1.

import cv2

from matplotlib import pyplot as plt

# Step 1: Read the image (in color)

image = cv2.imread('image1.png')  # Use the actual path of your image

# Step 2: Convert the image to grayscale

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Step 3: Display both images

plt.subplot(1, 2, 1)

plt.title('Original Image')

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))  # Convert BGR to RGB for correct color display in matplotlib

plt.axis('off')

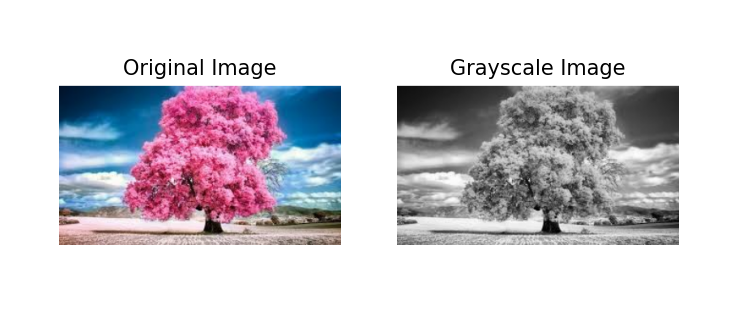
plt.subplot(1, 2, 2)

plt.title('Grayscale Image')

plt.imshow(gray\_image, cmap='gray')

plt.axis('off')

plt.show()



2.

import cv2

from matplotlib import pyplot as plt

# Step 1: Read the image

image = cv2.imread('image2.png')  # Replace with your image file name

# Step 2: Apply Gaussian Blur

blurred\_image = cv2.GaussianBlur(image, (7, 7), 0)  # (7,7) is the kernel size

# Step 3: Display original and blurred images

plt.subplot(1, 2, 1)

plt.title('Original Image')

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

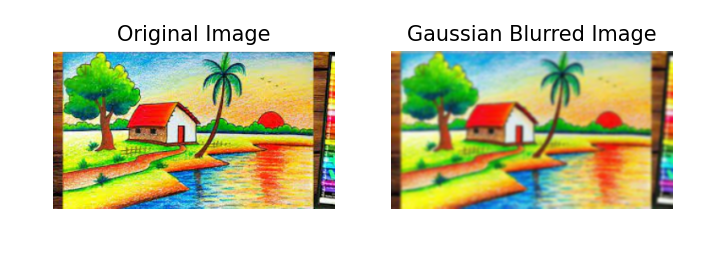
plt.subplot(1, 2, 2)

plt.title('Gaussian Blurred Image')

plt.imshow(cv2.cvtColor(blurred\_image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

plt.show()



3.

import cv2

from matplotlib import pyplot as plt

# Step 1: Read the image

image = cv2.imread('image3.png')  # Replace with your actual image file name

# Step 2: Convert to grayscale (Canny works on single channel images)

gray\_image = cv2.cvtColor(image, cv2.COLOR\_BGR2GRAY)

# Step 3: Apply Canny edge detection

edges = cv2.Canny(gray\_image, threshold1=100, threshold2=200)

# Step 4: Display original and edge-detected images

plt.subplot(1, 2, 1)

plt.title('Original Image')

plt.imshow(cv2.cvtColor(image, cv2.COLOR\_BGR2RGB))

plt.axis('off')

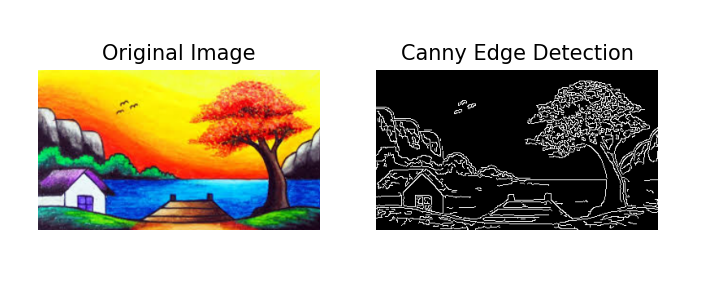
plt.subplot(1, 2, 2)

plt.title('Canny Edge Detection')

plt.imshow(edges, cmap='gray')

plt.axis('off')

plt.show()



4.

import cv2

import numpy as np

from matplotlib import pyplot as plt

image = cv2.imread('image4.png', cv2.IMREAD\_GRAYSCALE)

kernel = np.ones((5, 5), np.uint8)  # Corrected this line

dilated\_image = cv2.dilate(image, kernel, iterations=1)

plt.subplot(1, 2, 1)

plt.title('Original Image')

plt.imshow(image, cmap='gray')

plt.subplot(1, 2, 2)

plt.title('Dilated Image')

plt.imshow(dilated\_image, cmap='gray')

plt.show()



5.

import cv2

import numpy as np

from matplotlib import pyplot as plt

# Step 1: Read the image

image = cv2.imread('image5.png', cv2.IMREAD\_GRAYSCALE)  # Load in grayscale

# Step 2: Define kernel (structuring element)

kernel = np.ones((5, 5), np.uint8)

# Step 3: Apply erosion

eroded\_image = cv2.erode(image, kernel, iterations=1)

# Step 4: Display original and eroded images

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(image, cmap='gray')

plt.axis('off')

plt.subplot(1, 2, 2)

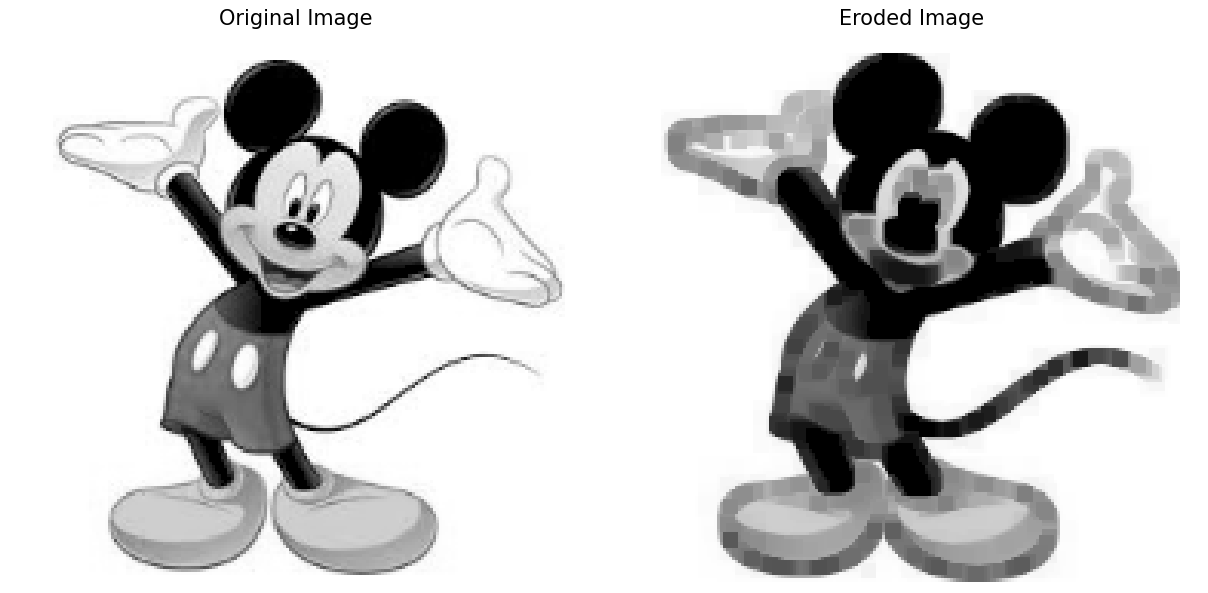
plt.title("Eroded Image")

plt.imshow(eroded\_image, cmap='gray')

plt.axis('off')

plt.tight\_layout()

plt.show()



6.

import cv2

# Step 1: Open the video file or camera

video\_path = 'video1.mp4'  # Replace with 0 for webcam

cap = cv2.VideoCapture(video\_path)

# Function to play video with given speed

def play\_video(speed\_delay, title):

    cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)  # Restart video

    print(f"Playing video: {title}")

    while cap.isOpened():

        ret, frame = cap.read()

        if not ret:

            break

        cv2.imshow(title, frame)

        if cv2.waitKey(speed\_delay) & 0xFF == ord('q'):

            break

    cv2.destroyAllWindows()

# Step 2: Display video in normal speed (delay ~25ms)

play\_video(speed\_delay=25, title="Normal Speed")

# Step 3: Display video in slow motion (delay ~100ms)

play\_video(speed\_delay=100, title="Slow Motion")

# Step 4: Display video in fast motion (delay ~5ms)

play\_video(speed\_delay=5, title="Fast Motion")

# Release the video capture

cap.release()

cv2.destroyAllWindows()



7.

import cv2

# Open webcam (use 0 for default camera)

cap = cv2.VideoCapture(0)

# Check if the webcam opened successfully

if not cap.isOpened():

    print("Error: Could not open webcam.")

    exit()

def display\_video(title, delay):

    print(f"Showing video: {title} (Press 'q' to exit)")

    while True:

        ret, frame = cap.read()

        if not ret:

            break

        cv2.imshow(title, frame)

        if cv2.waitKey(delay) & 0xFF == ord('q'):

            break

    cv2.destroyAllWindows()

# Normal speed (approx 30 FPS → ~33ms delay)

display\_video("Normal Speed", delay=33)

# Slow motion (lower FPS → ~150ms delay)

display\_video("Slow Motion", delay=150)

# Fast motion (higher FPS → ~5ms delay)

display\_video("Fast Motion", delay=5)

# Release the webcam

cap.release()

cv2.destroyAllWindows()



8.

import cv2

from matplotlib import pyplot as plt

# Step 1: Read the image

image = cv2.imread('image5.png')  # Replace with your image path

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  # Convert to RGB for matplotlib

# Step 2: Resize (scale) the image

# Scale down to half size

smaller = cv2.resize(image, None, fx=0.5, fy=0.5, interpolation=cv2.INTER\_AREA)

# Scale up to double size

bigger = cv2.resize(image, None, fx=2.0, fy=2.0, interpolation=cv2.INTER\_LINEAR)

# Step 3: Display all images

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)

plt.title("Original")

plt.imshow(image)

plt.axis('off')

plt.subplot(1, 3, 2)

plt.title("Smaller (0.5x)")

plt.imshow(smaller)

plt.axis('off')

plt.subplot(1, 3, 3)

plt.title("Bigger (2x)")

plt.imshow(bigger)

plt.axis('off')

plt.tight\_layout()

plt.show()



9.

import cv2

from matplotlib import pyplot as plt

# Step 1: Read the image

image = cv2.imread("image5.png")  # Update path

if image is None:

    print("Error: Image not found. Check the file path.")

    exit()

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  # For matplotlib display

# Step 2: Rotate the image

# 90 degrees clockwise

clockwise = cv2.rotate(image, cv2.ROTATE\_90\_CLOCKWISE)

# 90 degrees counter-clockwise

counter\_clockwise = cv2.rotate(image, cv2.ROTATE\_90\_COUNTERCLOCKWISE)

# Step 3: Display all images

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)

plt.title("Original")

plt.imshow(image)

plt.axis('off')

plt.subplot(1, 3, 2)

plt.title("Clockwise Rotation")

plt.imshow(clockwise)

plt.axis('off')

plt.subplot(1, 3, 3)

plt.title("Counter-Clockwise Rotation")

plt.imshow(counter\_clockwise)

plt.axis('off')

plt.tight\_layout()

plt.show()



10.

import cv2

import numpy as np

from matplotlib import pyplot as plt

# Step 1: Read the image

image = cv2.imread("image5.png")  # Update with your correct image path

if image is None:

    print("Error: Image not found.")

    exit()

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Step 2: Define translation matrix (move right by 100px and down by 50px)

rows, cols = image.shape[:2]

translation\_matrix = np.float32([[1, 0, 100], [0, 1, 50]])

# Step 3: Apply the translation

moved\_image = cv2.warpAffine(image, translation\_matrix, (cols + 100, rows + 50))

# Step 4: Display original and moved images

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.title("Original Image")

plt.imshow(image)

plt.axis('off')

plt.subplot(1, 2, 2)

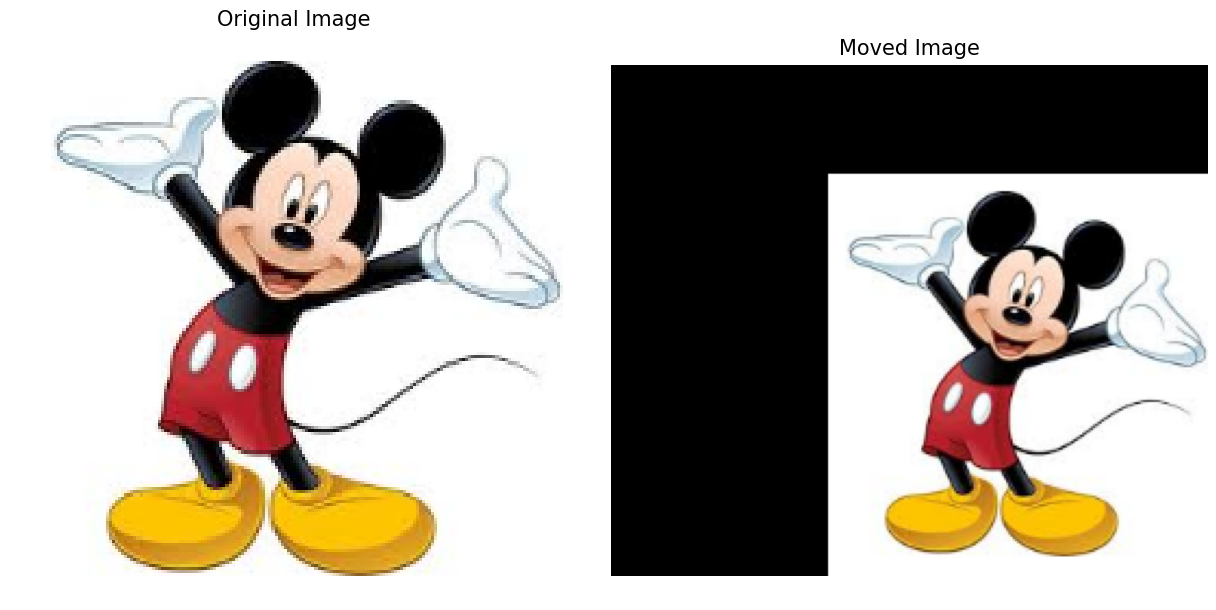
plt.title("Moved Image")

plt.imshow(moved\_image)

plt.axis('off')

plt.tight\_layout()

plt.show()



11.

import cv2

import numpy as np

img = cv2.imread('image6.png')

rows, cols = img.shape[:2]

# Source and destination points

pts1 = np.float32([[50, 50], [200, 50], [50, 200]])

pts2 = np.float32([[10, 100], [200, 50], [100, 250]])

# Affine transform matrix

M = cv2.getAffineTransform(pts1, pts2)

dst = cv2.warpAffine(img, M, (cols, rows))

cv2.imwrite('affine\_output.jpg', dst)





12.

import cv2

import numpy as np

img = cv2.imread('image6.png')

rows, cols = img.shape[:2]

pts1 = np.float32([[0, 0], [cols - 1, 0], [0, rows - 1], [cols - 1, rows - 1]])

pts2 = np.float32([[50, 50], [cols - 50, 30], [70, rows - 50], [cols - 50, rows - 30]])

# Perspective matrix

M = cv2.getPerspectiveTransform(pts1, pts2)

dst = cv2.warpPerspective(img, M, (cols, rows))

cv2.imwrite('perspective\_image.jpg', dst)





13.

import cv2

import numpy as np

cap = cv2.VideoCapture('video1.mp4')

if not cap.isOpened():

    print("Error: Could not open video file.")

    exit()

pts1 = np.float32([[100, 100], [400, 100], [100, 400], [400, 400]])

pts2 = np.float32([[150, 150], [350, 100], [130, 380], [370, 420]])

M = cv2.getPerspectiveTransform(pts1, pts2)

while cap.isOpened():

    ret, frame = cap.read()

    if not ret:

        break

    warped = cv2.warpPerspective(frame, M, (frame.shape[1], frame.shape[0]))

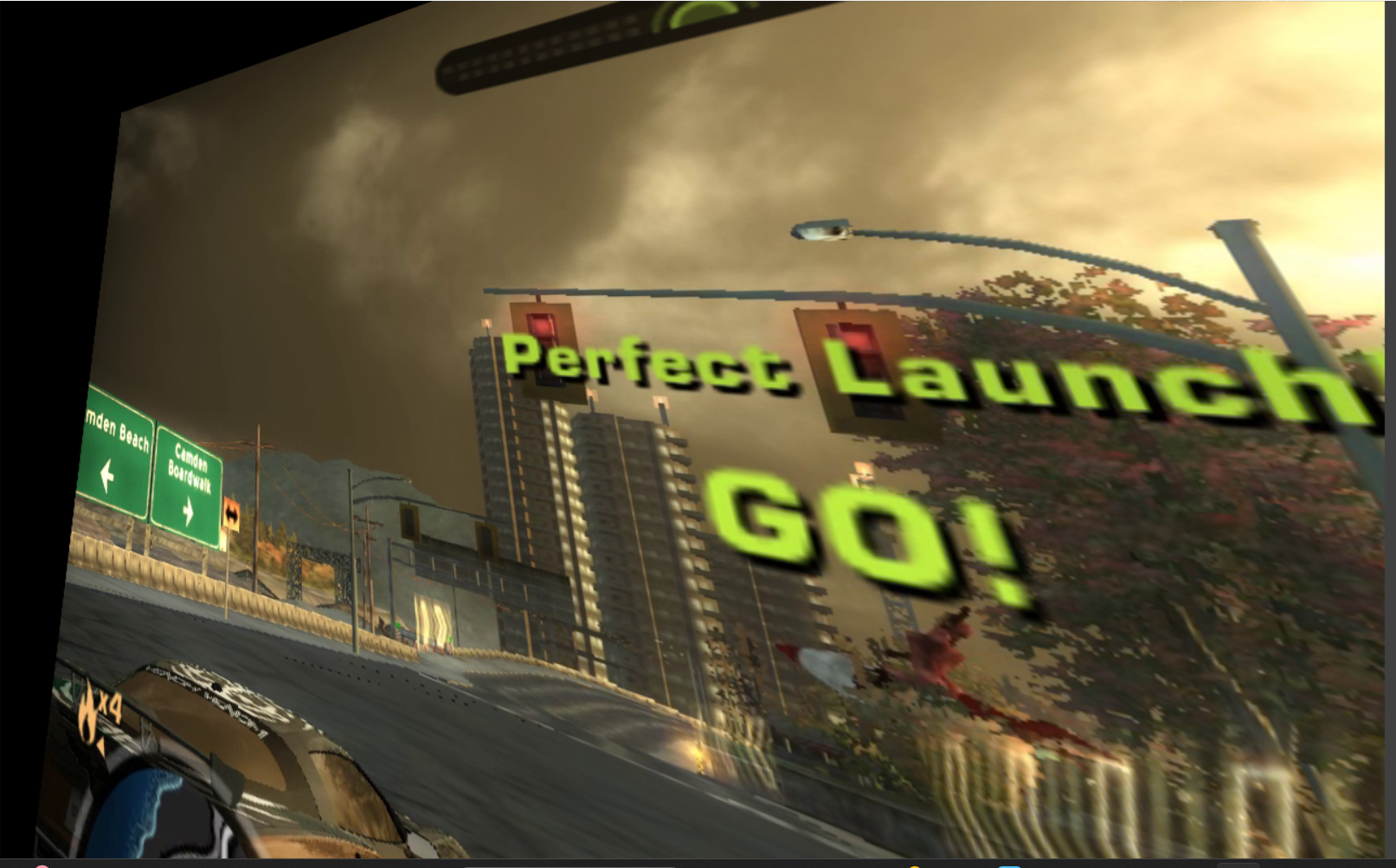
    cv2.imshow('Perspective Video', warped)

    if cv2.waitKey(1) & 0xFF == ord('q'):

        break

cap.release()

cv2.destroyAllWindows()



14.

import cv2

import numpy as np

# Use ORB to detect keypoints

img1 = cv2.imread('image7.png', 0)

img2 = cv2.imread('image8.png', 0)

# Check if images are loaded

if img1 is None or img2 is None:

    print("Error: Could not load one or both images.")

    exit()

orb = cv2.ORB\_create()

kp1, des1 = orb.detectAndCompute(img1, None)

kp2, des2 = orb.detectAndCompute(img2, None)

# Match features

bf = cv2.BFMatcher(cv2.NORM\_HAMMING, crossCheck=True)

matches = bf.match(des1, des2)

matches = sorted(matches, key=lambda x: x.distance)

# Use first 10 good matches

src\_pts = np.float32([kp1[m.queryIdx].pt for m in matches[:10]]).reshape(-1, 1, 2)

dst\_pts = np.float32([kp2[m.trainIdx].pt for m in matches[:10]]).reshape(-1, 1, 2)

# Get homography

H, \_ = cv2.findHomography(src\_pts, dst\_pts, cv2.RANSAC)

warped = cv2.warpPerspective(img1, H, (img2.shape[1], img2.shape[0]))

cv2.imwrite('homography\_output.jpg', warped)

cv2.destroyAllWindows()

15.

import cv2

import numpy as np

# Step 1: Read the image

img = cv2.imread('image6.png')

if img is None:

    print("Image not found. Make sure 'image6.png' is in the same folder.")

    exit()

# Step 2: Define 4 points in the original image

src\_pts = np.float32([

    [100, 100],

    [300, 100],

    [300, 300],

    [100, 300]

])

# Step 3: Define where those points should map to in the output image

dst\_pts = np.float32([

    [80, 150],

    [320, 100],

    [310, 320],

    [90, 330]

])

# Step 4: Calculate the Homography matrix using DLT method

H, status = cv2.findHomography(src\_pts, dst\_pts, method=0)  # method=0 means pure DLT

print("Homography Matrix (DLT):\n", H)

# Step 5: Apply the perspective transformation

warped = cv2.warpPerspective(img, H, (img.shape[1], img.shape[0]))

# Step 6: Show the result

cv2.imshow("Original Image", img)

cv2.imshow("Warped Image using DLT", warped)

cv2.waitKey(0)

cv2.destroyAllWindows()



16.

import cv2

# Step 1: Read the image in grayscale

img = cv2.imread('image6.png', cv2.IMREAD\_GRAYSCALE)

if img is None:

    print("Error: Image not found.")

    exit()

# Step 2: Apply Canny edge detection

edges = cv2.Canny(img, threshold1=100, threshold2=200)

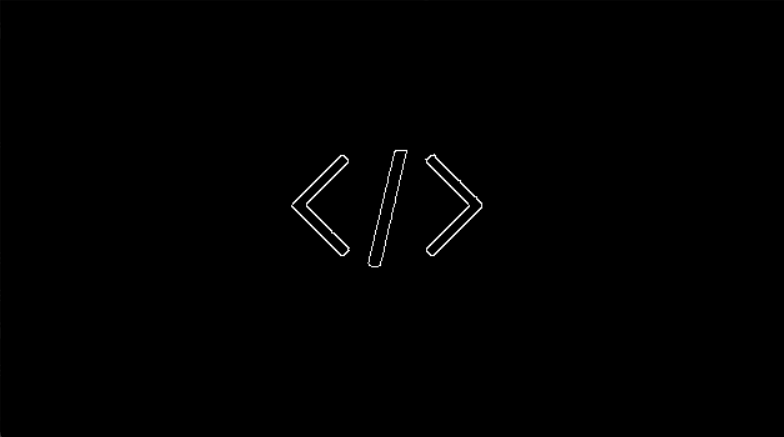
# Step 3: Show the original and edge-detected images

cv2.imshow('Original Image', img)

cv2.imshow('Canny Edge Detection', edges)

cv2.waitKey(0)

cv2.destroyAllWindows()



17.

import cv2

import numpy as np

# Step 1: Read the image in grayscale

img = cv2.imread('image6.png', cv2.IMREAD\_GRAYSCALE)

if img is None:

    print("Error: Image not found.")

    exit()

# Step 2: Apply Sobel operator along the X-axis

sobel\_x = cv2.Sobel(img, cv2.CV\_64F, dx=1, dy=0, ksize=3)

# Step 3: Convert the result to uint8 for display

sobel\_x\_abs = cv2.convertScaleAbs(sobel\_x)

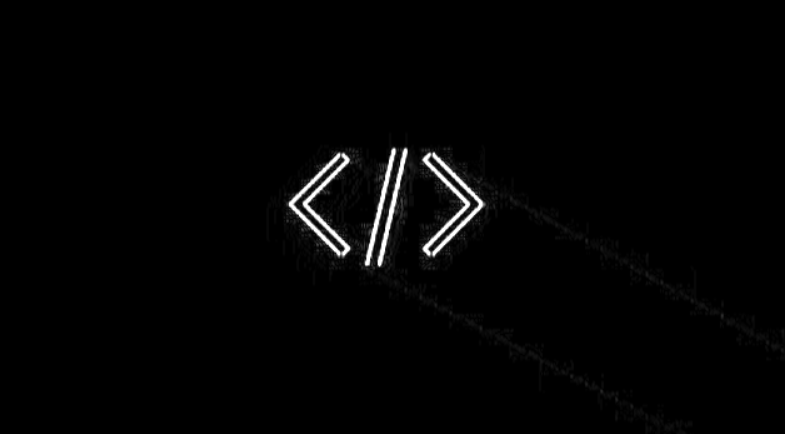
# Step 4: Show the result

cv2.imshow('Original Image', img)

cv2.imshow('Sobel X Edge Detection', sobel\_x\_abs)

cv2.waitKey(0)

cv2.destroyAllWindows()



18.

import cv2

import numpy as np

# Step 1: Read the image in grayscale

img = cv2.imread('image6.png', cv2.IMREAD\_GRAYSCALE)

if img is None:

    print("Error: Image not found.")

    exit()

# Step 2: Apply Sobel operator along the Y-axis

sobel\_y = cv2.Sobel(img, cv2.CV\_64F, dx=0, dy=1, ksize=3)

# Step 3: Convert the result to uint8 for display

sobel\_y\_abs = cv2.convertScaleAbs(sobel\_y)

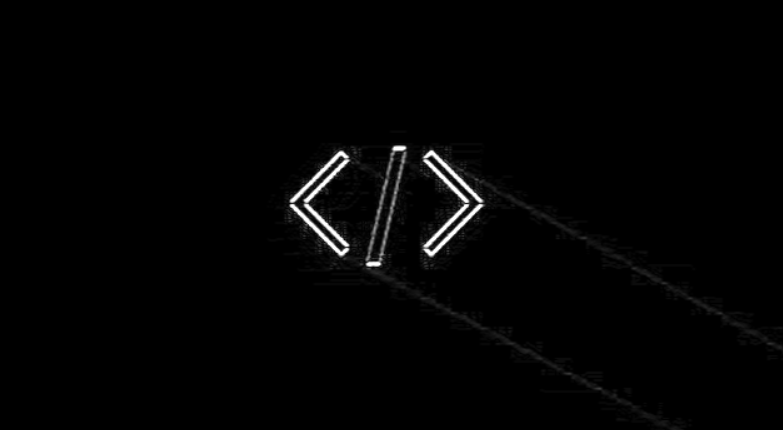
# Step 4: Show the result

cv2.imshow('Original Image', img)

cv2.imshow('Sobel Y Edge Detection', sobel\_y\_abs)

cv2.waitKey(0)

cv2.destroyAllWindows()



19.

import cv2

import numpy as np

# Step 1: Read the image in grayscale

img = cv2.imread('image6.png', cv2.IMREAD\_GRAYSCALE)

if img is None:

    print("Error: Image not found.")

    exit()

# Step 2: Apply Sobel operator along X and Y axes

sobel\_x = cv2.Sobel(img, cv2.CV\_64F, dx=1, dy=0, ksize=3)

sobel\_y = cv2.Sobel(img, cv2.CV\_64F, dx=0, dy=1, ksize=3)

# Step 3: Combine the two gradients (magnitude)

sobel\_xy = cv2.magnitude(sobel\_x, sobel\_y)

# Step 4: Convert to 8-bit image

sobel\_xy\_abs = cv2.convertScaleAbs(sobel\_xy)

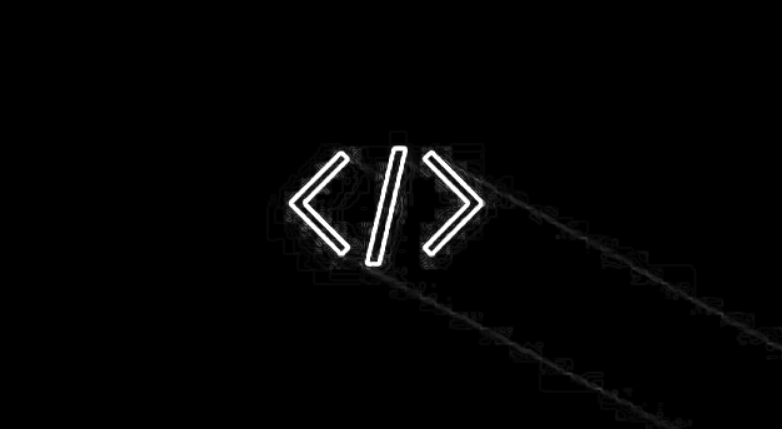
# Step 5: Show the result

cv2.imshow('Original Image', img)

cv2.imshow('Sobel XY Edge Detection', sobel\_xy\_abs)

cv2.waitKey(0)

cv2.destroyAllWindows()



20.

import cv2

import numpy as np

# Step 1: Read the image in grayscale

img = cv2.imread('image6.png', cv2.IMREAD\_GRAYSCALE)

if img is None:

    print("Error: Image not found.")

    exit()

# Step 2: Define Laplacian kernel with a negative center

laplacian\_kernel = np.array([[0,  1, 0],

                             [1, -4, 1],

                             [0,  1, 0]])

# Step 3: Apply the Laplacian filter

laplacian = cv2.filter2D(img, cv2.CV\_64F, laplacian\_kernel)

# Step 4: Sharpen the image by subtracting the Laplacian

sharpened = cv2.convertScaleAbs(img - laplacian)

# Step 5: Show the result

cv2.imshow('Original Image', img)

cv2.imshow('Sharpened Image (Laplacian - Negative Center)', sharpened)

cv2.waitKey(0)

cv2.destroyAllWindows()

