Imperial College London

8 – Feature Detection

Prof Peter YK Cheung

Dyson School of Design Engineering

URL: www.ee.ic.ac.uk/pcheung/teaching/DE4_DVS/
E-mail: p.cheung@imperial.ac.uk

What is meant by "discontinuity"?

- Discontinuity in intensity is normally identified by the 1st order and 2nd order derivatives (lecture 5 slides 13, 14).
- We use central difference to compute the 1st order derivative as:

$$\frac{\partial f(x)}{\partial x} = f'(x) = \frac{f(x+1) - f(x-1)}{2}$$

- The 2nd order derivative is given by:

$$\frac{\partial f^2(x)}{\partial x^2} = f''(x) = f(x+1) - 2f(x) + f(x-1)$$

- We rarely use 3rd order derivatives. Nevertheless, here it is just for information:

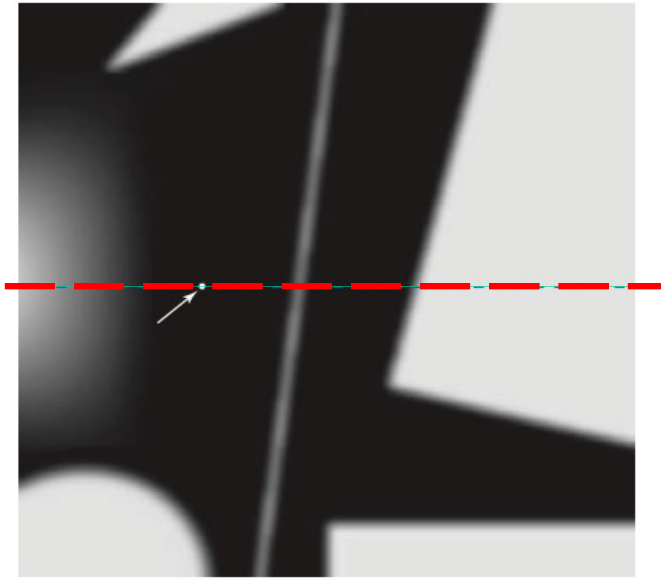
$$\frac{\partial f^3(x)}{\partial x^3} = f'''(x) = \frac{f(x+2) - 2f(x+1) + 2f(x-1) - f(x-2)}{2}$$

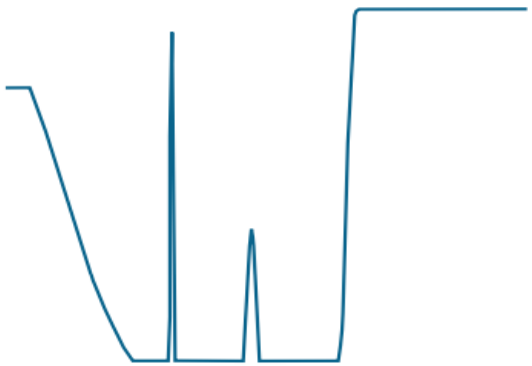
Digital Derivatives - coefficients

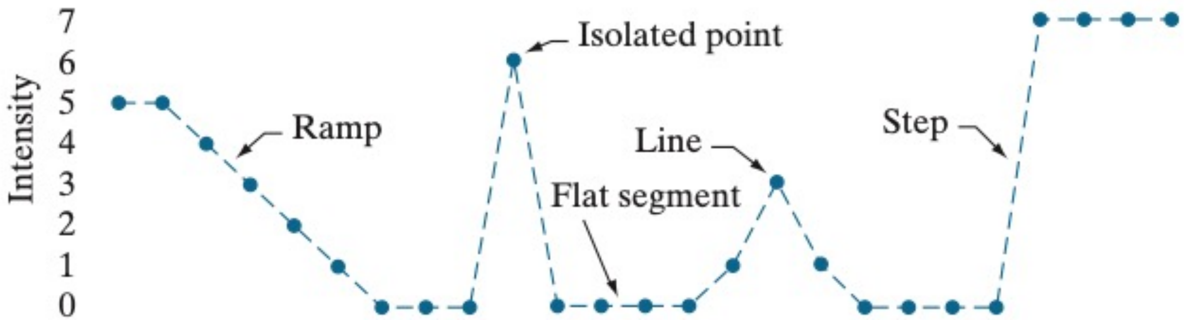
- To generalise, here is a table of the first four central digital derivatives coefficients:

	$f(x+2)$	$f(x+1)$	$f(x)$	$f(x-1)$	$f(x-2)$
$2f'(x)$		1	0	–1	
$f''(x)$		1	–2	1	
$2f'''(x)$	1	–2	0	2	–1
$f''''(x)$	1	–4	6	–4	1

Cross section of an image & derivatives







Intensity values	5	5	4	3	2	1	0	0	0	6	0	0	0	0	1	3	1	0	0	0	0	7	7	7	7	·	·
First derivative		−1	−1	−1	−1	−1	0	0	6	−6	0	0	0	1	2	−2	−1	0	0	0	7	0	0	0			
Second derivative		−1	0	0	0	0	1	0	6	−12	6	0	0	1	1	−4	1	1	0	0	7	−7	0	0			

Detection of Isolated point

- The obvious approach is to perform spatial filtering with a kernel that compute the 2nd order derivative (also called the Laplacian):

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{dx^2} + \frac{\partial^2 f}{dy^2}$$

$$= f(x+1,y) + f(x-1,y) + f(x,y+1) + f(x,y-1) - 4f(x,y)$$

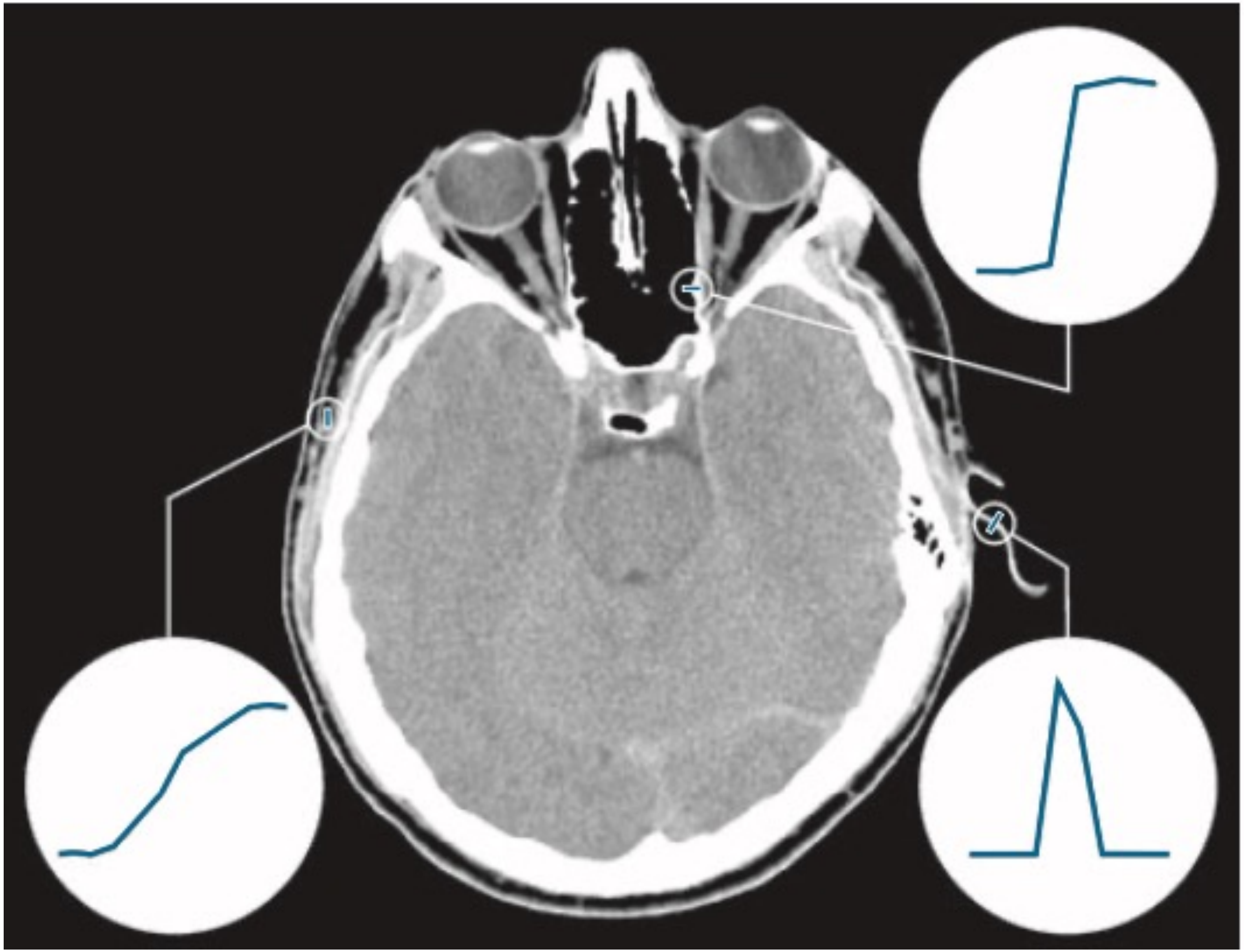
- This is equivalent to performing convolution with the filter kernel, but negate the output:

0	-1	0
-1	4	-1
0	-1	0

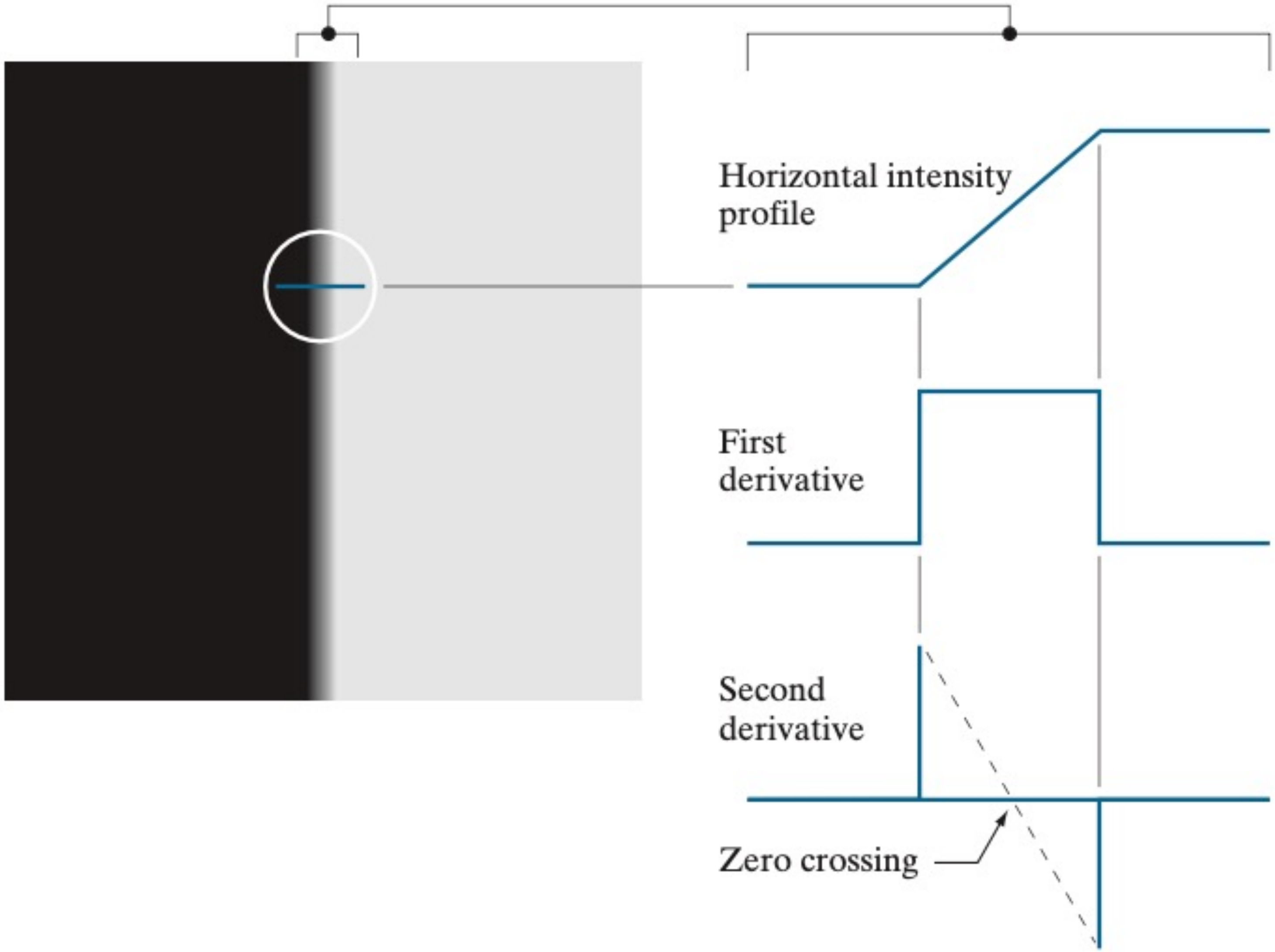
or

-1	-1	-1
-1	8	-1
-1	-1	-1

Three types of edges



Edge detection using derivatives



Sobel Edge detector kernels

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

−1	−2	−1
0	0	0
1	2	1

$$g_x = \frac{\partial f}{dx} = (w_7 + 2w_8 + w_9) - (w_1 + 2w_2 + w_3)$$

−1	0	1
−2	0	2
−1	0	1

$$g_y = \frac{\partial f}{dy} = (w_3 + 2w_6 + w_9) - (w_1 + 2w_4 + w_7)$$

Laplacian of a Gaussian (LoG) edge detector (1)

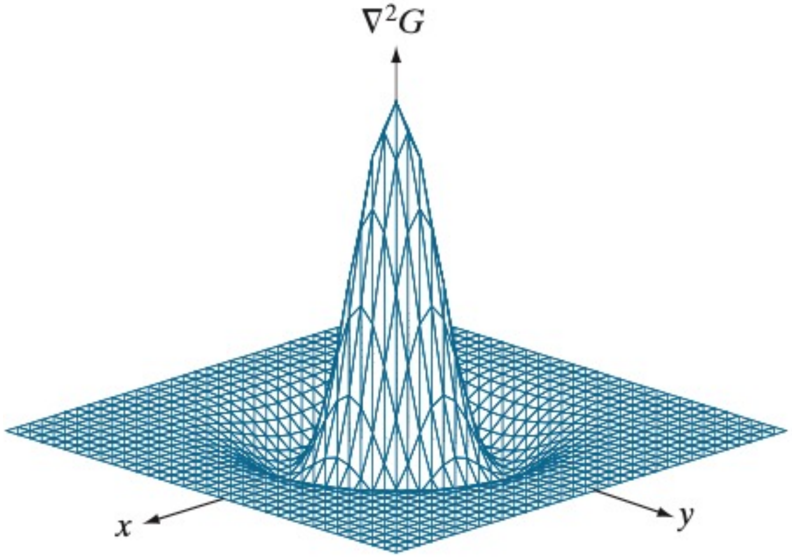
- Marr-Hildreth proposed an edge detector which is has two properties:
  1. Compute 1st or 2nd derivative at every point in the image
  2. Capable of being “tuned” to any scale or size
- The operator they proposed is the **Laplacian** (or 2nd derivative) **of** a **Gaussian** function.
- A 2D Gaussian function is defined as:

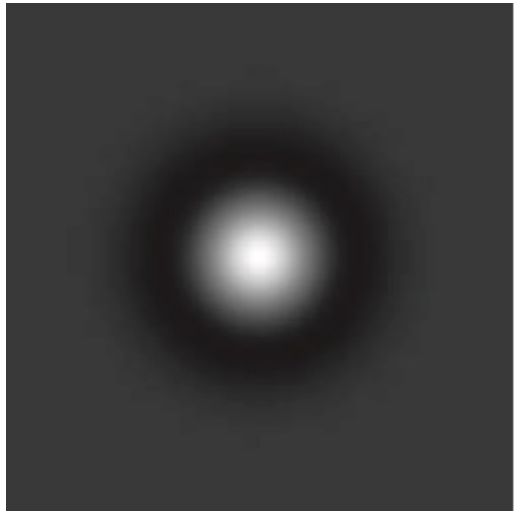
$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

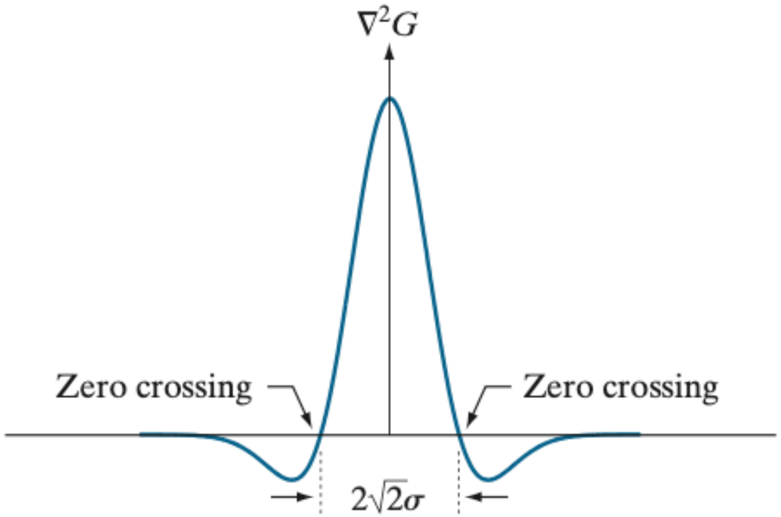
- The Laplacian of a Gaussian (LoG) is defined as:

$$\nabla^2 G(x,y) = \left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right) e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Laplacian of a Gaussian (LoG) edge detector (2)







0	0	−1	0	0
0	−1	−2	−1	0
−1	−2	16	−2	−1
0	−1	−2	−1	0
0	0	−1	0	0

Steps of the LoG algorithm

- The LoG algorithm includes these steps:

1. Convolving the LoG kernel with the image: $g(x,y) = [\nabla^2 G(x,y)] \star f(x,y)$.
2. Find the zero crossing of $g(x,y)$ to find the locations of edges in $f(x,y)$

- Since both the Laplacian and convolution operations are linear, we get:

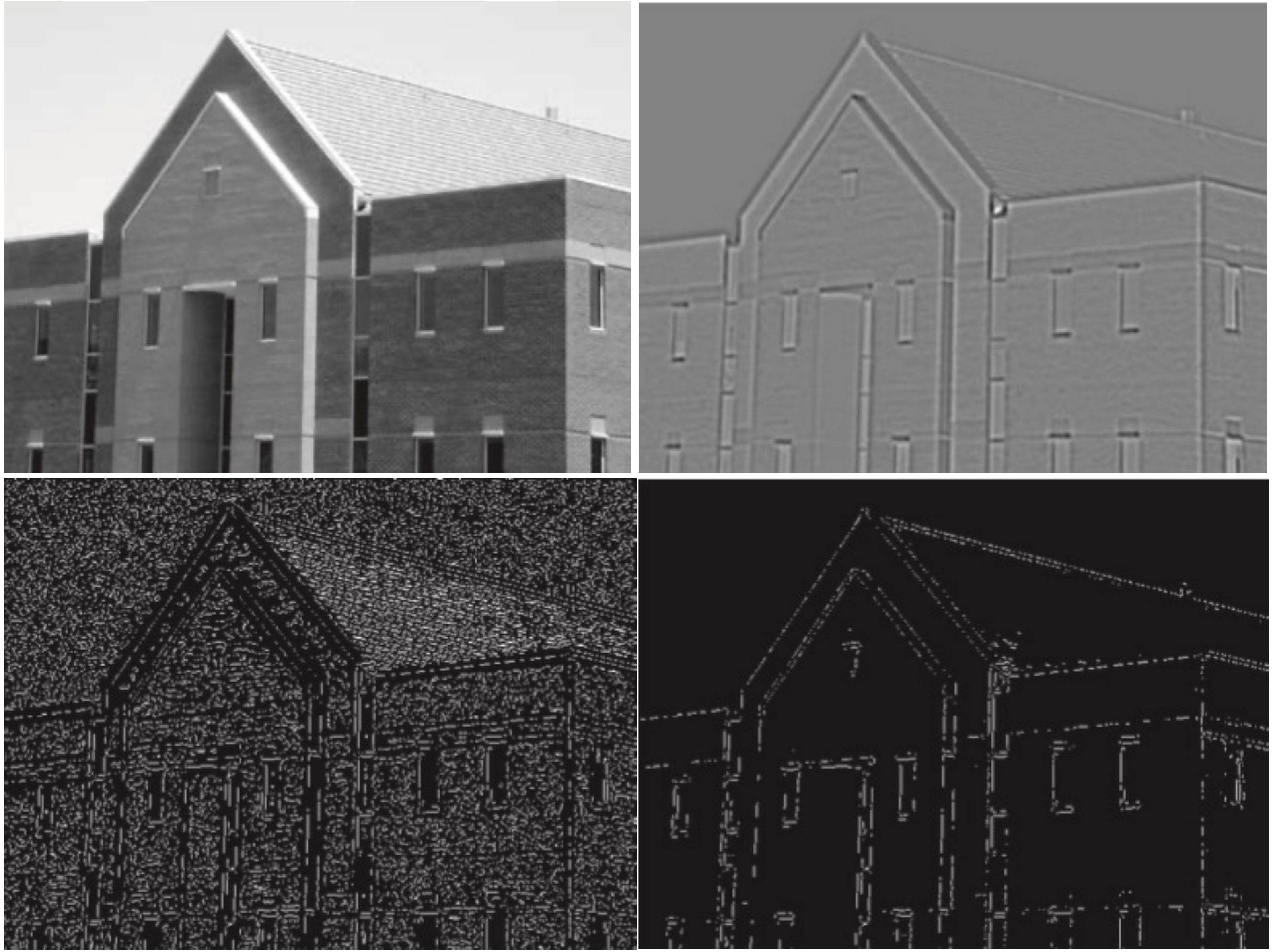
$$g(x,y) = [\nabla^2 G(x,y)] \star f(x,y) = \nabla^2 [G(x,y) \star f(x,y)]$$

- This implies that we can achieve the same results by:

1. Smooth the image with a Gaussian filter using convolution.
2. Compute the Laplacian of the results.
3. Find the zero crossing of the output of the Laplacian.

Example of using LoG for edge detection

$f(x,y)$



LoG output
$g(x,y)$

Zero crossing threshold = 0

Zero crossing threshold = 4% of max intensity of g

The Canny Edge Detector

- The Canny edge detector is based on three objectives:
  1. Low error rate: find all edges with no false and spurious results.
  2. Well localized edge points: location of edge points actually on edges.
  3. Single edge point response: return only one point for each true edge point.

- To achieve these objectives, Canny detector applies five steps:
  1. Apply **Gaussian filter** to smooth the image, thus removing noise.
  2. Find the intensity gradients of the filter image (i.e. 1^{st} derivative), including both the **gradient magnitude** and **direction**.
  3. Apply **non-maximum suppression** to thin the edges.
  4. Apply **double threshold** to determine potential edges.
  5. Using **hysteresis method**, follow the strong edge points to produce the final definitive edge.

Canny Detector – Step 2: Gradient Magnitude & Direction

- Step 1: The filtering the image $f(x, y)$ with a Gaussian filter is similar to that of LoG edge detector. It removes noise from the image. The result is $f_s(x, y)$.

- Step 2: Compute the gradient at each pixel. Need to compute BOTH magnitude and direction:

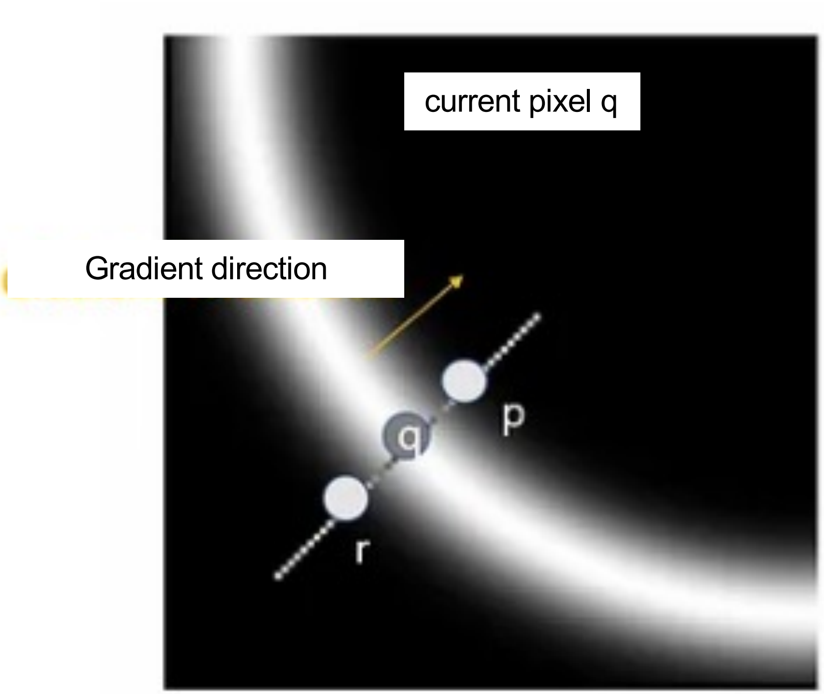
$$M_s(x, y) = \sqrt{\left(\frac{\partial f_s(x)}{\partial x}\right)^2 + \left(\frac{\partial f_s(y)}{\partial y}\right)^2}$$

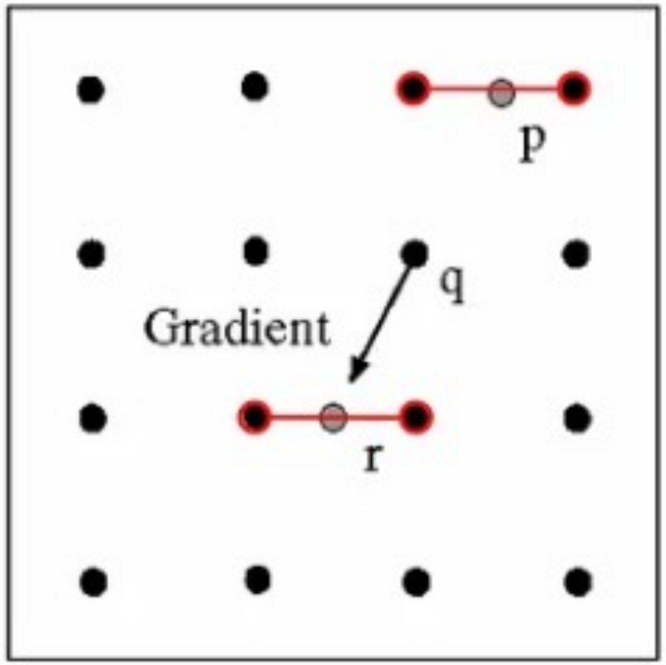
$$\alpha(x, y) = \tan^{-1}\left[\frac{\partial f_s(y)/\partial y}{\partial f_s(x)/\partial x}\right]$$

- Angle quantized to one of four directions: horizontal $(0°)$, vertical $(90°)$ and the two diagonals $(45°, 135°)$.

Canny Detector – Step 3: Non-Maximum Suppression

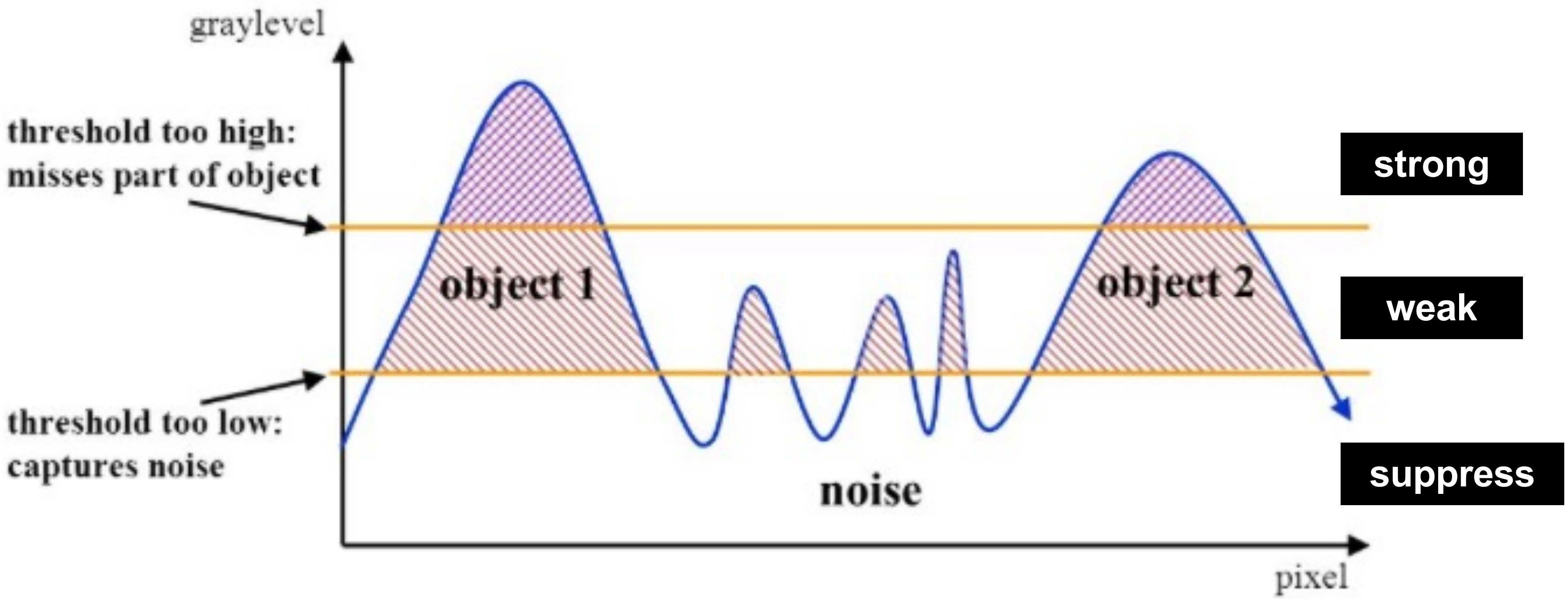
- Step 3 of Canny is to use an edge thinning method to combat the smoothing effect of Gaussian smoothing.
  1. Compare intensity at q with neighbours along gradient direction p and r.
  2. Since q is maximum, set p and r to zero.
  3. Repeat for all pixels.





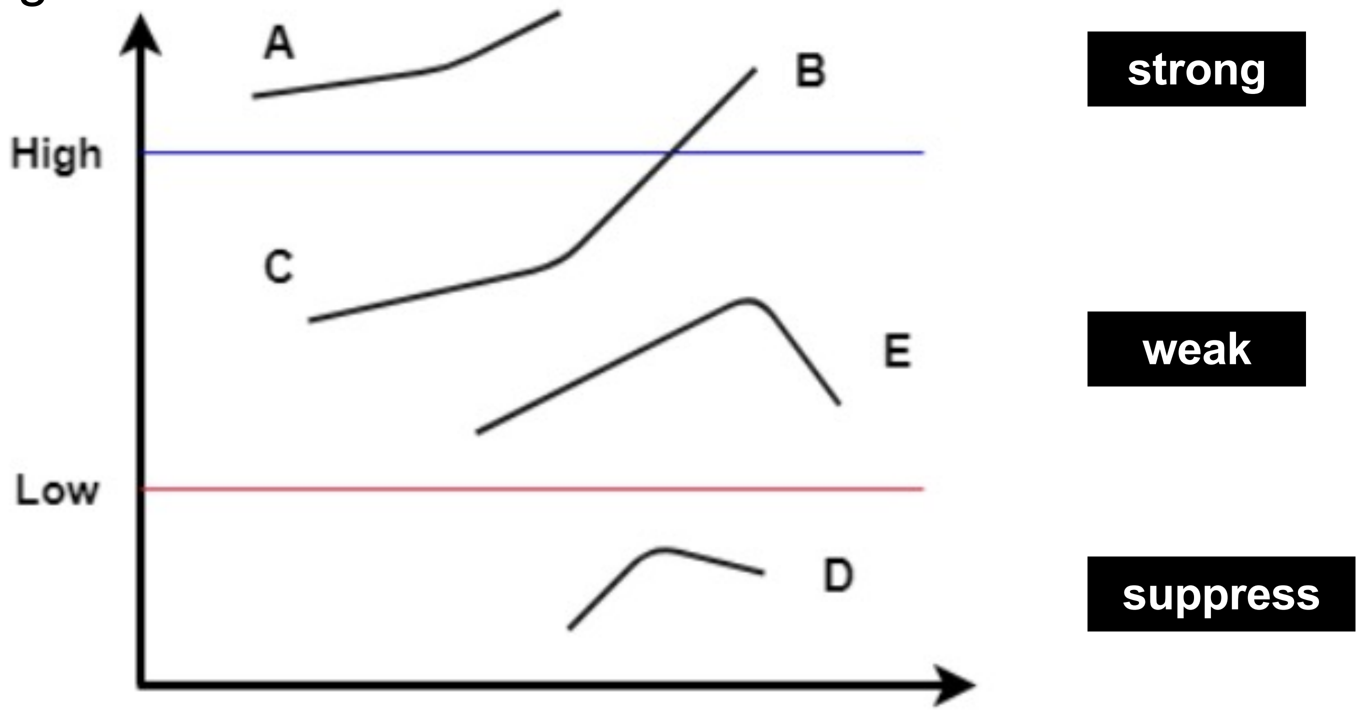
Canny Detector – Steps 4: Double thresholding

- Noise can produce false edges even after Step 3.
- Use higher and lower thresholds to categorize each pixel.
- Gradient magnitude > high threshold → strong pixel.
- Low threshold ≤ Gradient magnitude ≤ high threshold → weak pixel.
- Gradient magnitude < low threshold → suppress pixel.



Canny Detector – Steps 5: Edge tracking by Hysteresis

- Finally, edge is tracked by its neighbourhood connections (hysteresis).
- A pixels are all strong. So A must be an edge.
- D pixels are all suppressed and therefore are not considered in Step 5.
- E pixels are all weak and none of their neighbours are strong – suppress.
- B is strong, but C is weak. However, C pixels are neighbour to strong, so reclassified as strong.

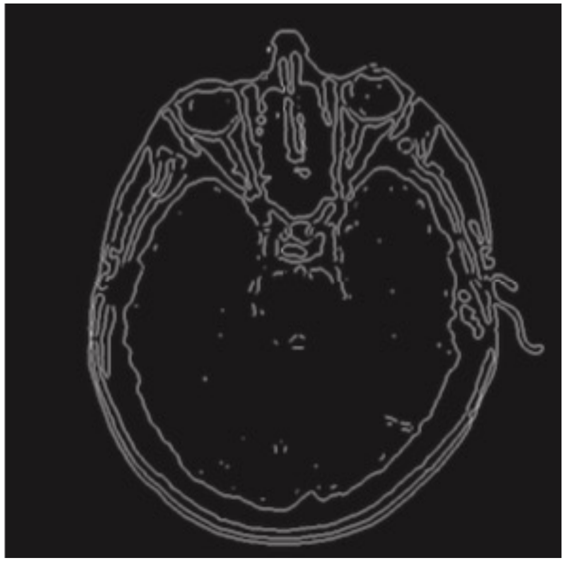


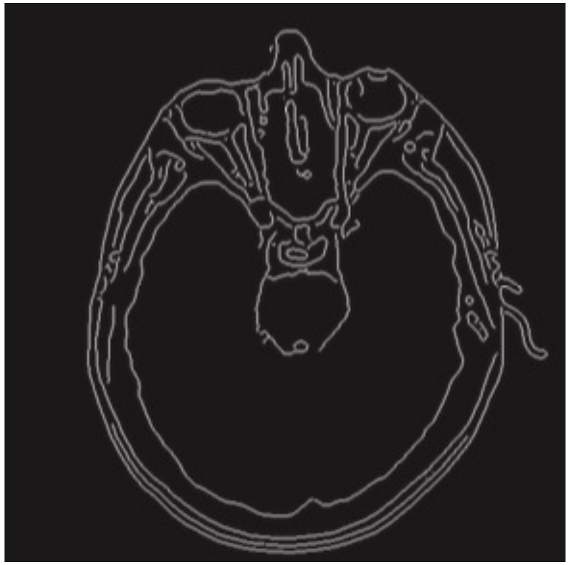
Compare Canny with other edge detection methods

$f(x, y)$

Edge detection with smoothing (5x5 square kernel) then thresholding

Edge detection with LoG method





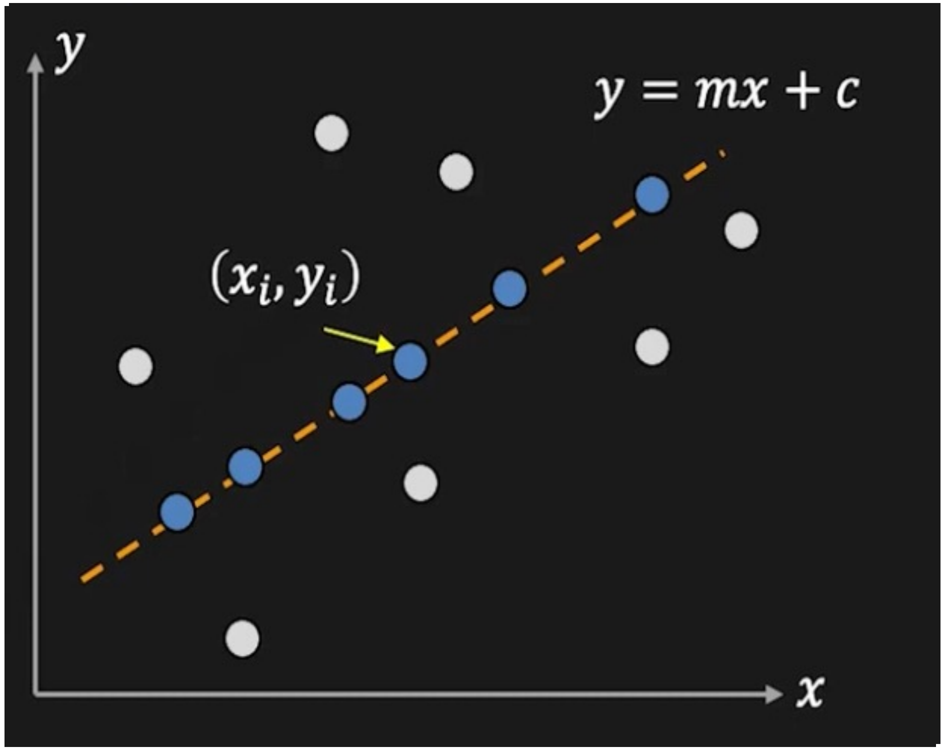
Edge detection with Canny method

Comparison of Edge detection methods

Method	Pros	Cons
Sobel	Simple; detect edges and the orientations	Sensitive to noise; inaccurate
Laplacian + zero crossing	Detect edges and direction; isotropic	Sensitive to noise; interaction between nearby edges
Laplacian of Gaussian (LoG)	Correct places of edges; handle different areas and scales	Malfunction at curves and corners; cannot find orientations
Canny	Low error rate; good localization; accurate; not sensitive to noise	More complex; sometimes produce false zero crossings

The Hough Transform – Basic Idea

- Previous method detected edge points (x_i, y_i) as shown here.
- How to detect line $y = mx + c$?

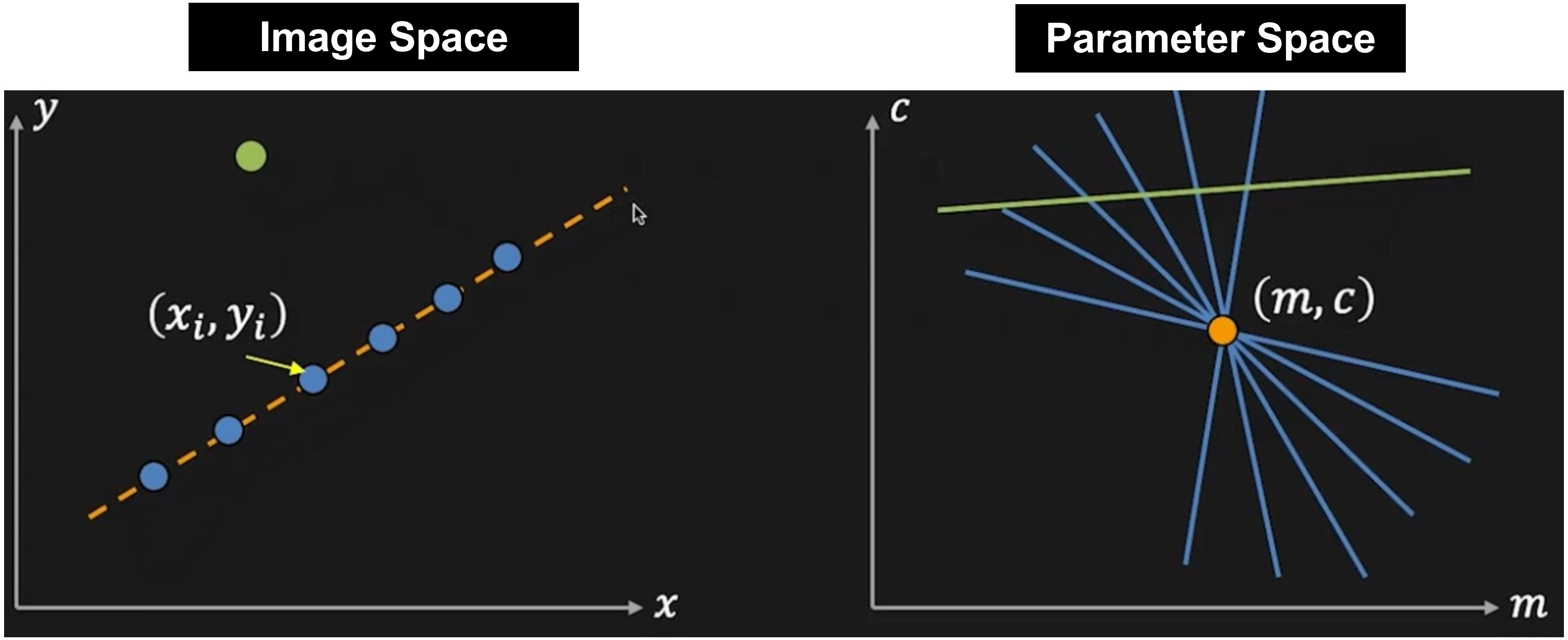


- Consider the point (x_i, y_i). Its equation is given by:

Parameter space

$$y_i = mx_i + c \quad \Longleftrightarrow \quad c = -mx_i + y_i$$

The Hough Transform – Image Space vs Parameter Space



$$y_i = mx_i + c \quad \longleftrightarrow \quad c = -mx_i + y_i$$

- All lines through edge point (x_i, y_i) maps onto the blue line in the parameter space.
- Another point on the line in image space maps to another line in the parameter space but intersect at the same (m, c) values.
- Now map a few more points on the edge line, will result in same intersection.

The Hough Transform – Line Detection Algorithm

- Step 1: Quantize parameter space (m, c).

- Step 2: Create a counting array $H(m, c)$.

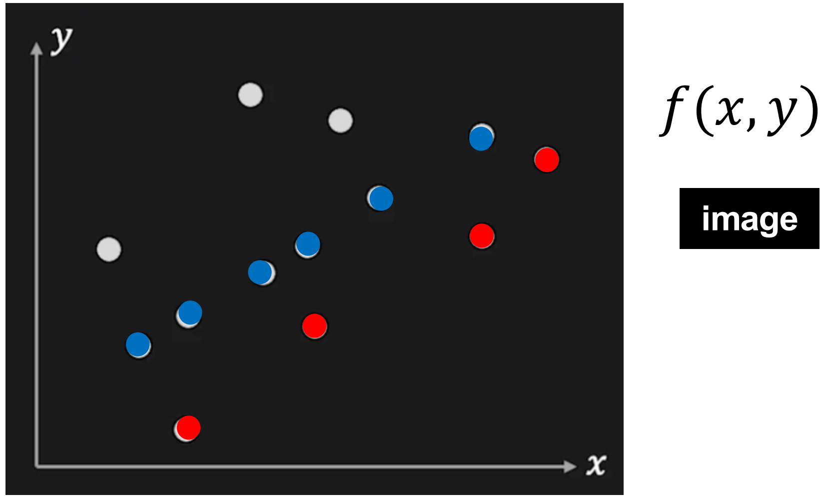
- Step 3: Set $H(m, c) = 0$ for all (m, c).

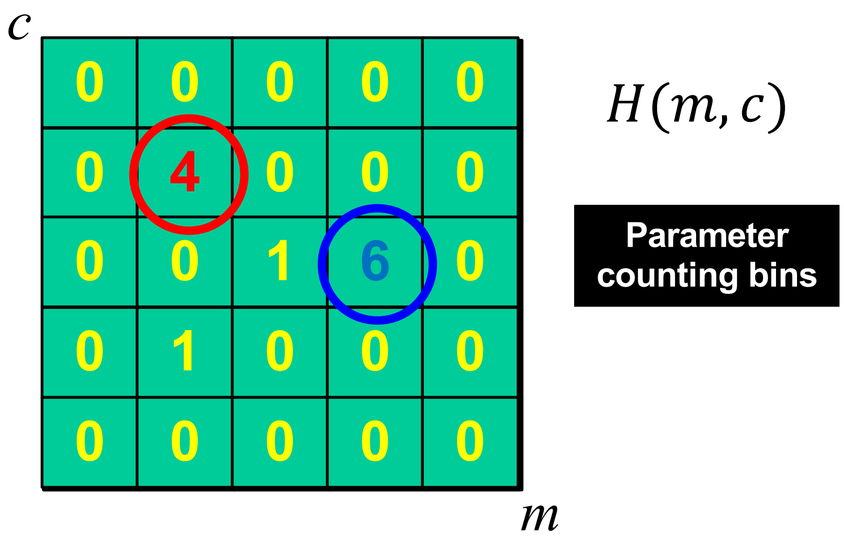
- Step 4: For each **edge point** (x_i, y_i)

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

  So for all points on the straight line, this increase the count at (m, c).

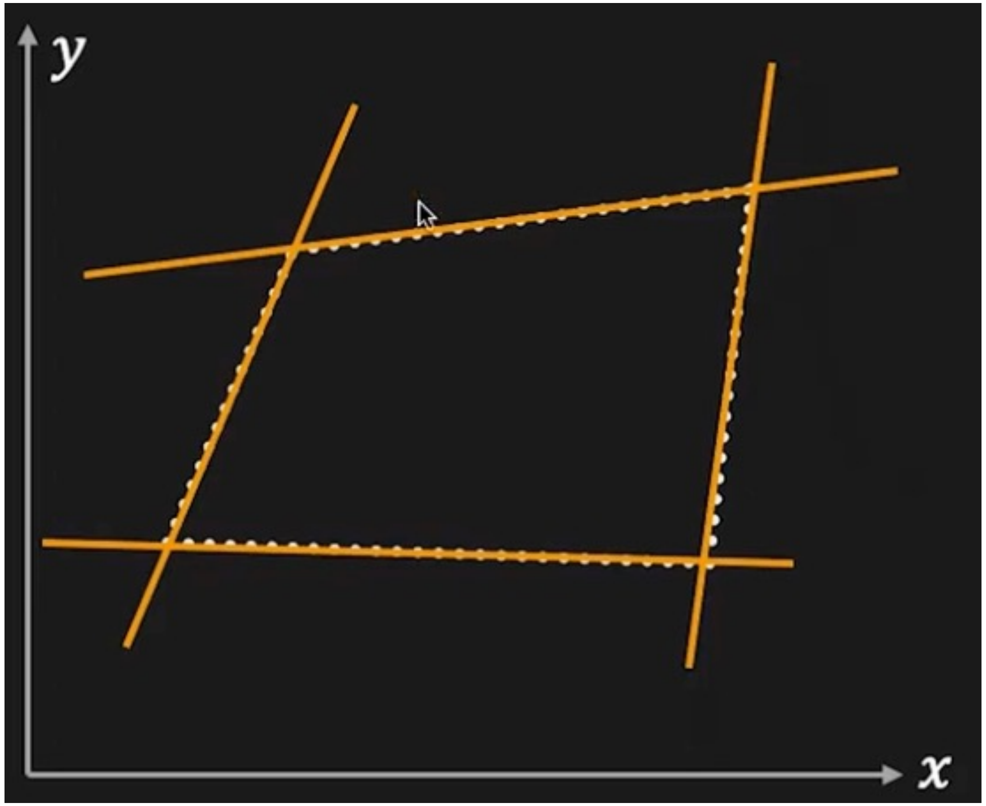
- Step 5: Identify the local maxima in $H(m, c)$.



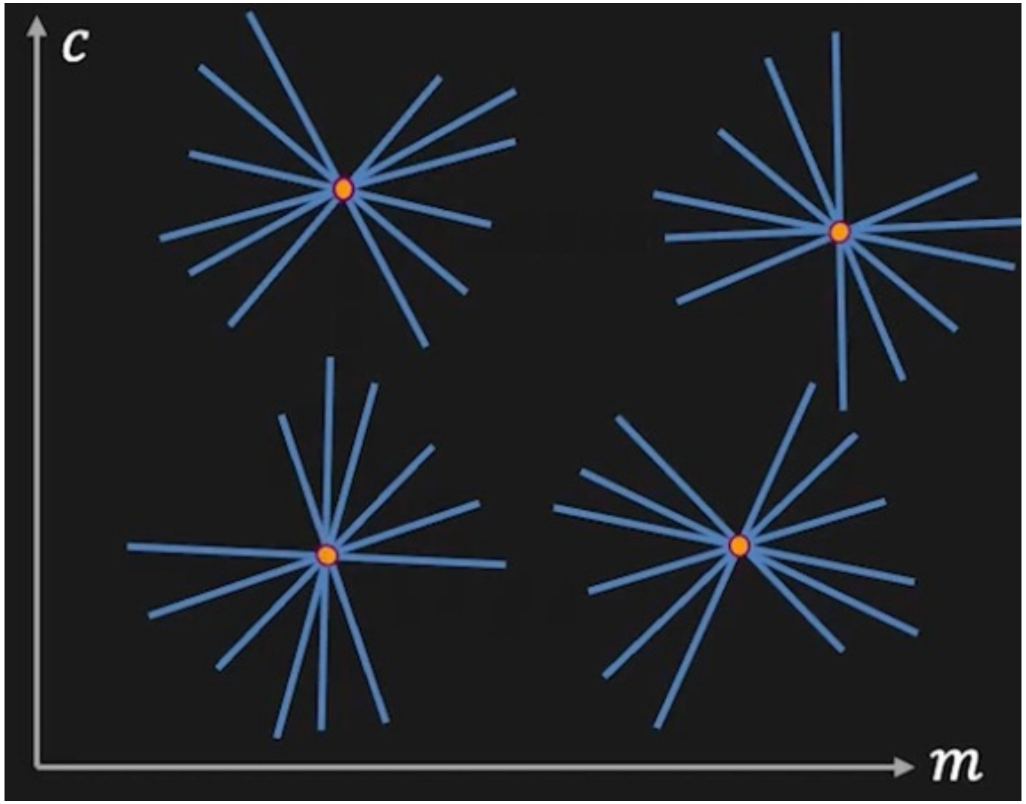


The Hough Transform – Multiple line detection

Image Space



Parameter Space



The Hough Transform – Better Parameters

- Problem with using (m, c) parameter space.
- Range of slope of line is huge:

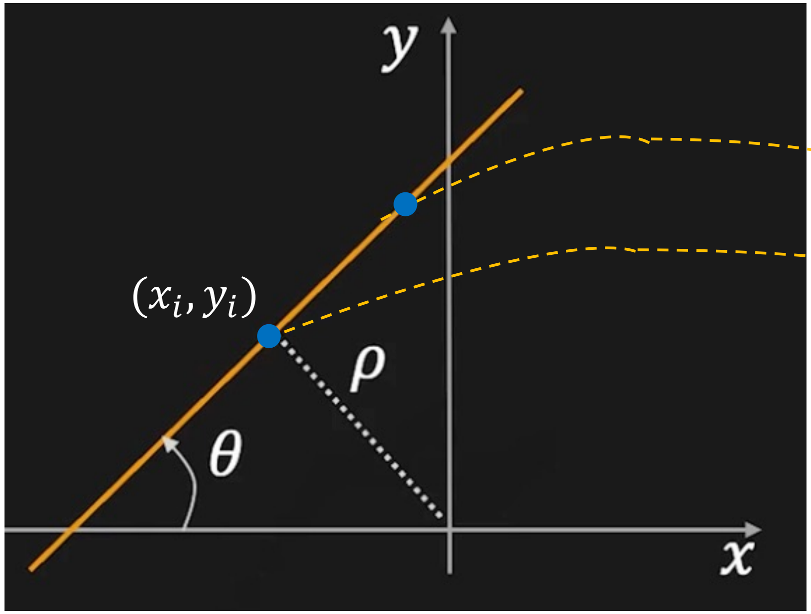
$$-\infty \leq m \leq \infty$$

- Not viable for limited size of counting array.
- **Solution**: do not map line to $y = mx + c$
- Instead use a different equation for a line:

$$x \sin\theta + y \cos\theta + \rho = 0$$

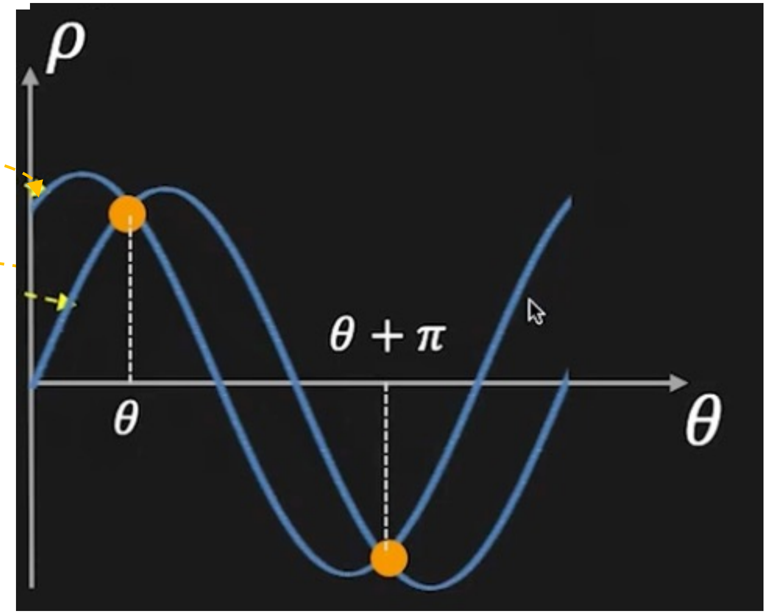
- The orientation θ is finite: $0 \leq \theta \leq \pi$
- The distance ρ from origin is also finite.

Image Space



$$x \sin\theta + y \cos\theta + \rho = 0$$

Parameter Space



$$x \sin\theta + y \cos\theta + \rho = 0$$

The Hough Transform – Design consideration

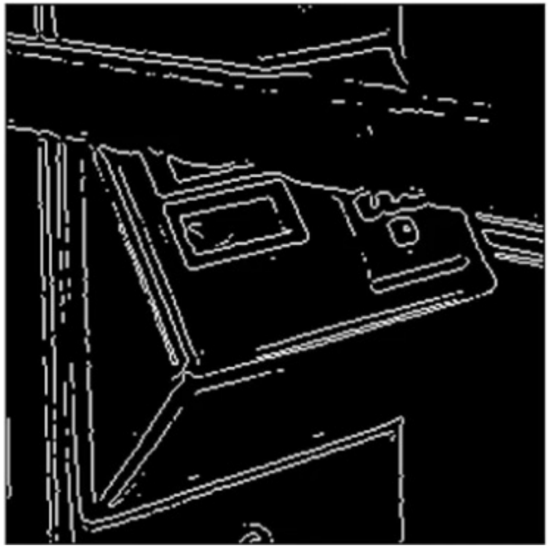
- What is the dimension of the parameter counting array?
  - Too many bins, noise will cause lines to be missed.
  - Too few bins, different lines will merge together.

- How many lines?
  - Count the peaks in the array (thresholding),

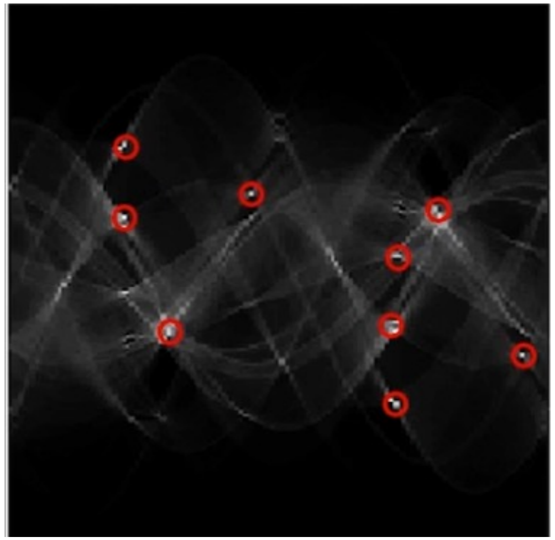
- How to handle inaccurate edge locations?
  - Increment nearby bins instead of just individual bins.

Example of Hough Transform in line detection

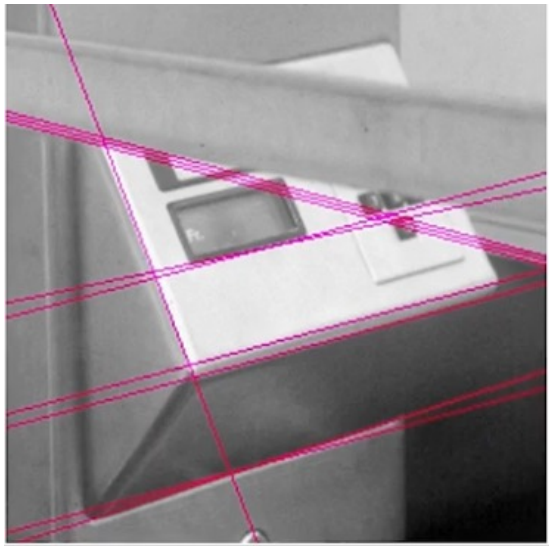
Input image



Detected edge



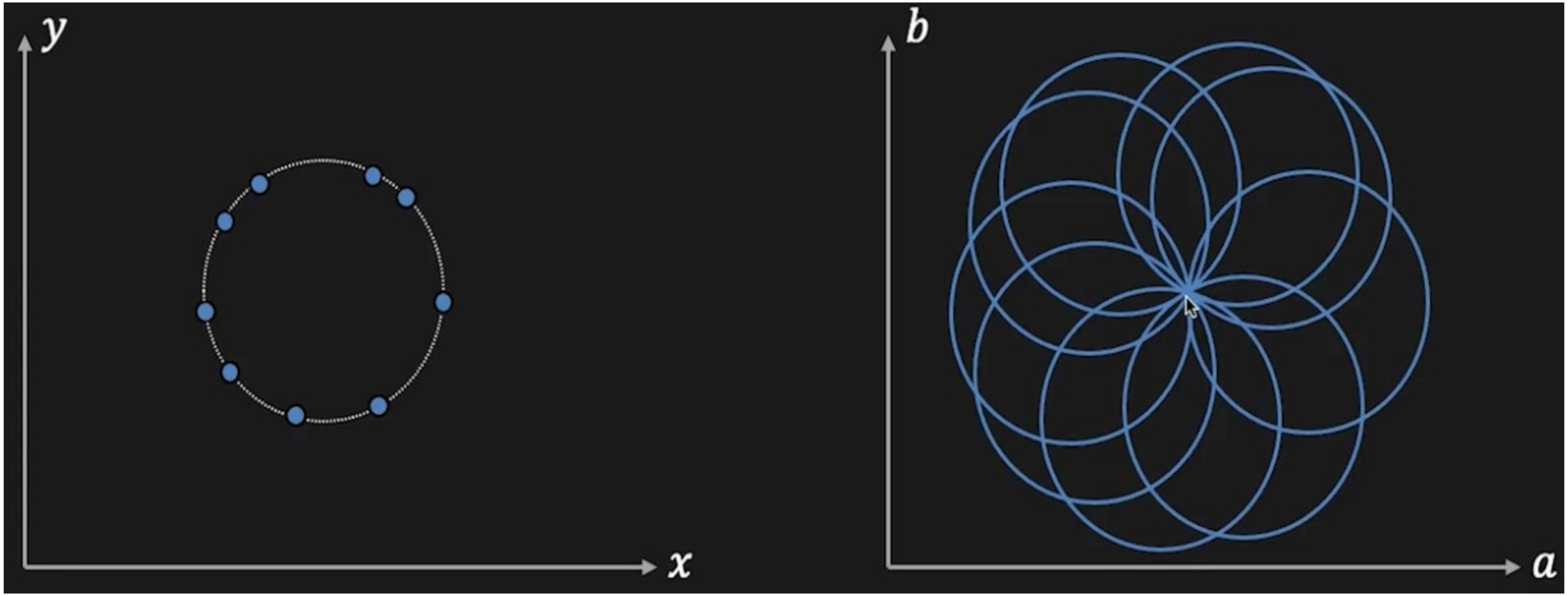
Hough Transform
$H(\theta, \rho)$



Detected lines

Hough Transform: Detection of Circle (known r)

Image Space | **Parameter Space**

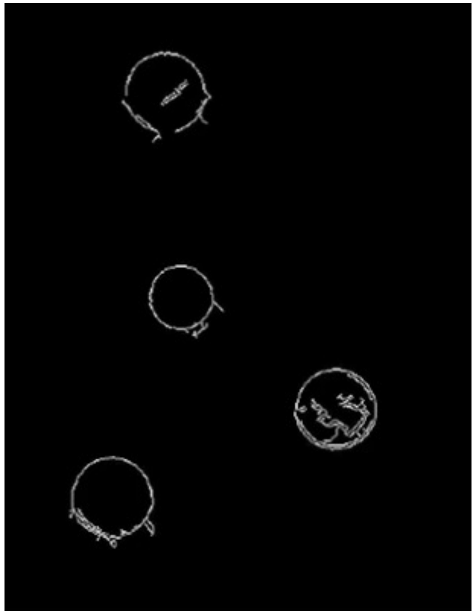


$(x_i - a)^2 + (y_i - b)^2 = r^2$ ⟷ $(a - x_i)^2 + (b - y_i)^2 = r^2$

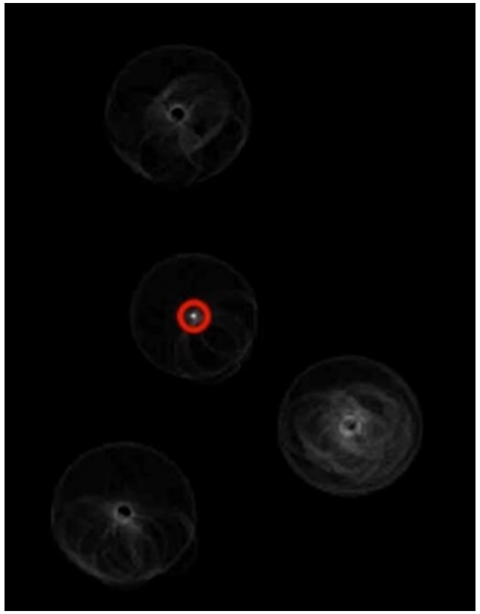
- For circle of known radius, and a give point (x_i, y_i).
- All circles through this point maps to a circle in the parameter space.
- The intersection of all edge points gives the parameter (a, b).

Hough Transform: Circle detection example

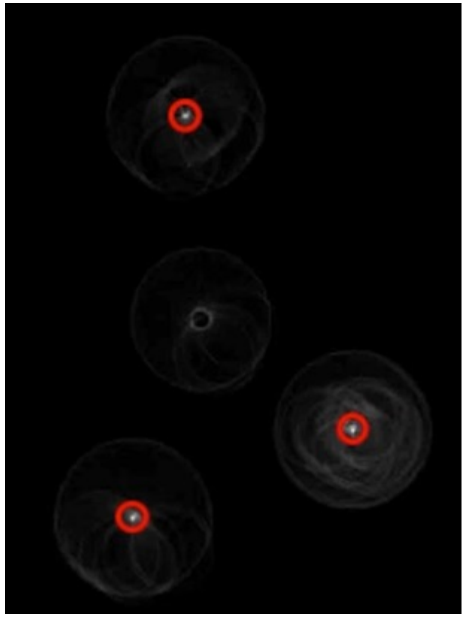
Image of coins



Edge points



Hough Transform $H_1(a, b)$ for penny ($r = r_1$)



Hough Transform $H_2(a, b)$ for quarters ($r = r_2$)