The National Higher School of Artificial Intelligence
المدرسة الوطنية العليا للذكاء الاصطناعي

Course
« *Computer Vision»*

Sid-Ahmed Berrani

2024-2025

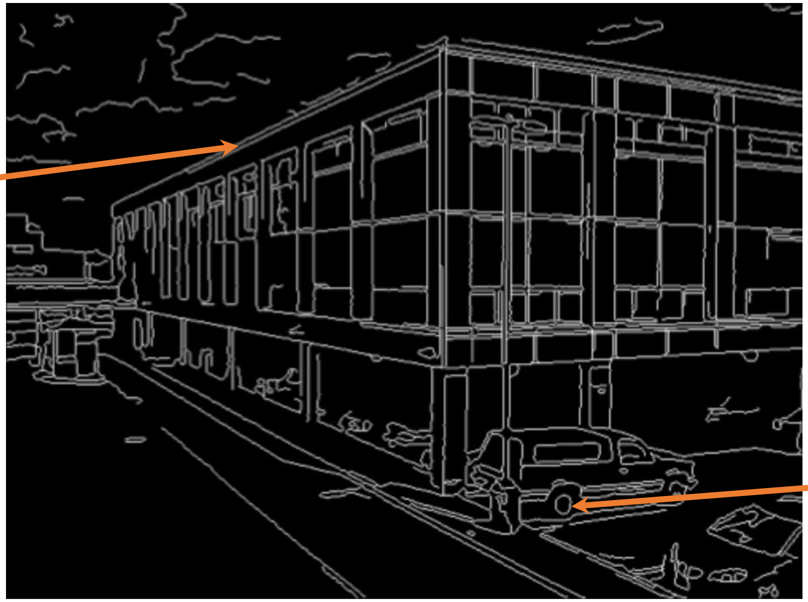
VI. Line/curve boundary detection

Basically:

We have a set of edges in an image **=>** fit a line, a circle or any geometrical shape.

Fitting is the process to decompose an image or a set of tokens (i.e. pixels, isolated points, sets of edge points...) into components that belong to circles, lines or any other shape.

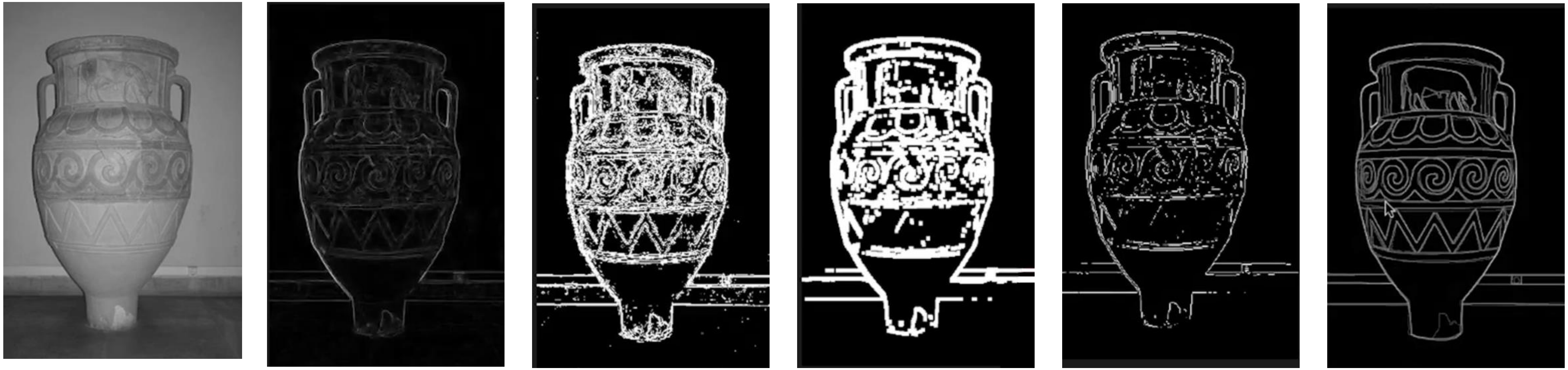
A line



A circle

VI. Line/curve boundary detection

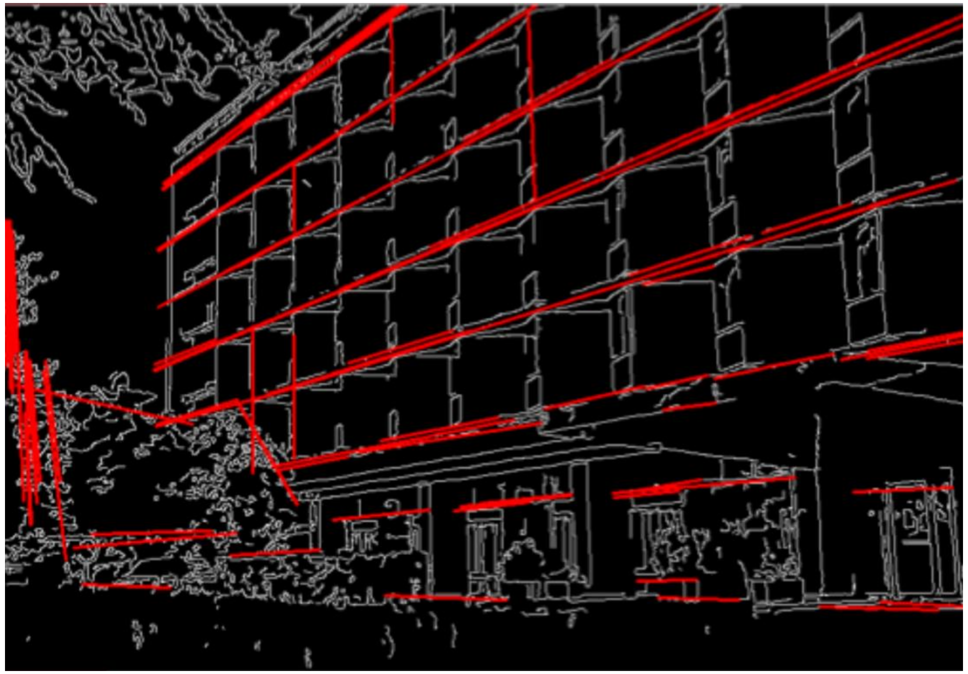
Edge post-processing



VI. Line/curve boundary detection

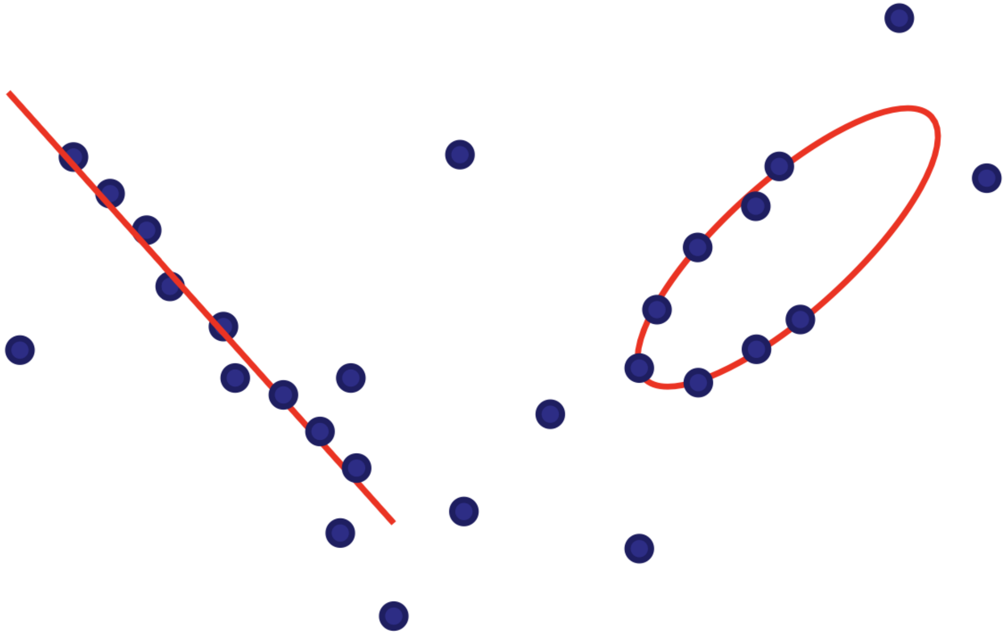
What is it used for?

- Image segmentation: produce compact representations that emphasize the relevant image structures.
- Image understanding.
- Analyzing and measuring man made objects (e.g. as part of quality insurance process).



VI. Line/curve boundary detection

⇒ **Fitting** involves determining what possible curves could have given rise to a set of tokens observed in an image.

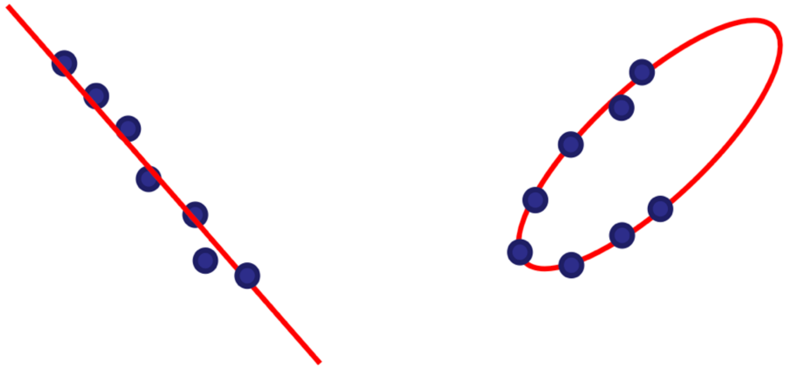


VI. Line/curve boundary detection

Fitting involves determining what possible curves could have given rise to a set of tokens observed in an image.

⇒ **Many sub-problems:**

1. Parameter estimation: if we already know the association between tokens and curves. We need to recover the parameters of each curve.

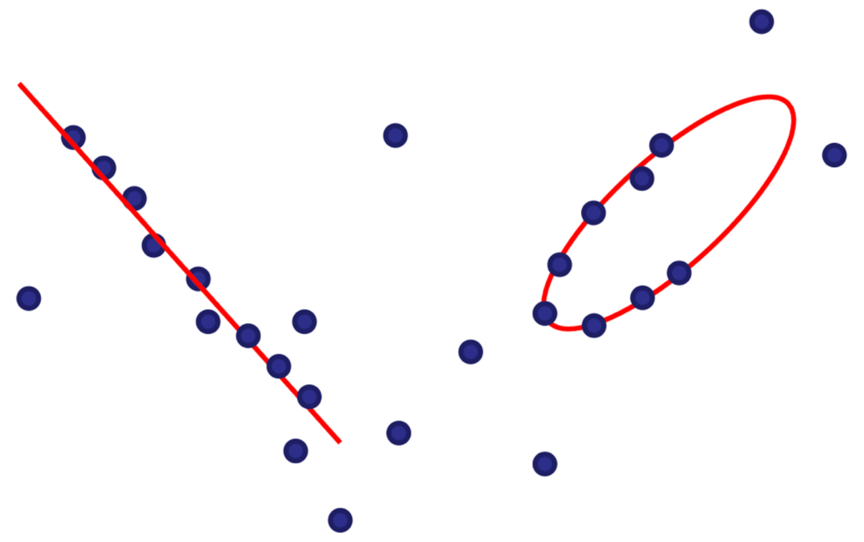


VI. Line/curve boundary detection

Fitting involves determining what possible curves could have given rise to a set of tokens observed in an image.

⇒ **Many sub-problems:**

2. Token-curve association: we assume to know only how many curves are present but not which token came from which curve. The association must be solved together with parameter estimation.



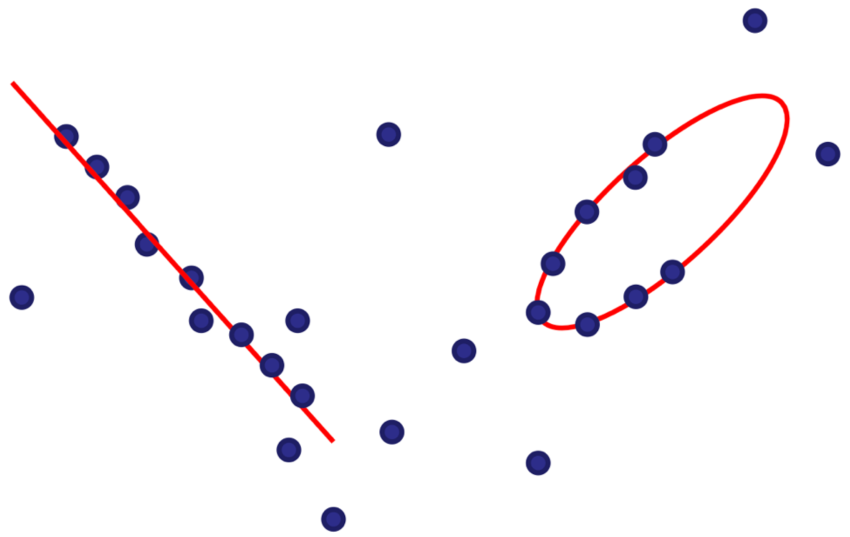
VI. Line/curve boundary detection

Fitting involves determining what possible curves could have given rise to a set of tokens observed in an image.

⇒ **Many sub-problems:**

3. Counting: we have no prior knowledge on the data, so we must figure out:

(i) How many curves are present,
(ii) The association between tokens and curves,
(iii) Curve parameters.



VI. Line/curve boundary detection

Parameter estimation:

- We have observed a set of points generated by a certain curve model with unknown parameters.

Goal: Find the best set of parameters that justify the observations.

Two approaches:

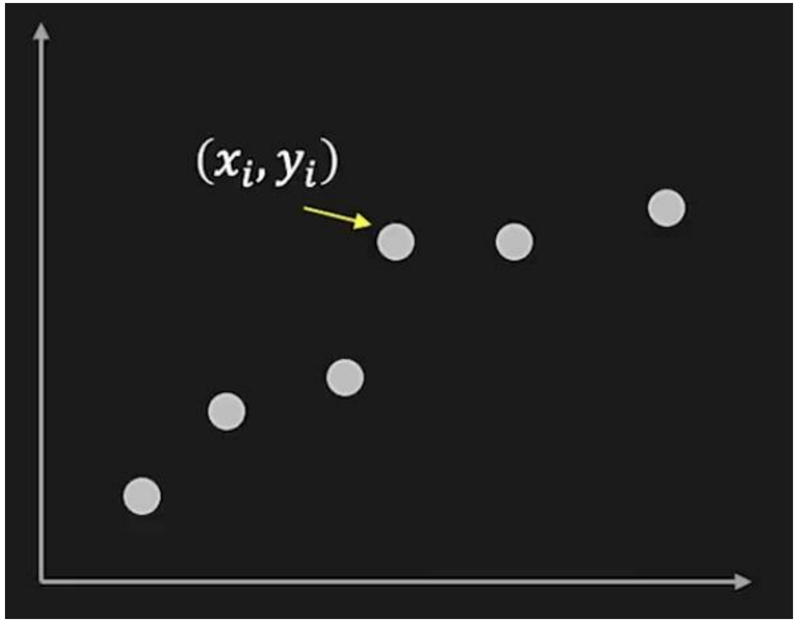
1. Minimize a loss function accounting for the distances between each point and the curve.
2. Describe the curve as a generative model and find the best parameters maximizing the probability of generating the observed data.

In some cases (like 2D lines) the two approaches yield to the same result

VI. Line/curve boundary detection

Let's start with fitting lines to edges…

Edges: points (x_i, y_i)

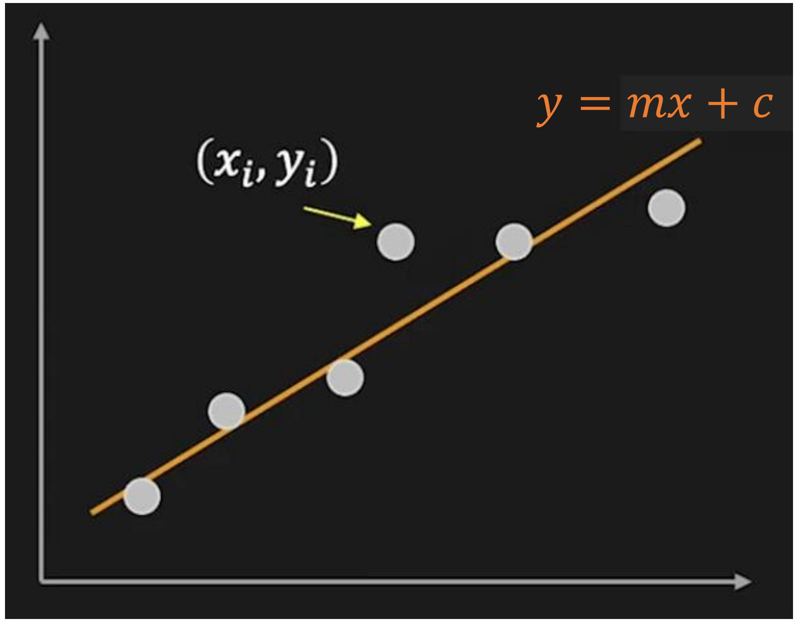


VI. Line/curve boundary detection

Let's start with fitting lines to edges…

Edges: points (x_i, y_i)

Goal: find m and c.

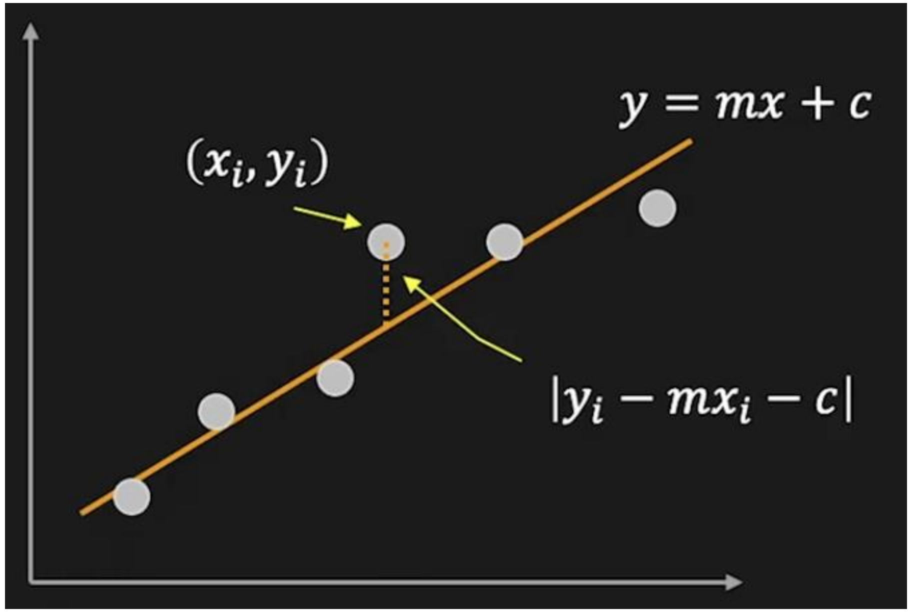


VI. Line/curve boundary detection

Let's start with fitting lines to edges...

Edges: points (x_i, y_i)

Goal: find m and c.



The idea:

Minimize average squared **vertical** distances.

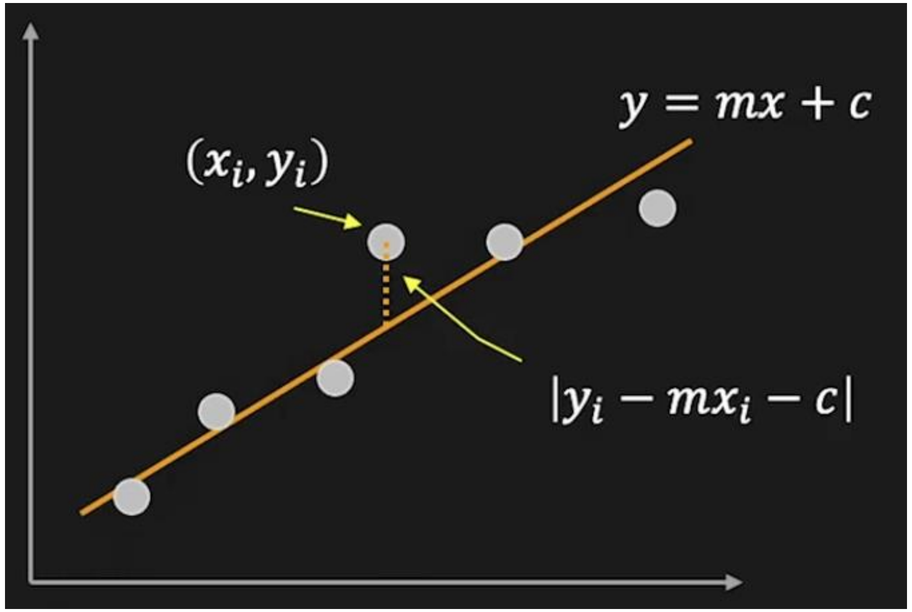
$$E = \frac{1}{N}\sum_i (y_i - mx_i - c)^2$$

The solution: least square method

$$\frac{\partial E}{\partial m} = \frac{-2}{N}\sum_i x_i(y_i - mx_i - c) \qquad \frac{\partial E}{\partial c} = \frac{-2}{N}\sum_i (y_i - mx_i - c)$$

VI. Line/curve boundary detection

Let's start with fitting lines to edges…



The idea:

Minimize average squared **vertical** distances.

$$E = \frac{1}{N}\sum_i (y_i - mx_i - c)^2$$

The solution: least square method

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

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

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

Where $\bar{x} = \frac{1}{N}\sum_i x_i \qquad \bar{y} = \frac{1}{N}\sum_i y_i$

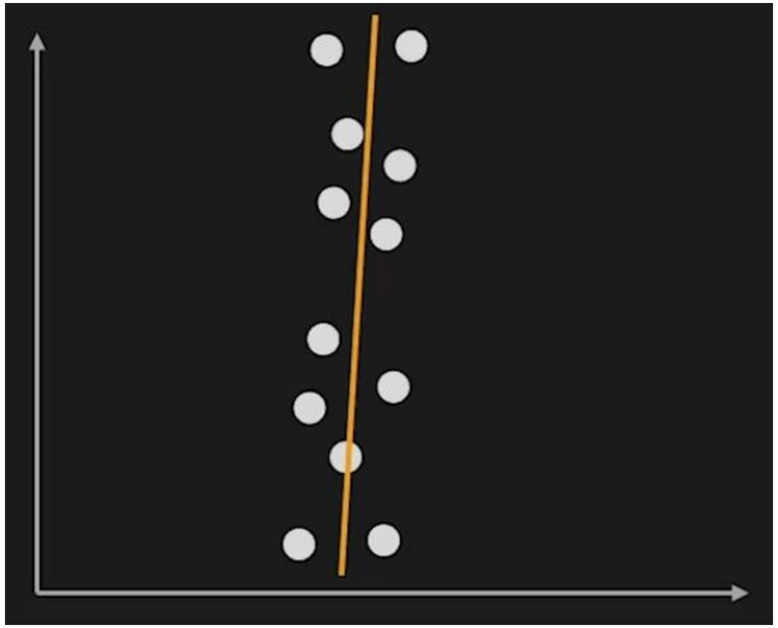
VI. Line/curve boundary detection

Let's start with fitting lines to edges…

The idea:

Minimize average squared **vertical** distances.

$$E = \frac{1}{N}\sum_i (y_i - mx_i - c)^2$$



The solution: least square method

The problem:

Vertical lines.

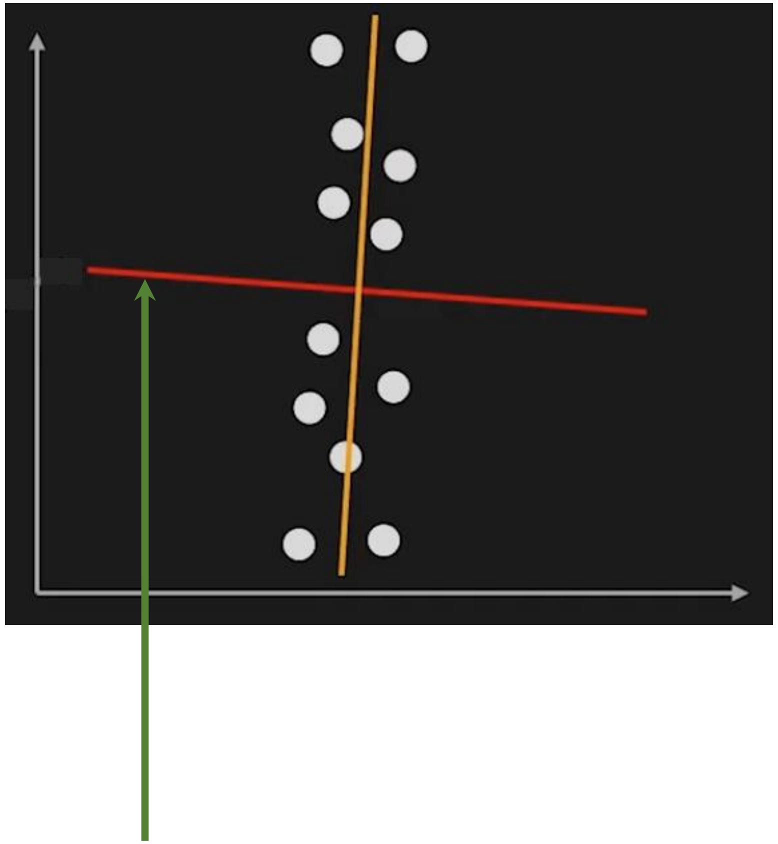
VI. Line/curve boundary detection

Let's start with fitting lines to edges…

The idea:

Minimize average squared **vertical** distances.

$$E = \frac{1}{N}\sum_i (y_i - mx_i - c)^2$$



The line that minimizes E

The solution: least square method

The problem:

Vertical lines.

An alternative solution:

Minimizing the perpendicular distance.

VI. Line/curve boundary detection

Fitting lines to edges

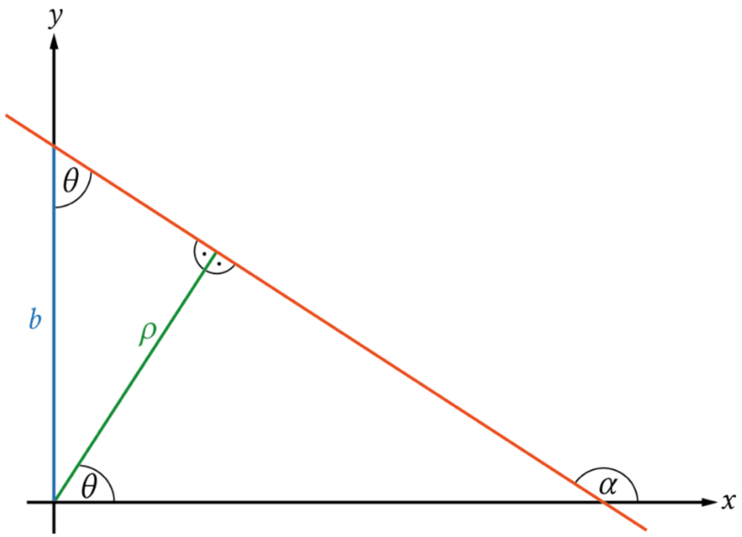
Minimizing the perpendicular distance

Switch to polar form:

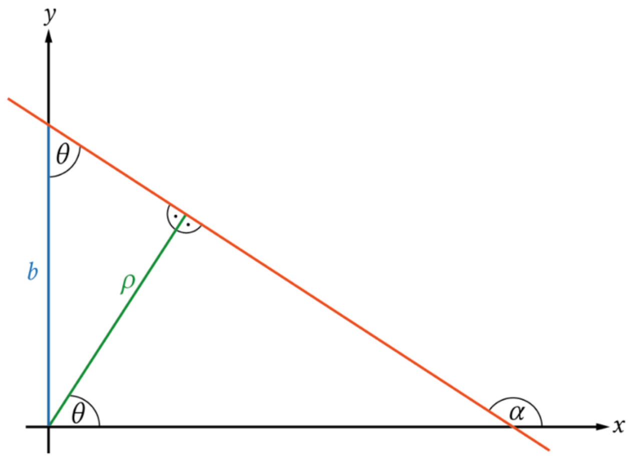
$$x \cos\theta + y \sin\theta = \rho$$

- θ: angle between the x-axis and the line's normal vector
- ρ: distance from the origin to the line along the normal

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$



VI. Line/curve boundary detection



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

Fitting lines to edges

Minimizing the perpendicular distance

The **signed distance** from a point (x_i, y_i) to the line is:

$$d_i = x_i \cos\theta + y_i \sin\theta - \rho$$

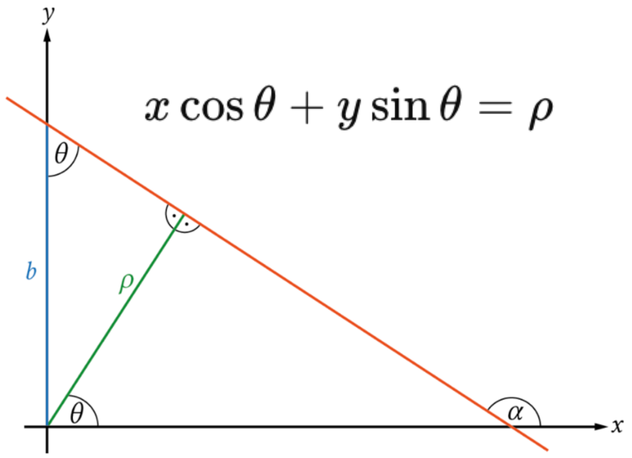
$\Rightarrow$ Minimize the **sum of squared distances**:

$$E(\theta, \rho) = \sum_{i=1}^{n} (x_i \cos\theta + y_i \sin\theta - \rho)^2$$

$\Rightarrow$ Solution:

- Perform a 2D PCA over the points (after centering them)
- The normal direction of the line is the eigenvector of the **smallest eigenvalue of the** covariance matrix from the centered data

VI. Line/curve boundary detection



$$E(\theta,\rho) = \sum_{i=1}^{n} (x_i\cos\theta + y_i\sin\theta - \rho)^2$$

Fitting lines to edges

Minimizing the perpendicular distance

$\Rightarrow$ Solution: 2D PCA

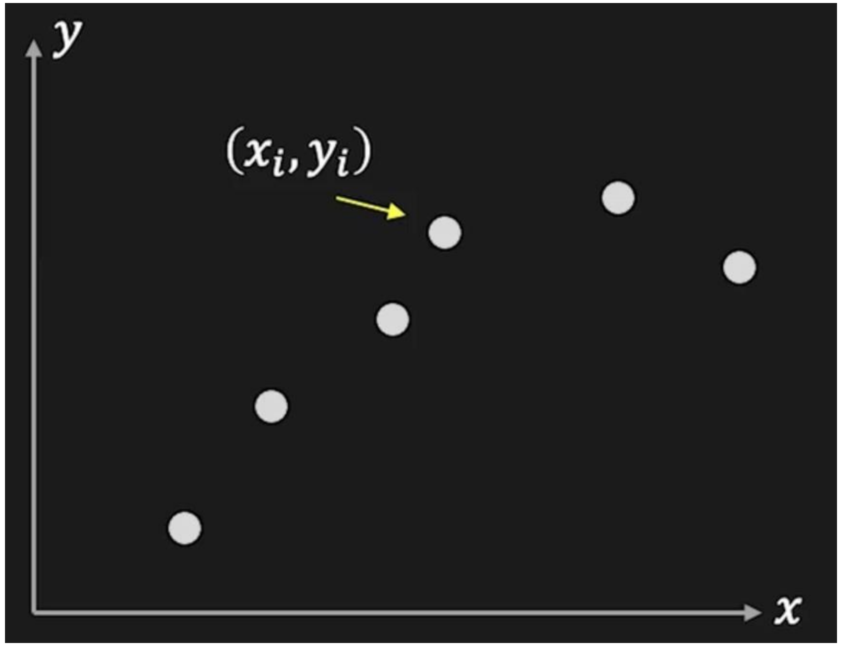
1. Compute the mean $(\bar{x}, \bar{y})$
2. Center the data: $x_i' = x_i - \bar{x}$, $y_i' = y_i - \bar{y}$
3. Form the 2×2 covariance matrix from the centered data
4. Compute eigenvectors/values — pick the eigenvector of the **smallest eigenvalue** as the normal direction
5. Compute $\rho = \bar{x}\cos\theta + \bar{y}\sin\theta$

VI. Line/curve boundary detection

Fitting curves to edges

Edges: points (x_i, y_i)

Goal: find a polynomial that best fits the edge points.



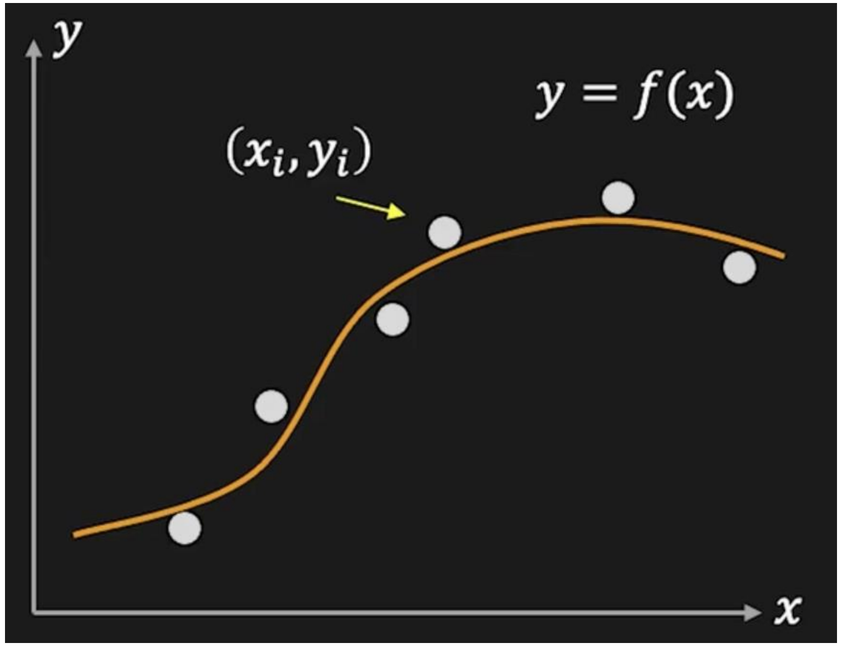
VI. Line/curve boundary detection

Fitting curves to edges

Edges: points (x_i, y_i)

Goal: find a polynomial that best fits the edge points.

$$y = f(x) = ax^3 + bx^2 + cx + d$$



$\Rightarrow$ Minimize:

$$E = \frac{1}{N}\sum_i \left(y_i - ax_i^3 - bx_i^2 - cx_i - d\right)^2$$

$\Rightarrow$ Solve the linear system using the least square method:

$$\frac{\partial E}{\partial a} = 0; \quad \frac{\partial E}{\partial b} = 0; \quad \frac{\partial E}{\partial c} = 0; \quad \frac{\partial E}{\partial d} = 0;$$

VI. Line/curve boundary detection

Fitting curves to edges

Edges: points (x_i, y_i)

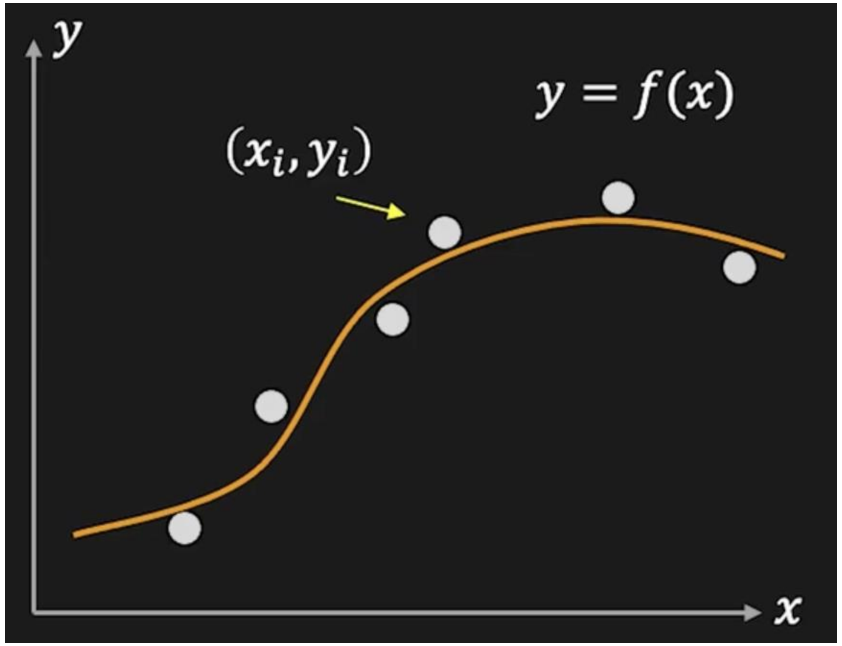
Goal: find a polynomial that best fits the edge points with a more general solution

$$y_0 = ax_0^3 + bx_0^2 + cx_0 + d$$

$$y_1 = ax_1^3 + bx_1^2 + cx_1 + d$$

$$\dots$$

$$y_n = ax_n^3 + bx_n^2 + cx_n + d$$



VI. Line/curve boundary detection

Fitting curves to edges

Edges: points (x_i, y_i)

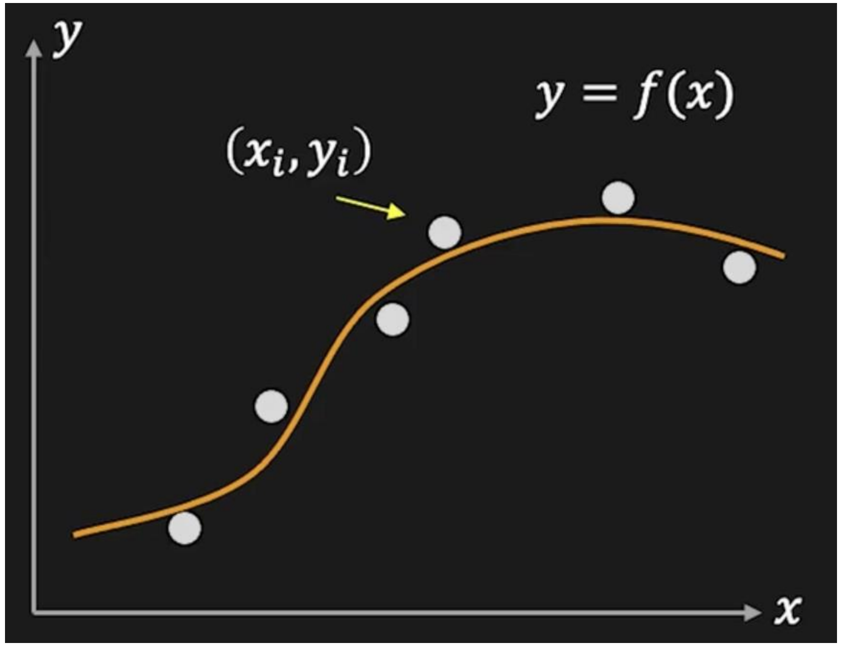
Goal: find a polynomial that best fits the edge points with a more general solution

$$y_0 = ax_0^3 + bx_0^2 + cx_0 + d$$

$$y_1 = ax_1^3 + bx_1^2 + cx_1 + d$$

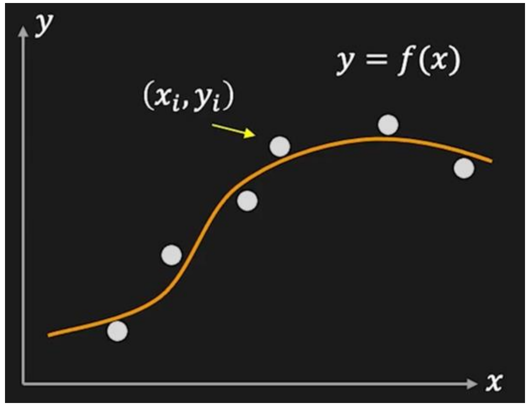
$$\dots$$

$$y_n = ax_n^3 + bx_n^2 + cx_n + d$$



$\Rightarrow$ Given many (x_i, y_i), this is an over-determined linear system with 4 unknowns (a, b, c, d).

VI. Line/curve boundary detection



$\Rightarrow$ Solving a linear system

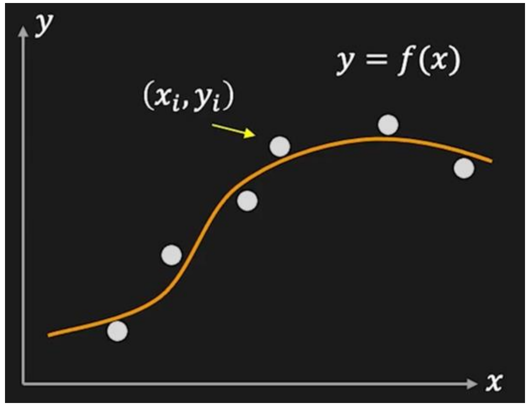
An over-determined linear system with m unknowns $\{a_j\}$ where $j = 0 \ldots m$ and n observations $\{(x_{i0}, \ldots, x_{im}, y_i)\}$ where $i = 0 \ldots n$ can be written in a matrix form:

$$\begin{bmatrix} x_{00} & x_{01} & \cdots & x_{0m} \\ x_{10} & x_{11} & \cdots & x_{1m} \\ \vdots & \vdots & \vdots & \vdots \\ x_{n0} & x_{n1} & \cdots & x_{nm} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix} \quad \Rightarrow \quad Xa = y$$

$X_{n \times m}$: Known $\qquad$ $a_{m \times 1}$: Unknown $\qquad$ $y_{n \times 1}$: Known

$\Rightarrow$ $X_{n \times m}$ is not necessarily a square matrix, thus it cannot be inverted.

VI. Line/curve boundary detection



$\Rightarrow$ Solving a linear system

We can use the least squares solution:

$$Xa = y$$

$$X^T X a = X^T y$$

$X^T X$ is a $m \times m$ square matrix

$$(X^T X)^{-1} X^T X\, a = (X^T X)^{-1} X^T y$$

$$a = (X^T X)^{-1} X^T y$$

$$X^+ = (X^T X)^{-1} X^T$$

(Pseudo inverse)

VI. Line/curve boundary detection

The main problem with boundary detection:

How to know which edges in an image actually correspond to the boundary we are looking for.

=> One solution: **The Hough transform**

Ref.: Hough, P. V. C. Method and means for recognizing complex patterns, U.S. Patent 3,069,654, Dec. 18, 1962.

VII. The Hough transform

The Hough transform was originally introduced by **Paul Hough** in 1962:

U.S. patent 3,069,654 titled "Method and Means for Recognizing Complex Patterns".

The method was later generalized by **Richard Duda and Peter Hart** in 1972 to detect arbitrary shapes, particularly lines, in digital images:

Use of the **Hough transformation** to **detect lines** and **curves** in pictures

RO Duda, PE Hart - Communications of the ACM, 1972 - dl.acm.org

… to the region of the 0-p plane near this **curve**. If we **find** a cell with count k near this **curve**, then we are assured that k figure points lie on a **line** passing (nearly) through the point (x0, y0). …

VII. The Hough transform

Definition and goal:

- The Hough Transform is a **feature extraction technique** used in image analysis, computer vision, and digital image processing.
- Its main goal is to detect **simple geometric shapes** (such as lines, circles, or ellipses) in an image.

VII. The Hough transform

Advantages over existing methods:

- **Robust to noise and gaps:** Unlike edge-following algorithms, the Hough Transform can detect lines even if the line is partially broken or noisy.
- **Simple mathematical formulation** for complex pattern recognition tasks.
- **Works in cluttered scenes**: Efficiently finds shapes even in the presence of many irrelevant edges.
- **Detects multiple instances** of the same shape in a single pass.

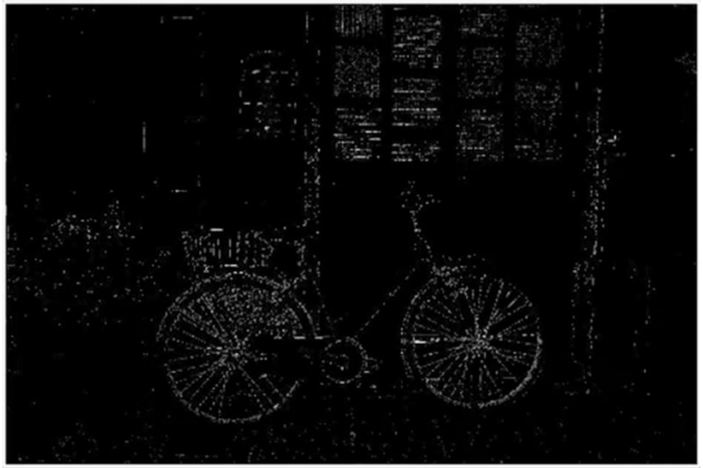
VII. The Hough transform

Applications:

- **Lane detection** in autonomous driving systems (detecting road boundaries or lanes).
- **Object recognition** where objects are approximated by geometric shapes.
- **Medical image analysis** (e.g., detecting circular features like tumors or bones).
- **Robotics** for identifying structured environments.
- **Industrial inspection** to detect regular shapes in manufactured parts.
- **Document image analysis** to detect lines, text baselines, or page segmentation.
- …

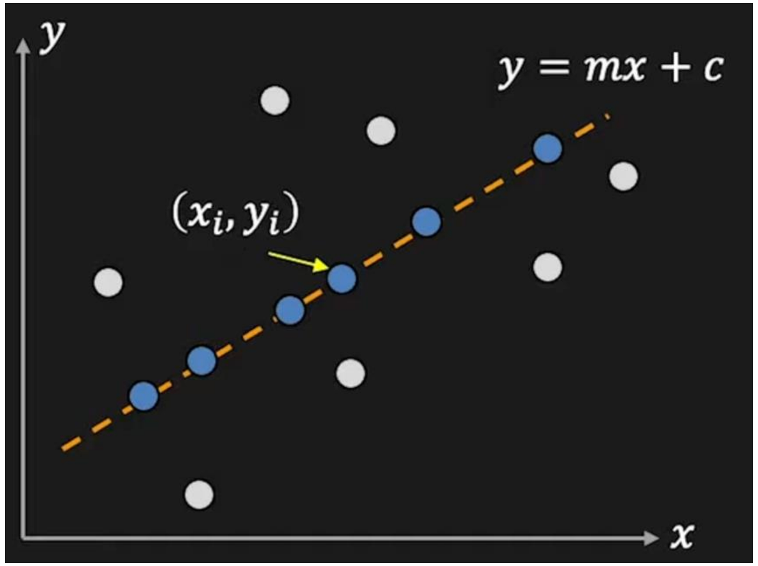
VII. The Hough transform

Challenges:



- Which data to fit to?
- Only part of the model is visible: data is incomplete
- Noise.

VII. The Hough transform



Let's start with the straight line…

Edges: points (x_i, y_i)

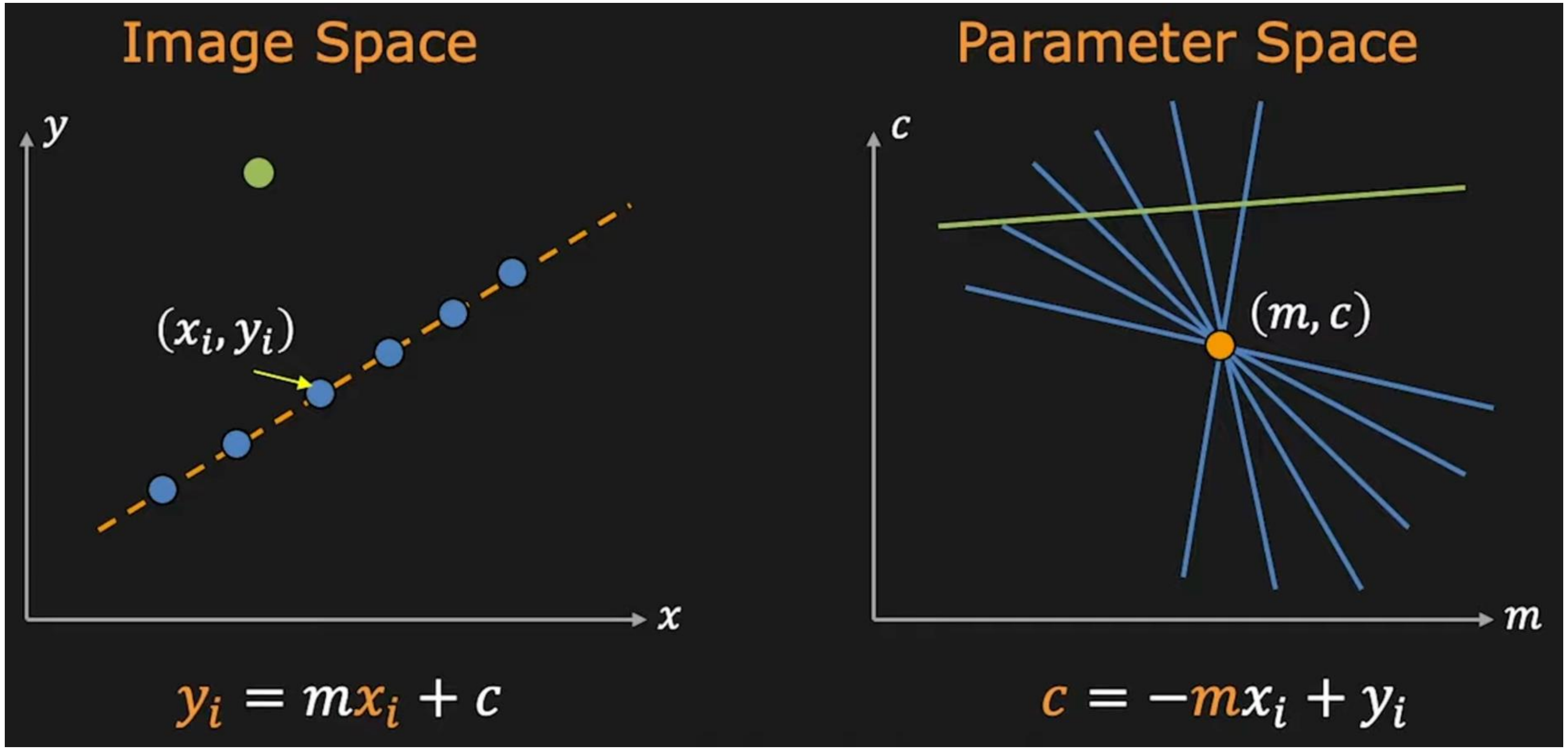
Task: detect the line $y = mx + c$

If we consider a point (x_i, y_i):

$$y_i = mx_i + c \qquad \Leftrightarrow \qquad c = -mx_i + y_i$$

VII. The Hough transform

The principle:



Point ⟷ **Line**

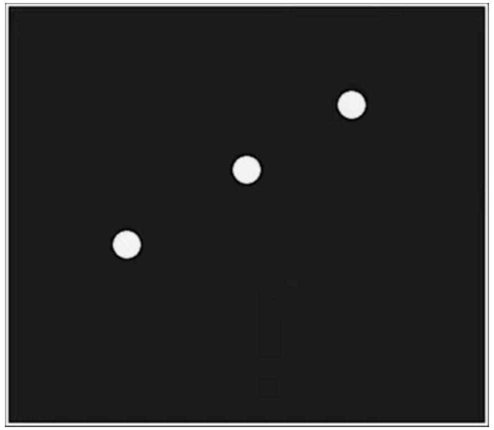
Line ⟷ **Point**

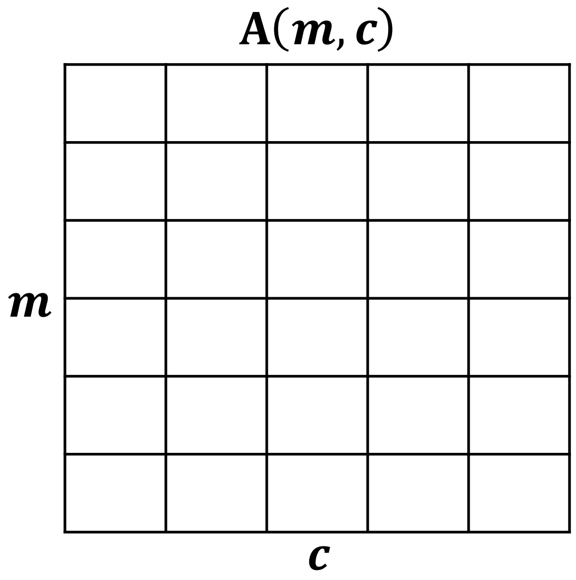
VII. The Hough transform

Line detection algorithm

1. Quantize parameter space (m, c)
2. Create an accumulator array $\mathrm{A}(m, c)$
3. Set $\mathrm{A}(m, c) = 0$ for all positions in the array

Image





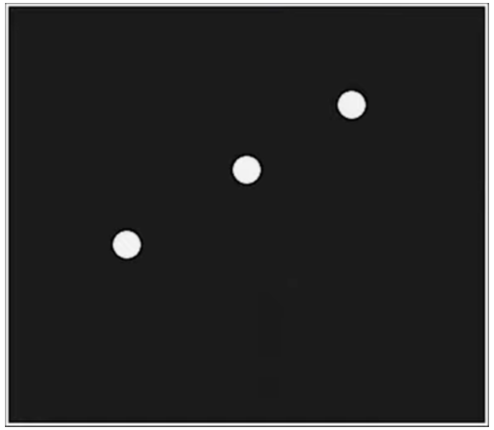
VII. The Hough transform

Line detection algorithm

1. Quantize parameter space (m, c)
2. Create an accumulator array $\mathrm{A}(m, c)$
3. Set $\mathrm{A}(m, c) = 0$ for all positions in the array
4. For each edge point (x_i, y_i):

   $\mathrm{A}(m, c) \mathrel{+}= 1$ if (m, c) lies in the line $\mathrm{c} = -mx_i + y_i$

Image



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
0	0	0	0	0

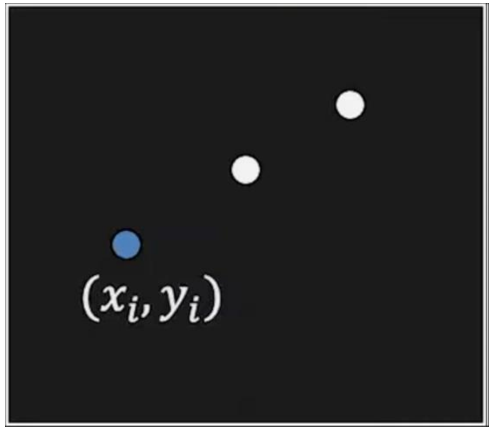
VII. The Hough transform

Line detection algorithm

1. Quantize parameter space (m, c)
2. Create an accumulator array $\mathrm{A}(m, c)$
3. Set $\mathrm{A}(m, c) = 0$ for all positions in the array
4. For each edge point (x_i, y_i):

   $\mathrm{A}(m, c) \mathrel{+}= 1$ if (m, c) lies in the line $c = -mx_i + y_i$

Image



$\mathbf{A}(\boldsymbol{m}, \boldsymbol{c})$

$\boldsymbol{m}$					
	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**
	0	0	0	0	0

$\boldsymbol{c}$

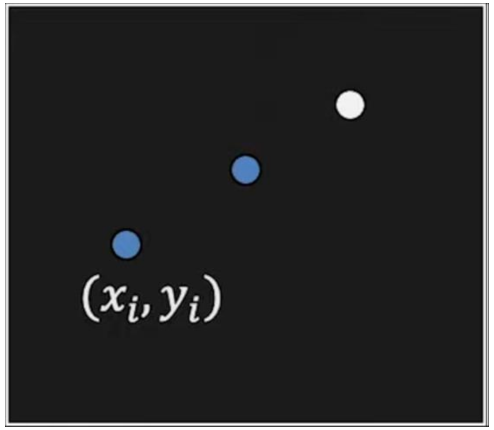
VII. The Hough transform

Line detection algorithm

1. Quantize parameter space (m, c)
2. Create an accumulator array $\mathrm{A}(m, c)$
3. Set $\mathrm{A}(m, c) = 0$ for all positions in the array
4. For each edge point (x_i, y_i):

   $\mathrm{A}(m, c) \mathrel{+}= 1$ if (m, c) lies in the line $\mathrm{c} = -mx_i + y_i$

Image



$\mathbf{A}(\boldsymbol{m}, \boldsymbol{c})$

$\boldsymbol{m}$

1	0	0	**1**	0
0	**1**	0	**1**	0
0	0	**1**	**1**	0
0	0	0	**2**	0
0	0	0	**1**	**1**
0	0	0	**1**	0

$\boldsymbol{c}$

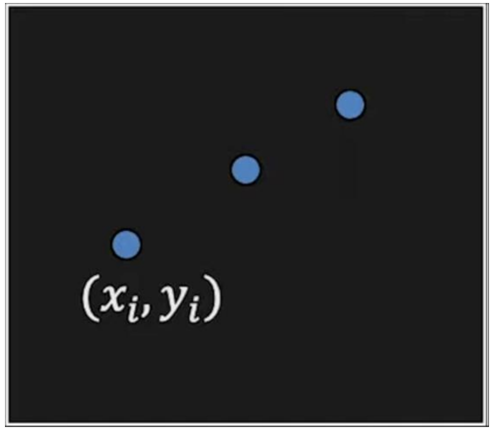
VII. The Hough transform

Line detection algorithm

1. Quantize parameter space (m, c)
2. Create an accumulator array $\mathrm{A}(m, c)$
3. Set $\mathrm{A}(m, c) = 0$ for all positions in the array
4. For each edge point (x_i, y_i):

   $\mathrm{A}(m, c) \mathrel{+}= 1$ if (m, c) lies in the line $\mathrm{c} = -mx_i + y_i$

Image



$\mathbf{A}(\boldsymbol{m}, \boldsymbol{c})$

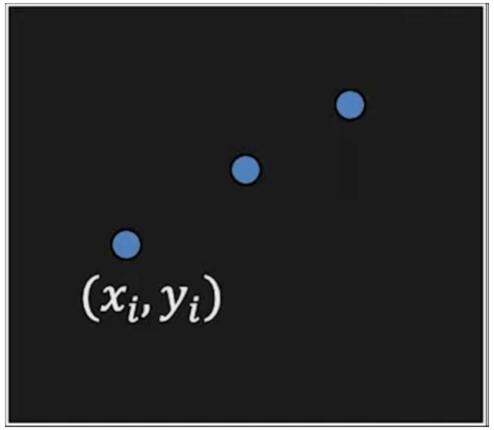
$\boldsymbol{m}$ \ $\boldsymbol{c}$					
	1	0	0	**1**	0
	0	**1**	0	**1**	0
	0	0	**1**	**1**	0
	1	**1**	**1**	**3**	**1**
	0	0	0	**1**	**1**
	0	0	0	**1**	0

VII. The Hough transform

Line detection algorithm

1. Quantize parameter space (m, c)
2. Create an accumulator array $\mathrm{A}(m, c)$
3. Set $\mathrm{A}(m, c) = 0$ for all positions in the array
4. For each edge point (x_i, y_i):
   $\mathrm{A}(m, c) \mathrel{+}= 1$ if (m, c) lies in the line $\mathrm{c} = -mx_i + y_i$
5. Find local maxima in $\mathrm{A}(m, c)$

Image



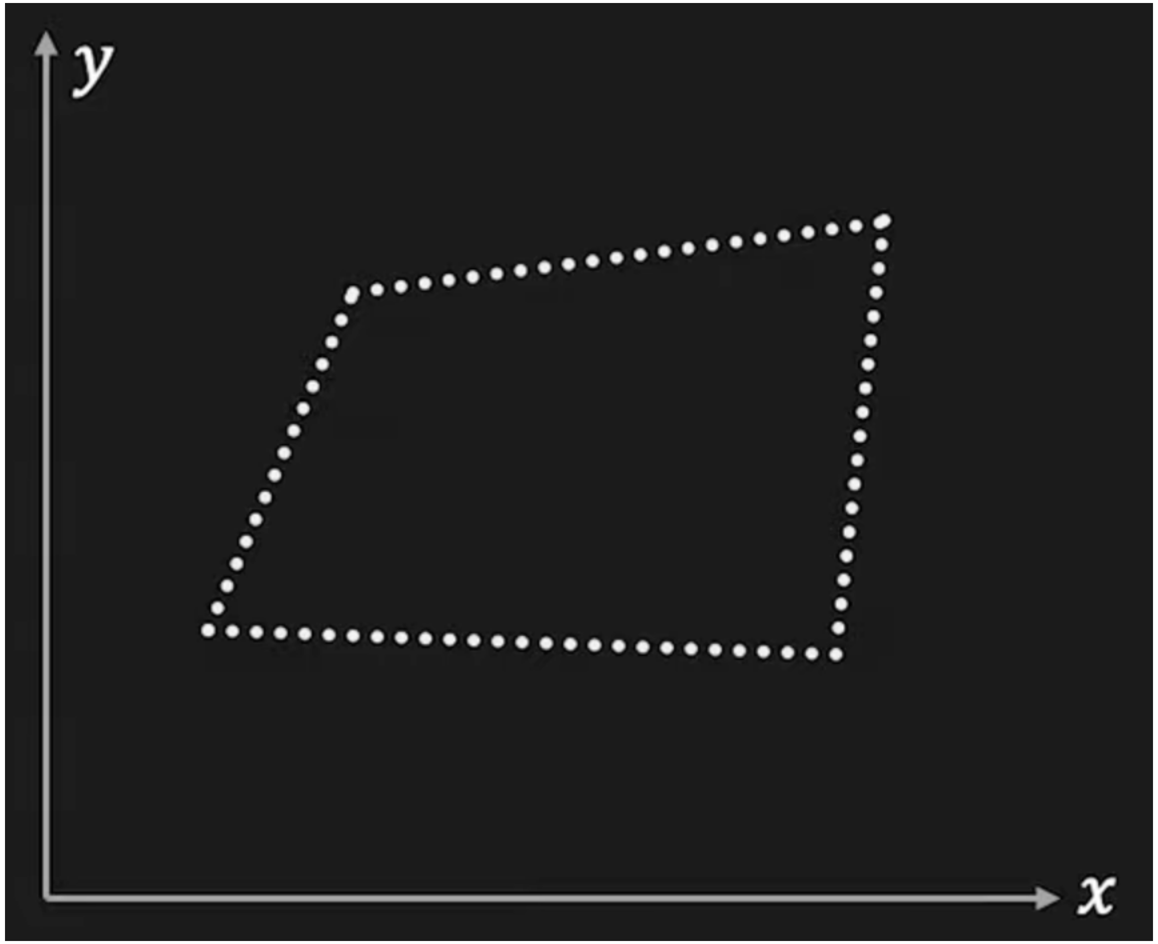
$\mathbf{A}(\boldsymbol{m}, \boldsymbol{c})$

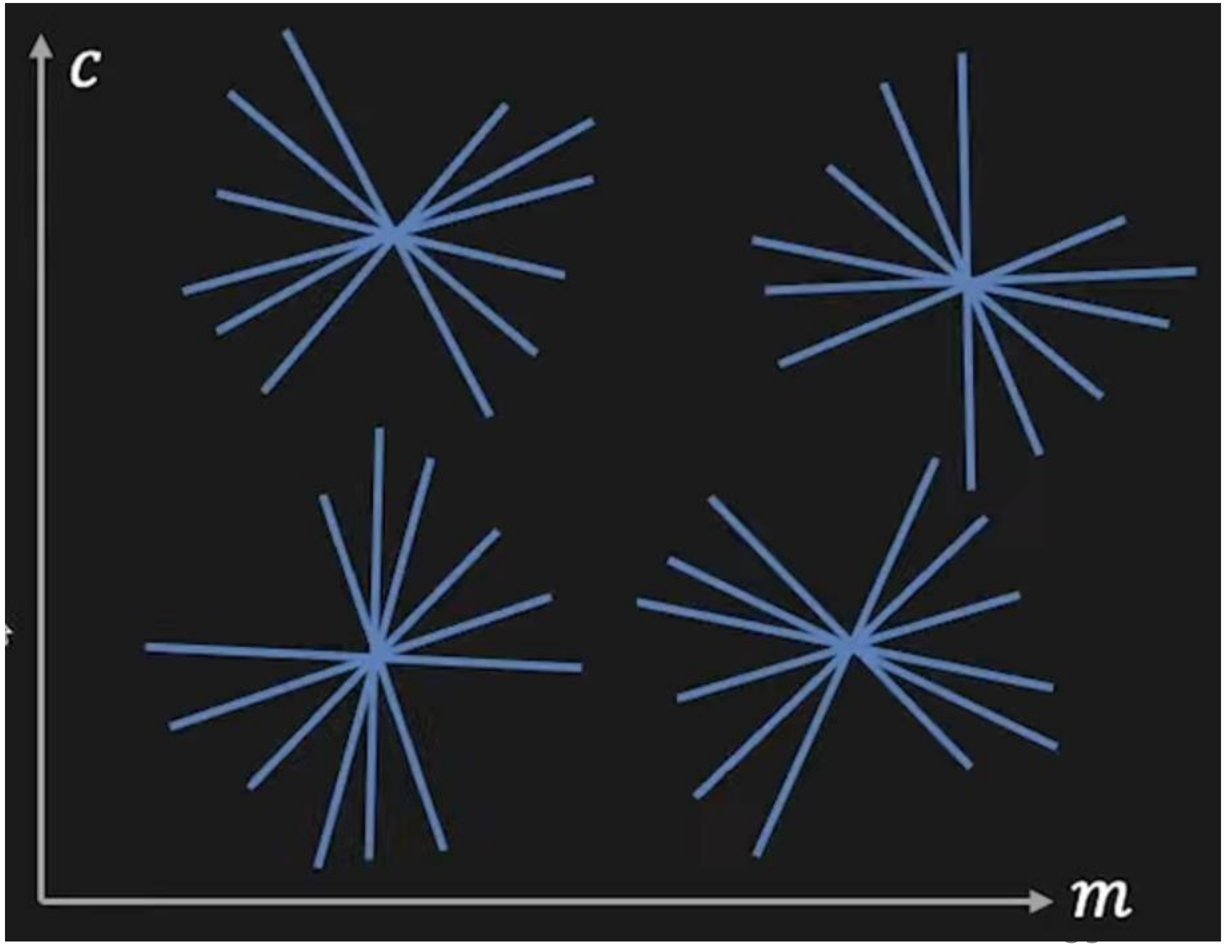
	1	0	0	1	0
	0	1	0	1	0
	0	0	1	1	0
$\boldsymbol{m}$	1	1	1	$\boxed{3}$	1
	0	0	0	1	1
	0	0	0	1	0

$\boldsymbol{c}$

VII. The Hough transform

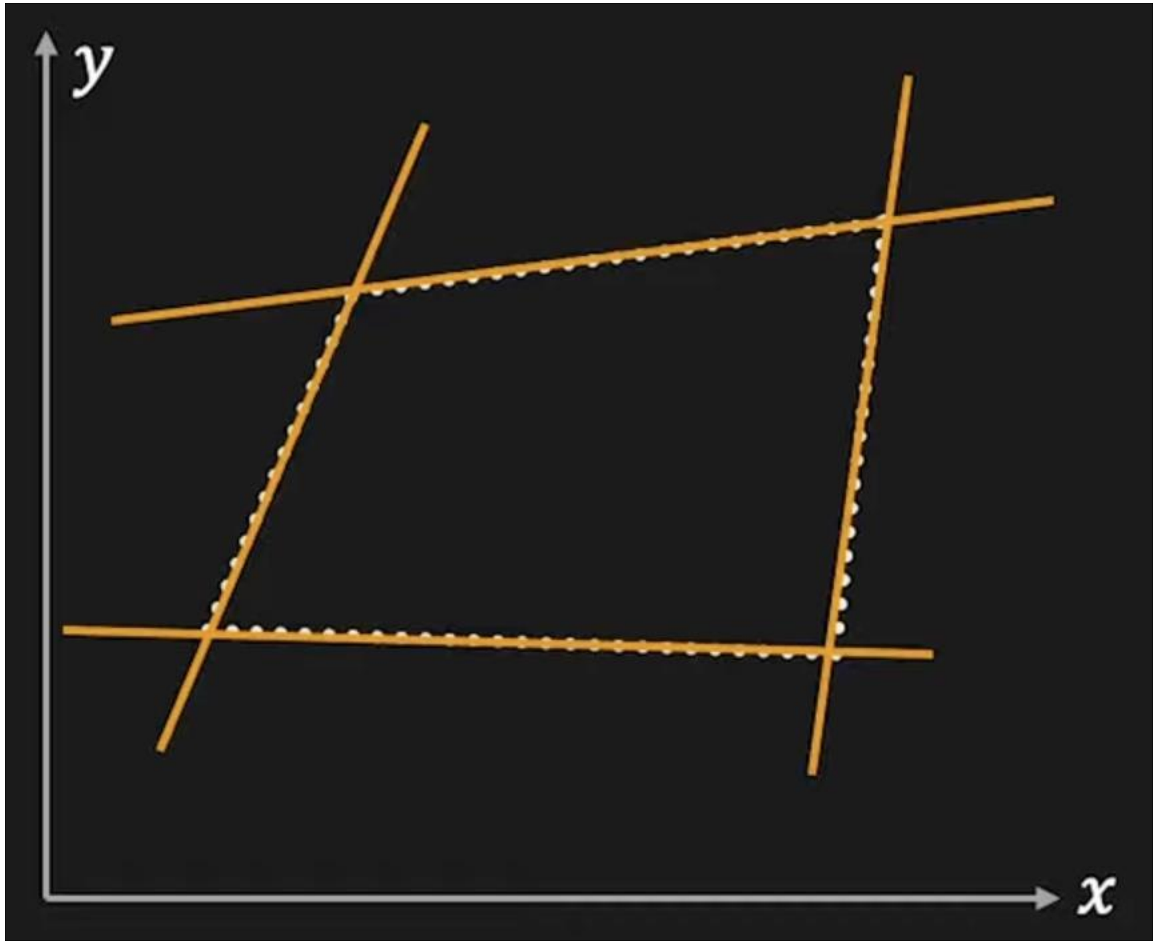
The case of multiple lines per image

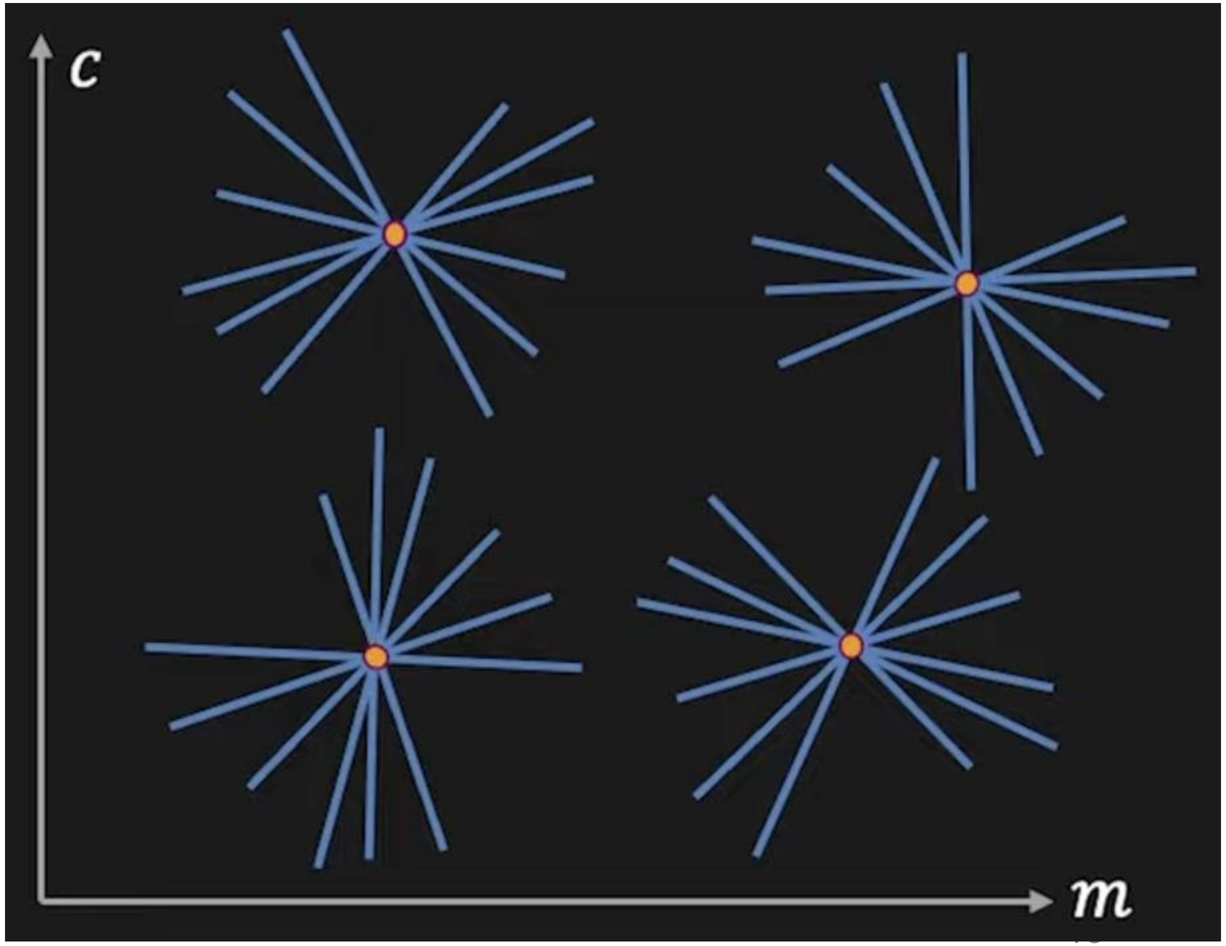




VII. The Hough transform

The case of multiple lines per image

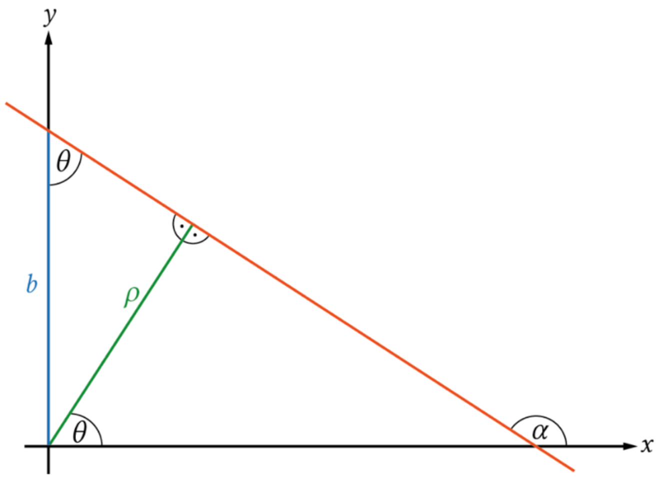




VII. The Hough transform

A parametrization problem

- The parameter m (the slope) varies between $-\infty$ and $+\infty$.
- A fine quantization => a very large accumulator.
- A huge amount of memory required
- A very costly line detection algorithm.



The solution:

- Instead of representing the line in the cartesian domain, use a **polar representation**.
- Work on parameters θ and ρ:

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

$$\pi \geq \theta \geq 0$$

ρ is finite

VII. The Hough transform

The algorithm:

- Initialize $\mathrm{A}(\rho, \theta) = 0$
- For each edge point $p_i(x_i, y_i)$ in the image

  For $\theta = 0$ to π

  $$\rho = x_i cos(\theta) + y_i sin(\theta)$$

  $$\mathrm{A}(\rho, \theta) += 1$$

- Find (ρ, θ) for which $\mathrm{A}(\rho, \theta)$ is maximum.
- The detected line is given by $\rho = x\ cos(\theta) + y\ sin(\theta)$

VII. The Hough transform

Possible extensions & optimizations

- Along with the edges, make use of the gradient direction

  => Reduces the need to iterate overall possible values of θ.

  The gradient direction at each edge point to directly infer the most likely orientation of the line passing through that point.

- Give more votes to strongest edges: Makes use of the gradient magnitude.

- Change the sampling of (ρ, θ) to trade-off resolution with computing time:

  High resolution -> Dispersion of votes (different line may be merged)

  Low resolution -> Cannot distinguish similar lines (noise may causes lines to be missed)

VII. The Hough transform

Possible extensions & optimizations

- How many lines do we have?

  Count the peaks in the accumulator array.

  Need a peak finding technique (non-maximal suppression for corner corner detection).

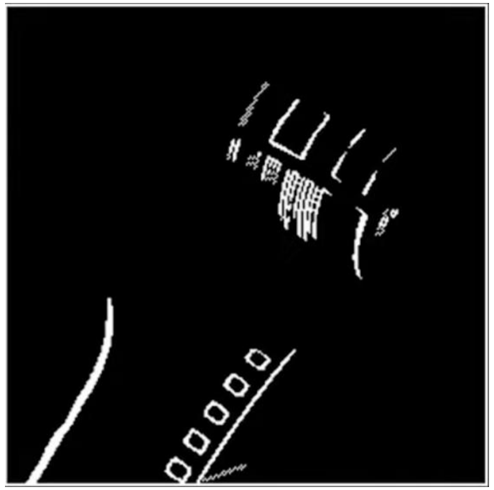
VII. The Hough transform

Examples

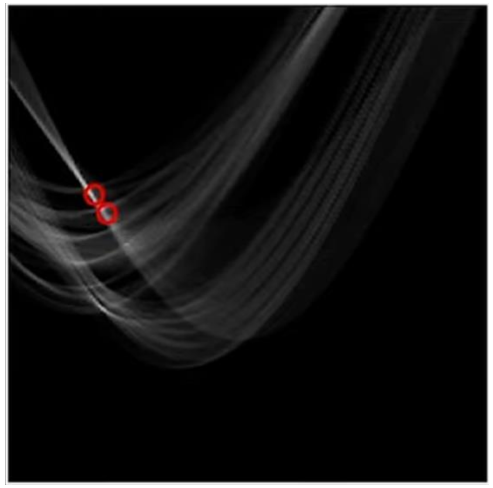
Original image

Gradient

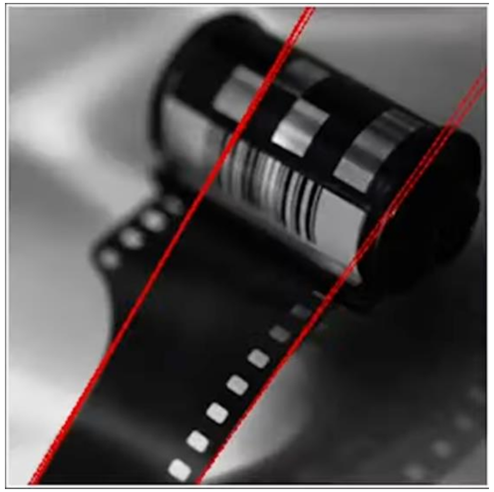
Edges



Hough Transform

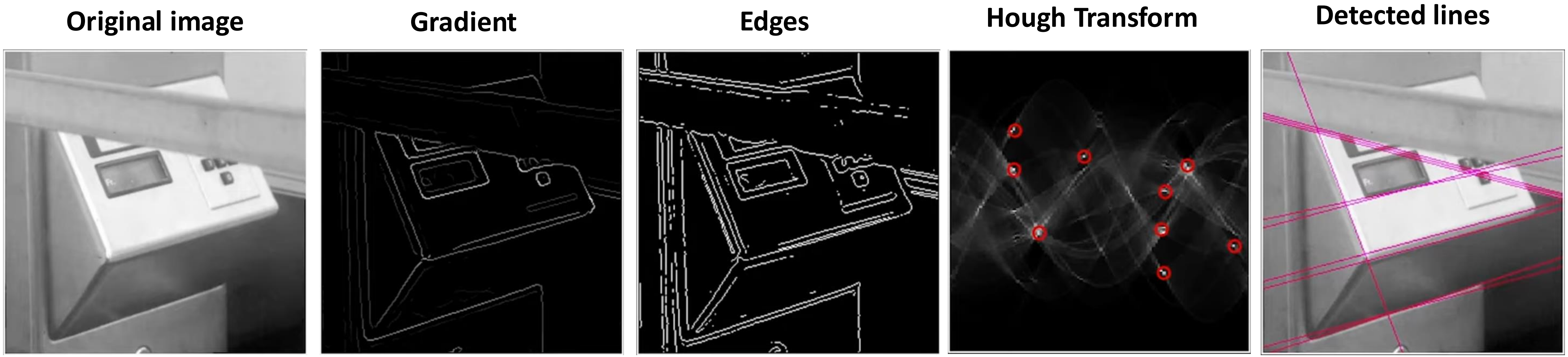


Detected lines



VII. The Hough transform

Examples

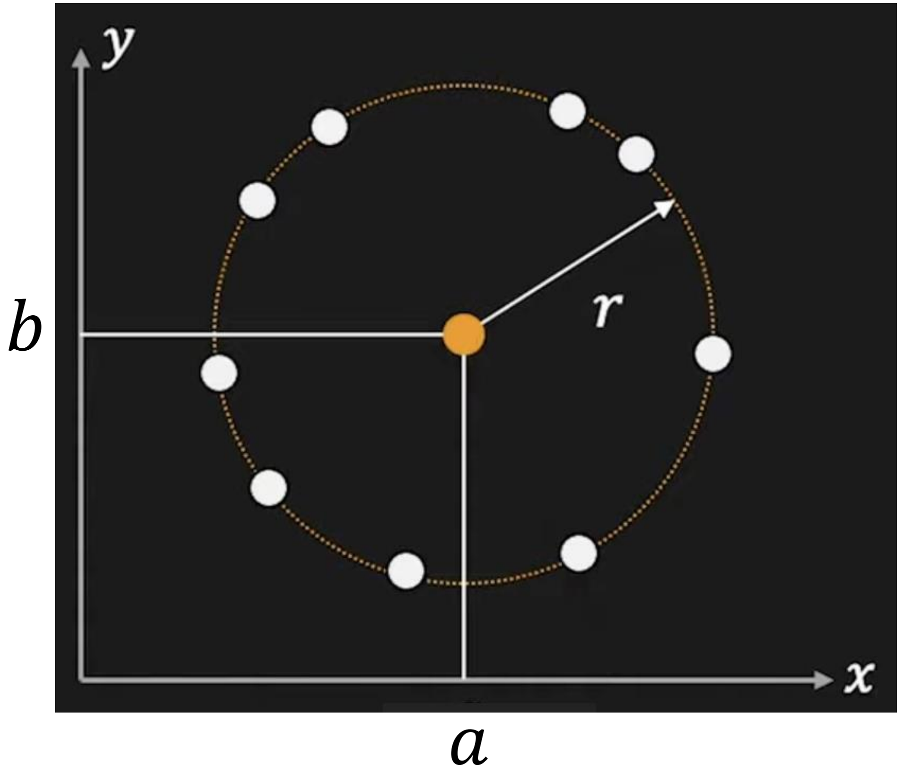


VII. The Hough transform

Application to circle detection

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

$\Rightarrow$ Problem: find the three parameters a, b and r given a set of edge points.



VII. The Hough transform

Application to circle detection

- Let's make the problem simpler. We assume the radius r known.
- The accumulator array concerns only parameters a and b (the center of the circle).

$$(\boldsymbol{x_i} - a)^2 + (\boldsymbol{y_i} - b)^2 = r^2$$

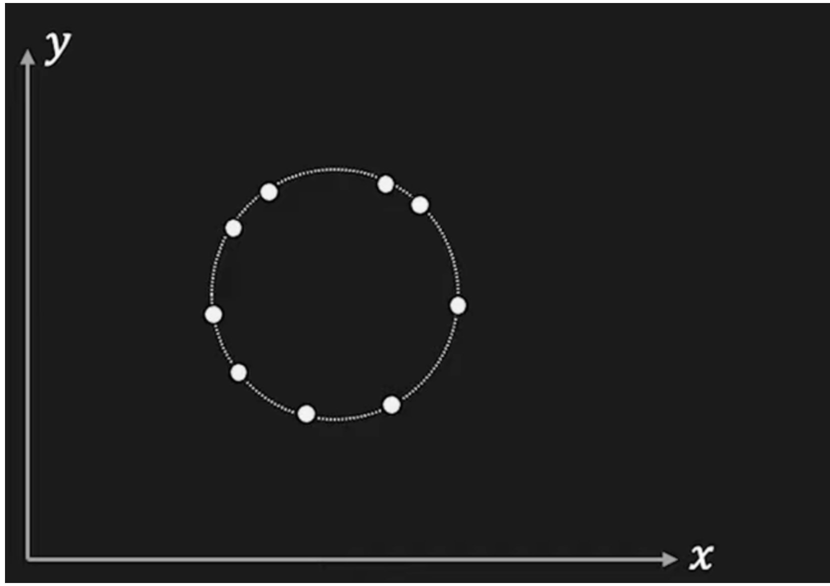
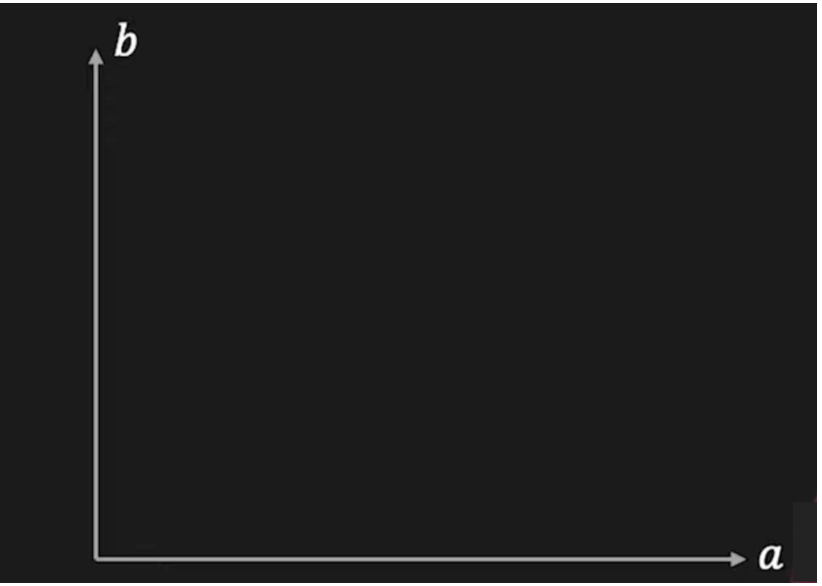


Image space

$$(x_i - \boldsymbol{a})^2 + (\boldsymbol{y_i} - \boldsymbol{b})^{\boldsymbol{2}} = r^2$$



Parameter space

VII. The Hough transform

Application to circle detection

- Let's make the problem simpler. We assume the radius r known.
- The accumulator array concerns only parameters a and b (the center of the circle).

$$(\boldsymbol{x_i} - a)^2 + (\boldsymbol{y_i} - b)^2 = r^2 \qquad (x_i - \boldsymbol{a})^2 + (y_i - \boldsymbol{b})^2 = r^2$$

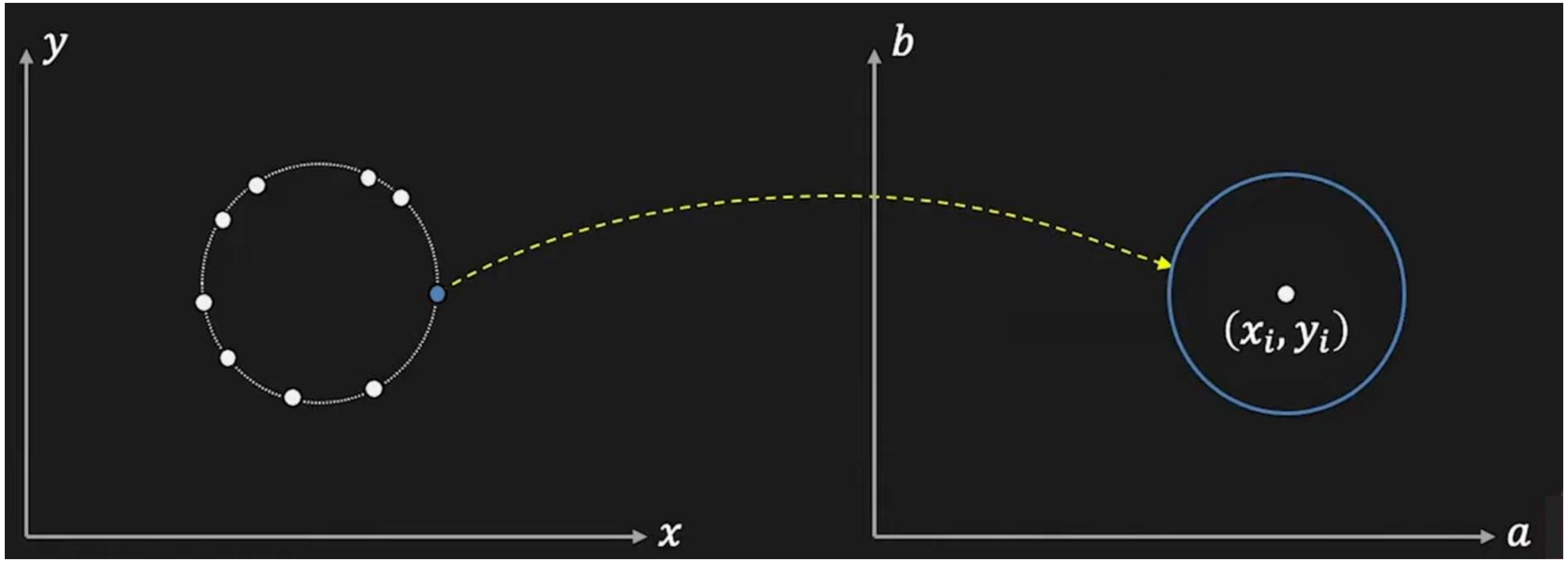


Image space **Parameter space**

VII. The Hough transform

Application to circle detection

- Let’s make the problem simpler. We assume the radius r known.
- The accumulator array concerns only parameters a and b (the center of the circle).

$$(\boldsymbol{x_i} - a)^2 + (\boldsymbol{y_i} - b)^2 = r^2 \qquad\qquad (x_i - \boldsymbol{a})^2 + (\boldsymbol{y_i} - \boldsymbol{b})^2 = r^2$$

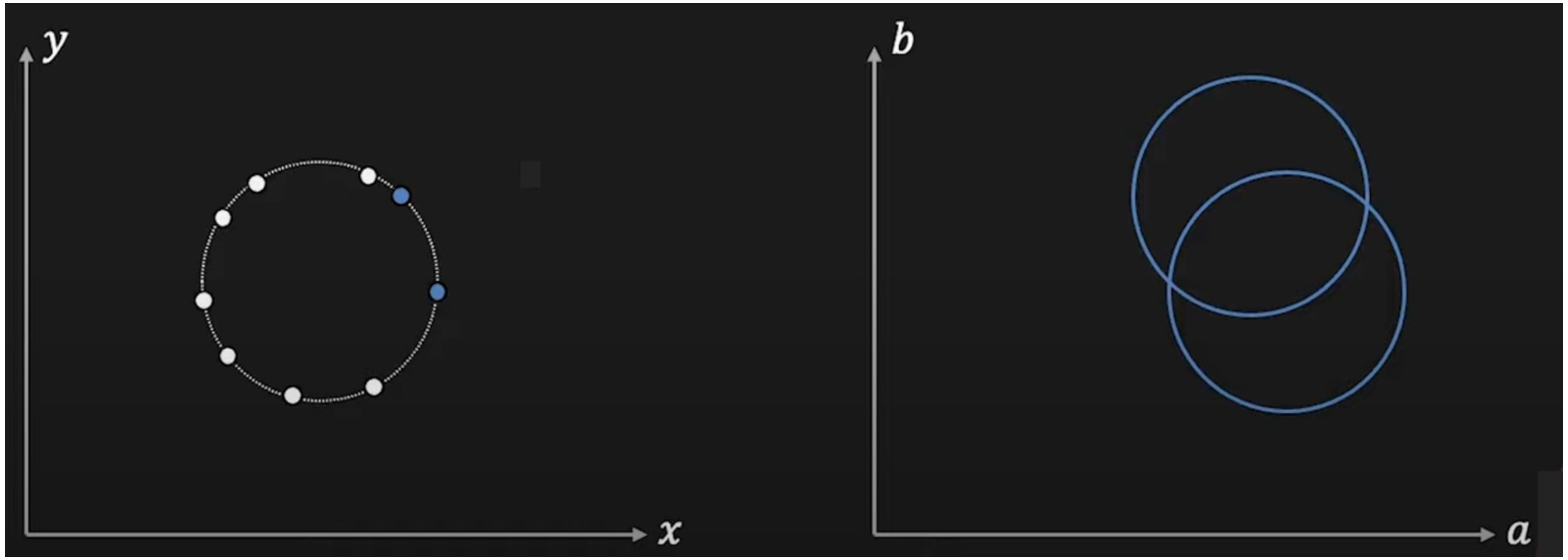


Image space **Parameter space**

VII. The Hough transform

Application to circle detection

- Let's make the problem simpler. We assume the radius r known.
- The accumulator array concerns only parameters a and b (the center of the circle).

$$(\boldsymbol{x_i} - a)^2 + (\boldsymbol{y_i} - b)^2 = r^2 \qquad (x_i - \boldsymbol{a})^2 + (\boldsymbol{y_i} - \boldsymbol{b})^\mathbf{2} = r^2$$

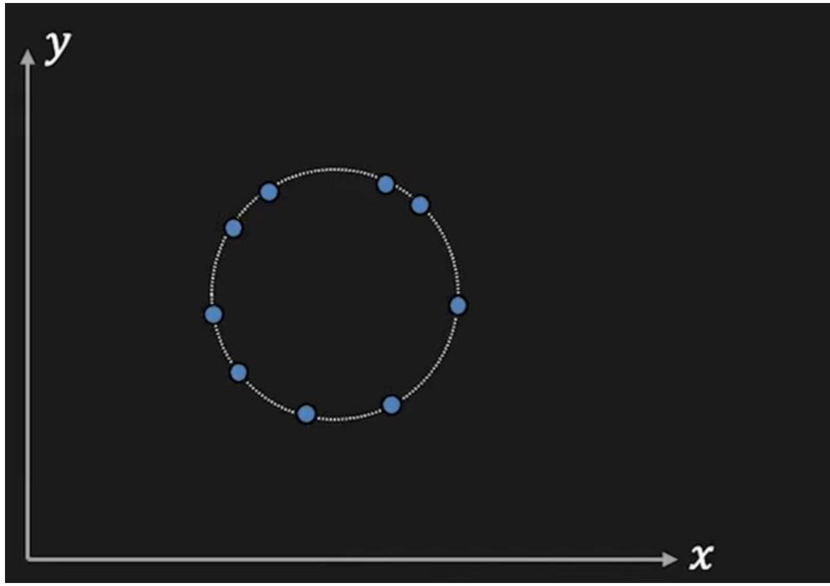
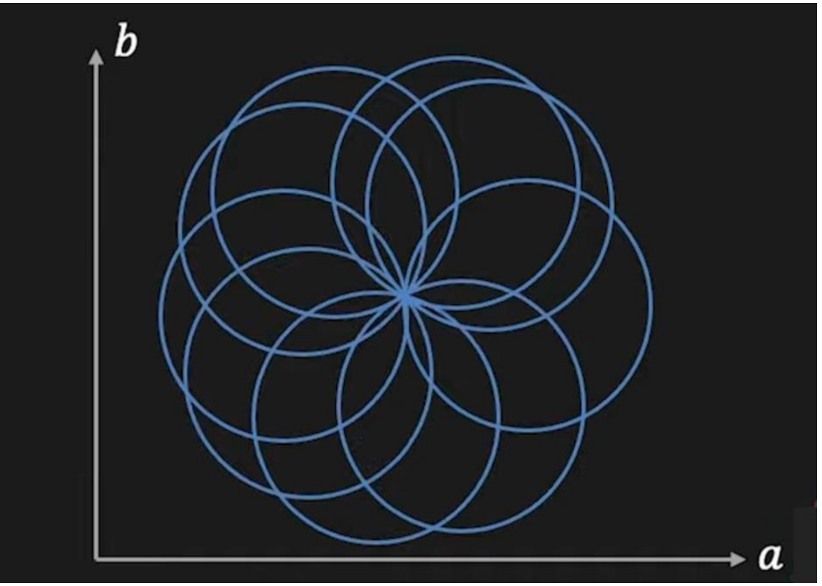


Image space

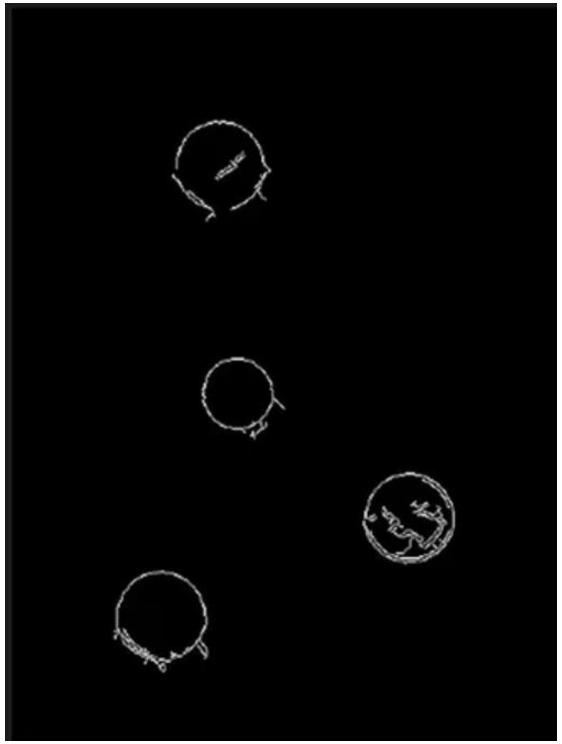


Parameter space

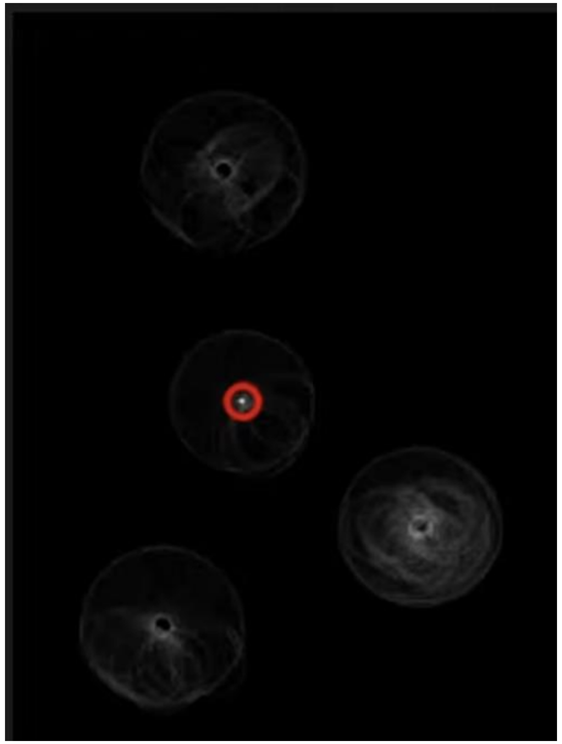
VII. The Hough transform

An example

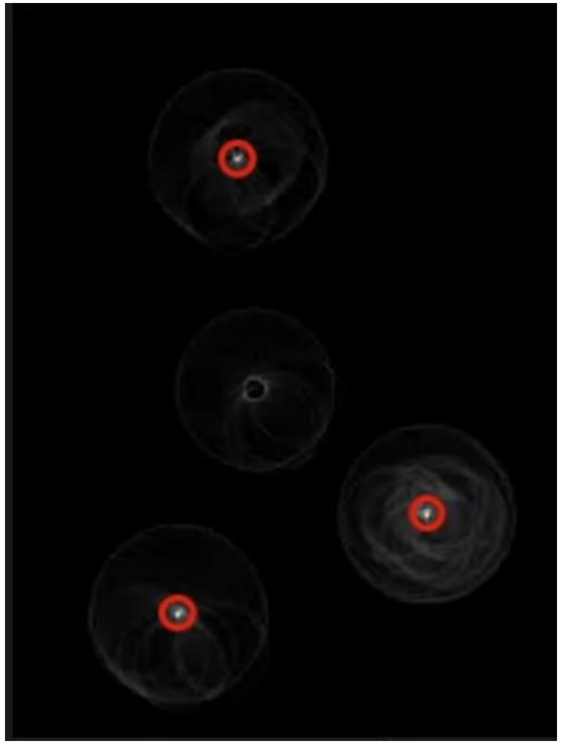
Original image



Edges



Hough Transform
$r = r_1$



Hough Transform
$r = r_2$

VII. The Hough transform

Application to circle detection

- Now, if the radius r is unknown.
- The accumulator array concerns parameters a, b and also the radius r.

$$(\boldsymbol{x_i} - a)^2 + (\boldsymbol{y_i} - b)^2 = r^2 \qquad (x_i - \boldsymbol{a})^2 + (\boldsymbol{y_i} - \boldsymbol{b})^2 = \boldsymbol{r}^2$$

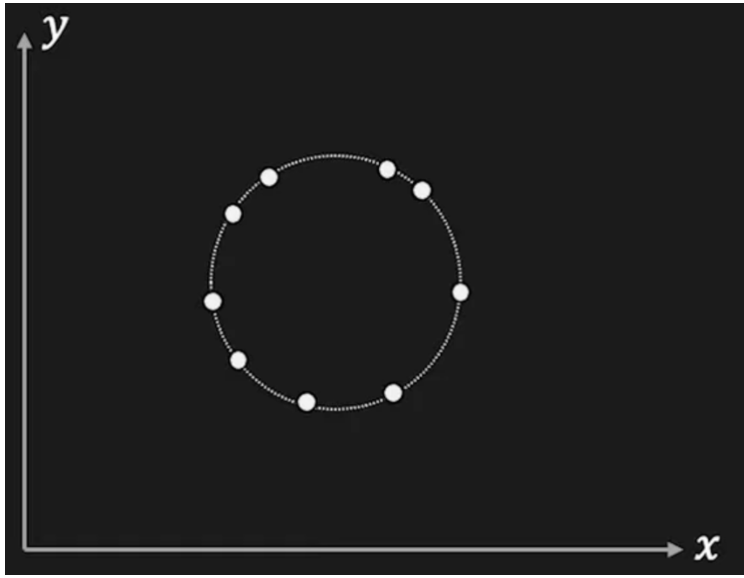
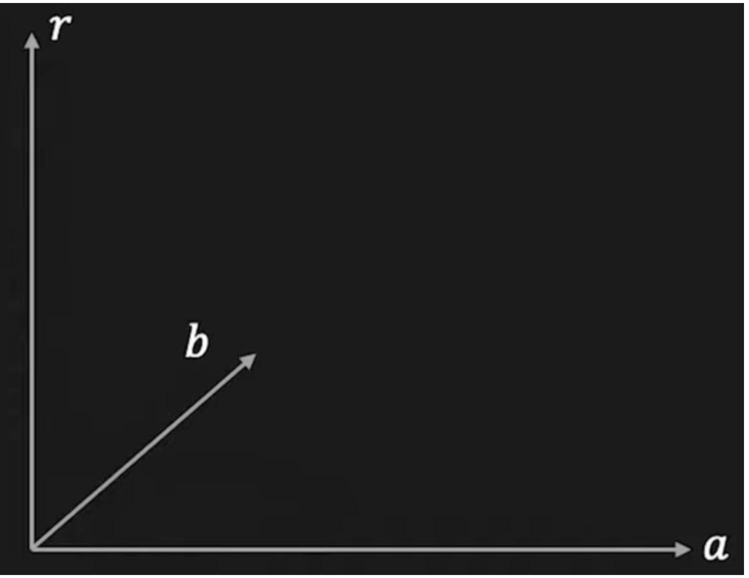


Image space



Parameter space

VII. The Hough transform

Application to circle detection

- Now, if the radius r is unknown.
- The accumulator array concerns parameters a, b and also the radius r.

$(\boldsymbol{x_i} - a)^2 + (\boldsymbol{y_i} - b)^2 = r^2$ $\qquad$ $(x_i - \boldsymbol{a})^2 + (\boldsymbol{y_i} - \boldsymbol{b})^2 = \boldsymbol{r}^2$

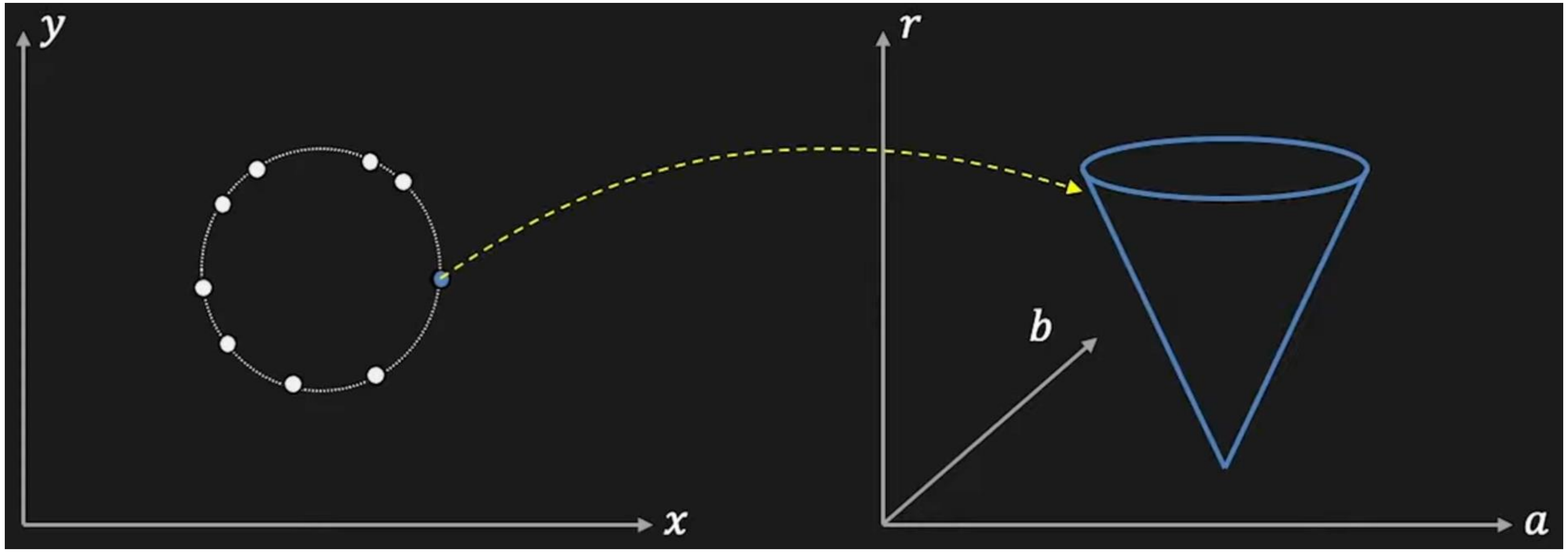


Image space $\qquad$ **Parameter space**