******

***AI385 lab 8***

**Hough Transform**

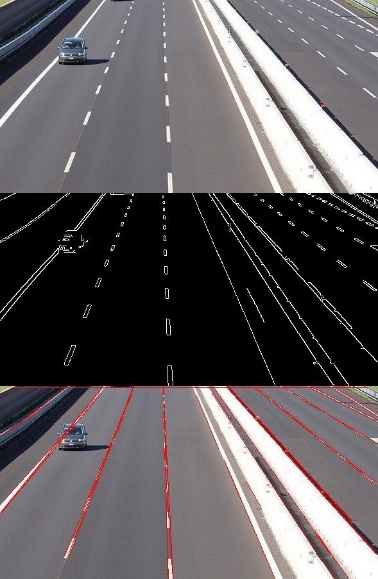
**Learning Objectives:**

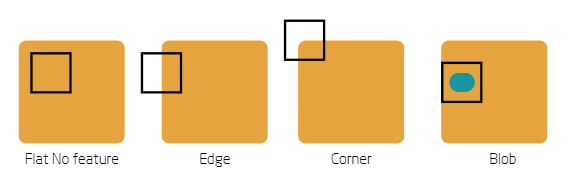
* *To use OpenCV for understanding the concept of the Hough Transform.*
* Implement **Hough Line Transform** for edge-based boundary detection.
* Implement **Hough Circle Transform** for circular boundary detection.
* Apply boundary detection to a real-world medical data.

**Introduction**

Hough Line Transform is a powerful technique in computer vision used to detect **lines and curves** in an image. It is widely used in applications such as **lane detection in self-driving cars, document analysis, object detection, and medical imaging**.

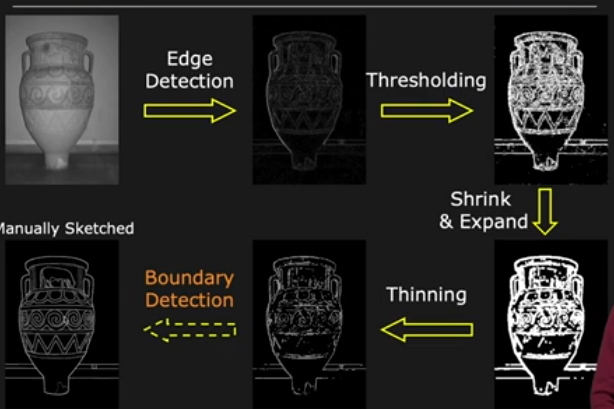
* First let’s review (edges – corners – blob) ,and we have new concept called *boundary* which is the edges and corners of an object
* Now Imagine you have an edges and corners of an object (image) and you want fit a line to these set of edges





A close-up of a yellow square

AI-generated content may be incorrect.

* To Summarize this is our problem: **(WE NEED A METHOD THAT ENABLE US TO CONNECT DETECTED FEATURES IN ORDER TO YIELD THE BOUNDRY OF THE OBJECT)**
* ****this problem call *boundary detection*

**1- Understanding Hough Line Transform**

**📌 Why Is Hough Transform Important?**

Hough Transform is a feature extraction technique used to detect geometric shapes like **lines, circles, and ellipses**. Unlike edge detection techniques, it can identify **incomplete and occluded lines**, making it robust for real-world applications.

**📌 How Does Hough Line Transform Work?**

1. **Edge Detection:** Convert an image to grayscale and apply **Canny Edge Detection** to highlight edges.
2. **Hough Space Representation:** Transform image space (x, y) into **parameter space (ρ, θ)**, where:
   1. **ρ** (rho) is the perpendicular distance from the origin to the line.
   2. **θ** (theta) is the angle between the x-axis and the perpendicular line to the detected edge.
3. **Voting Mechanism:** Every detected edge point **votes** for possible lines that pass through it.
4. **Line Detection:** The peaks in Hough space represent the most probable lines in the image.

**2- Detecting Boundaries Using Hough Line Transform**

**🛠️ Task: Implement Hough Transform for Boundary Lines**

This task will help you detect straight-line boundaries in an image using Hough Line Transform.

**Step-by-Step Explanation:**

* Convert the image to grayscale.
* Apply Canny Edge Detection to highlight edges.
* Use Hough Line Transform to detect straight lines.
* Convert detected ρ, θ values into (x, y) line coordinates.
* Draw the detected lines on the image.

import cv2

import numpy as np

# Load an image in grayscale

gray = cv2.imread(r'C:\Users\m.nasif\Desktop\OneDrive - University of Prince Mugrin\CV\Lab 78\chess.jpg', cv2.IMREAD\_GRAYSCALE)

# Apply Canny Edge Detection

edges = cv2.Canny(gray, 50, 150)

# Apply Standard Hough Line Transform

lines = cv2.HoughLines(edges, rho=1, theta=np.pi/180, threshold=100)

# Convert edges to color image to draw lines

output = cv2.cvtColor(edges, cv2.COLOR\_GRAY2BGR)

# Draw detected lines

if lines is not None:

    for line in lines:

        rho, theta = line[0]

        a = np.cos(theta)

        b = np.sin(theta)

        x0 = a \* rho  # x-coordinate of the line's intersection with the normal

        y0 = b \* rho  # y-coordinate of the line's intersection with the normal

        x1 = int(x0 + 1000 \* (-b))  # Compute x-coordinates for line extension

        y1 = int(y0 + 1000 \* (a))   # Compute y-coordinates for line extension

        x2 = int(x0 - 1000 \* (-b))

        y2 = int(y0 - 1000 \* (a))

        cv2.line(output, (x1, y1), (x2, y2), (0, 0, 255), 2)  # Draw line in red

# Display results

cv2.imshow('Edges', edges)

cv2.imshow('HoughLines - Standard', output)

cv2.waitKey(0)

cv2.destroyAllWindows()

**📌 Adjusting Parameters of** cv2.HoughLines() **for Different Results**

* **Increase threshold** → Detect fewer but stronger lines (useful in noisy images).
* **Decrease threshold** → Detect more lines (can introduce false positives).
* **Increase rho** → Lower precision but faster execution.
* **Decrease rho** → Higher precision but slower computation.
* **Use theta=np.pi/90** instead of np.pi/180 → Detects lines at a **2-degree** resolution.

**3- Detecting Circular Boundaries Using Hough Circle Transform**

**🛠️ Task: Implement Hough Circle Transform**

This task will help you detect circular boundaries using **Hough Circle Transform**.

**Step-by-Step Explanation:**

* Apply **Canny Edge Detection** to detect edges.
* Use **Hough Circle Transform** to detect circular boundaries.
* Set parameters like:
  + param1 → Edge detection threshold.
  + param2 → Circle detection threshold.
  + minRadius, maxRadius → Define the range of detected circle sizes.
* Draw detected circles on the image.

import cv2

import numpy as np

# Load an image in grayscale

gray = cv2.imread(r'C:\Users\m.nasif\Desktop\OneDrive - University of Prince Mugrin\CV\Lab 78\temp1.png', cv2.IMREAD\_GRAYSCALE)

# Apply Gaussian Blur to reduce noise

blurred = cv2.GaussianBlur(gray, (9, 9), 2)

# Apply Hough Circle Transform

circles = cv2.HoughCircles(blurred,

                           method=cv2.HOUGH\_GRADIENT,

                           dp=1.2,

                           minDist=30,

                           param1=50,

                           param2=30,

                           minRadius=10,

                           maxRadius=100)

# Convert to color image for drawing

output = cv2.cvtColor(gray, cv2.COLOR\_GRAY2BGR)

# Draw detected circles

if circles is not None:

    circles = np.uint16(np.around(circles))

    for i in circles[0, :]:

        cv2.circle(output, (i[0], i[1]), i[2], (0, 255, 0), 2)  # Draw circle in green

        cv2.circle(output, (i[0], i[1]), 2, (0, 0, 255), 3)  # Draw center in red

# Display results

cv2.imshow('Blurred', blurred)

cv2.imshow('HoughCircles - Detected Circles', output)

cv2.waitKey(0)

cv2.destroyAllWindows()

**📌 Adjusting Parameters of** cv2.HoughCircles() **for Different Results**

* **Increase minDist** → Prevents detecting too many nearby circles (useful for detecting **individual objects** like coins).
* **Decrease minDist** → Detects circles even if they are close to each other (useful for **clustered objects**).
* **Increase param2** → Only strong, well-defined circles will be detected.
* **Decrease param2** → More circles will be detected, including weak or false positives.
* **Increase dp** → Speeds up the algorithm by reducing resolution.
* **Increase minRadius and maxRadius** → Restricts detection to specific-sized circles.

***Exercises***

**Exercise 1: Boundries Detection:**

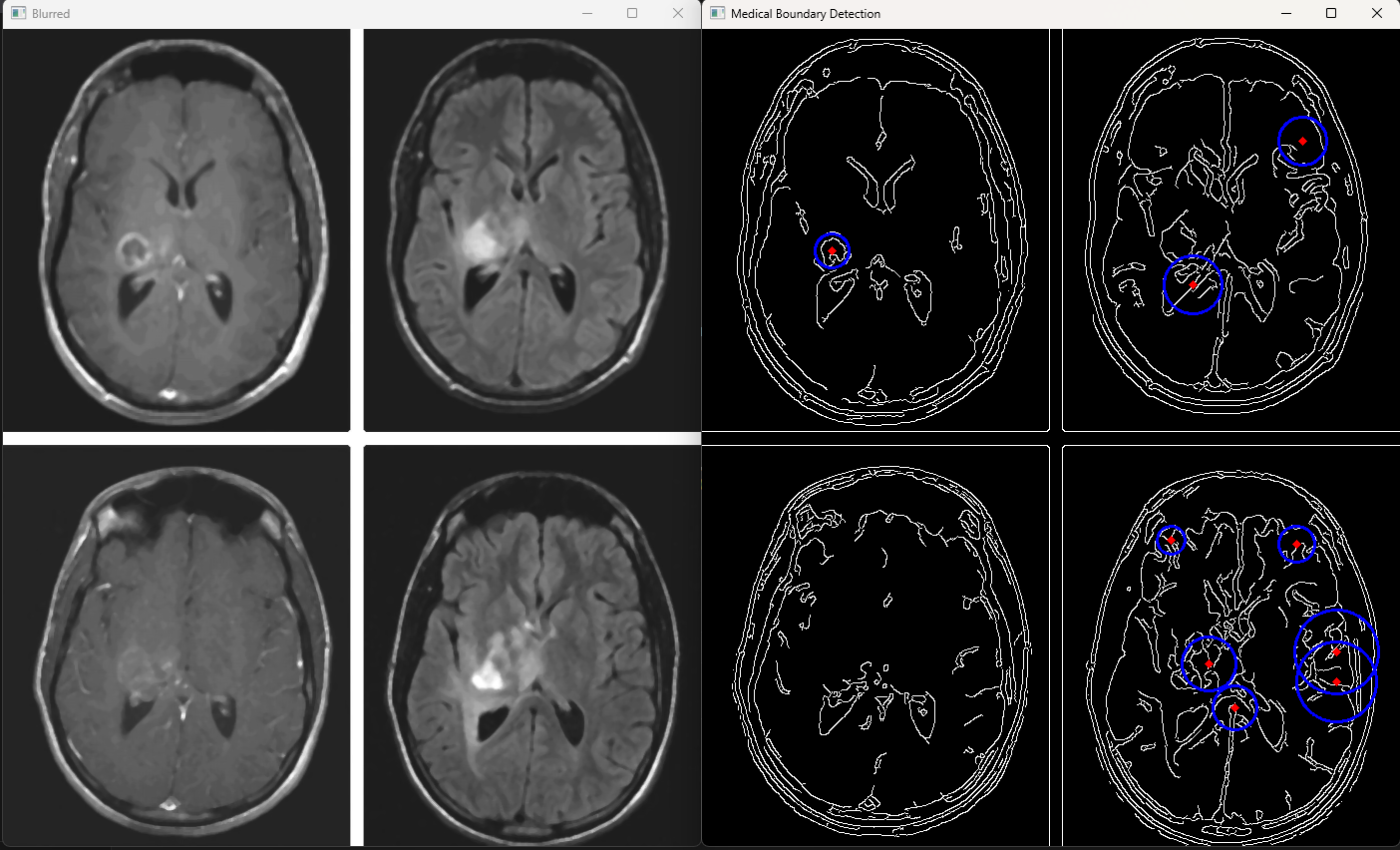
1. **Modify the parameters of the** cv2.HoughLines() **function and observe any changes**
2. **Modify the parameters of the** cv2.HoughCircles() **function and observe any changes**

**Exercise 2: Detect Circular Boundaries in Medical Images**

**apply Hough Transform to medical images to detect circular structures like tumors, blood vessels, or cells.**

Step-by-Step Explanation:

* Apply Median Blurring to remove noise.
* Use Canny Edge Detection to highlight edges.
* Detect circular boundaries using cv2.HoughCircles().
* Draw detected boundaries on the medical image.

****

*Expected output*