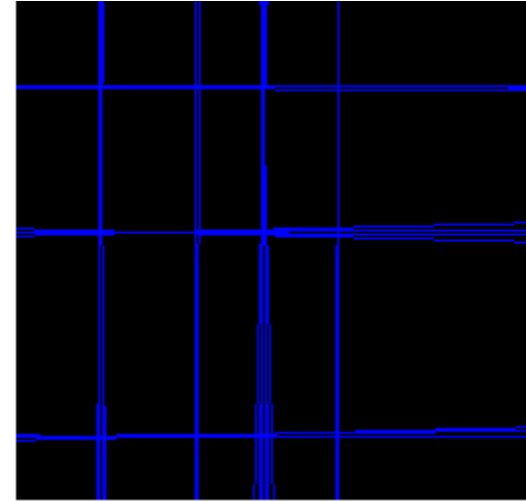
input image



Hough space



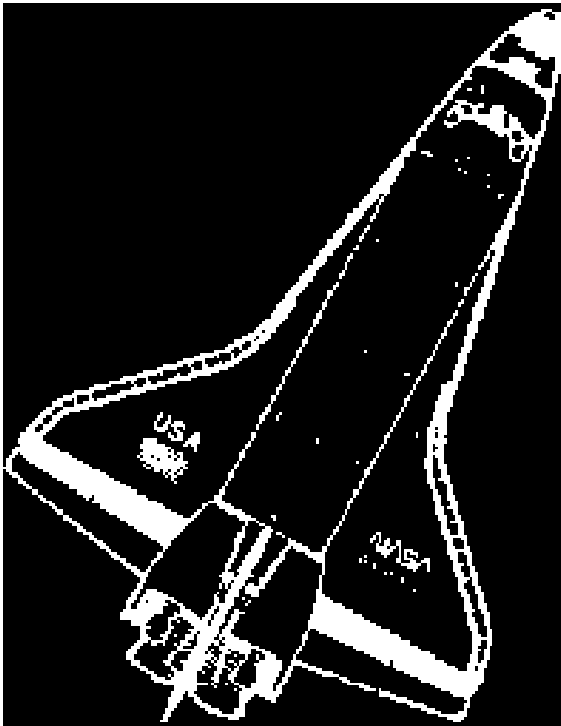
lines detected



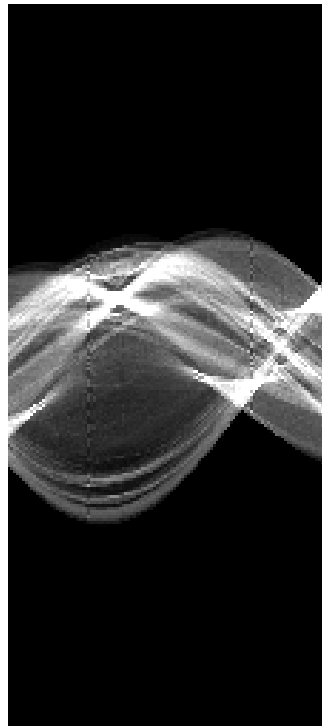
# Examples

---

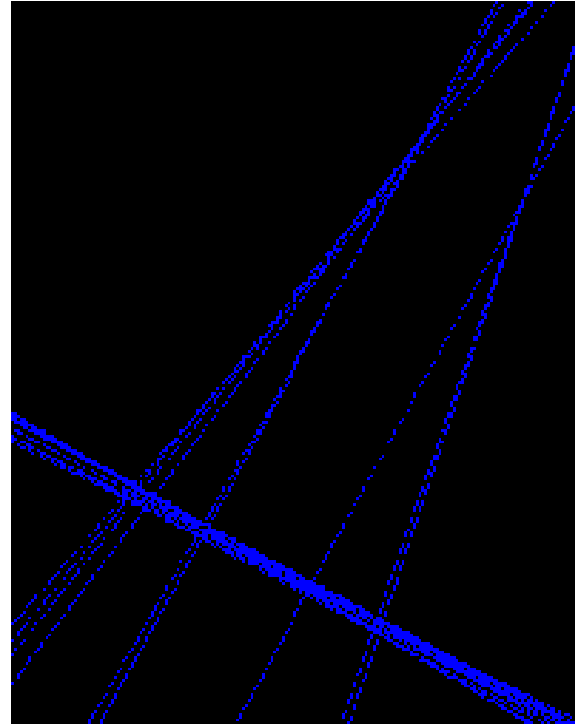
input image



Hough space



lines detected



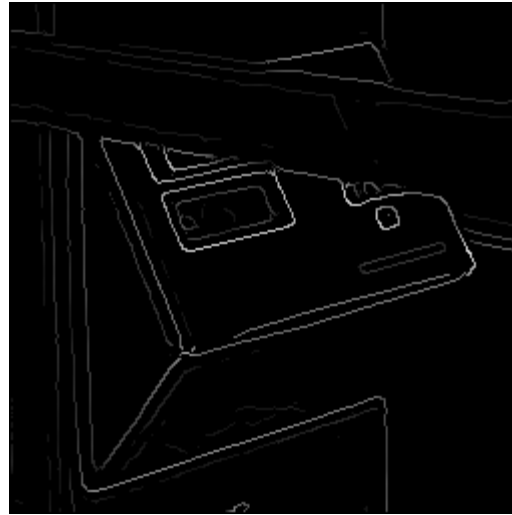


# Examples

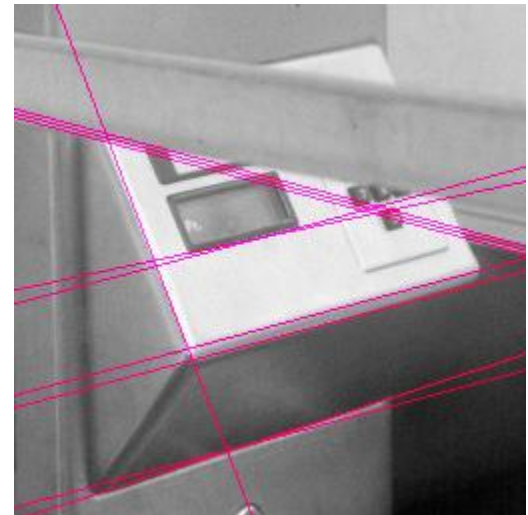
---



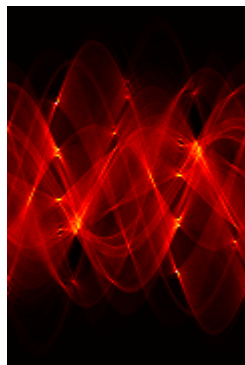
Original



Edge Detection



Found Lines



Parameter Space

# Algorithm

---

1. Quantize the parameter space

int P[0,  $\rho_{\max}$ ][0,  $\theta_{\max}$ ]; // accumulators

2. For each edge point  $(x, y)$  {

For ( $\theta = 0$ ;  $\theta \leq \theta_{\max}$ ;  $\theta = \theta + \Delta\theta$ ) {

$\rho = x \cos \theta + y \sin \theta$  // round off to integer

(P[ $\rho$ ][ $\theta$ ])++;

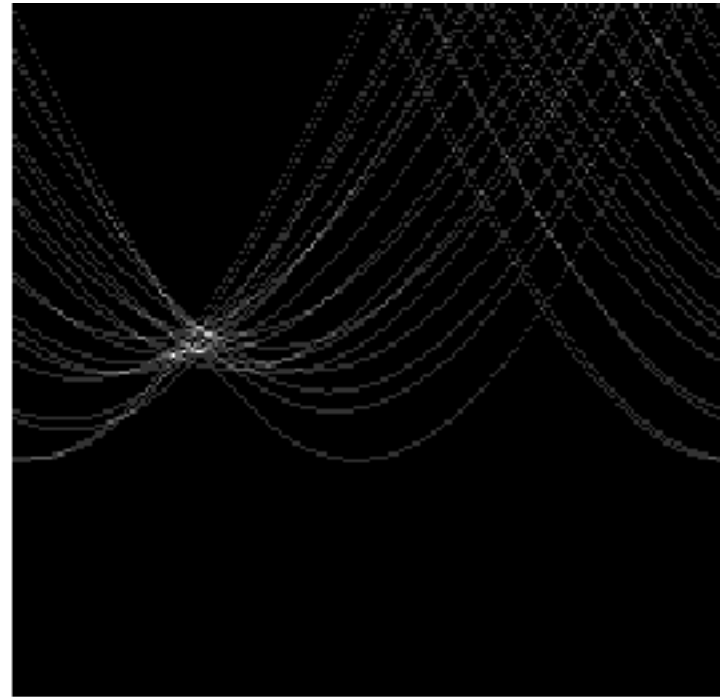
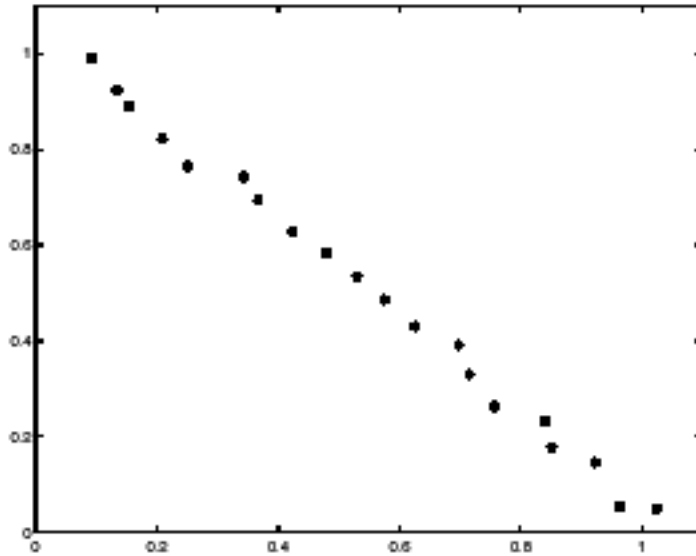
}

}

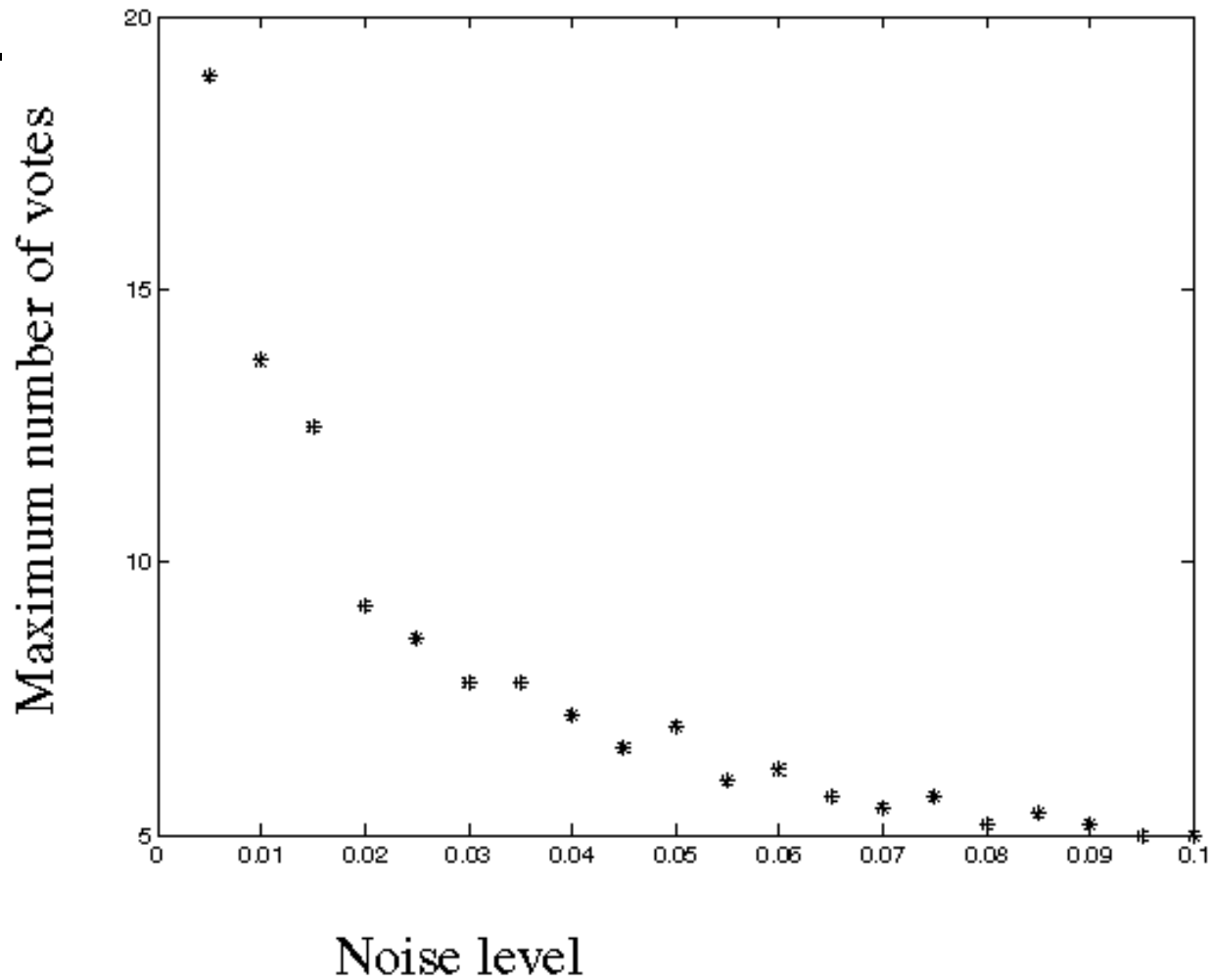
3. Find the peaks in P[ $\rho$ ][ $\theta$ ].

# Cell Size

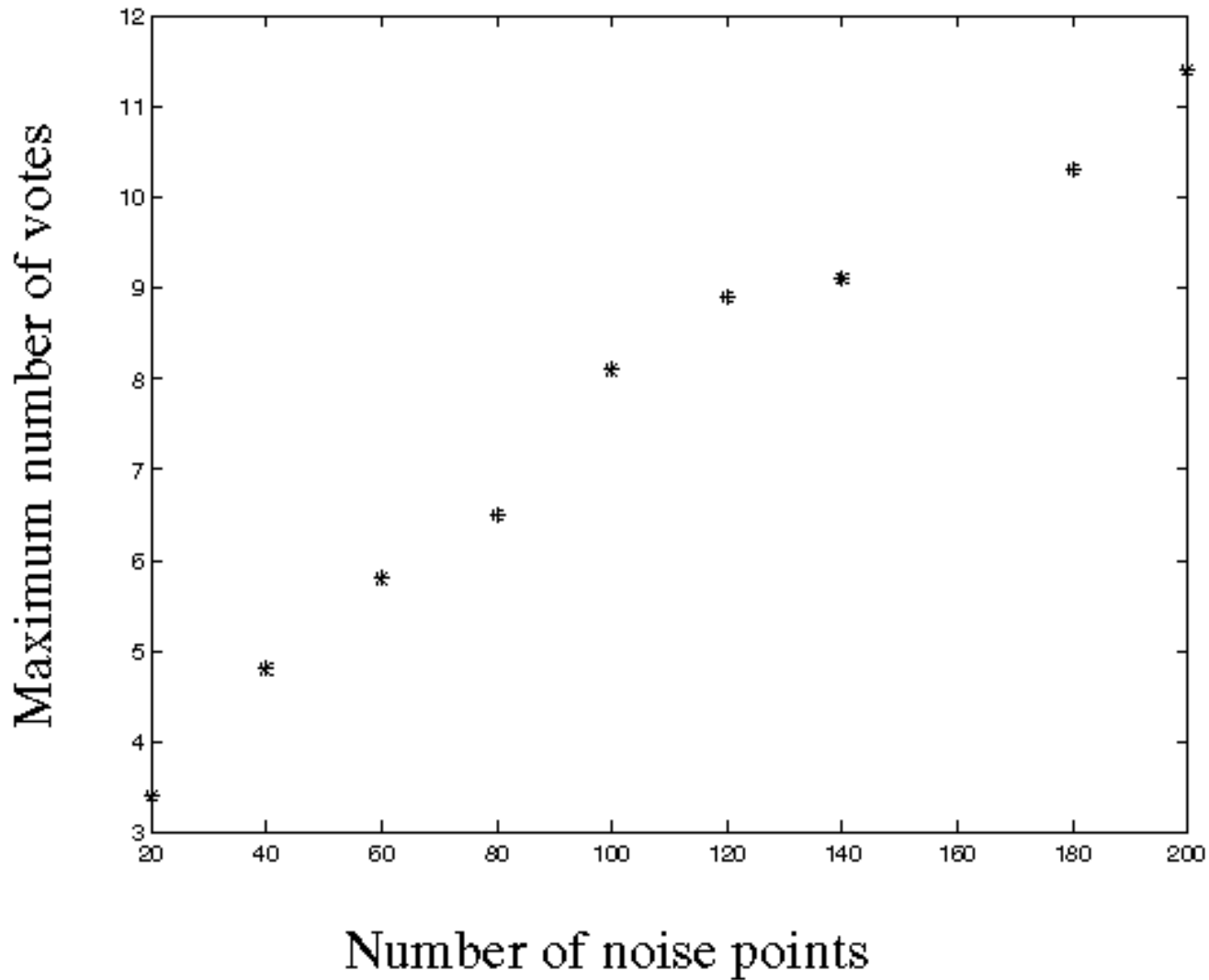
---



Choose the parameter cell size such that the algorithm is robust to noise.



Fewer votes land in a single bin when noise increases.



Adding more clutter increases number of bins with false peaks.

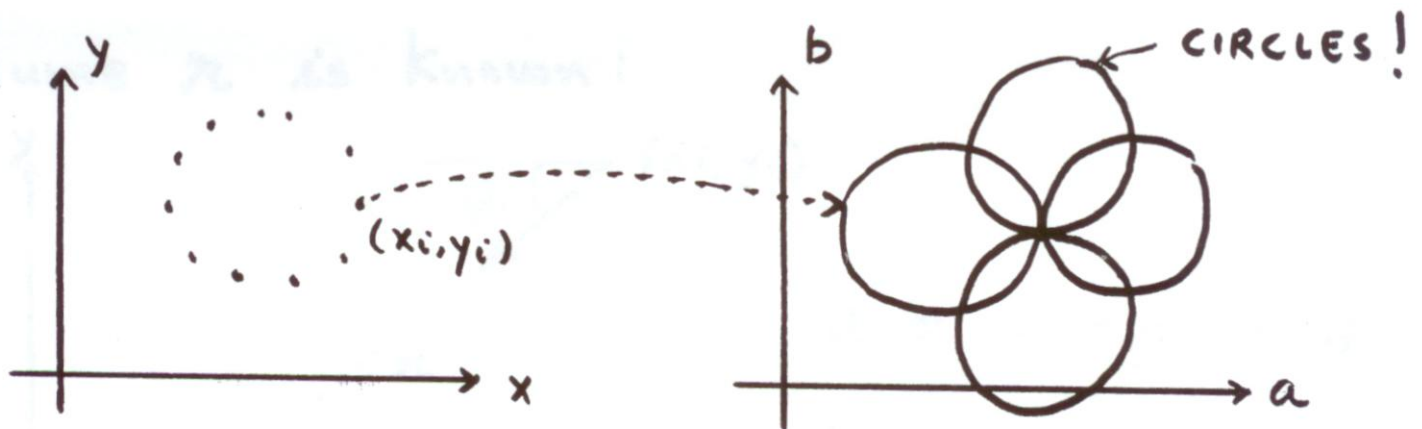
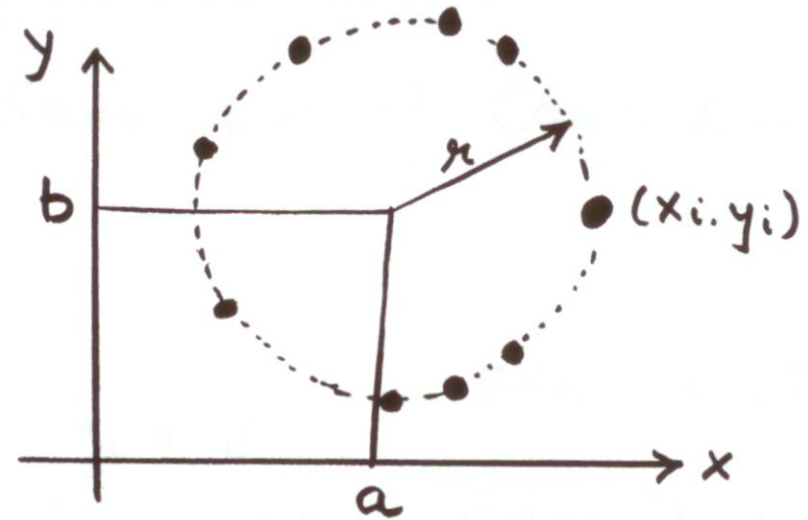
# Finding Circles by Hough Transform

Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$

If radius is known: (2D Hough Space)

Accumulator Array  $A(a, b)$



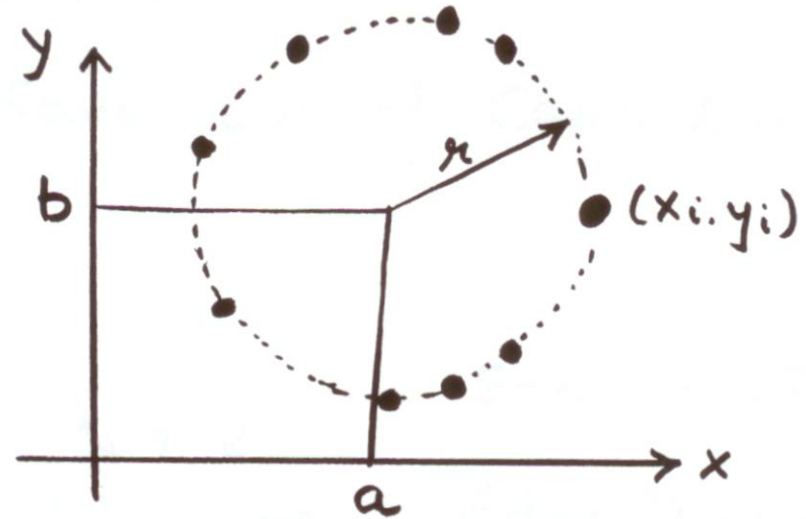


# Finding Circles by Hough Transform

---

Equation of Circle:

$$(x_i - a)^2 + (y_i - b)^2 = r^2$$



If radius is not known: 3D Hough Space!

Use Accumulator array  $A(a, b, r)$

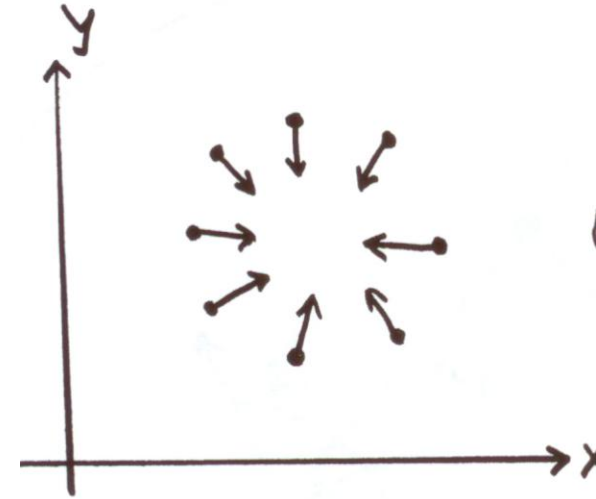
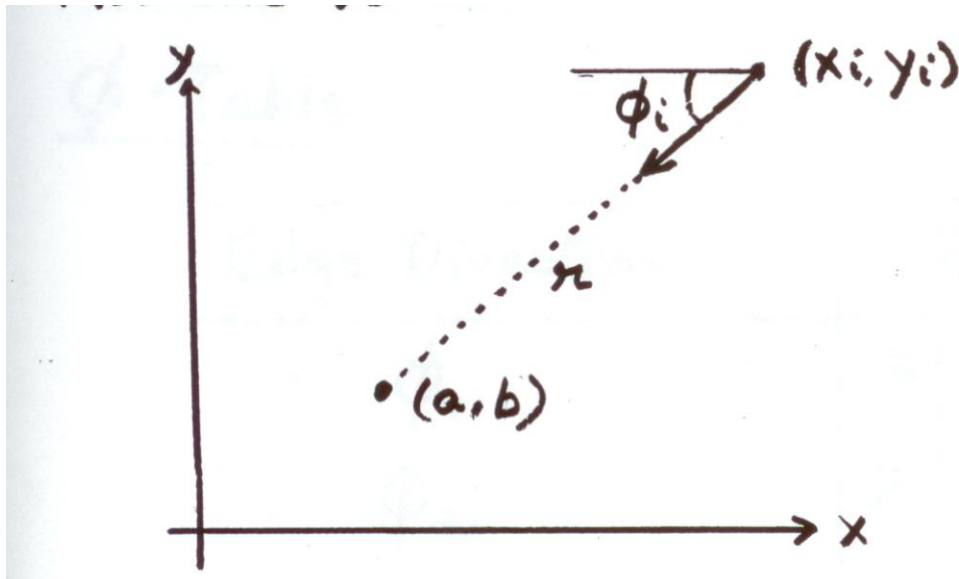
# Using Gradient Information

- Gradient information can save lot of computation:

Edge Location  $(x_i, y_i)$

Edge Direction  $\phi_i$

Assume radius is known:



$$a = x - r \cos \phi$$

$$b = y - r \sin \phi$$

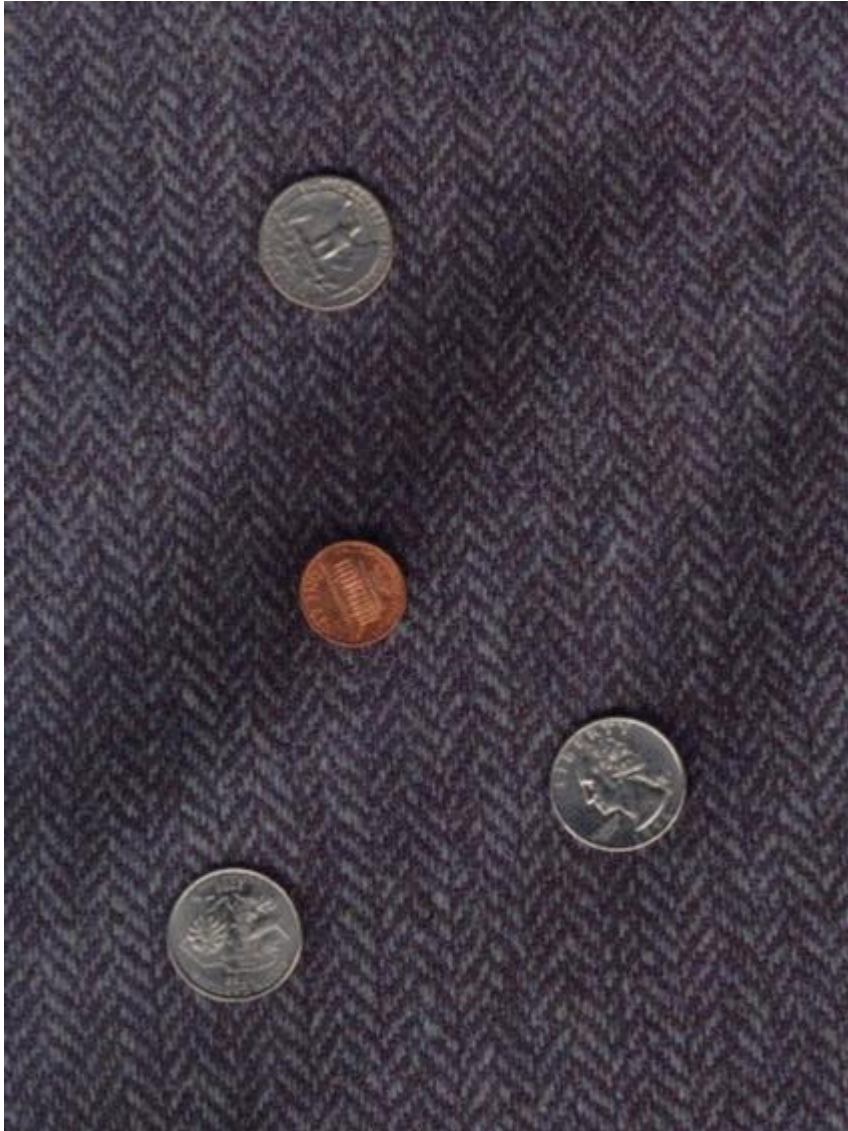
Need to increment only one point in Accumulator!!

If radius is not known, accumulator is 2d using gradients

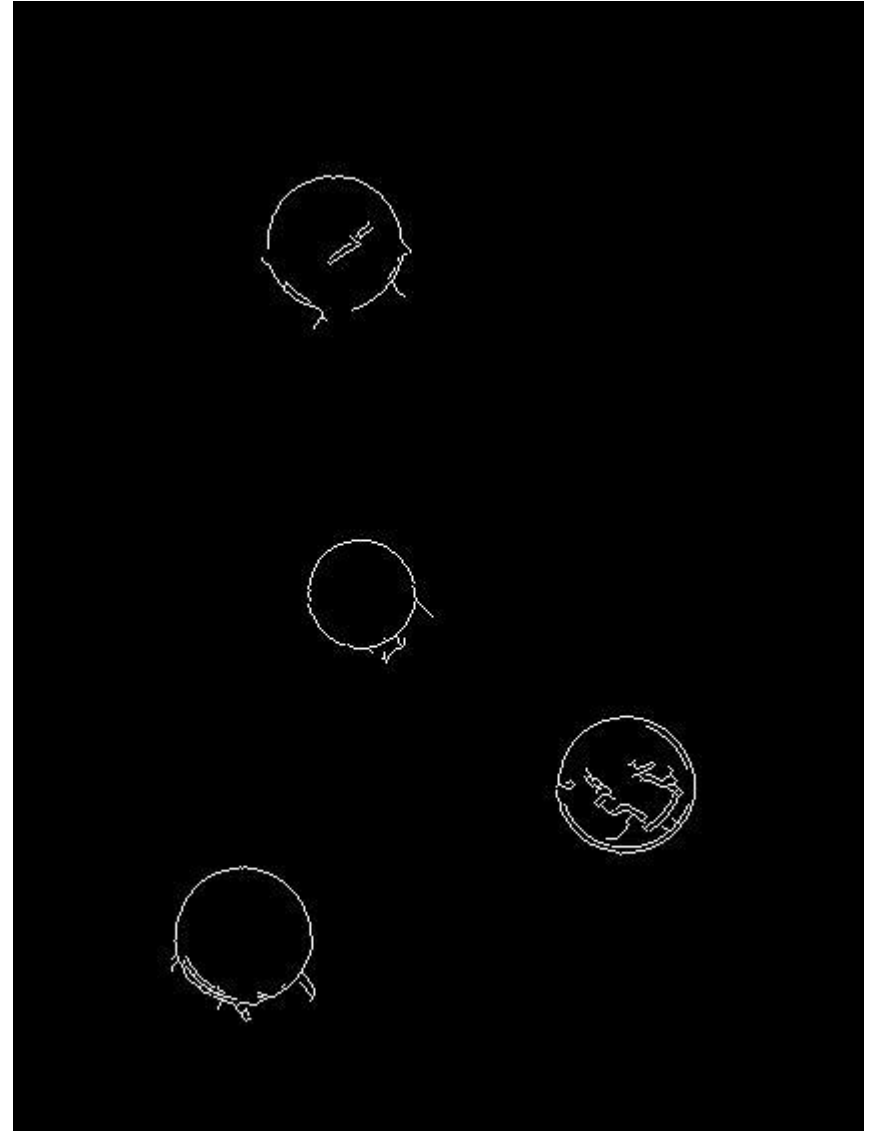
# Finding Coins

---

Original



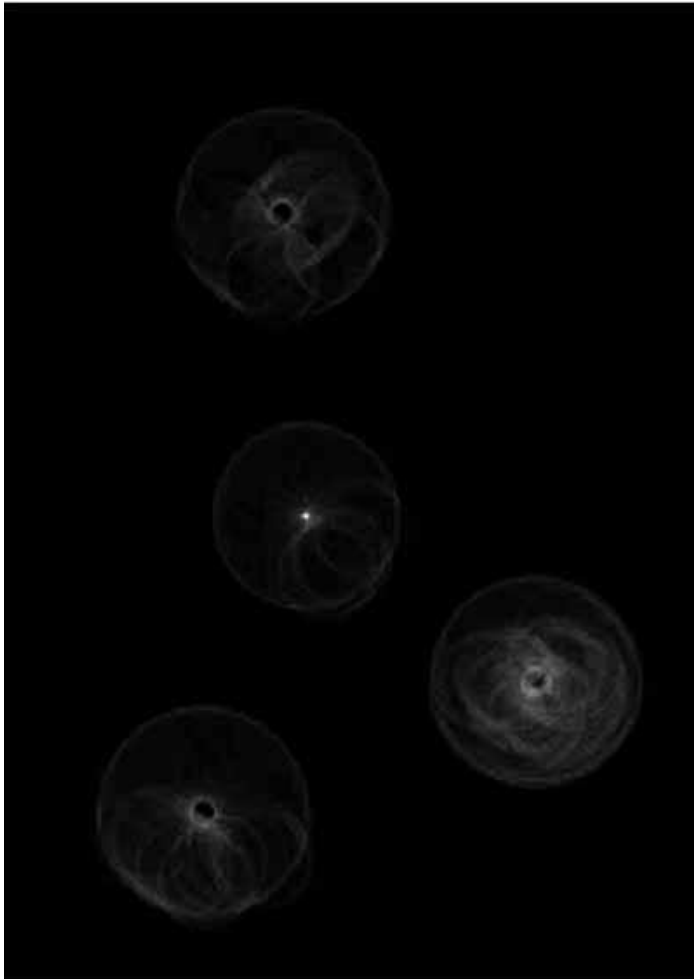
Edges (note noise)



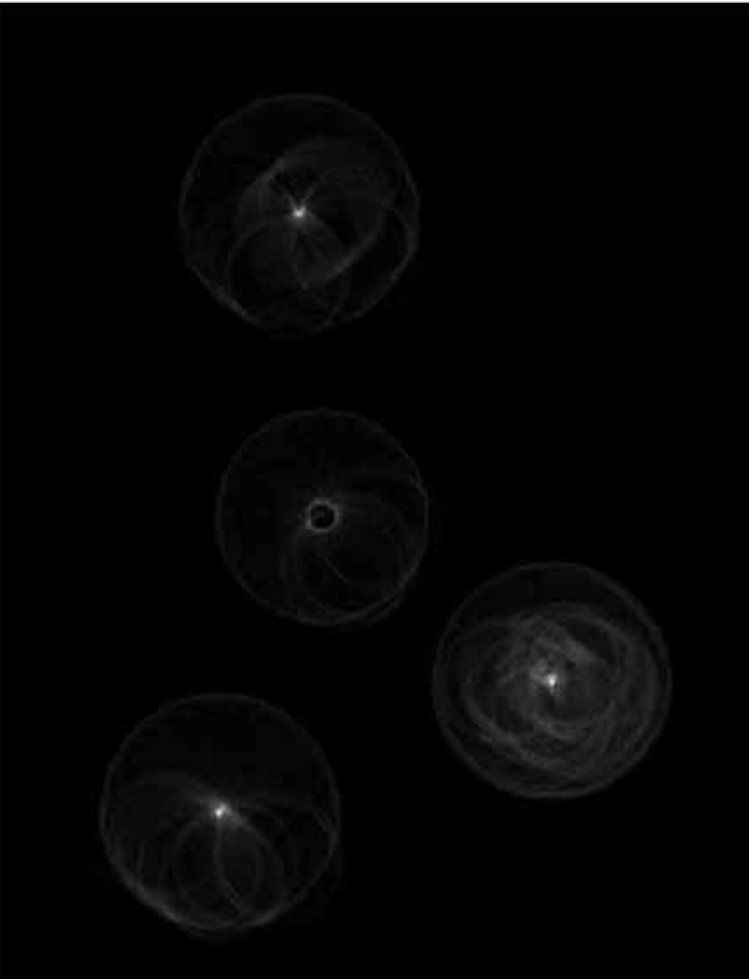
# Finding Coins (Continued)

---

Penny

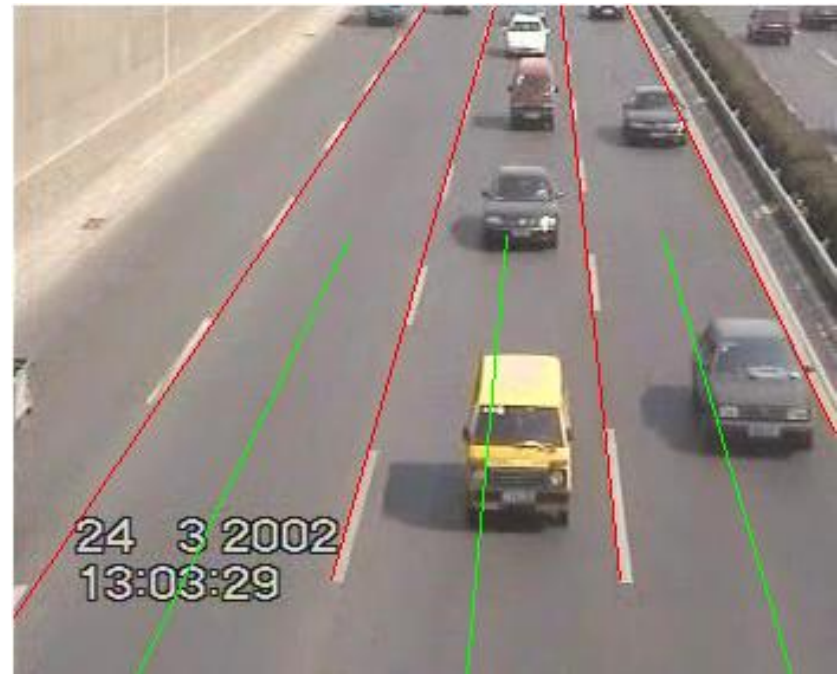


Quarters



# Application: Lane Detection

---



# Hough Characteristics

---

- Detects all the curves in an image at once
- Running time proportional to the number of edge points that are in the image
- Can deal with disconnected edge points
  - Does not assume (require) any connectivity for edges
- Accumulator dimension (space) proportional to number of parameters that define the curve
  - Works well for lines (only 2d accumulator array necessary)
- Not easy to extend to more complex curves because of the space requirements
  - Can use image gradient to decrease space requirements
- Using gradients works well for circles



# Probabilistic Hough Transform

---

- Given a set of  $p$  edge points in an image
  - Goal is to find a particular curve (line, or circle)
  - Idea is that given  $n$  edge points ( $n$  is 2 for line or 3 for circle) we can create a unique curve through just these points
- Do while we have enough edge points
  - For  $K$  times (a parameter)
    - Choose  $n$  random edge points (2 for line and 3 for circle)
    - Create a unique line or circle through these points
    - Count the number of edge points that are within  $d$  pixels (another parameter) of that unique line or circle
    - Save the best curve (has most points within distance  $d$ )
  - Endfor
- remove the edge points found for best curve
- Enddo

# Probabilistic Hough Transform

---

- Two parameters distance  $d$ , and #samples  $K$ 
  - Distance  $d$  is typically set in range 1 to 5 pixels
  - #samples  $K$  depends on how many curves you expect there to be in the image
- Given expectation of at most  $n$  curves in the image you can compute a value for  $K$ 
  - $K$  is an exponential function of  $n$ , the degrees of freedom (dof) of the curve, which is 2 for a line and 3 for a circle
- Running time  $O(n K p)$  where  $K$  is number of samples, and  $p$  is the number of edge points
- Space requirements are low so you could use this for complex curves (like ellipse, 5 dof)

# Probabilistic HT relative to ordinary HT

---

- Ordinary HT space requirements where  $q$  is grid size are  $q^2$  for line and  $q^3$  for a circle
  - Running time is  $O(q p)$  with  $p$  edge points
- Probabilistic HT space requirements are simply  $O(p)$ , the number of edge points
  - Running time is  $O(n K p)$ ,  $n$  curves  $p$  edge points,  $K$  samples
- Which is faster for lines and circles?
  - Depends on how many lines and circles exist ( $n$ )
  - Remember for Prob. HT value of  $K$  is an exponential function of the number of expected lines or circles
  - With a small number of curves  $K$  is small, and Prob. HT is faster, large number of curves  $K$  is large and HT is faster
- For curves like ellipse Prob. HT is only choice