Welcome to Computer Vision

Computer Vision

Dr. Muhammad Tahir

DEPARTMENT OF COMPUTER SCIENCE,

FAST-NUCES, Peshawar

Course Details

LECTURES: Monday
& Wednesday

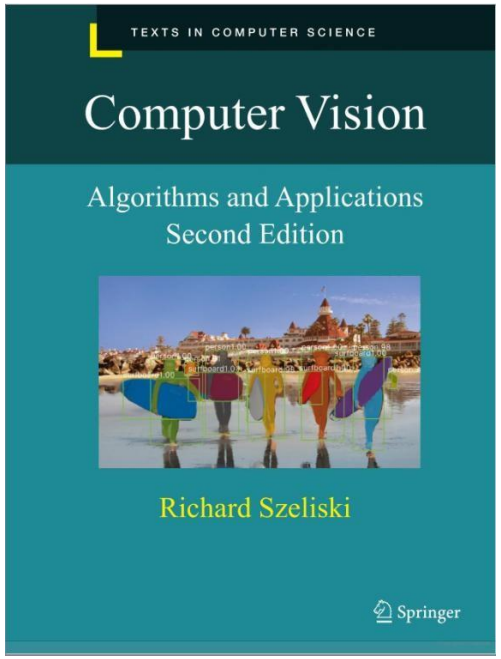
TIMINGS:

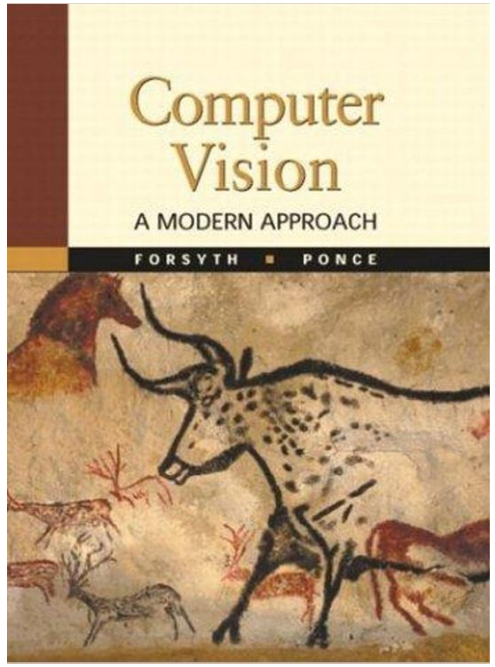
9:30 am – 11:00 am

MY OFFICE:

OFFICE HOURS:

EMAIL: m.tahir@nu.edu.pk





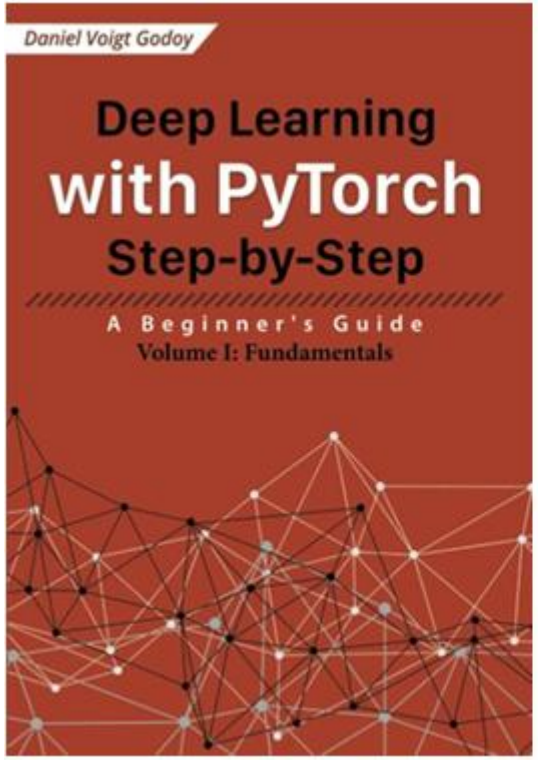
References

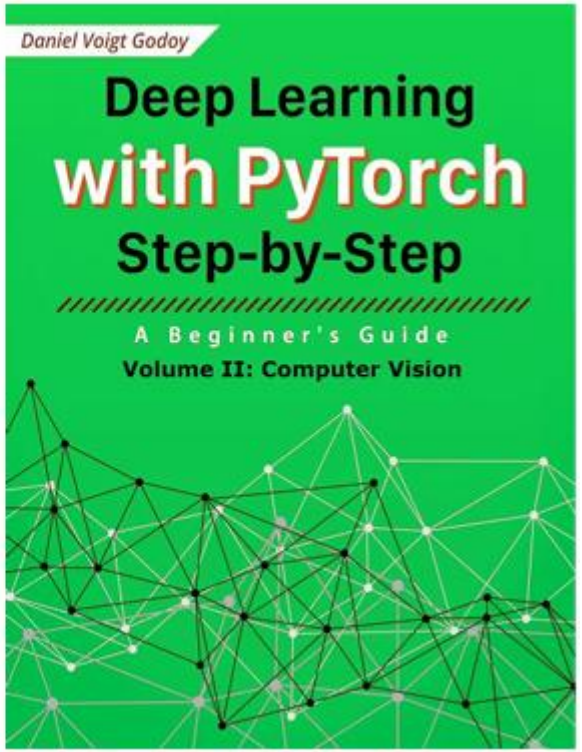
The material in these slides are based on:

1. Rick Szeliski's book: Computer Vision: Algorithms and Applications
2. Forsythe and Ponce: Computer Vision: A Modern Approach

Recommended Books

Deep Learning with PyTorch Step-by-Step by Daniel Voigt Godoy





Course Learning Outcomes

No	CLO (Tentative)	Domain	Taxonomy Level	PLO
1	Understanding basics of Computer Vision: algorithms, tools, and techniques	Cognitive	2	
2	Develop solutions for image/video understanding and recognition	Cognitive	3	
3	Design solutions to solve practical Computer Vision problems	Cognitive	3	

Outline

- Canny Edge Detection
- Hough Transform
- Model Fitting

Gaussian Filtering

- Gaussian filtering is a **smoothing technique** used in image processing to reduce noise and detail.
- Instead of using a simple average of neighboring pixels (like an averaging filter), Gaussian filtering gives **higher weights to pixels closer to the center** and **lower weights to pixels further away**.

- The filter is based on the **Gaussian function**:

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

where:

- σ= standard deviation (controls how spread the filter is).
- $(x, y) =$ pixel coordinates relative to the center.

- This makes the kernel look like a "bell curve" in 2D.

Gaussian Filtering

- A Gaussian kernel gives less weight to pixels further from the center of the window

- This kernel is an approximation of a Gaussian function.

$F[x, y]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

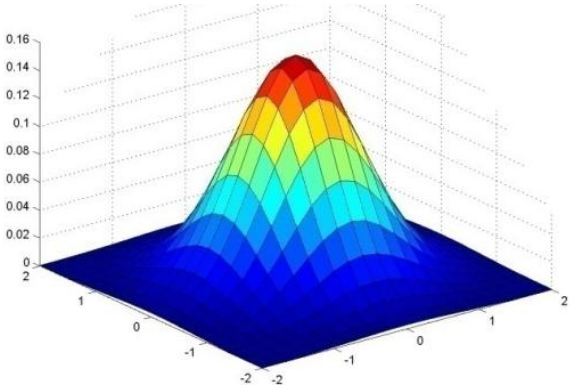
$$\frac{1}{16}\begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & 4 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

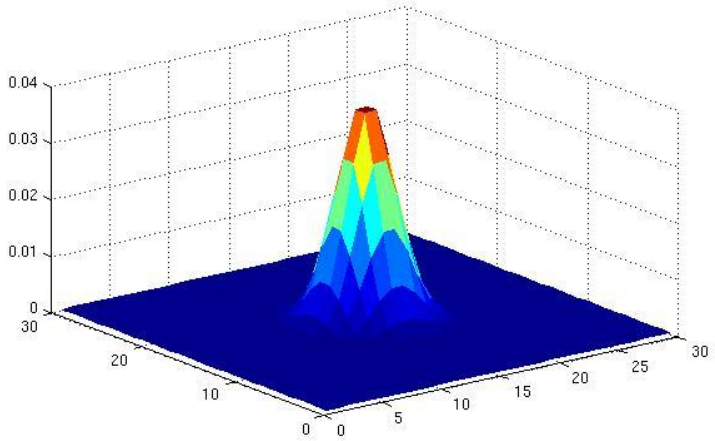
$H[u, v]$

Gaussian Filtering

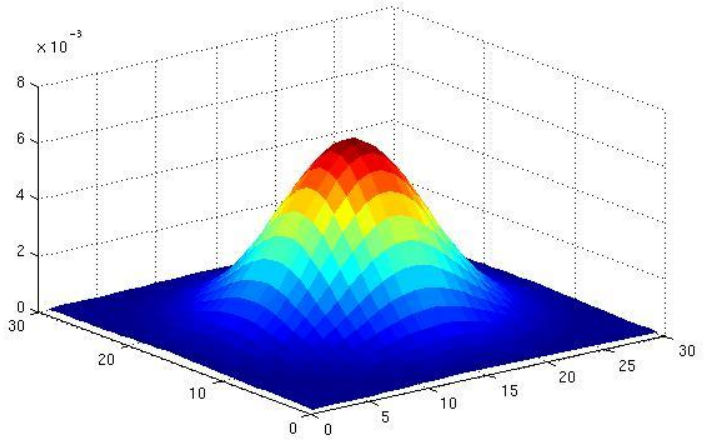
Parameter **σ** is the "scale" / "width" / "spread" of the Gaussian kernel and controls the **amount of smoothing**.

$$G_\sigma = \frac{1}{2\pi\sigma^2}\, e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$





σ = 2 with 30 x 30 kernel



σ = 5 with 30 x 30 kernel

Gaussian Vs Averaging

Gaussian Smoothing

Smoothing by Averaging

Canny Edge Detector

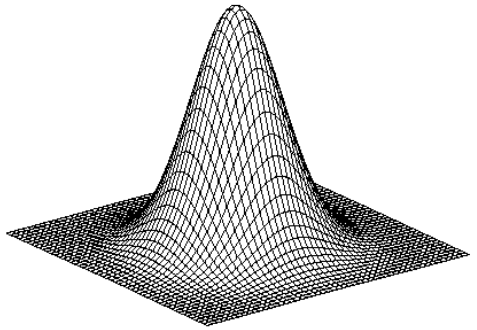
Canny Edge Detector

- The **Canny edge detector** is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images.
- It was developed by John F. Canny in 1986.
- This is probably the most widely used edge detector in computer vision.

Canny Edge Detector

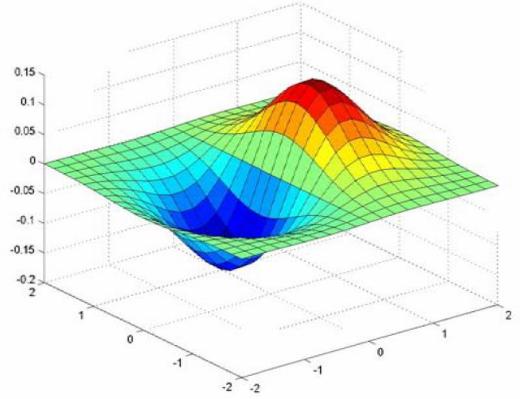
1. Filter image with x, y derivatives of Gaussian (i.e. Apply Sobel Operator)
2. Find magnitude and orientation of gradient
3. Non-maximum suppression:
    - Thin multi-pixel wide "ridges" down to single pixel width
4. Thresholding and linking (hysteresis):
    - Define two thresholds: low and high
    - Use the high threshold to start edge curves and the low threshold to continue them

Step 01: Derivative of Gaussian filter



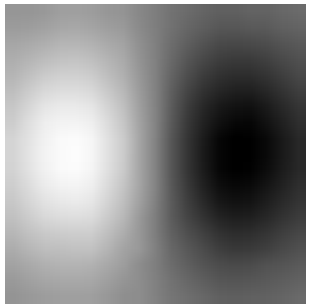
2D-gaussian

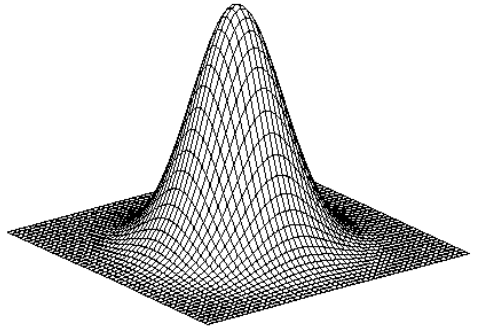
* [1 0 -1] =



x - derivative

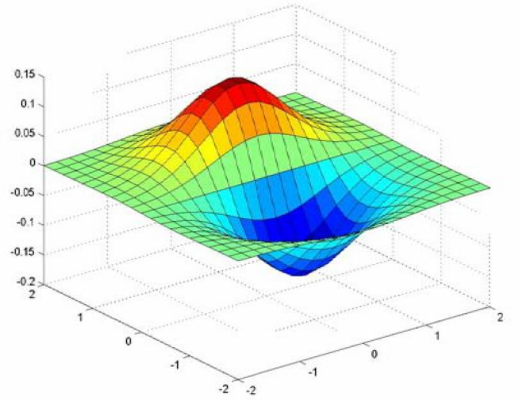
$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$





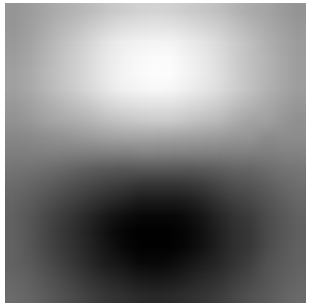
2D-gaussian

$$* \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} =$$



y - derivative

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$



Derivation of Sobel Operator from Gaussian Filter

- The **first plot** (leftmost) shows a **2D Gaussian surface** (bell-shaped).
- When we take the **derivative in the x-direction** (∂/∂x), we get the second 3D plot (colored).
- It now has a **positive lobe and a negative lobe**, meaning it responds to intensity changes along **x-axis**.
- The **matrix (kernel)** shown is:

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

- This is the **Sobel operator in the x-direction**, which emphasizes **horizontal gradients (vertical edges)**.
- The final grayscale image on the right shows how this filter responds to an image. It highlights vertical edges (light-to-dark or dark-to-light transitions across columns).

Derivation of Sobel Operator from Gaussian Filter

- Again, we start with the Gaussian.
- Taking the **derivative in the y-direction** (∂/∂y) gives the second plot
- The **positive and negative lobes are flipped vertically**.
- The **matrix (kernel)** shown is:

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

- This is the **Sobel operator in the y-direction**, which emphasizes **vertical gradients (horizontal edges)**.
- The grayscale output image highlights **horizontal edges**.

Effect of σ (Gaussian kernel spread/size)

original

Canny with $\sigma = 1$

Canny with $\sigma = 2$

The choice of σ depends on desired behavior

- large σ detects large scale edges
- small σ detects fine features

What scale to choose for σ?

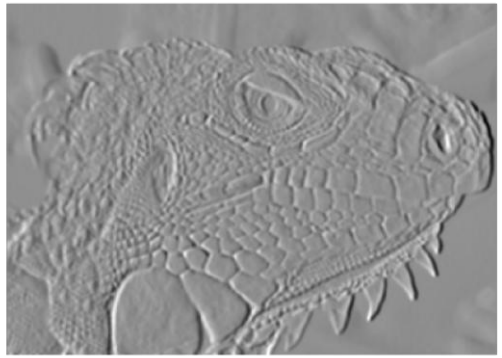
Depends what we're looking for

Sobel Operator and Gaussian

- The Gaussian smooths the image (reduces noise).
- Derivatives (∂/∂x and ∂/∂y) extract **intensity changes** — i.e., edges.
- Sobel operators combine **Gaussian smoothing + derivative approximation**.
- Final edge maps show locations **where intensity changes strongly in horizontal or vertical directions**.

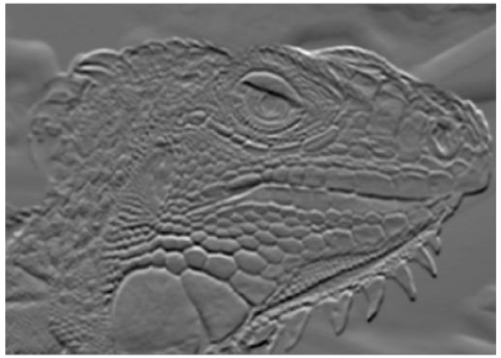
Step 02: Compute gradients

original image



X-Derivative of Gaussian

$$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

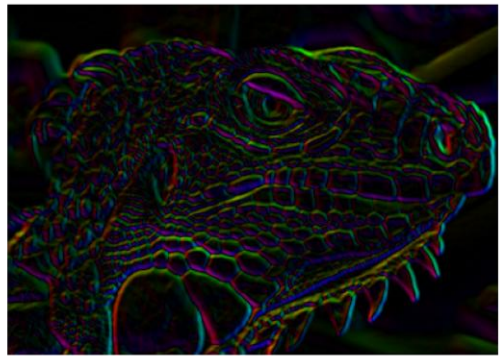


Y-Derivative of Gaussian

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Gradient Magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



Gradient Orientation

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

Step 03: Non-Maximum Suppression

- Non-maximum suppression is an edge thinning technique.
- After applying gradient calculation, the edge extracted from the gradient value is still quite blurred. There should only be one accurate response to the edge.
- Edge occurs where gradient reaches a maxima
- Non-maximum suppression can help to suppress all the gradient values (by setting them to 0) except the local maxima
- Suppress non-maxima gradient even if it passes threshold

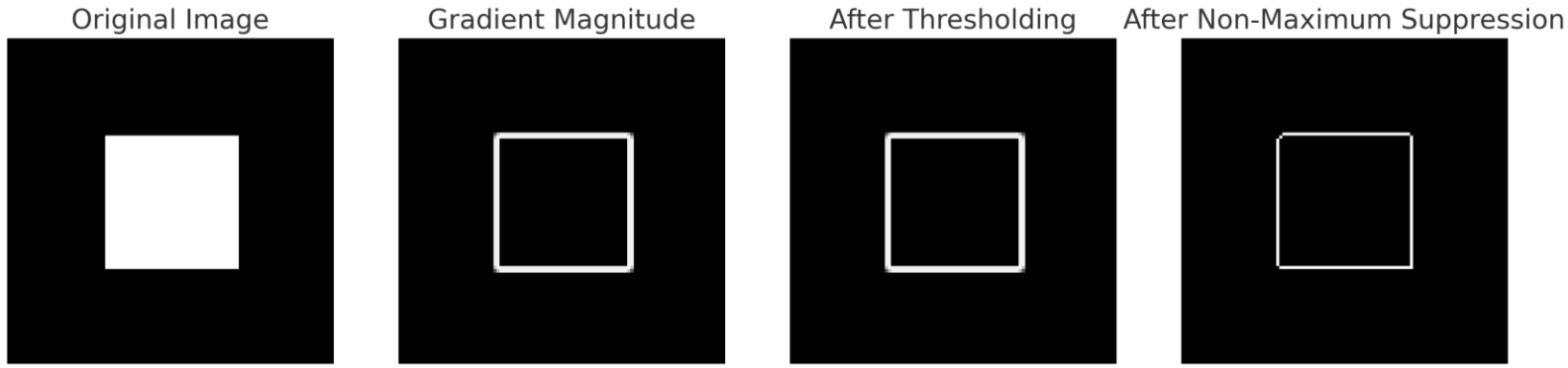
Only eight angle directions possible
Suppress all pixels in each direction which are not maxima
Do this in each marked pixel neighborhood

Non-Maximum Suppression: Algorithm

- Compare the edge strength (gradient) of the current pixel with the edge strength of the pixel in the positive and negative gradient directions.
- If the edge strength of the current pixel is the largest compared to the other pixels in the mask with the same direction, the pixel value will be preserved. Otherwise, the value will be suppressed.

- Examples
  - A pixel that is pointing in the y-direction will be compared to the pixel above and below it in the vertical axis

Non-Maximum Suppression



- **Original Image** → White square on black background.
- **Gradient Magnitude** → Thick edges appear because gradients are strong on both sides of the transition.
- **After Thresholding** → Many edge pixels remain (still thick).
- **After Non-Maximum Suppression** → Only the thinnest, sharpest edge pixels (local maxima) are kept, giving a **1-pixel thin edge**.
- This demonstrates why we compute gradients: their magnitude shows where edges exist, and non-maximum suppression ensures we keep only the strongest response along each edge direction.

Non-maximum suppression

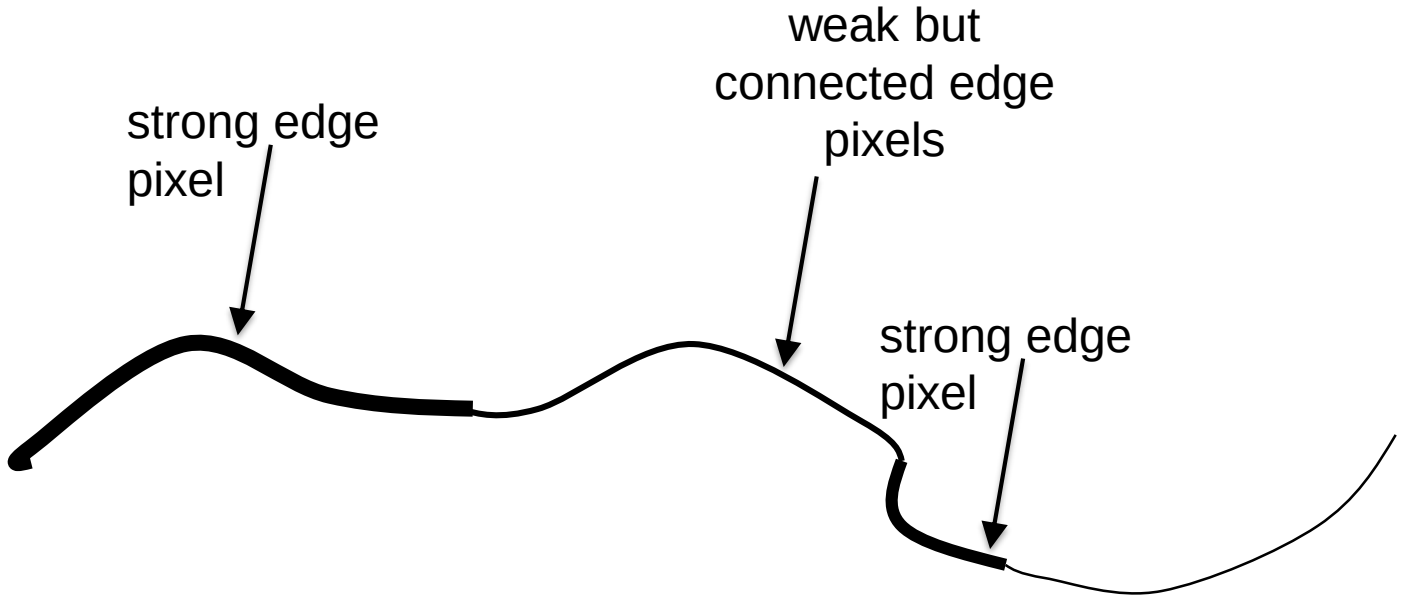
Before

After

Step 04: Thresholding (Hysteresis)

- Define two thresholds: Low and High
  - If less than Low, not an edge
  - If greater than High, strong edge
  - If between Low and High, weak edge
    - Consider its neighbors iteratively then declare it as "edge pixel" if it is connected to an 'strong edge pixel' directly or via pixels between Low and High

Thresholding (Hysteresis)



use a high threshold to start edge curves and a low threshold to continue them.

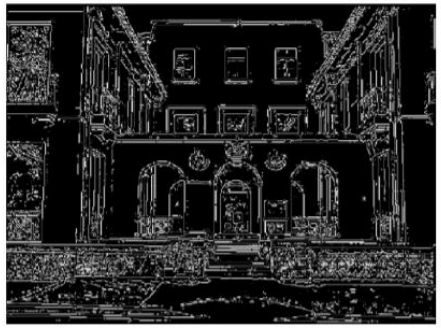
Thresholding Edges

- Still, some noise
- Only want strong edges
- 2 thresholds, 3 cases
  - R > T: strong edge
  - R < T but R > t: weak edge
  - R < t: no edge
- Why two thresholds?

Thresholding (Hysteresis)

original image

high threshold
(strong edges)



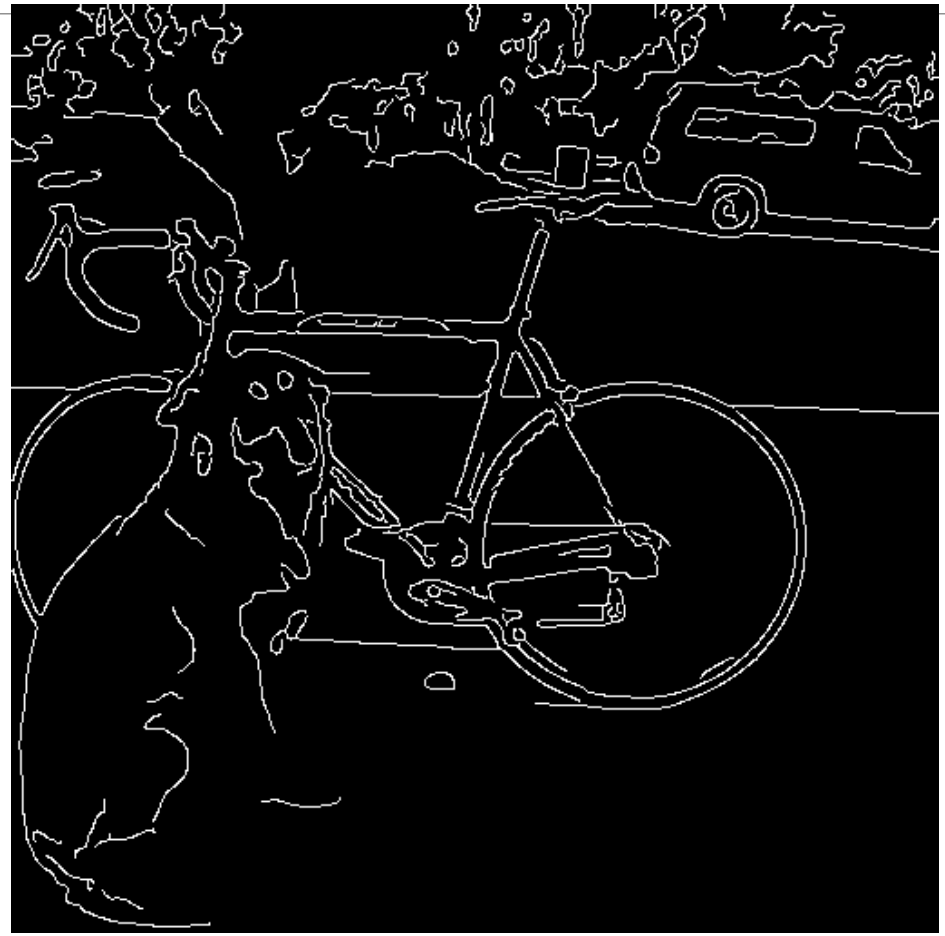
low threshold
(weak edges)

hysteresis threshold

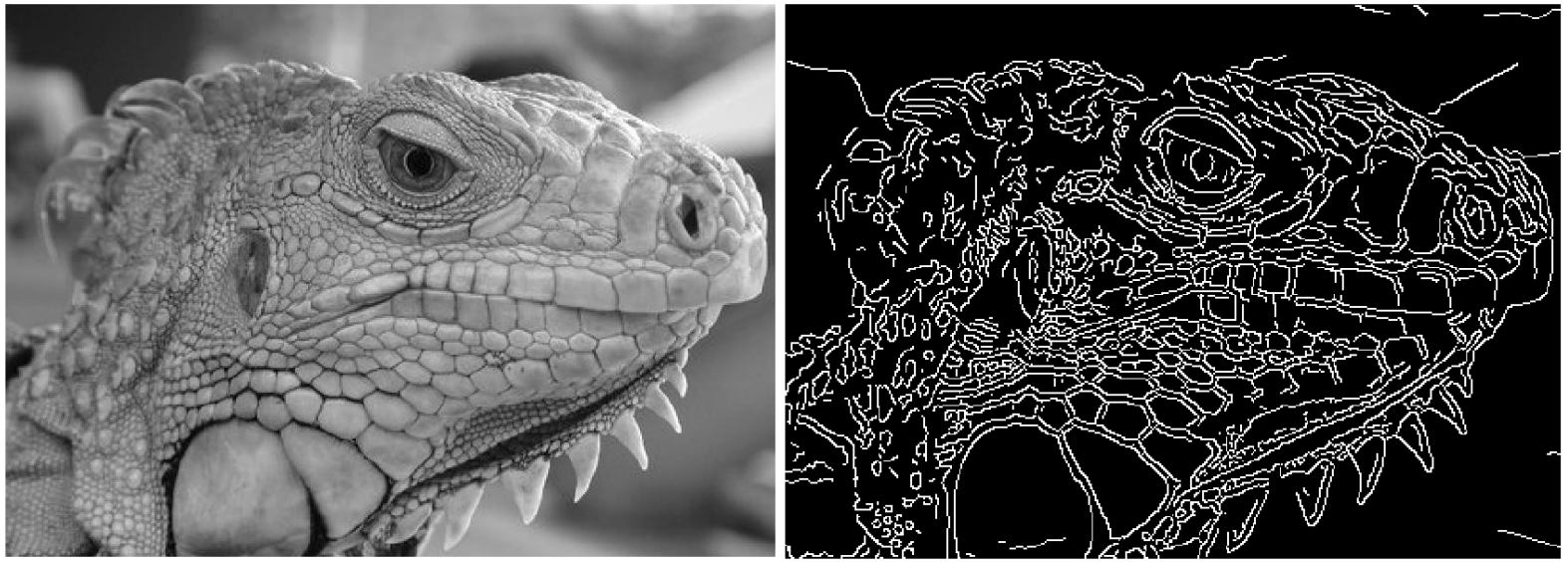
Source: L. Fei-Fei

Connecting edges

- Strong edges are edges!
- Weak edges are edges
  iff they connect to strong edges
- Look in some neighborhood
  (usually 8 closest)



Canny Edge Detector



Cartesian Space

- The "normal" coordinate system we use to represent points in an image or plane.
- This is the **image space** — where shapes and edges actually appear.
- For a 2D image, every pixel is defined by its (x, y) location.

Example: A line in Cartesian space is expressed as:

$$y = mx + b$$

where m is slope and b is intercept.

Parametric Space (Hough Space)

- A space where **parameters of shapes** (like lines, circles) are represented instead of pixel coordinates.
- **For lines:** Instead of (x, y), we describe a line by: $\rho = x\cos\theta + y\sin\theta$

where:

$\rho =$ perpendicular distance from the origin,

$\theta =$ angle of the perpendicular.

$\rightarrow$ Now a **single line** in Cartesian space corresponds to a **single point** (ρ, θ) in parametric space.

- **For circles:** A circle can be described by parameters $(\mathrm{a}, \mathrm{b}, \mathrm{r})$,where:

$a, b =$ center,

$r =$ radius.

So, the **parametric space** is 3D for circles.

Cartesian Space Vs Parametric Space

Aspect	**Cartesian Space (Image Space)**	**Parametric Space (Hough/Feature Space)**
What is plotted?	Pixel coordinates (x, y)	Shape parameters (e.g., ρ, θ for lines; $\mathrm{a}, \mathrm{b}, \mathrm{r}$ for circles)
Use	Actual image, raw features	Detecting shapes via parameter voting
Line representation	Infinite points (x, y) along the line	One point (ρ, θ) in parametric space
Circle representation	Points satisfying $(x - a)^2 + (y - b)^2 = r^2$	One point $(\mathrm{a}, \mathrm{b}, \mathrm{r})$ in parameter space

Cartesian Space Vs Parametric Space

In Cartesian space: A line is a straight line drawn through pixels.

In Parametric space: That same line is a single point (ρ, θ).

Conversely: A point in Cartesian space maps to a sinusoidal curve in parametric space.

Where multiple curves intersect in parametric space $\rightarrow$ that's the detected line in Cartesian space.

Representation

- In Cartesian space, a line like $y = 0.5x + 2$ is unique.
- In Parametric space, the same line can be represented by different pairs (ρ, θ), because:
  - (ρ, θ) and $(-\rho, \theta + 180°)$ describe the same line.
  - This redundancy happens because a line can be referenced by the perpendicular from either side.

Representation in the Cartesian Space

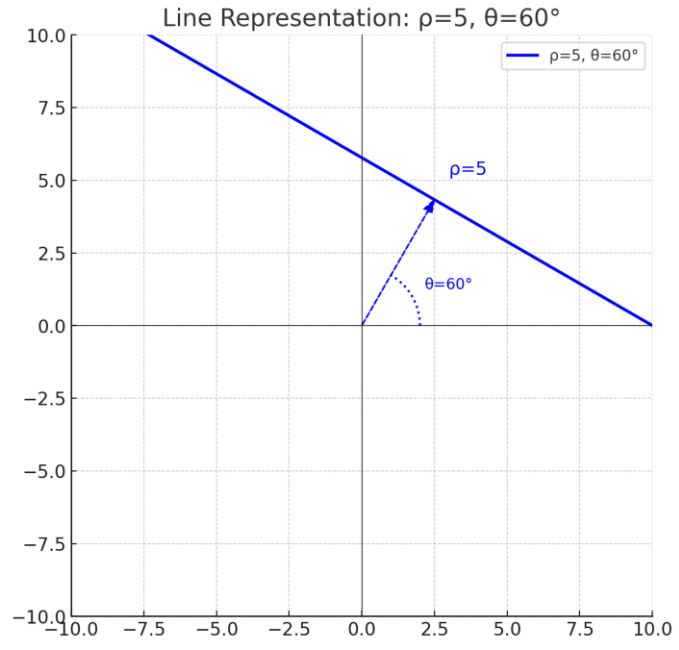
$$(\rho, \theta) \equiv (-\rho, \theta + 180°)$$
$$(\rho, \theta) = (5, 60°)$$

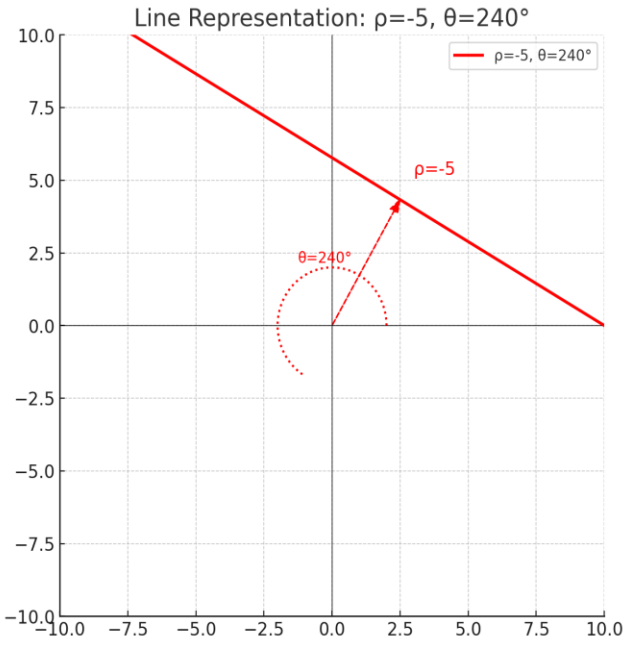
- This means: the line is at a perpendicular distance of 5 units from the origin, and the perpendicular makes a 60° angle with the x−axis.

  The equivalent representation is:

  $$(-5, 240°)$$

- Here, instead of moving **outward 5 units at 60°**, we move **inward −5 units**, which geometrically is the same as moving outward 5 units but rotated by 180°.



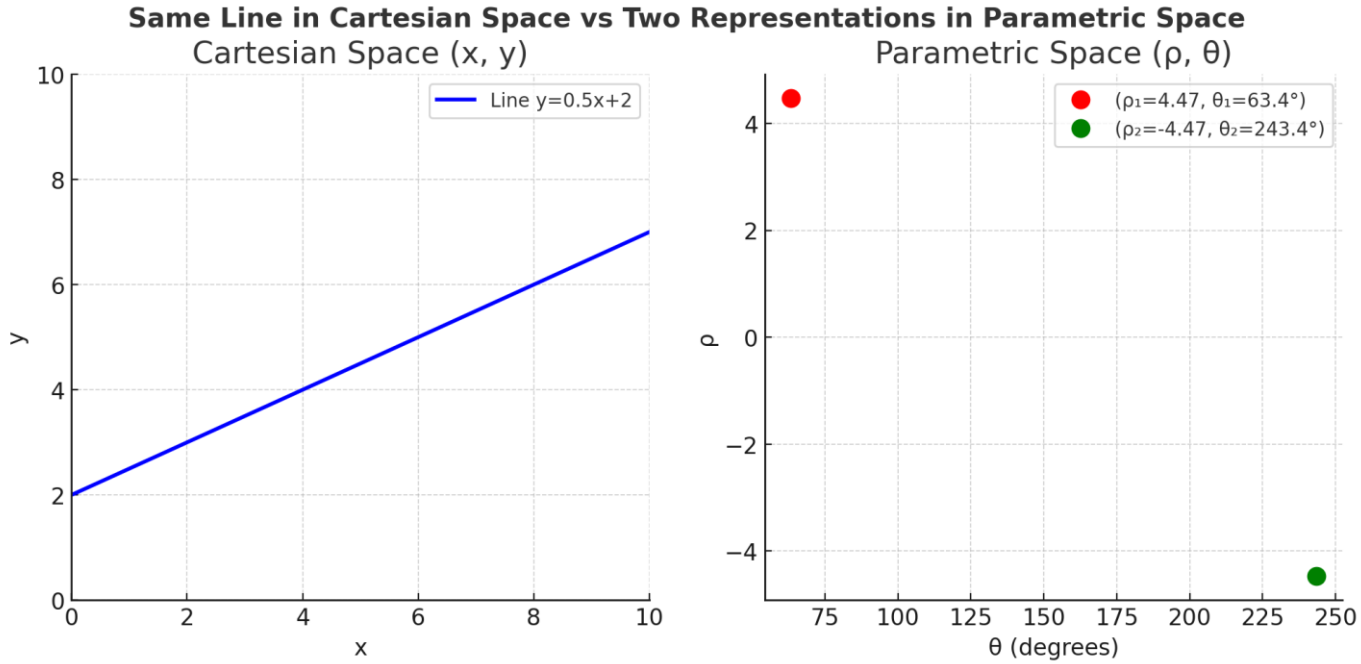


Representation

Left: The line drawn in Cartesian space. A single straight line $y = 0.5x + 2$

Right: Two equivalent parametric representations (ρ_1, θ_1) and (ρ_2, θ_2) plotted in parameter space, both describing the same line.

This shows the redundancy: (ρ, θ) and $(-\rho, \theta + 180°)$ are equivalent.



Boundary Detection

Difficulties for the fitting Approach

- You need to find the two wheels of the bicycle on the left in the edge map provided on the right

- The Problems:
  - **Extraneous Data**
    - Which points do we actually fit the circles to?
  - **Incomplete Data**
    - Some part of the wheel may not be visible or may be occluded by some other object
  - **Noise**

Which edge in an image corresponds to a boundary?

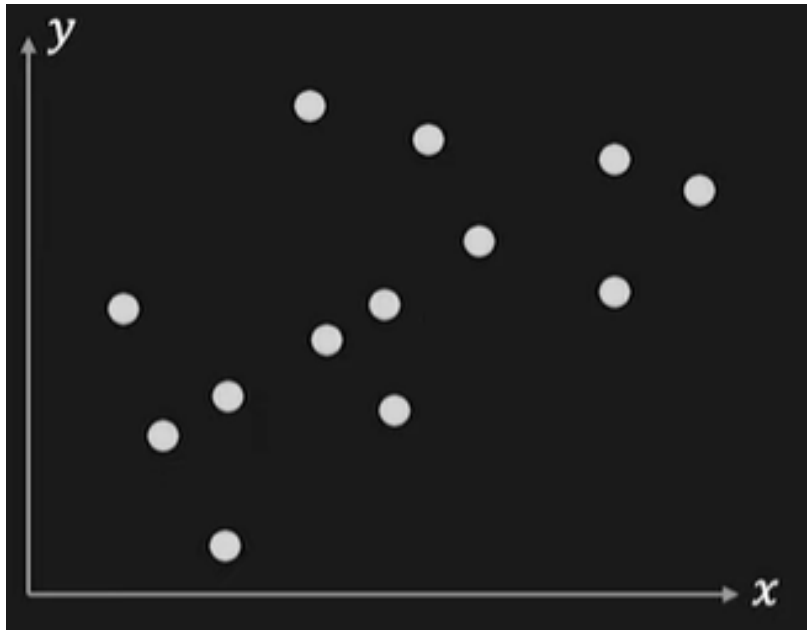
Hough Transform

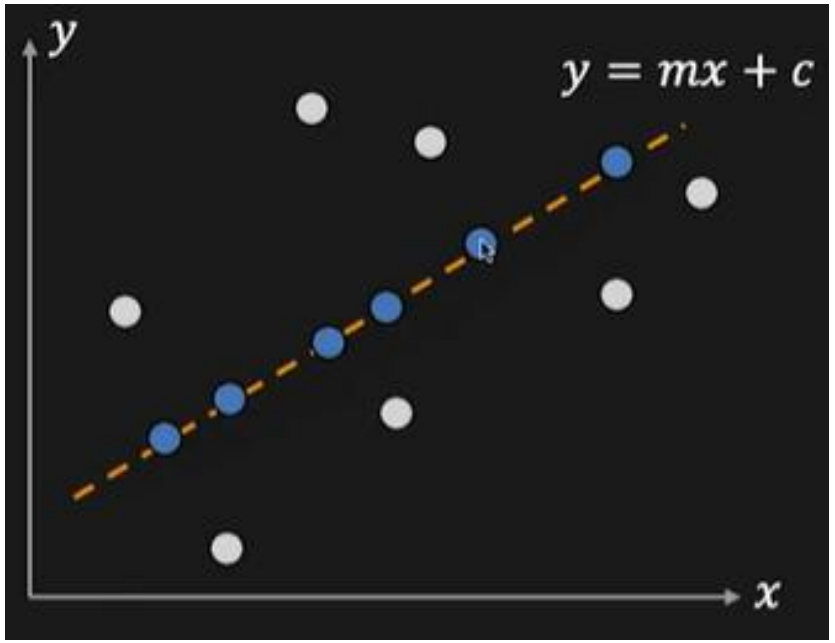
- Hough Transform provides a powerful way of solving this problem i.e. which edge belongs to the boundary?
- Boundary can be described using a small number of parameters

Detecting a Line using Hough Transform

Given Edge Points (x_i, y_i)

Task: Detect line $y = mx + c$





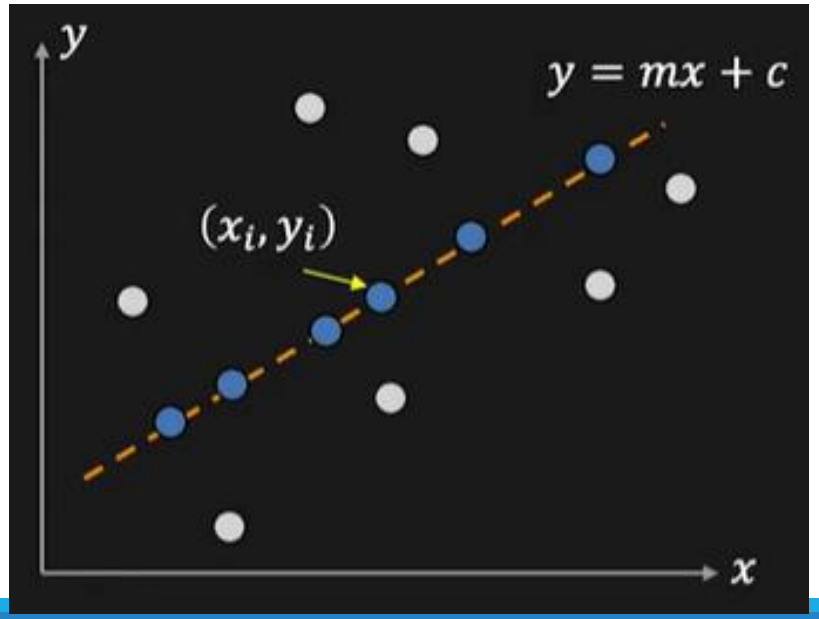
Detecting a Line using Hough Transform

Consider a point (x_i, y_i)

$$y_i = mx_i + c$$

Or

$$c = -mx_i + y_i$$



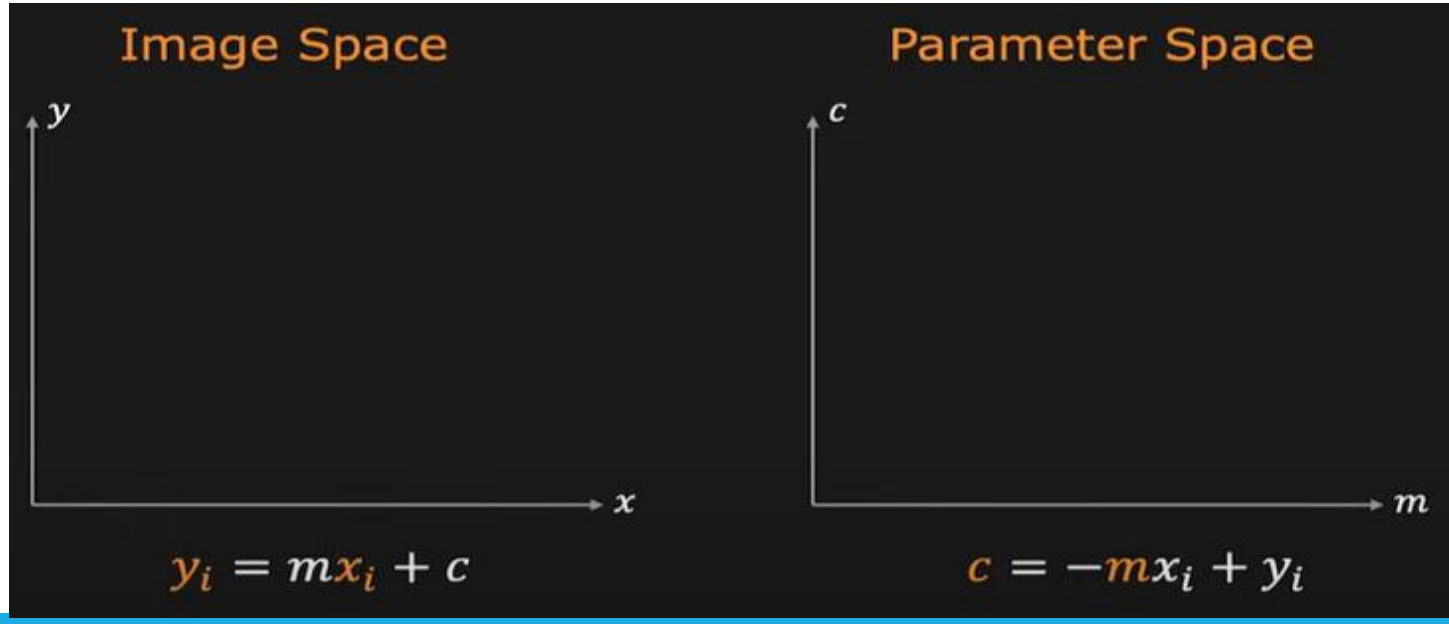


Image Space and Parameter Space

Let’s consider a straight line in the image space

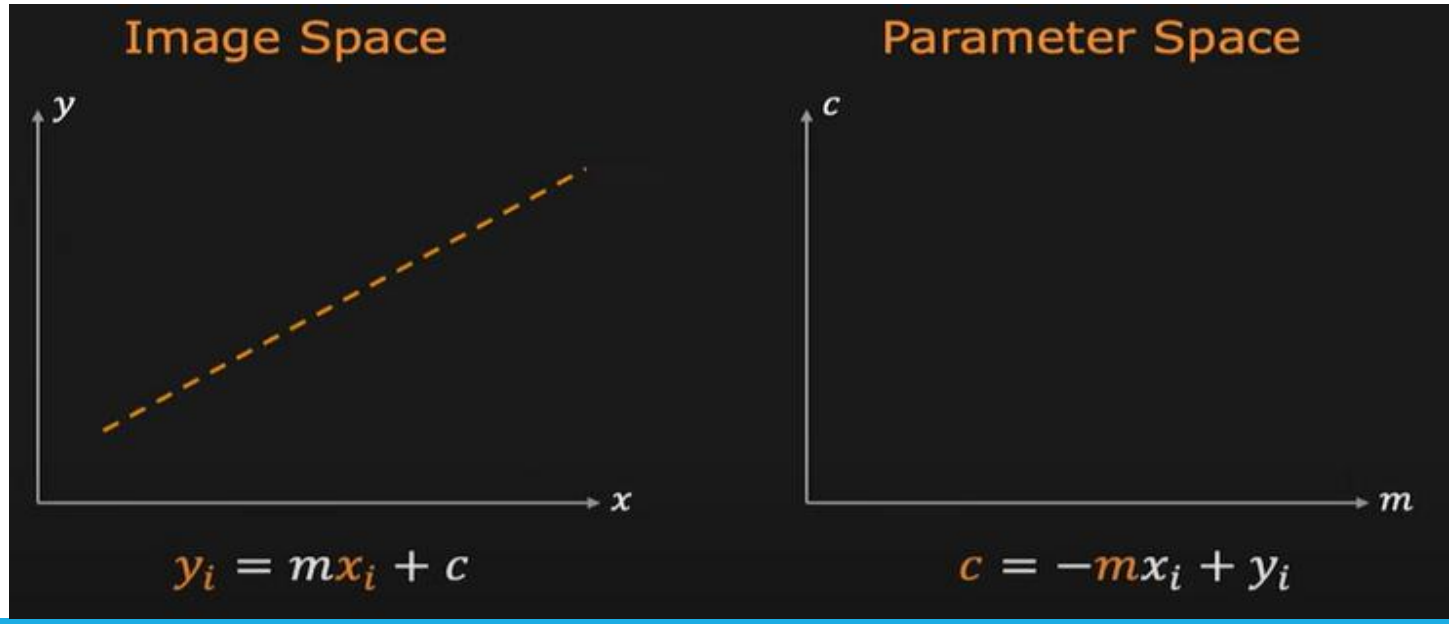


Image Space and Parameter Space

Let's consider a point (x_i, y_i) on this straight line

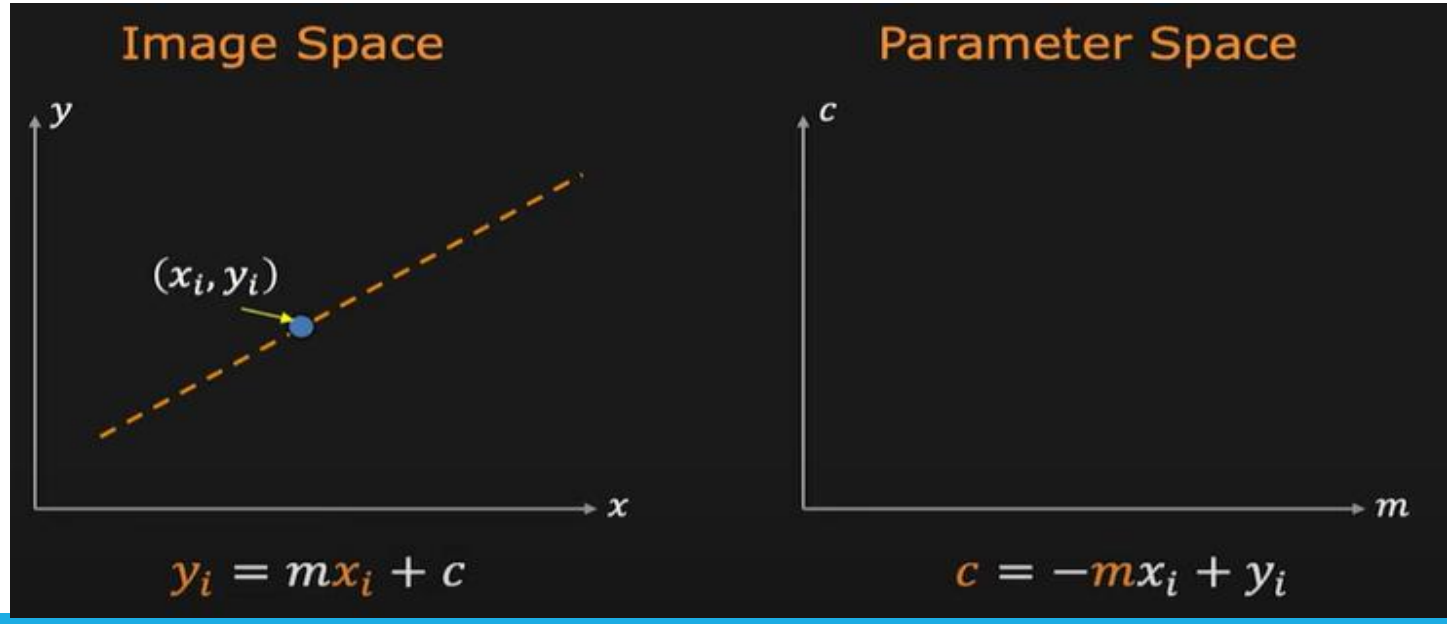


Image Space and Parameter Space

- When this point (x_i, y_i) is plugged into the other equation in Parameter Space
- Line in the m-c space corresponds to all the lines that pass through (x_i, y_i) in the image space

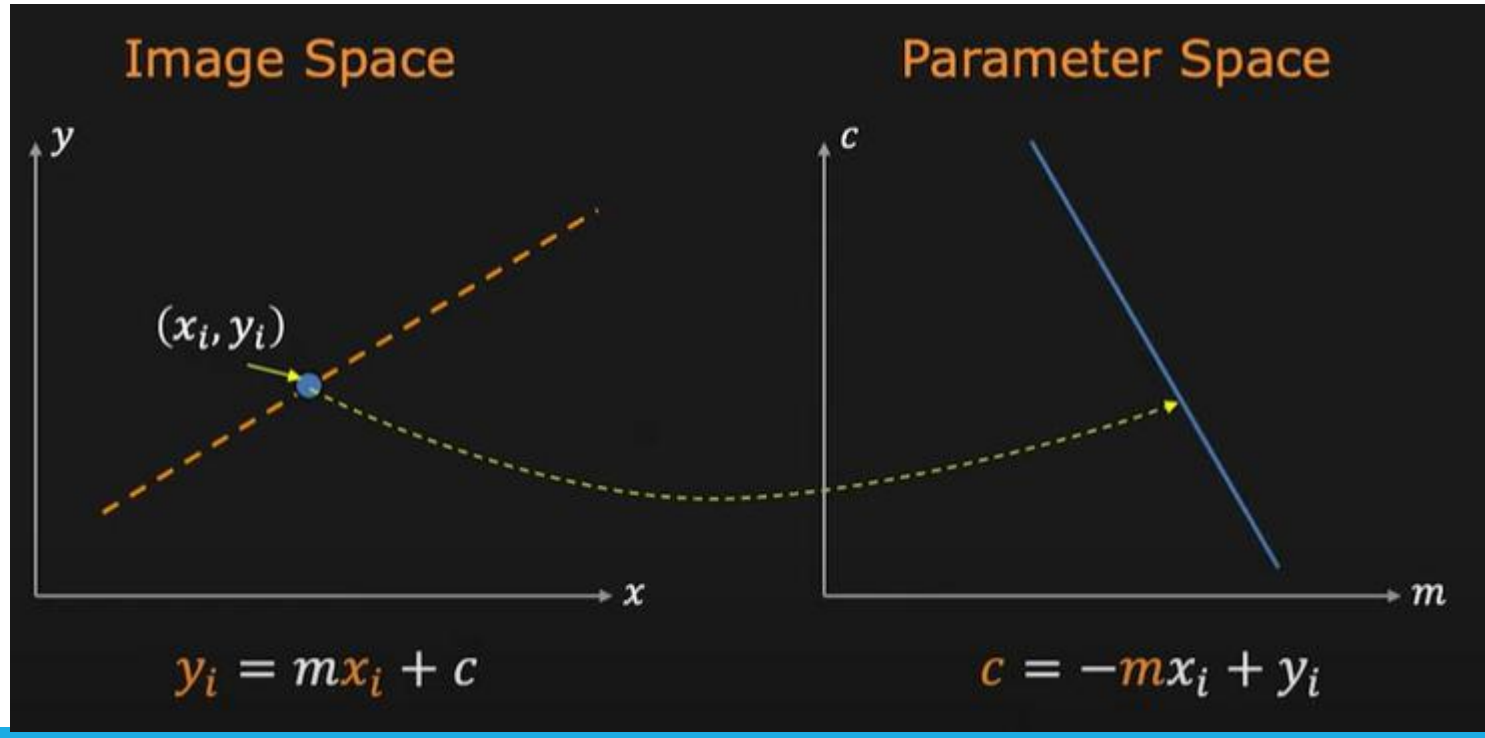


Image Space and Parameter Space

Let's consider another point

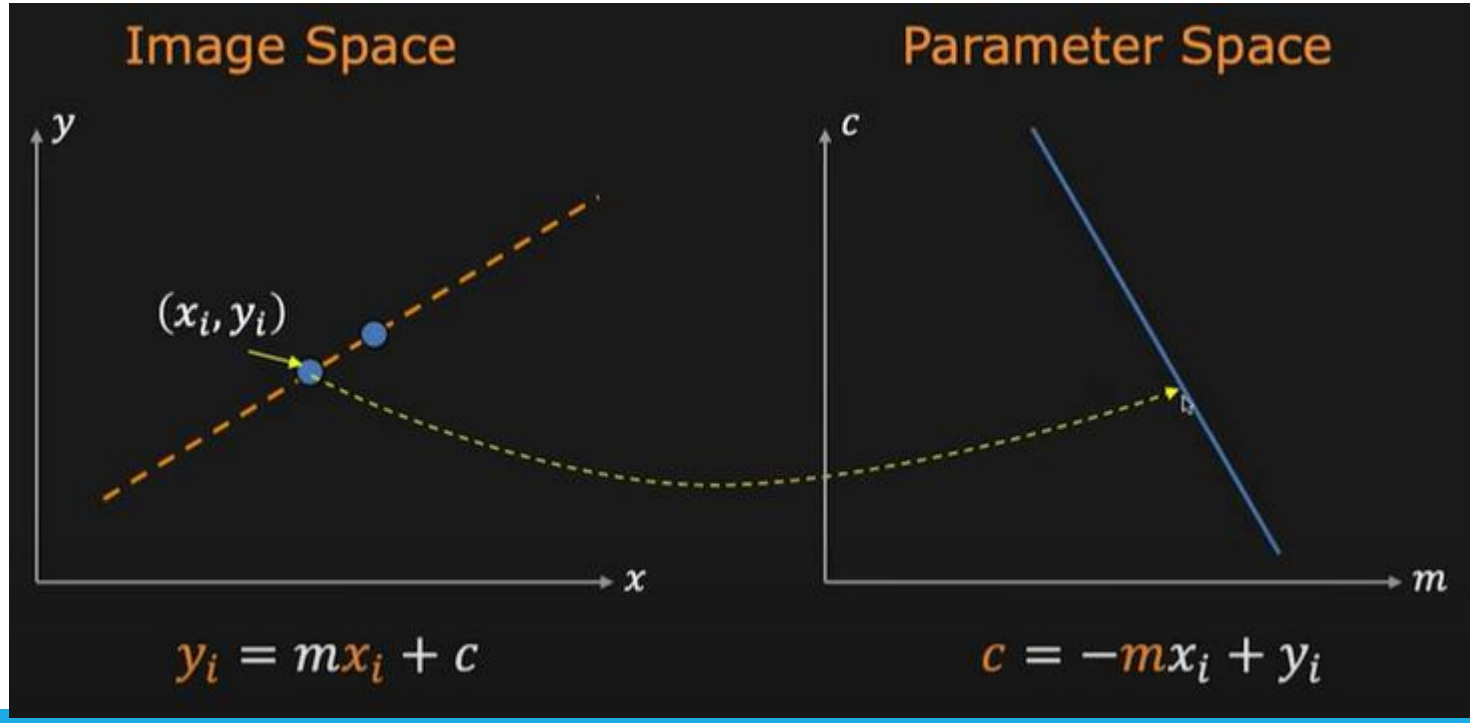


Image Space and Parameter Space

This new point gives us another line in the m-c space

This line corresponds to all m-c values in the parameter space

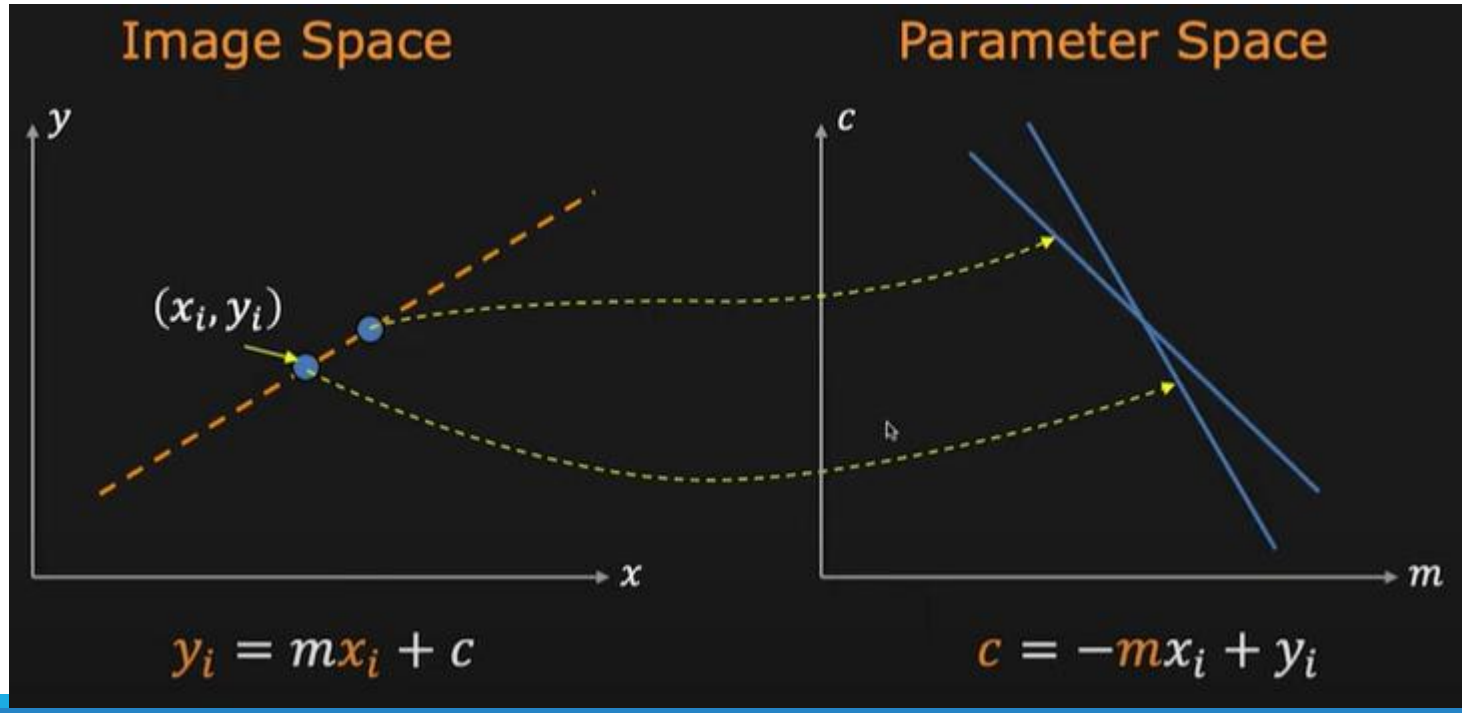


Image Space and Parameter Space

There is only one line that both the points in image space share and that is the point of intersection in the parameter space

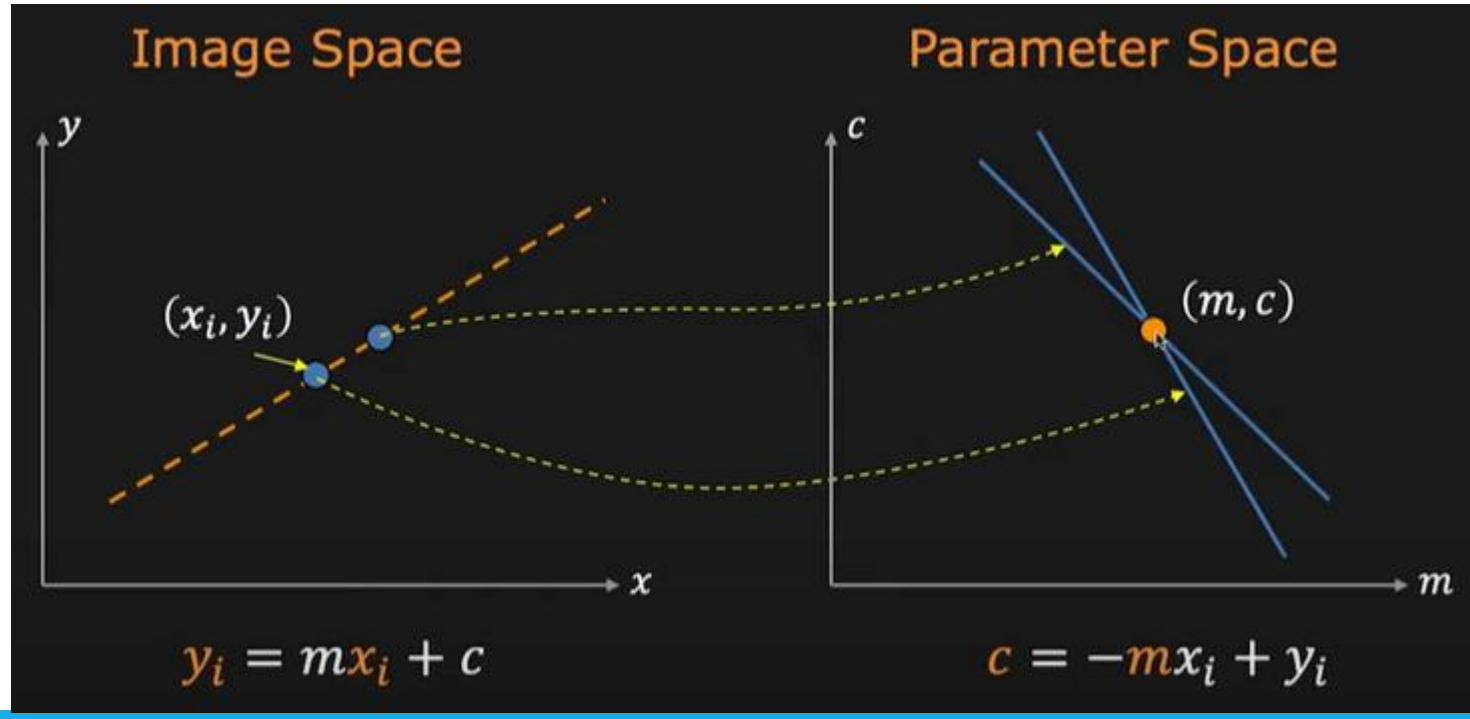


Image Space and Parameter Space

If we add more points, there will be more lines in the parameter space

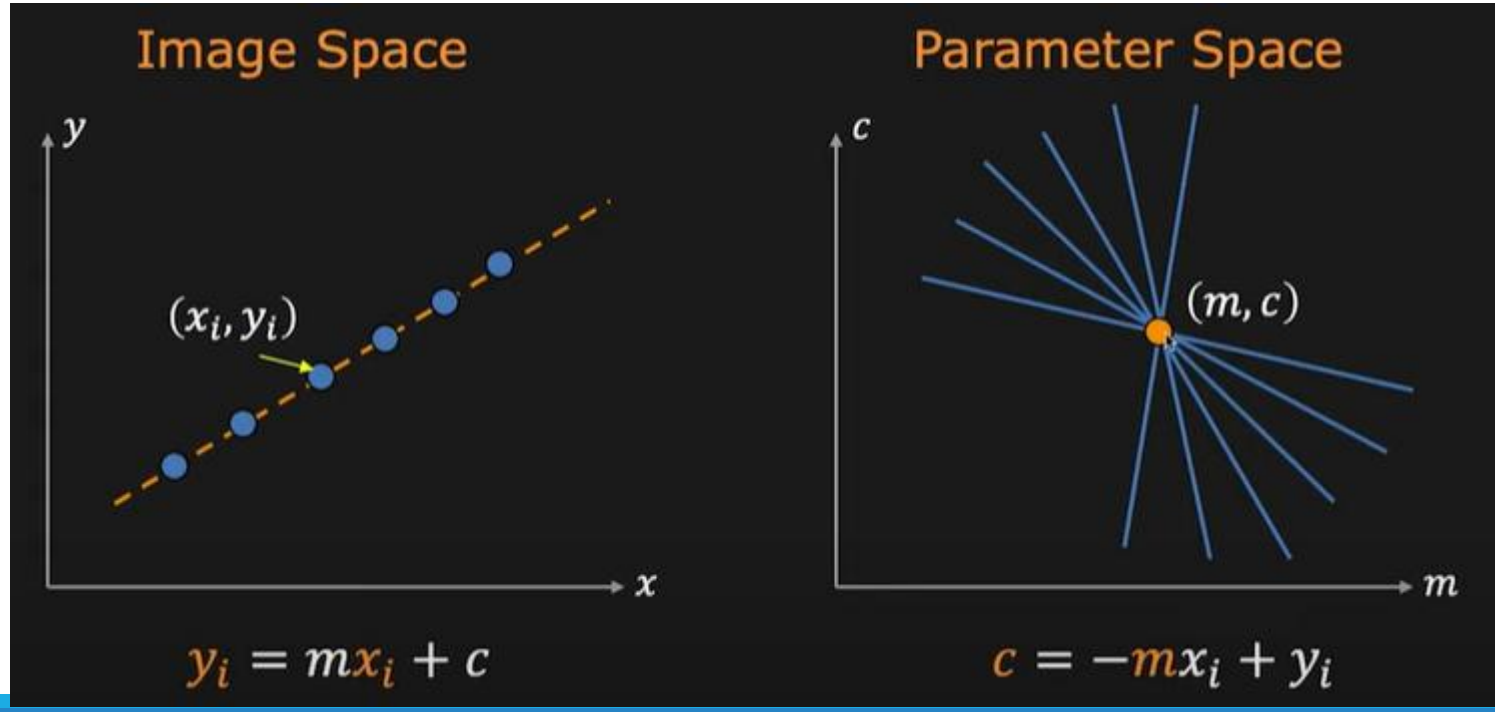


Image Space and Parameter Space

If we take another point that is not on the straight line, it will also lead to a line in the m-c space, but it will not belong to the straight line

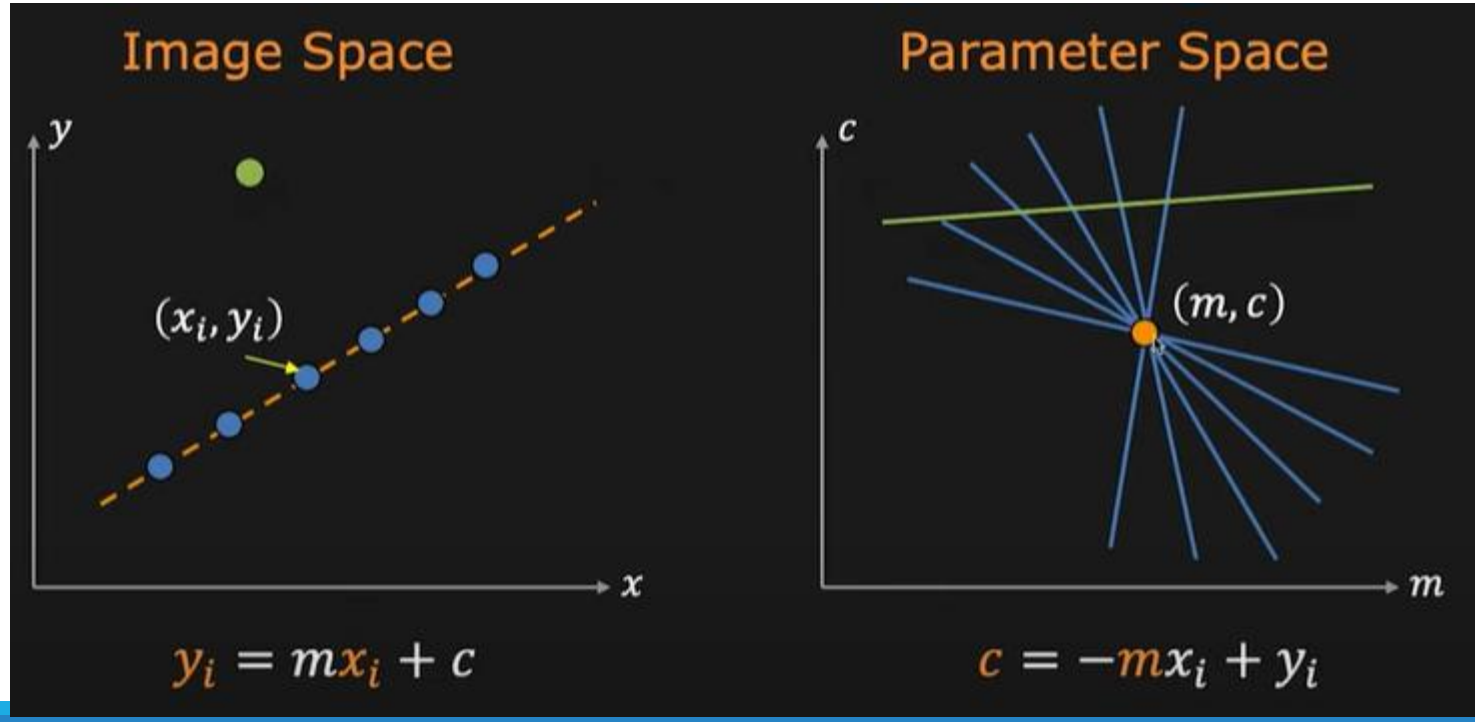


Image Space and Parameter Space

Point in image space maps to a Line in Parameter space

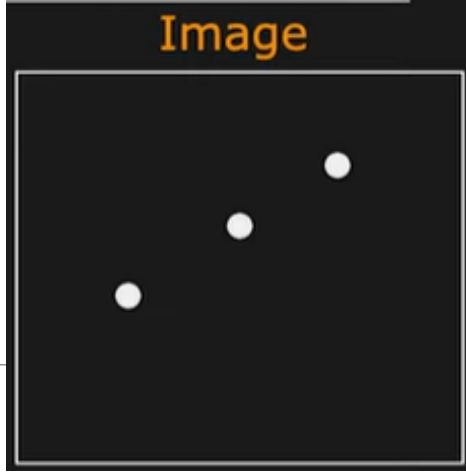
Line in image space maps to a Point in Parameter space

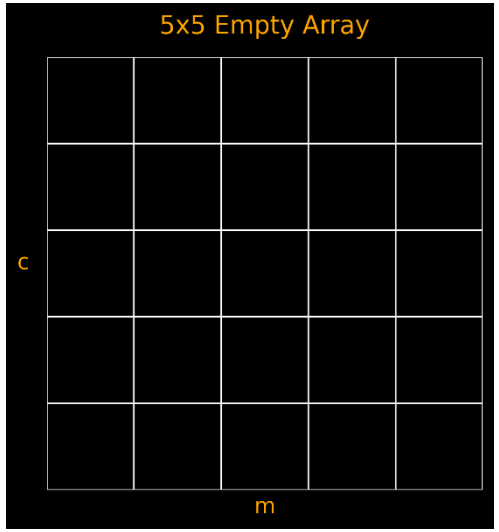
Line Detection Algorithm

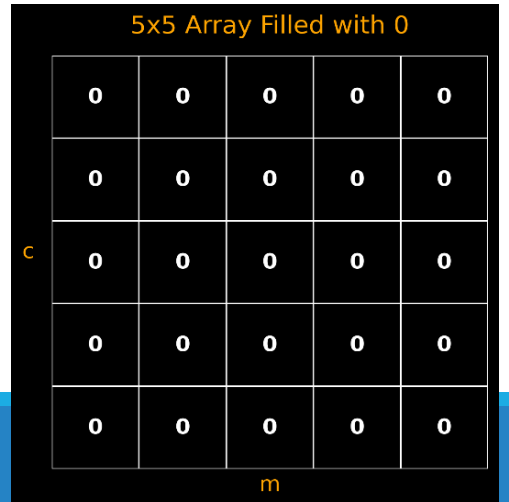
Step 1: Quantize parameter space (m, c)

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

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







Line Detection Algorithm

Step 1: Quantize parameter space (m, c)

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

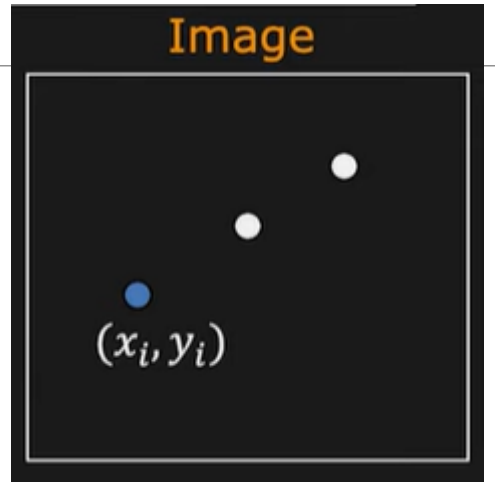
Step 3: Set $A(m, c) = 0$ for all (m, c)

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

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

if (m, c) lies on the line:

$$c = -mx_i + y_i$$



5x5 Array: Off-diagonal (anti-diagonal) = 1

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

c

m

Step-by-step with (m, c)

for each edge pixel (x_i, y_i) , add one vote to every$(\boldsymbol{m}, \boldsymbol{c})$ bin that could produce a line through that point.

1. **Choose bin grids (quantize).**

   Example (5×5):

   $m \in \{-2, -1, 0, 1, 2\}$ (5 bins),

   $c \in \{0, 1, 2, 3, 4\}$ (5 bins).

2. **Init the accumulator.**

   A is a 5×5 zero matrix where rows index m and columns index c.

3. **For one edge point**(x_i, y_i)**:**

   Loop over the **m** bins, compute the **c** that makes a line through the point:

$$c = y_i - m\, x_i$$

Quantize that c to the nearest/containing c-bin (round or floor—pick one consistently) and increment that cell.

Take $(x_i, y_i) = (1, 2)$
For each m:

m	$c = y_i - mx_i$	Bin c	cell incremented
-2	4	4	$A[m = -2, c = 4] \mathrel{+}= 1$
-1	3	3	$A[-1, 3] \mathrel{+}= 1$
0	2	2	$A[0, 2] \mathrel{+}= 1$
1	1	1	$A[1, 1] \mathrel{+}= 1$
2	0	0	$A[2, 0] \mathrel{+}= 1$

Line Detection Algorithm

Step 1: Quantize parameter space (m, c)

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

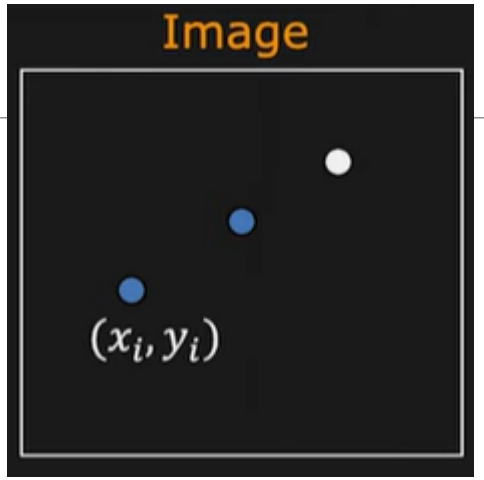
Step 3: Set $A(m, c) = 0$ for all (m, c)

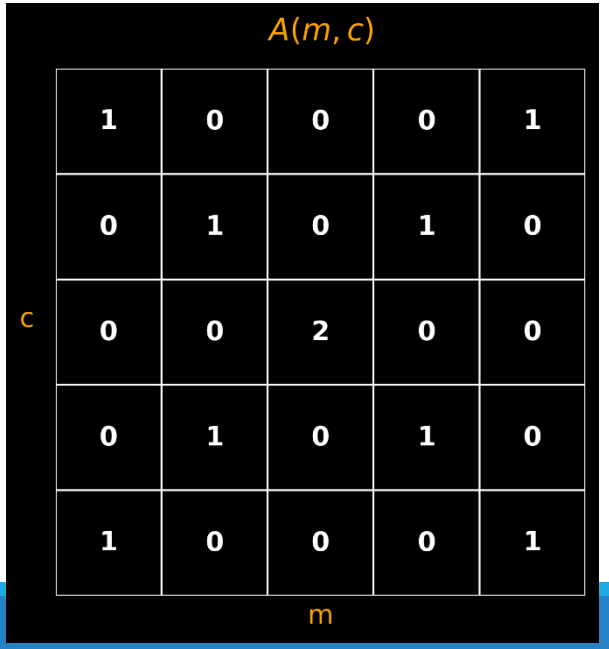
Step 4: For each edge point (x_i, y_i),

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

if (m, c) lies on the line:

$$c = -mx_i + y_i$$

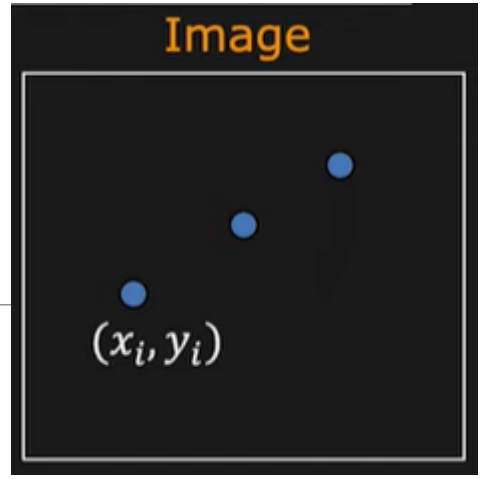




Take $(x_i, y_i) = (2, 3)$
For each m:

m	$c = 3 - 2m$	Bin c	Update
-2	7	out of range	—
-1	5	out of range	—
0	3	valid	$(\mathrm{m} = 1, \mathrm{c} = 3)$
1	1	valid	$(\mathrm{m} = 1, \mathrm{c} = 1)$
2	-1	out of range	—

Line Detection Algorithm



Based on these observation, we can design an algorithm

Step 1: Quantize parameter space (m, c)

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

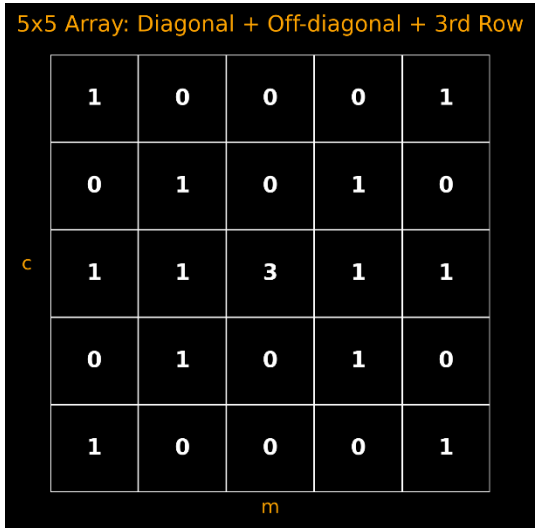
Step 3: Set $A(m, c) = 0$ for all (m, c)

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 $A(m, c)$

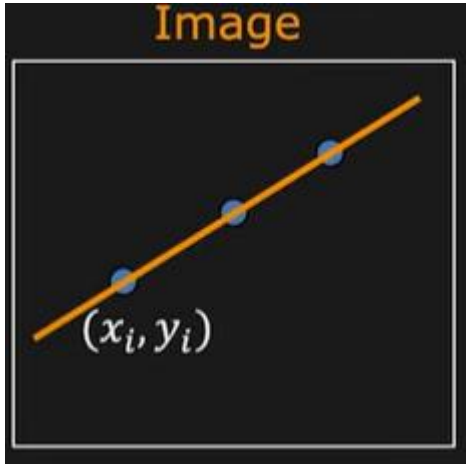


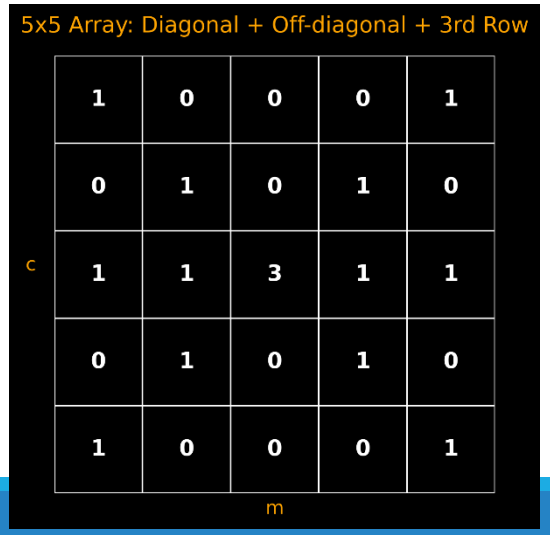
Take $(x_i, y_i) = (3, 4)$

m	$c = 4 - 3m$	Bin c	Update
-2	10	out of range	—
-1	7	out of range	—
0	4	valid	$(\mathrm{m} = 0, \mathrm{c} = 4)$
1	1	valid	$(\mathrm{m} = 1, \mathrm{c} = 1)$
2	-2	out of range	—

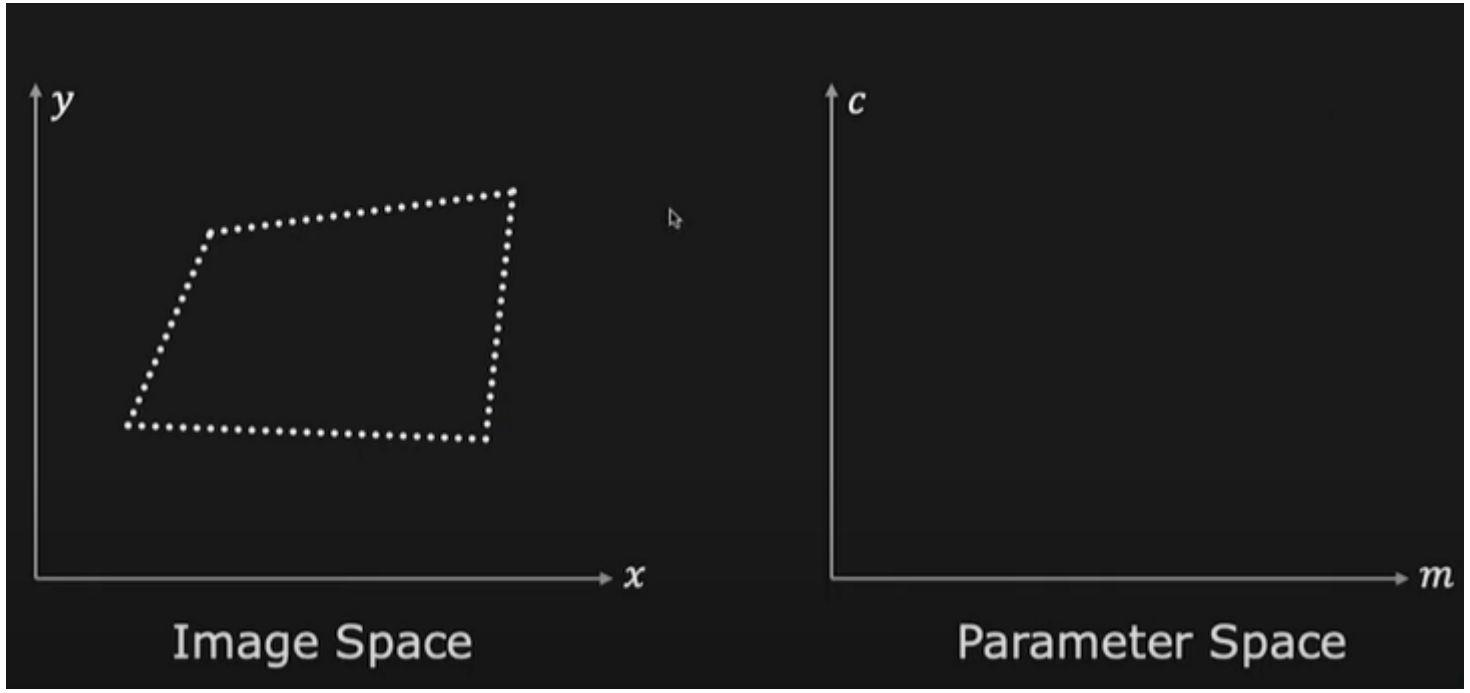
Line Detection Algorithm

So, the point of intersection will give us a straight line

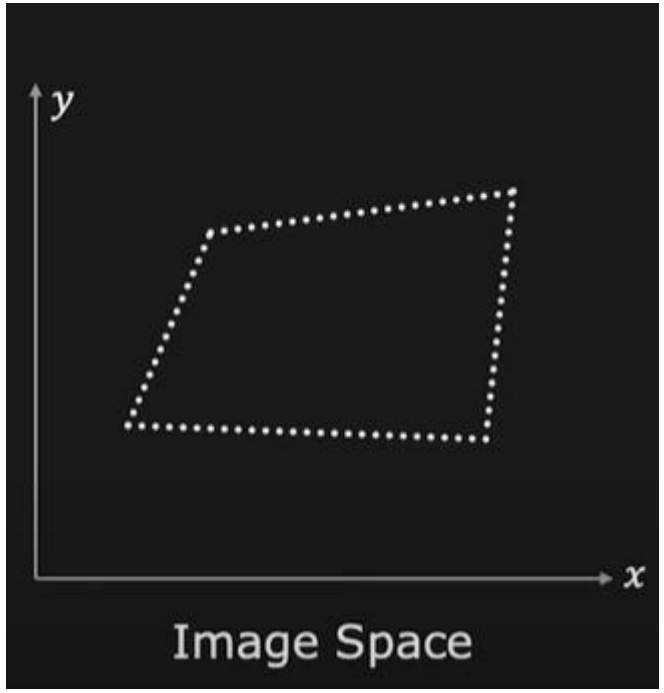


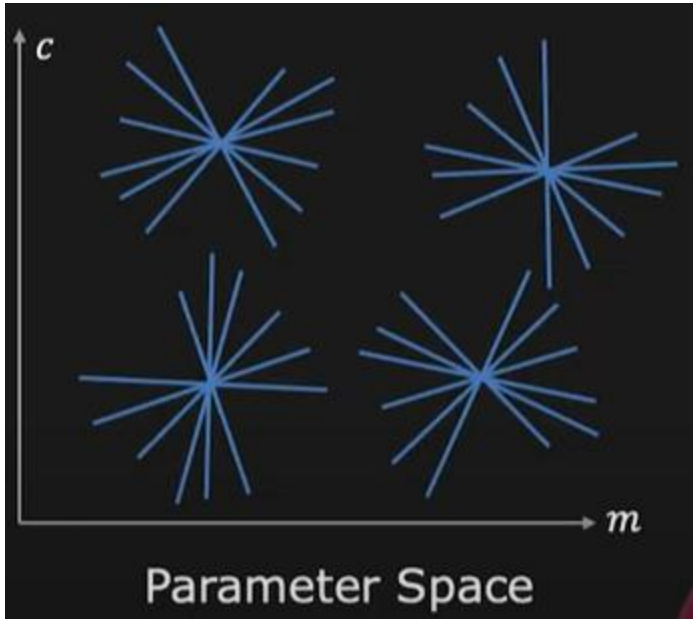


Multiple Line Detection

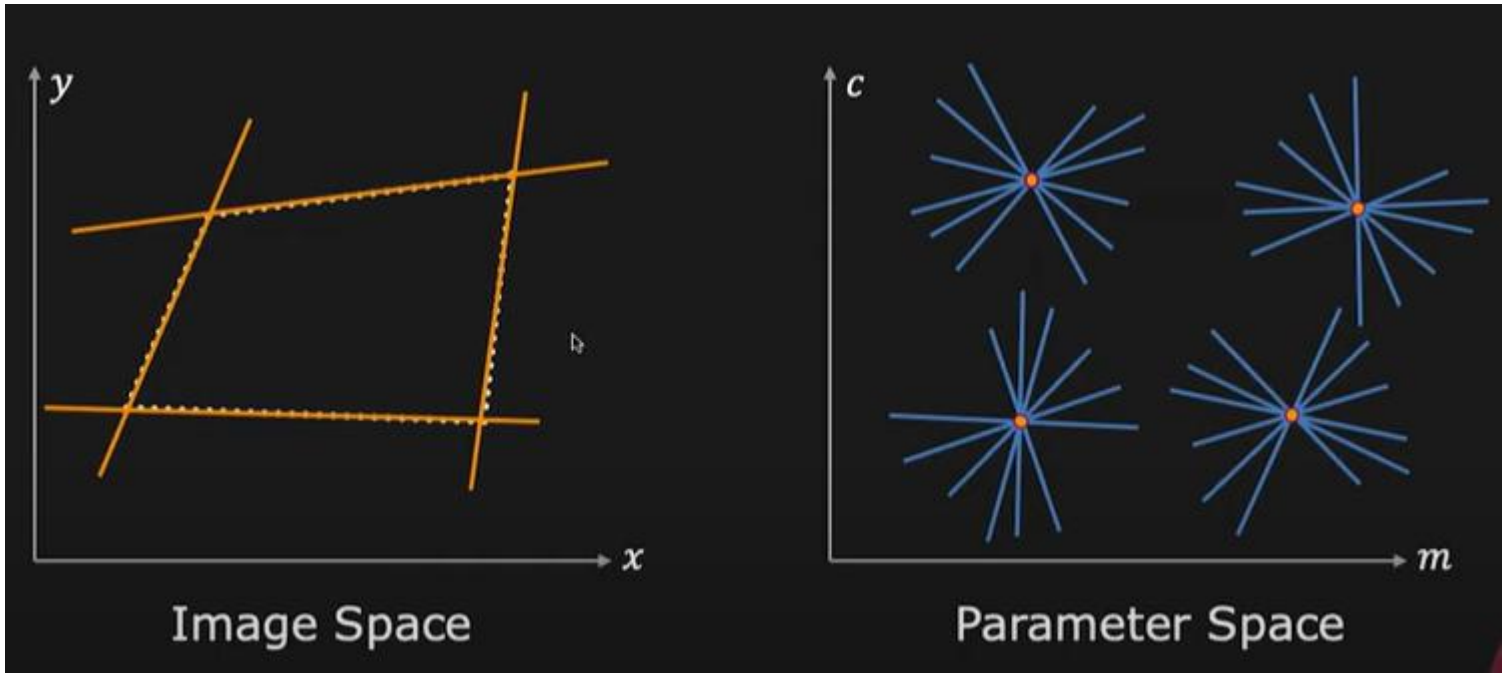


Multiple Line Detection





Multiple Line Detection

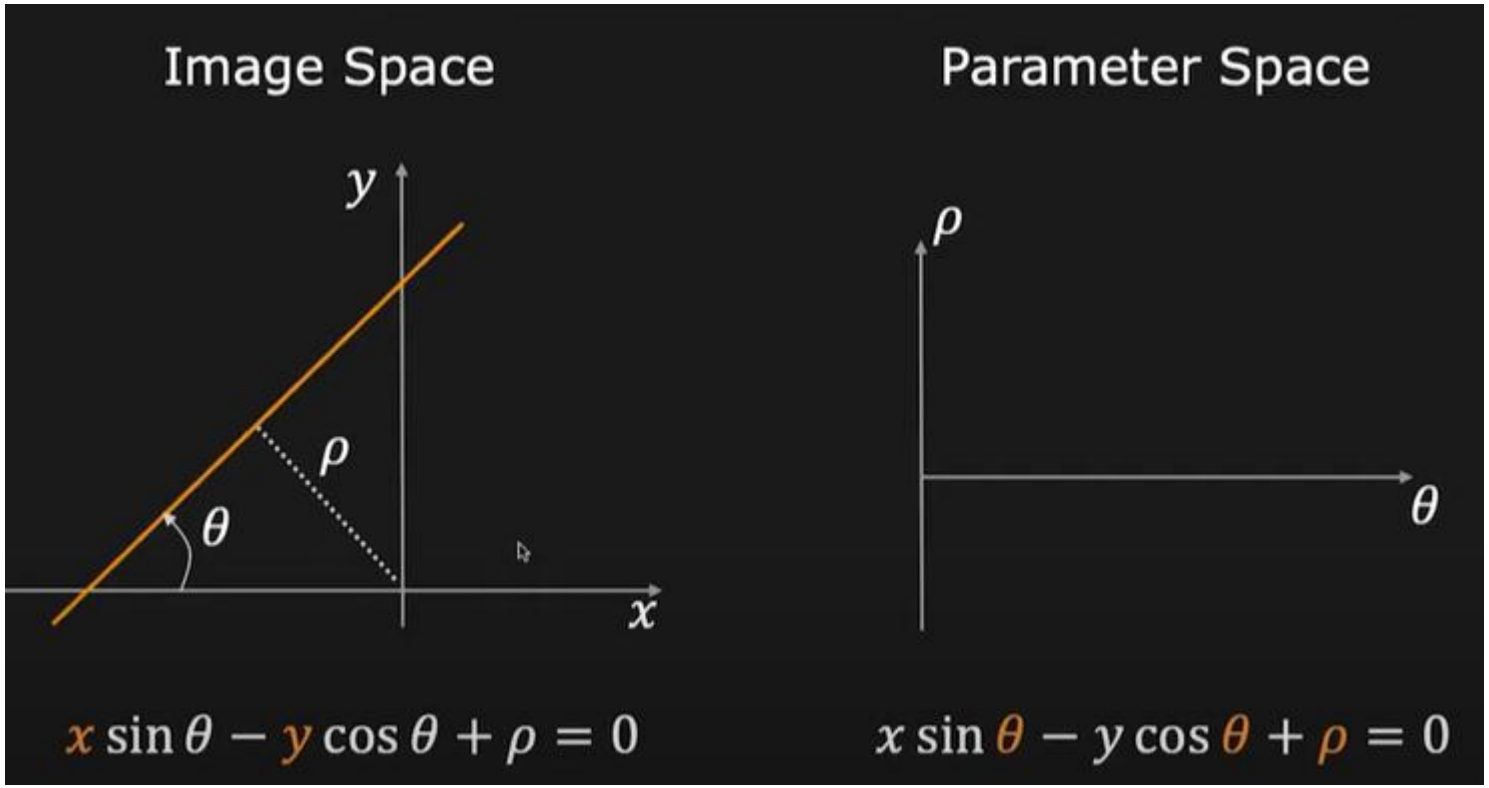


Better Parameterization

- **Issue:**
- Slope of the line $-\infty \leq m \leq \infty$
  - Large Accumulator
  - More memory and Computation

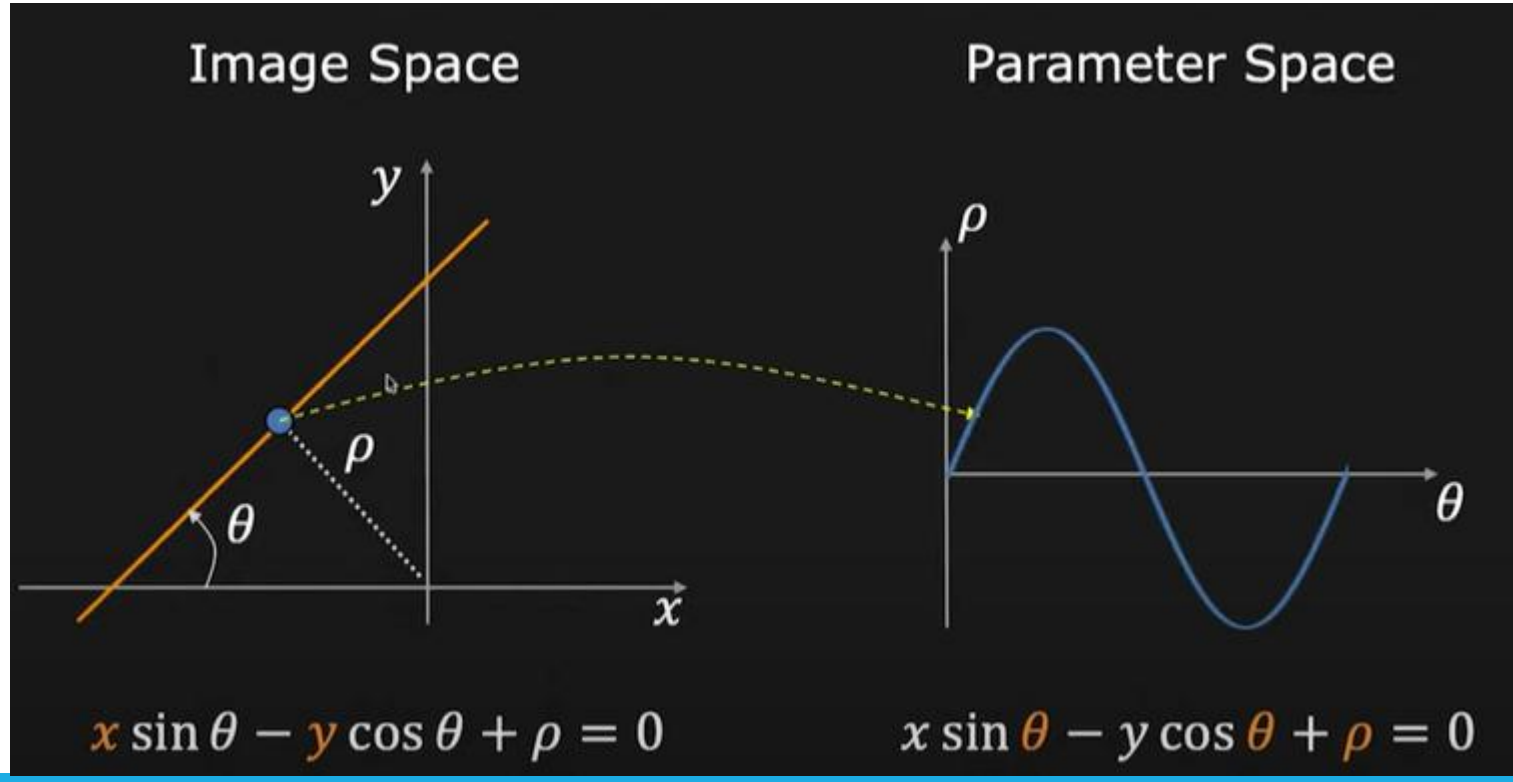
- **Solution:**
  - Use $x \sin\theta - y \cos\theta + \rho = 0$
  - Orientation θ is finite $0 \leq \theta \leq \pi$
  - Distance ρ is finite (distance of the straight line from the origine)

Better Parameterization

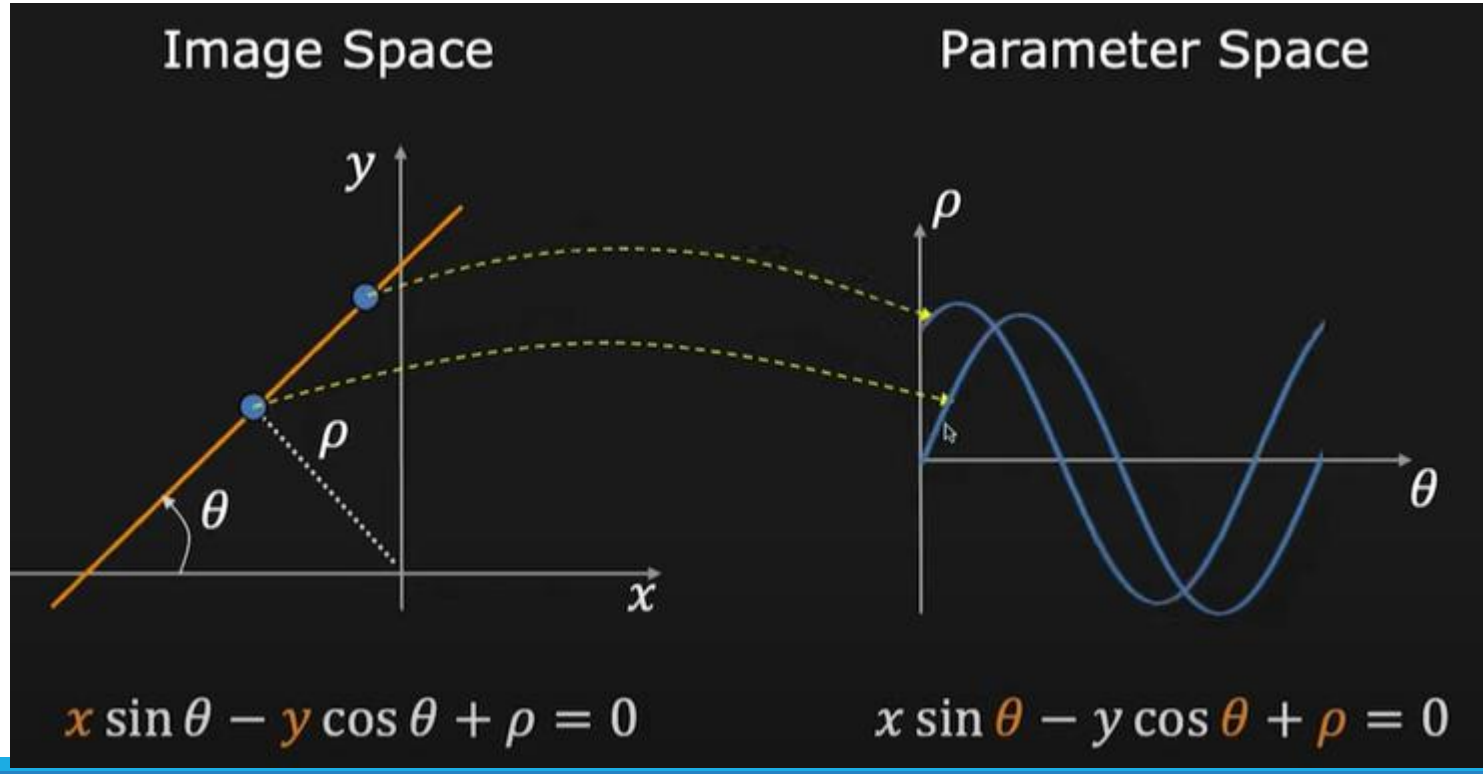


Working with Hough Transform

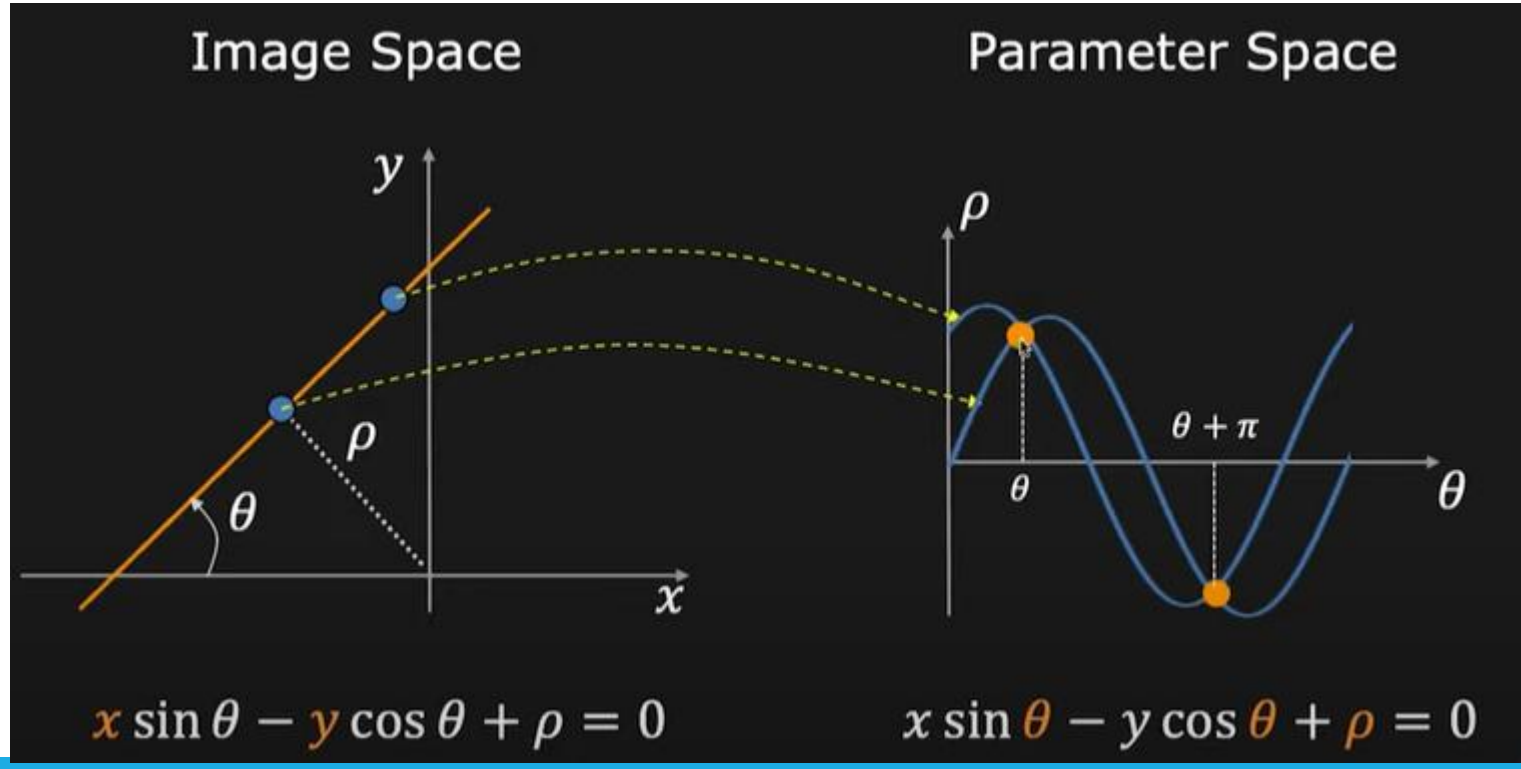
For a single point in an image space, we get a sinusoidal in the parameter space



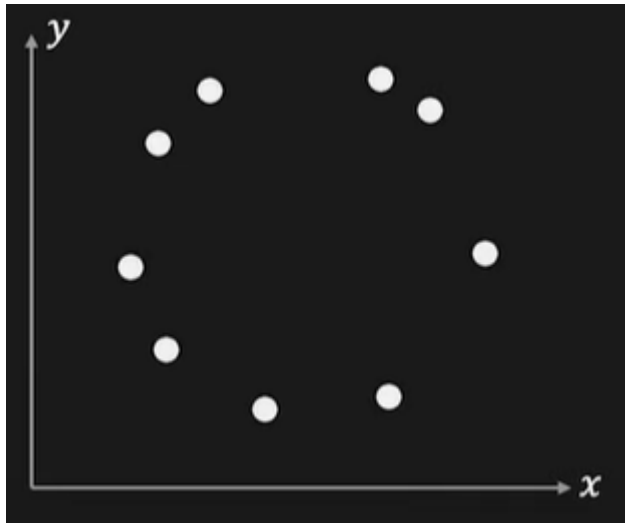
Working with Hough Transform



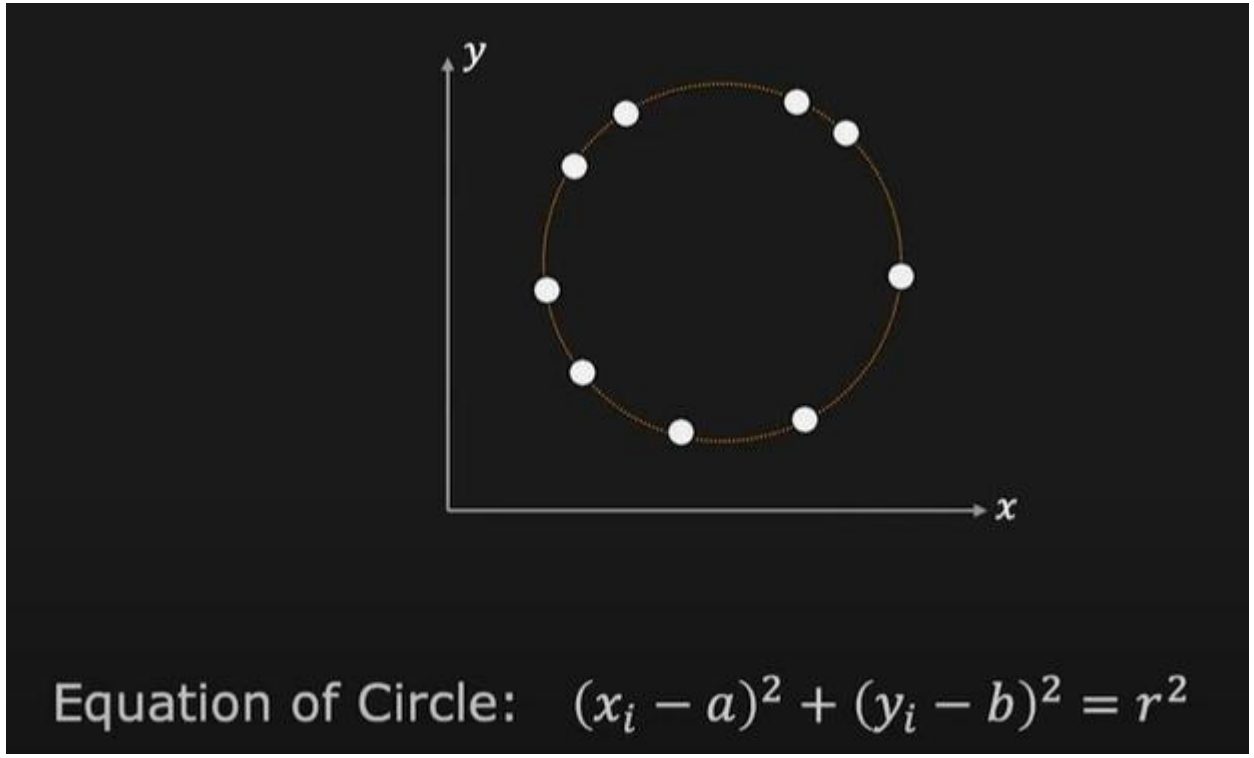
Working with Hough Transform



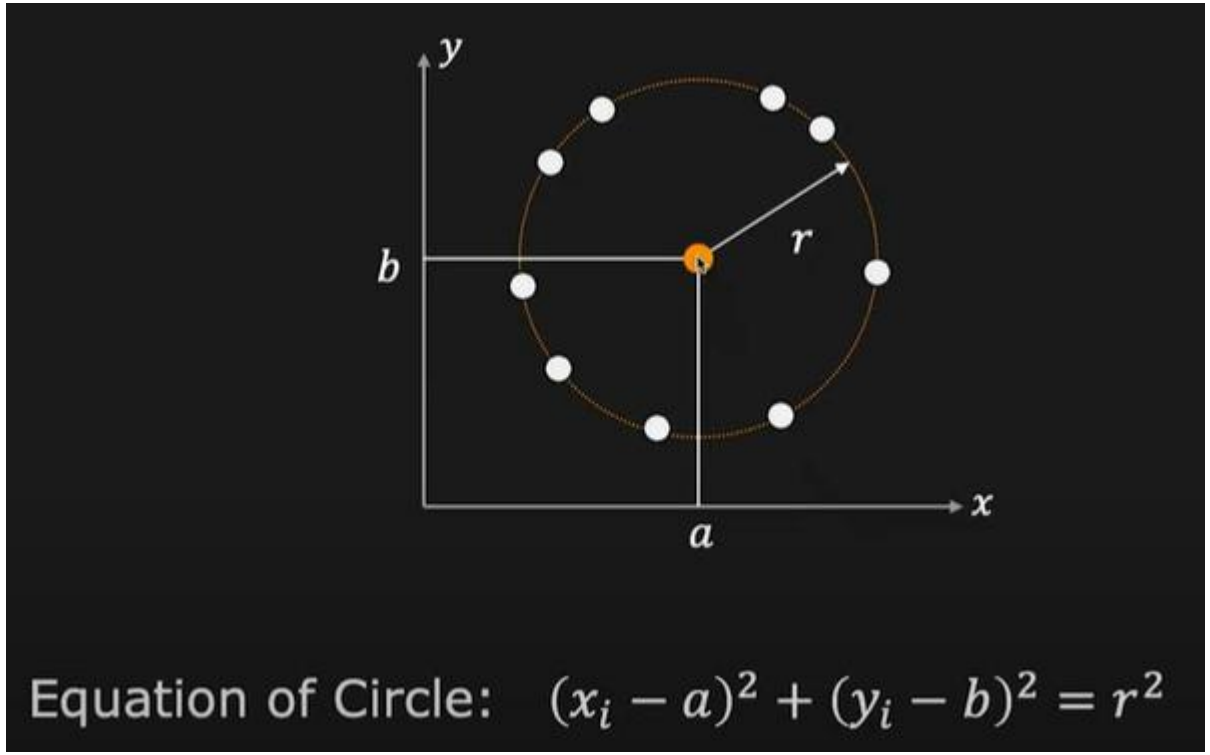
Circle Detection



Circle Detection

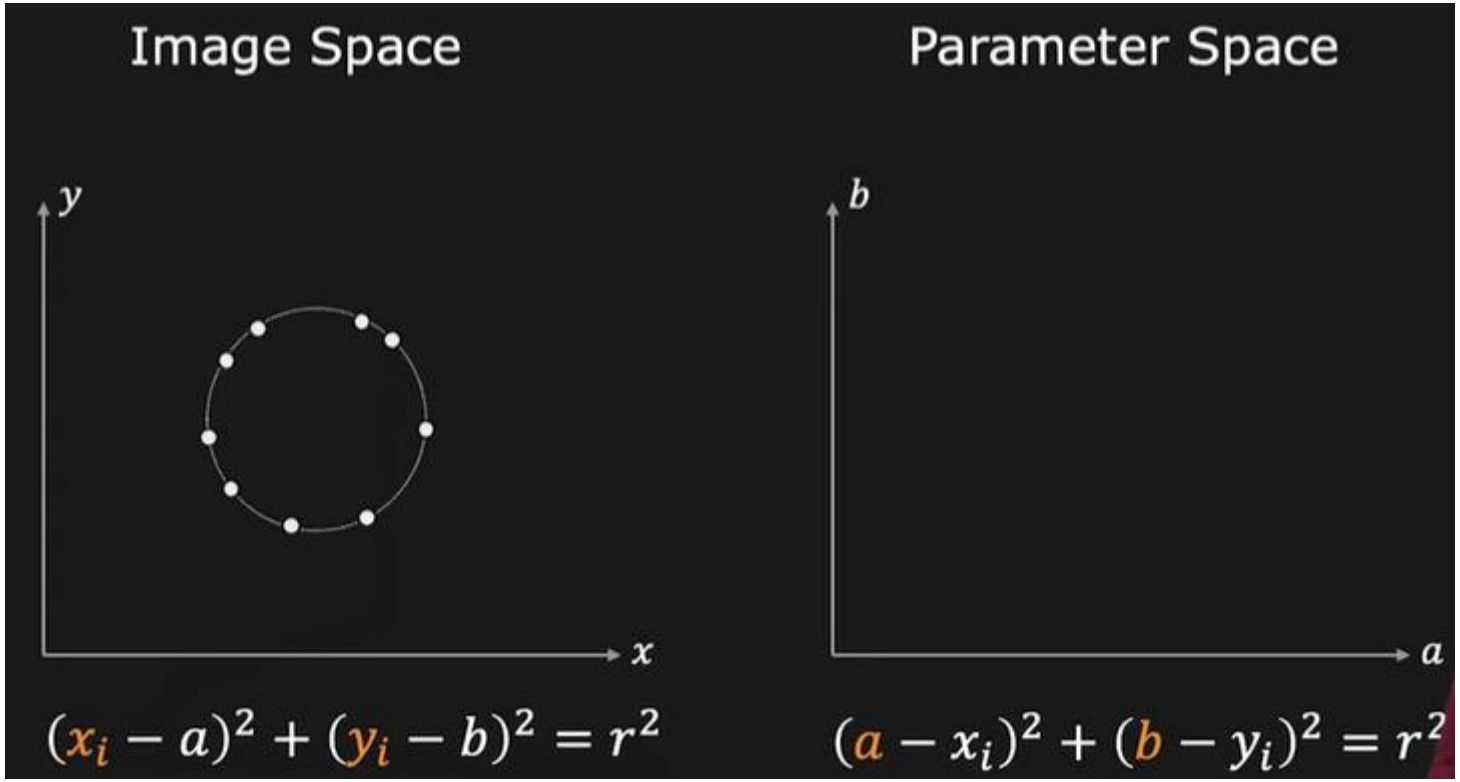


Circle Detection

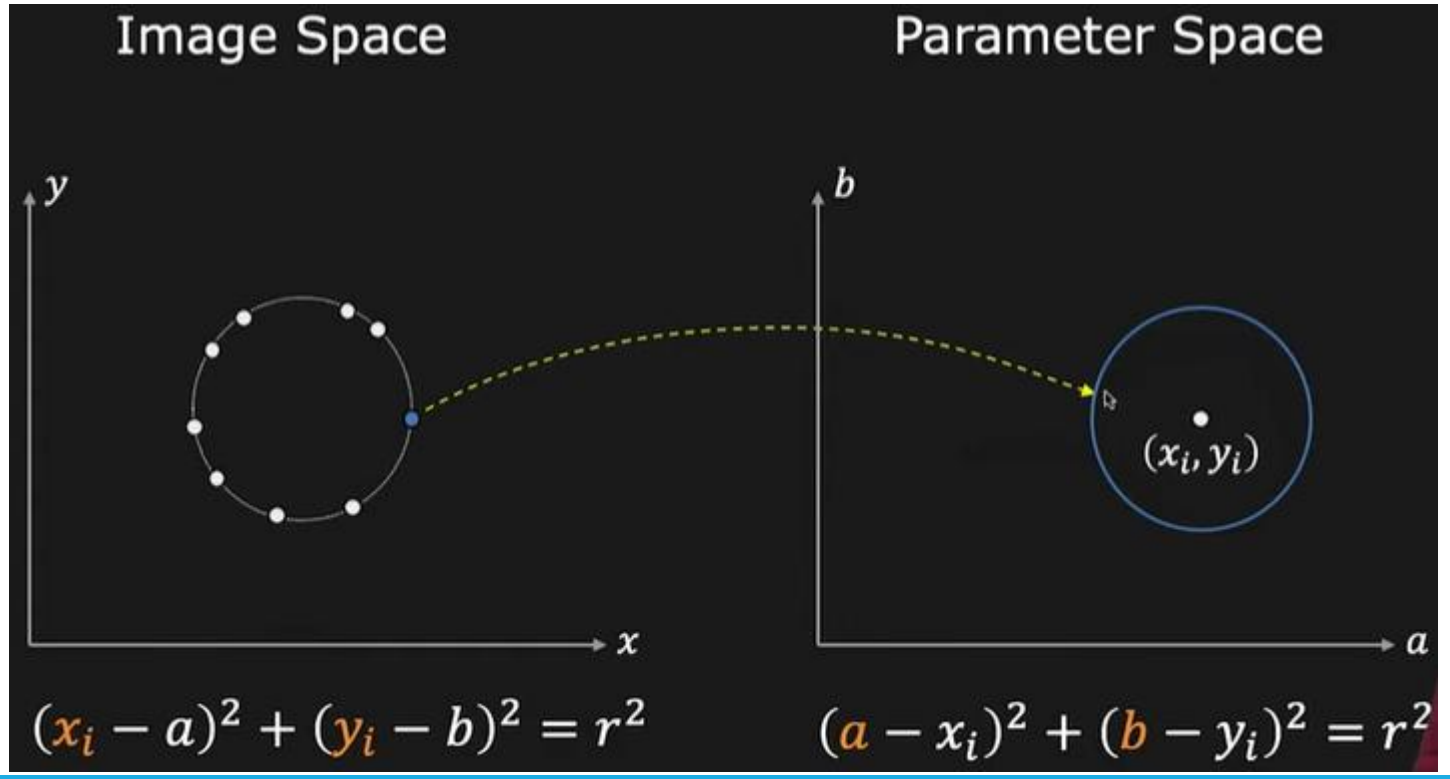


Circle Detection

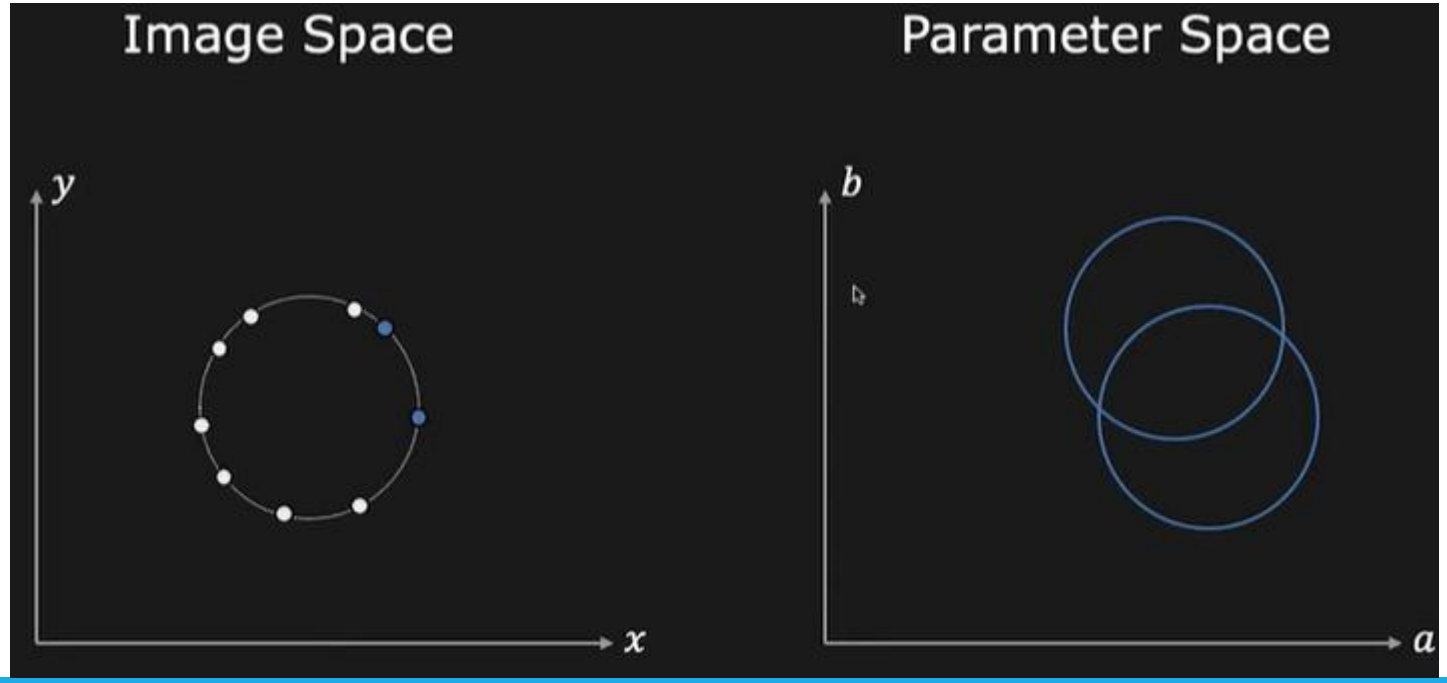
If radius r is known Accumulator array $A(a, b)$



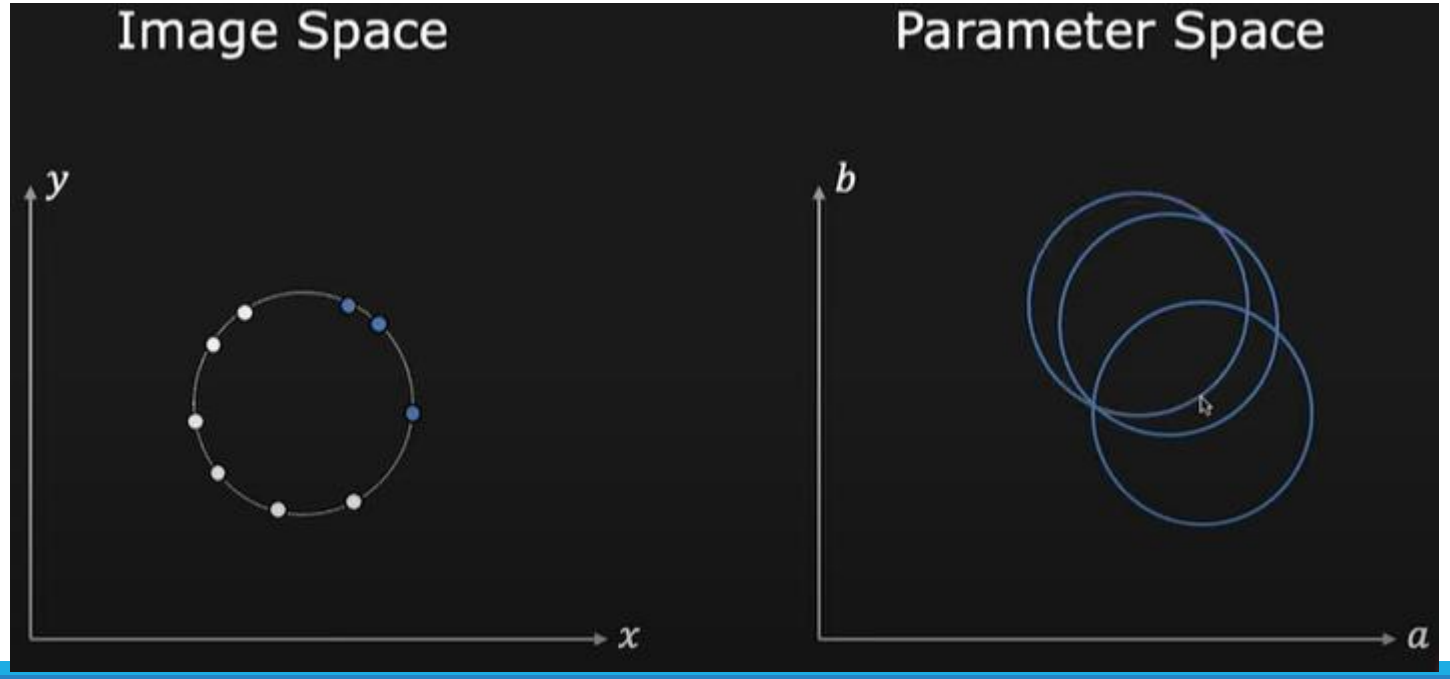
Circle Detection



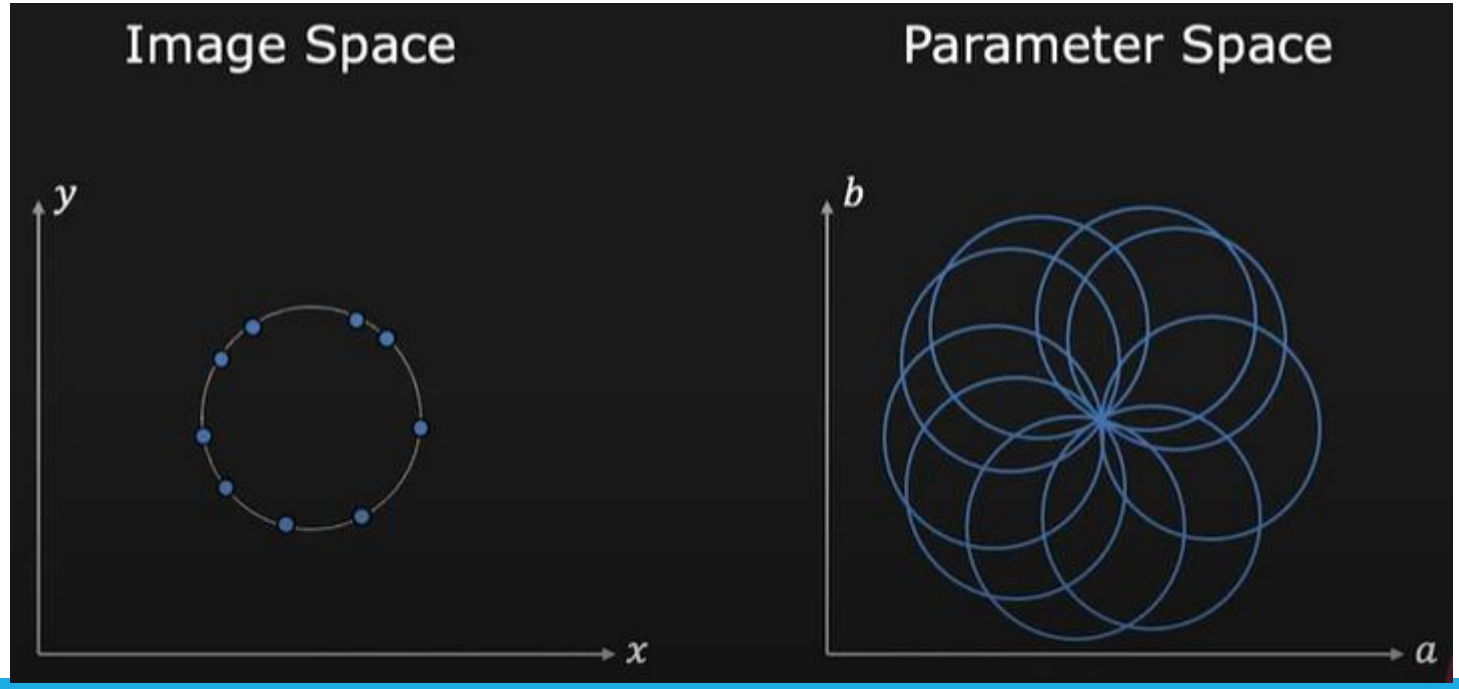
Circle Detection



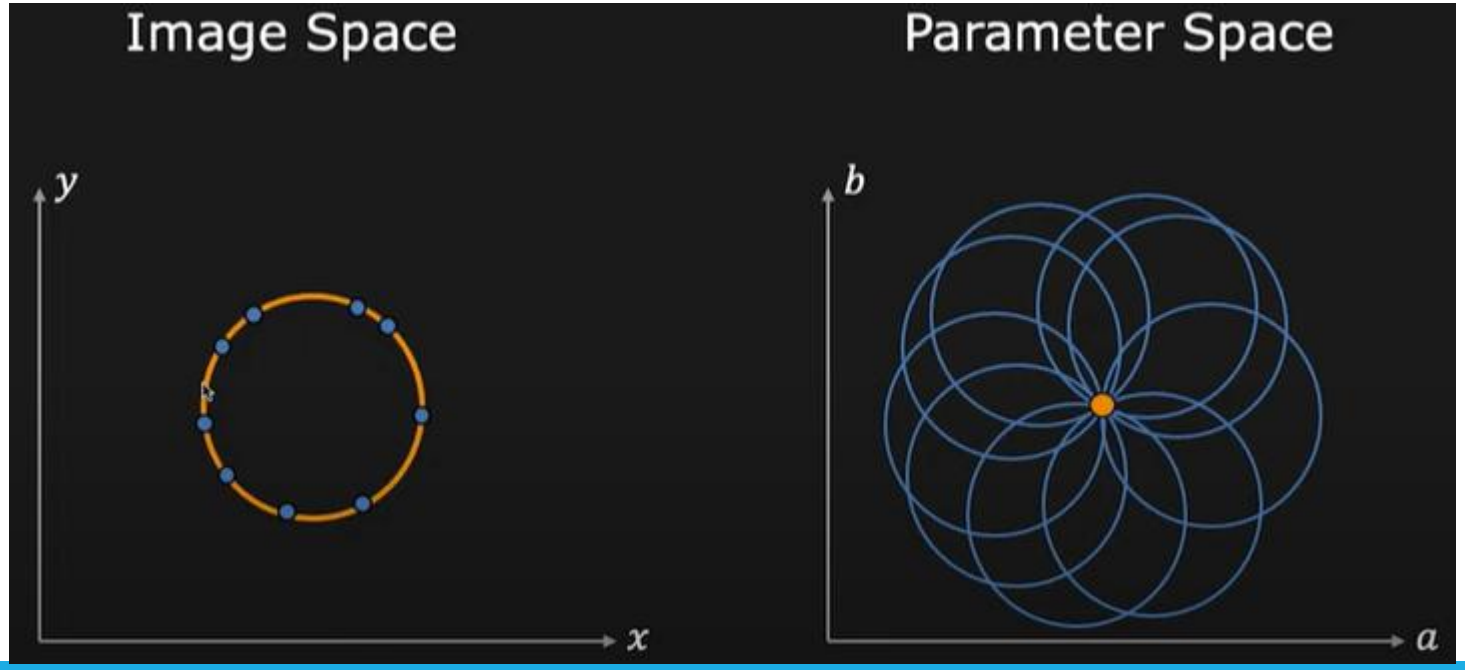
Circle Detection



Circle Detection

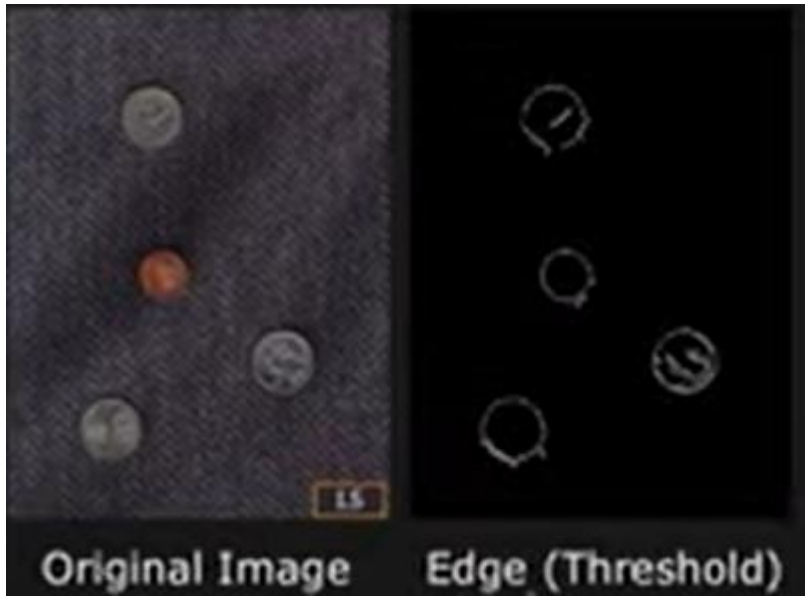


Circle Detection

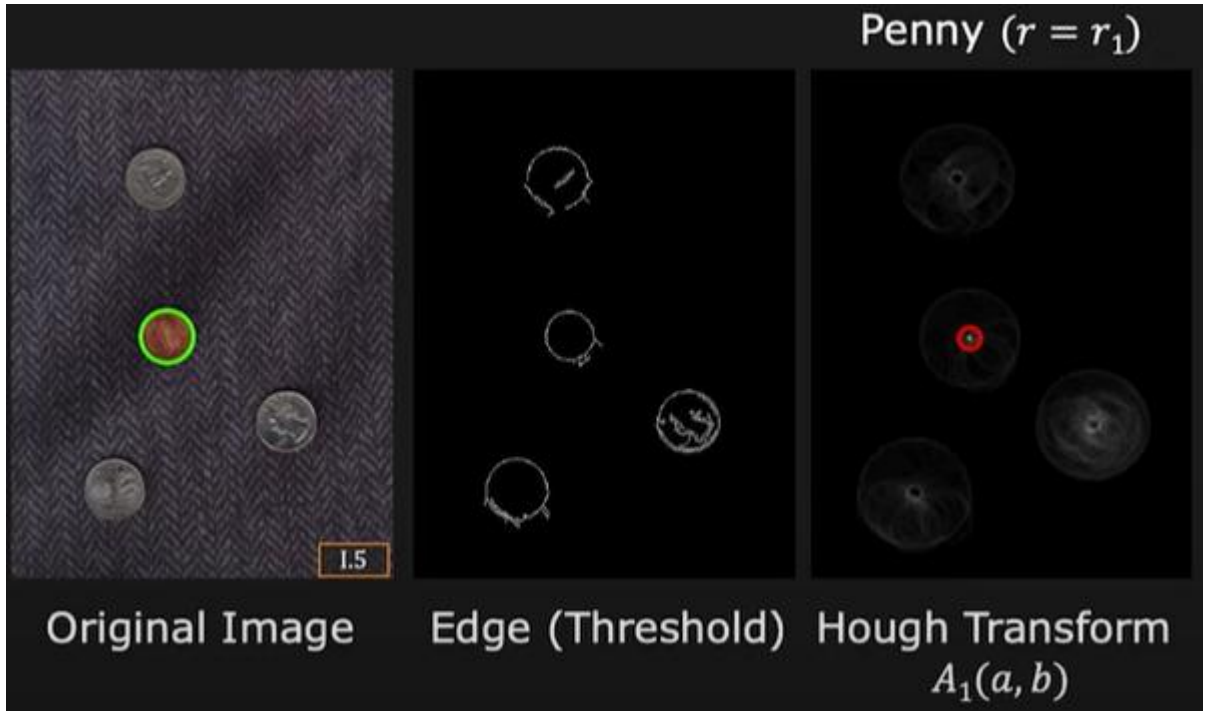


Example

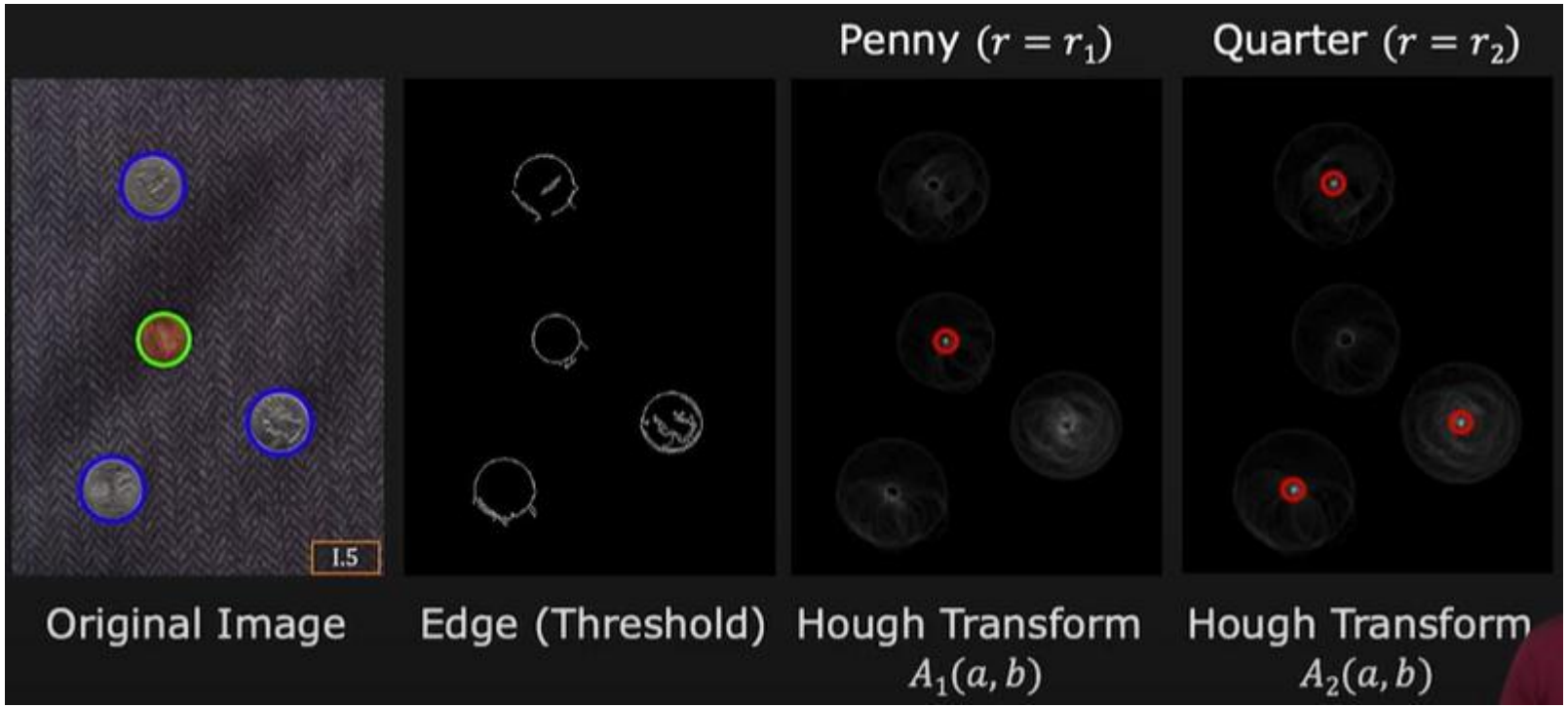
Example



Example


Penny ($r = r_1$)
1.5
Original Image
Edge (Threshold)
Hough Transform
$A_1(a, b)$

Example


Penny ($r = r_1$)
Quarter ($r = r_2$)
Original Image
Edge (Threshold)
Hough Transform
$A_1(a, b)$
Hough Transform
$A_2(a, b)$

Hough Transform - Applications

Hough transform in computer vision

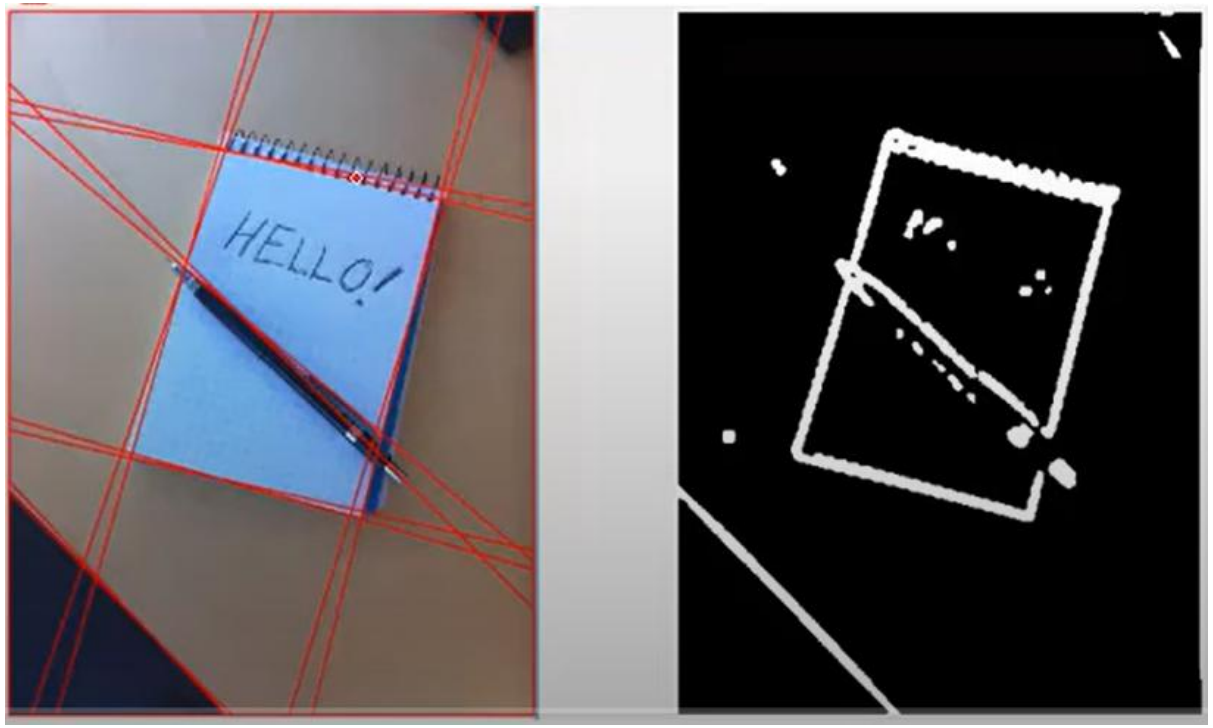
The Hough Transform is a popular technique in computer vision and image processing, used for detecting geometric shapes like lines, circles, and other parametric curves. Named after Paul Hough, who introduced the concept in 1962, the transform has evolved and found numerous applications in various domains such as medical imaging, robotics, and autonomous driving. In this article, we will discuss how Hough transformation is utilized in computer vision.

What is Hough Transform?

A feature extraction method called the Hough Transform is used to find basic shapes in a picture, like circles, lines, and ellipses. Fundamentally, it transfers these shapes' representation from the spatial domain to the parameter space, allowing for effective detection even in the face of distortions like noise or occlusion.

Example

- Detection of features such as corners and lines from images is an essential process. This can be further used for feature matching and object detection.
- The idea of the Hough Transform is that every edge point in the edge map is transformed to all possible lines that could pass through that point.

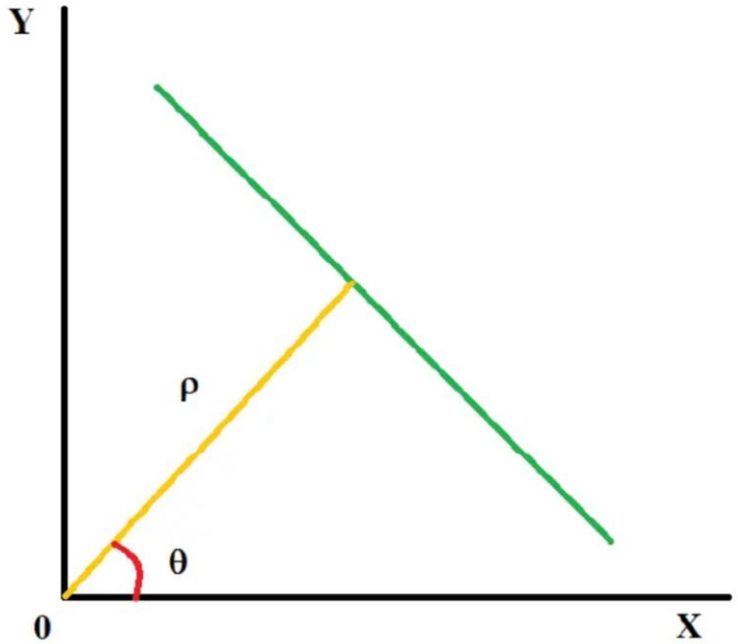


Introduction

- Hough Transform is a popular line detection algorithm that works by mapping points in an image to lines in parameter space.
- The Hough transform can detect lines of any orientation and can work well in images with a large amount of noise.
- To understand how this algorithm works we first need to understand how lines are defined in a polar system.

Line Representation

A line is described by ρ the perpendicular distance from the origin and θ the angle made by the perpendicular with the axis as shown in figure below.



Equation of the Line

- The equation of the line is therefore given as.

$$y = -\frac{\cos\theta}{\sin\theta}x + \frac{\rho}{\sin\theta}$$

- By rearranging the equation, we get

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

- From the above equation, we can say that all the points having the same values of ρ and θ constitute a single line.
- The basis of our algorithm is computing the value of ρ for each point in the image for all possible values of θ.

Hough Transform

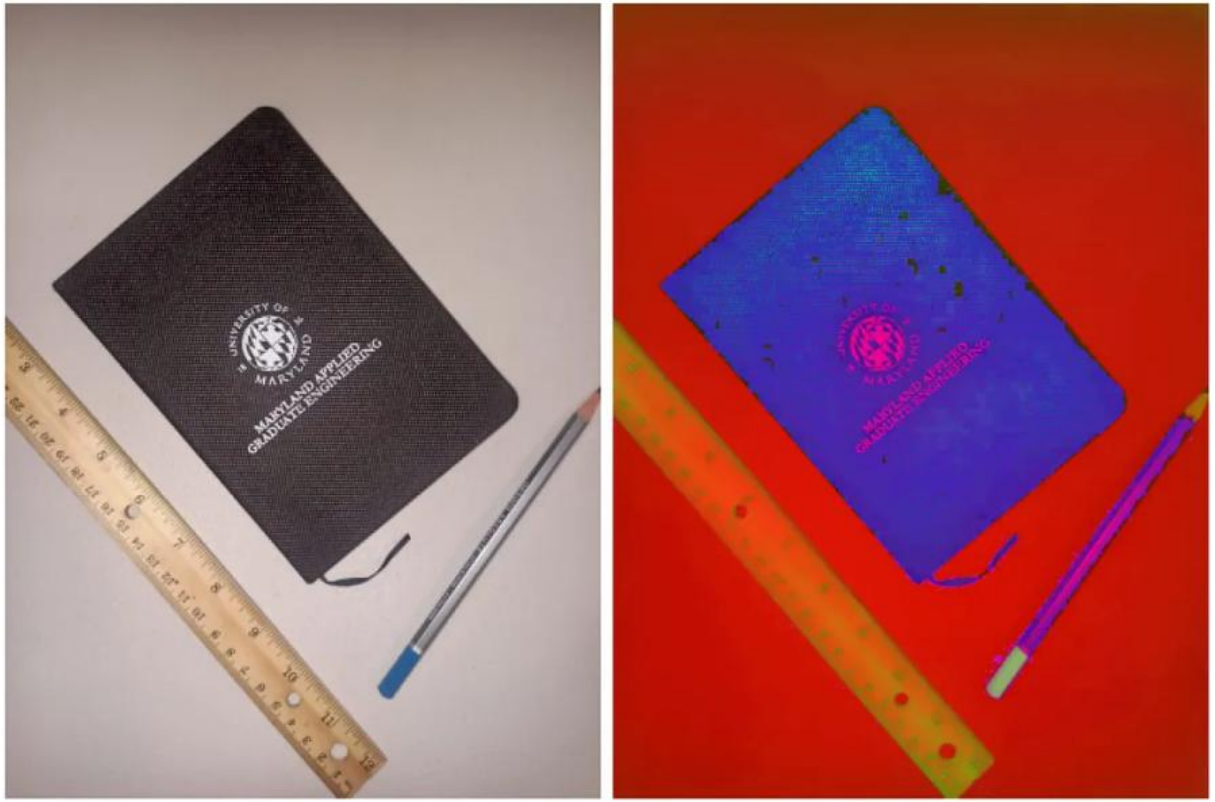
- We start by creating a parameter space (Hough Space).
- The parameter space is a 2D matrix of ρ and θ, where θ ranges between $0–180°$.
- We run this algorithm after detecting the edges of the image using an edge detection algorithm such as Canny edges.
- The pixels with a value of $255°$ are considered edges.
- We then scan the image pixel by pixel to find these pixels and using values of theta from $0°$ to $180°$ we compute ρ for each pixel.
- For pixels on the same line/edge, the value of θ and ρ will be the same.
- We upvote these indices in the Hough Space by 1.
- Finally, the value of ρ and θ with votes above a certain threshold are considered as lines.

To detect the corners of an object(book) in an image

Image Processing Pipeline

1. Convert the image to HSV

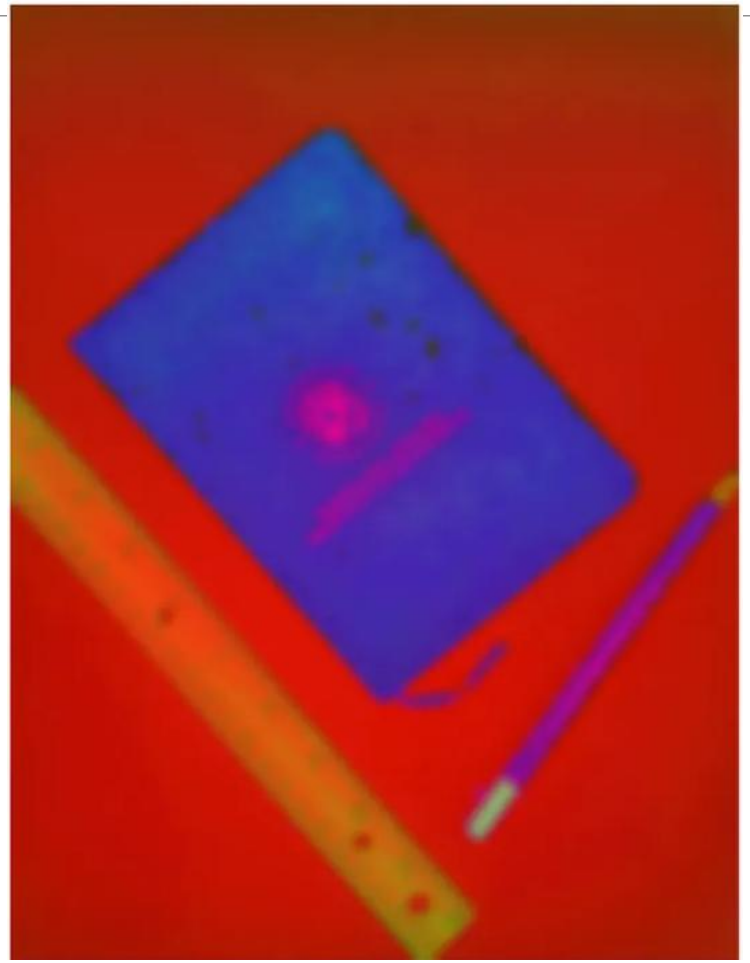
As it becomes difficult to work with RGB we convert the image to HSV color space so that we can easily filter the book within an HSV range. As shown in Figure below.



(a) BGR (b) HSV

2. Apply Gaussian blur to smooth the image

We apply Gaussian blur to smooth out the rough edges due to noise in the image as shown in figure

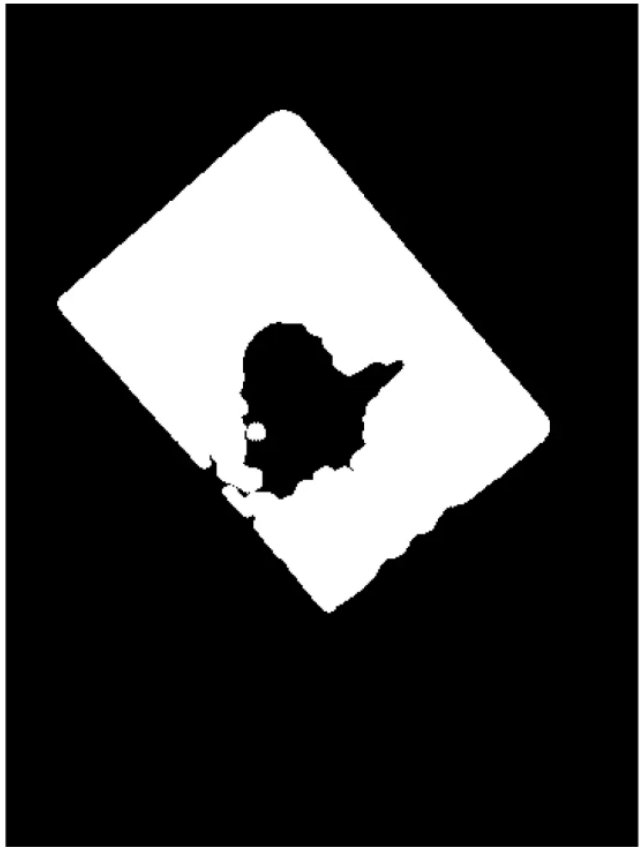


3. Create an image mask

We use the *inRange* function to create a mask. This allows us to get rid of the surrounding objects in the image. As shown in figure

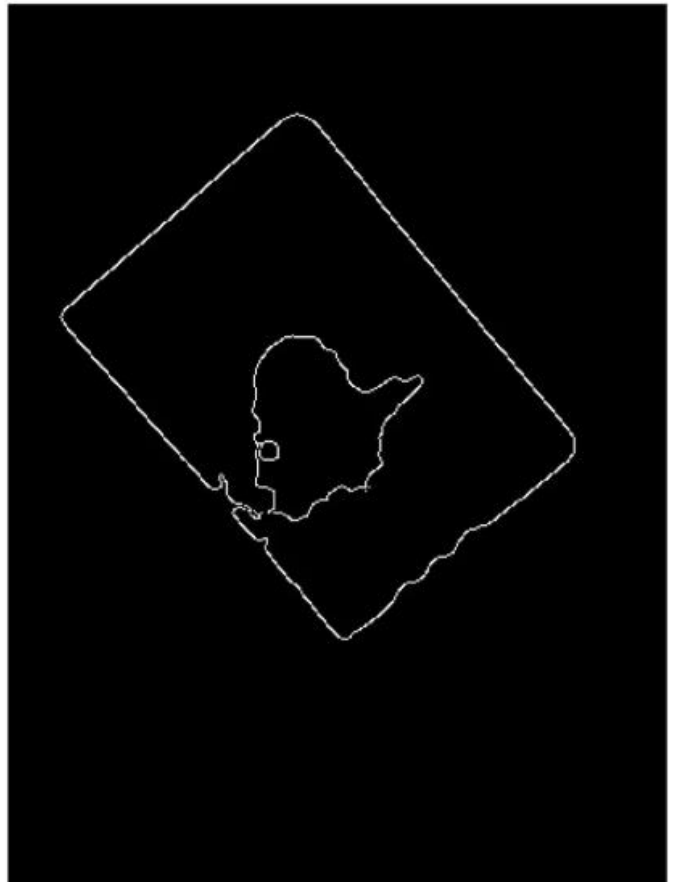
4. Dilate the mask to fill the gaps

As we can see gaps in our mask, we use *dilation* to fill these gaps.



5. Use canny edges to get the edges of the mask

The canny edge detector is used to detect the edges. This is possible because of the high contrast between the mask and the surrounding.



6. Create a Hough Space for detecting the lines

We use Hough space to detect the lines this works as follows:

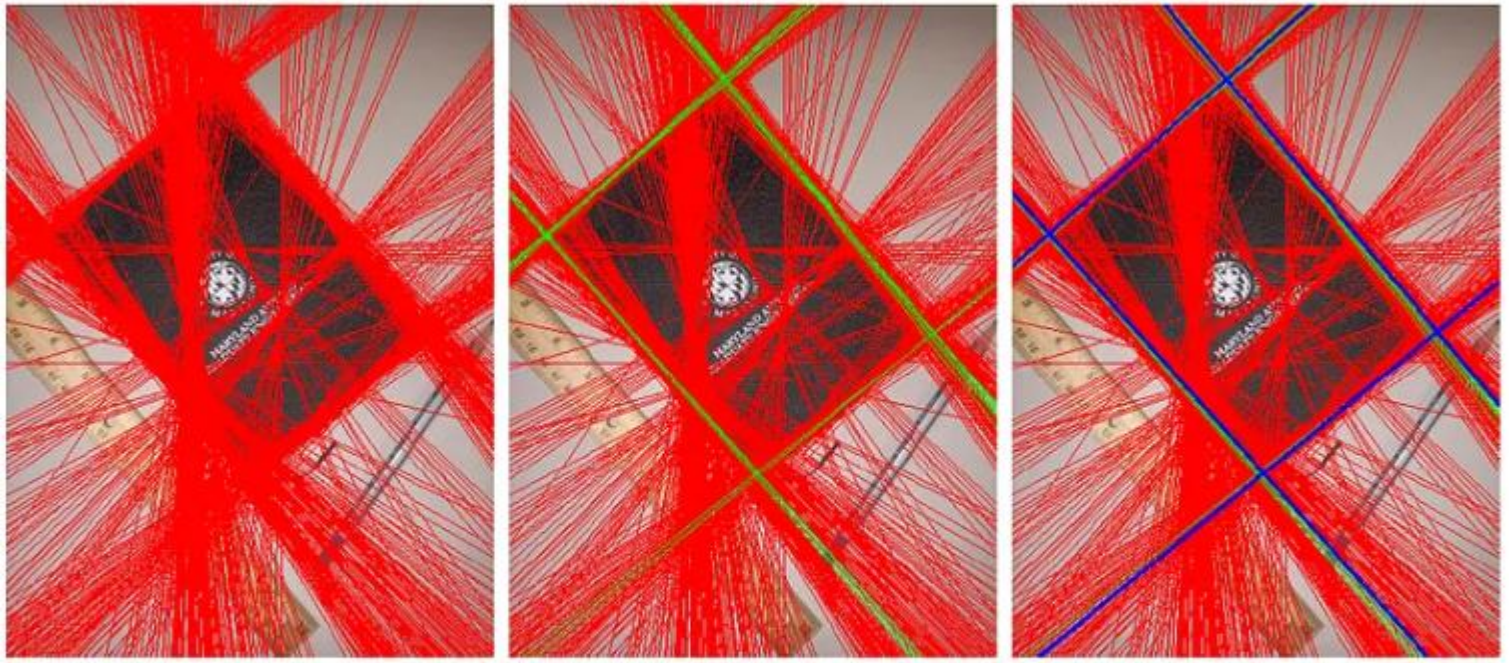
- We create a data structure to store all possible ρ (distance from the origin) and θ (angle with horizontal) along with the votes.
- We iterate through each pixel in the image and detect every bright pixel (value=255). Then we compute the value of ρ using the formula

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

Where θ is in the range $0–180°$ and increment the value corresponding to ρ and θ

$$H[\rho, \theta] \mathrel{+}= 1$$

- As there will be multiple line detections for all the possible edges in the image as shown in the figure
- we need to filter out the unnecessary edges only.
- We use a threshold to get the prominent lines that correspond to our required object
- When we perform canny edge detection, we do not get an edge represented as a line of single pixels, rather we get a cluster of pixels stacked together to form a line.
- Therefore, when we run our Hough algorithm these edges contribute to numerous lines for the same edge.
- To solve this problem, we cluster the neighbour values of ρ and θ in the Hough Space and average their values and get the sum of their upvotes. This results in the merging of the lines depicting the same edge as shown in the figure below.



(a) All Lines Detected (Red)

(b) Filtered with Threshold (Green)

(c) Merging Lines (Blue)

Line Selection/Thresholding

7. Solve the lines to get the intersections(corners)

From the prominent lines obtained in Hough Space, we create a combination of two lines and solve them using linear algebra. These combinations are selected using Permutation and Combinations, and line combinations that cannot be solved (Parallel Lines) are rejected. This gives us points of intersection which are the corners of the book

Example - Road Lane Detection

Implementation of Hough transform in computer vision

The Python code implementation for line detection utilizing the Hough Transform on this image and OpenCV is described in detail below.

1) Import necessary libraries

This code imports OpenCV for image processing and the NumPy library for numerical computations.

```
import numpy as np
import cv2
```

2) Read the image

```
img = cv2.imread('lane_hough.jpg', cv2.IMREAD_COLOR) # lane_hough.jpg is the filename
```

3) Convert the Image to Grayscale
Convert the loaded image to grayscale for edge detection.

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

4) Apply Canny Edge Detector:
Detect edges in the grayscale image using the Canny edge detection method.

```
edges = cv2.Canny(gray, 50, 200)
```

5) Detect Lines using Probabilistic Hough Transform:
In order to find lines in the edge-detected image, use the 'cv2.HoughLinesP' function. An array of lines is returned by this method, with each line's end points (x1, y1, x2, y2) serving as its representation.

```
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 68, minLineLength=15, maxLineGap=250)
```

6) Draw Detected Lines on the Original Image:
Draw each identified line using 'cv2.line' on the original image after iterating over them. The lines' thickness is set to 3 pixels, and their color is set to blue (255, 0, 0).

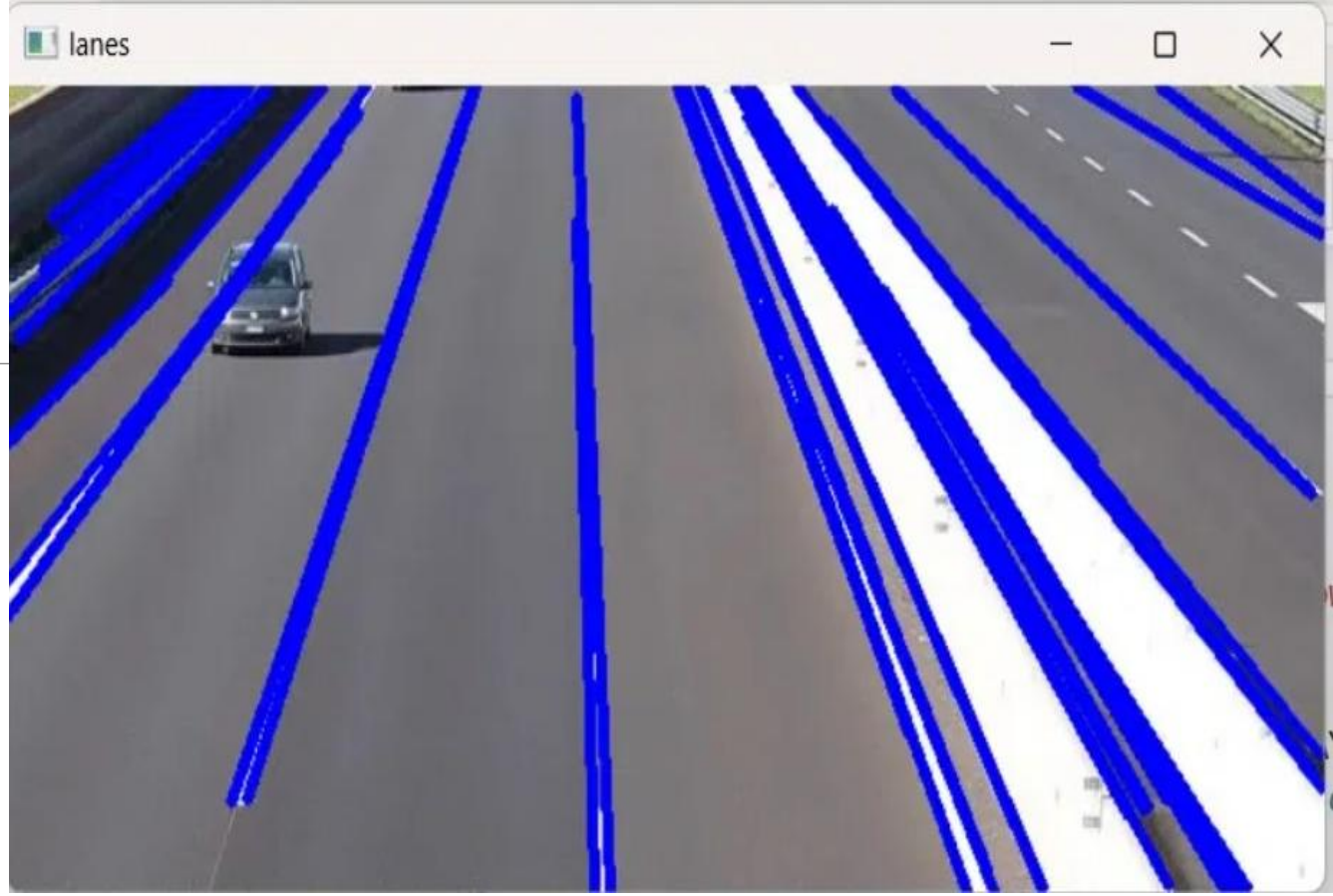
```
for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3)
```

7) Display the Result:

A statement explaining that line detection is being done should be printed. Next, use 'cv2.imshow' to display the image with the lines that have been detected. Finally, use 'cv2.waitKey(0)' and 'cv2.destroyAllWindows()' to wait for a key press to close the window.

```
print("Line Detection using Hough Transform")
cv2.imshow('lanes', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

complete code




```
import numpy as np
import cv2
# Read image
# road.png is the filename
img = cv2.imread('lane_hough.jpg',cv2.IMREAD_COLOR)

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# Find the edges in the image using canny detector
edges = cv2.Canny(gray, 50, 200)

# Detect points that form a line
lines = cv2.HoughLinesP(edges, 1, np.pi/180, 68, minLineLength=15, maxLineGap=250)
#lines = cv2.HoughLinesP(edges, 1, np.pi/180, minLineLength=10, maxLineGap=250)
# Draw lines on the image
for line in lines:
   x1, y1, x2, y2 = line[0]
   cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3)
# Show result
print("Line Detection using Hough Transform")
cv2.imshow('lanes',img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Applications in Computer Vision

In computer vision, the Hough Transform has several uses, some of which are as follows:
Edge Detection: The Hough Transform is an essential part of edge detection algorithms that facilitate the extraction of significant information from images by identifying lines or curves in the image.
Object Recognition: To aid in the identification and categorization of items, the Hough Transform can be utilized in object recognition tasks to pinpoint particular forms within an image.
Lane detection: To help autonomous cars stay in their assigned lanes, lane markers on the road are commonly detected using the Hough Transform.
Medical Imaging: The Hough Transform can be used to identify and evaluate different anatomical features in medical imaging applications, such as MRI or CT scans, which can help with diagnosis and therapy planning.
In the manufacturing sector, the Hough Transform can be applied to **quality control** tasks like measuring component dimensions or looking for flaws.

Thank you