EX NO:01 **IMAGE PROCESSING OPERATIONS**
DATE:

AIM:

To perform various image processing tasks using OpenCV and PIL in Python.

ALGORITHM

1. Image Enhancement:
• Adjust image brightness, sharpness, color, and contrast using PIL's ImageEnhance module.
• Apply enhancements sequentially by specifying the enhancement factor (e.g., brightness: 2.8) to improve image quality.
• Create ImageEnhance objects for brightness, sharpness, color, and contrast adjustments.

2. Image Blurring:
• Apply Gaussian, median, and bilateral blurring to an image using OpenCV.
• Specify the blur kernel size and other parameters to control the amount of blurring.
• Apply chosen blurring techniques (Gaussian, median, bilateral) with specified kernel sizes and parameters.

3. Image Filtering:
• Apply various image filters like sharpening and smoothing using PIL's ImageFilter module.
• Choose from filters like FIND_EDGES and SMOOTH_MORE to achieve desired visual effects.

4. Histogram Equalization:
• Perform histogram equalization on a grayscale image to enhance contrast.
• Compare the original and equalized histograms for visual improvement.
• Use equalizeHist() to perform histogram equalization.

5. Bitwise Operations:
• Apply bitwise NOT operation to invert the colors in a binary image.
• Display the original and inverted images.
• Apply bitwise NOT operation using cv2.bitwise_not().

6. Image Addition and Subtraction:
• Add and subtract two images pixel-wise using OpenCV.
• Visualize the resulting images representing image addition and subtraction.
• Perform pixel-wise addition or subtraction using cv2.add() and cv2.subtract().

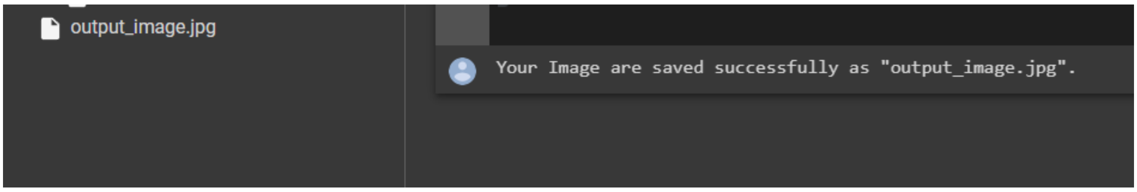
7. Image Resizing and Scaling:
• Resize images with different scaling factors and interpolation methods using OpenCV.
• Display the original image, a smaller version, and a stretched version with appropriate titles.
• Use cv2.resize() with appropriate parameters (scaling factors or target size) and interpolation method.

1) READ, WRITE AND SAVE AN IMAGE.

CODE:

```
import cv2
image = cv2.imread('/content/sample_data/LONDON.jpeg')
if image is None:
   print('Error: Could not able to open or read your image.')
else:
   cv2.imwrite('output_image.jpg', image)
   saved_image = cv2.imread('output_image.jpg')
   if saved_image is not None:
      print('Your Image are saved successfully as "output_image.jpg".')
   else:
      print('Error: Could not able to save your image.')
```

OUTPUT:

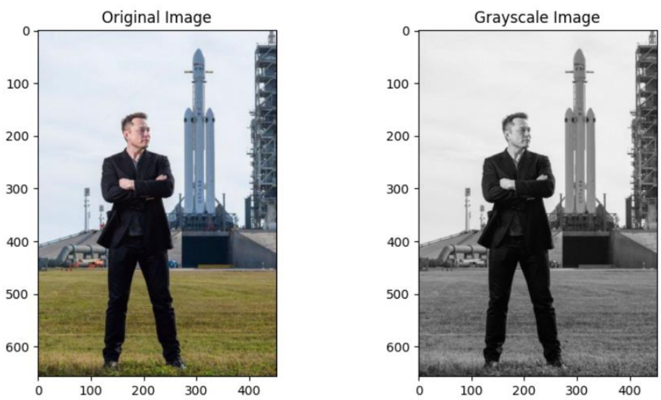


2) CONVERT AN IMAGE INTO GRAY SCALE.

CODE:

```
import cv2
import matplotlib.pyplot as plt
original_image = cv2.imread('/content/sample_data/elon.jpg')
gray_image = cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY)
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(gray_image, cmap='gray')
plt.title('Grayscale Image')
plt.show()
```

OUTPUT:

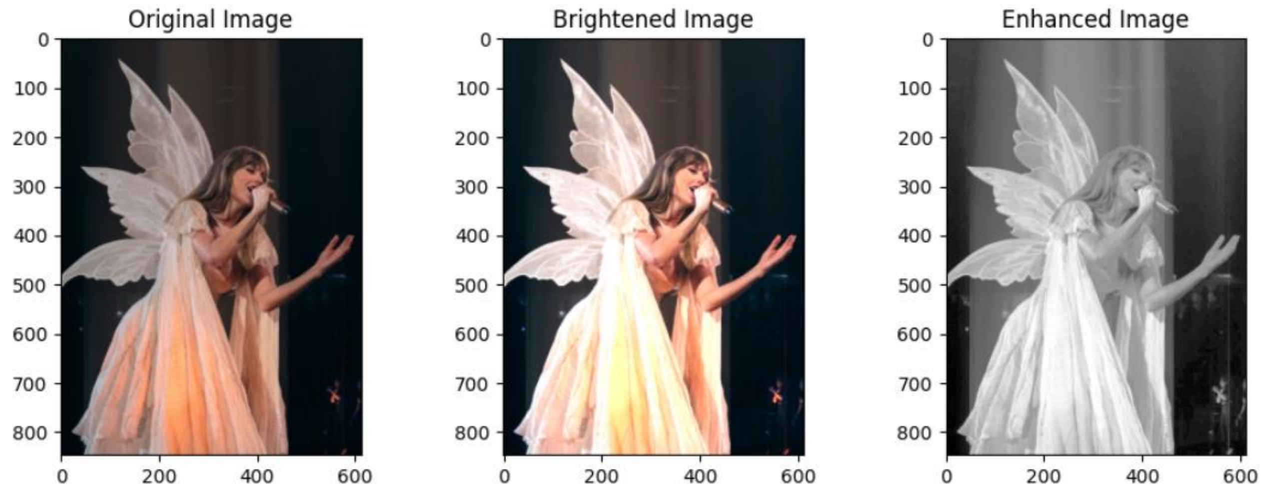


3) IMAGE ENCHACEMENT

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
original_image = cv2.imread('/content/sample_data/Taylor swift.jpg')
brightness_factor = 1.5
brightened_image = cv2.convertScaleAbs(original_image, alpha=brightness_factor, beta=0)
plt.figure(figsize=(12, 4))
plt.subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(brightened_image, cv2.COLOR_BGR2RGB))
plt.title('Brightened Image')
enhanced_image = cv2.equalizeHist(cv2.cvtColor(original_image, cv2.COLOR_BGR2GRAY))
plt.subplot(1, 3, 3)
plt.imshow(enhanced_image, cmap='gray')
plt.title('Enhanced Image')
plt.show()
```

OUTPUT:

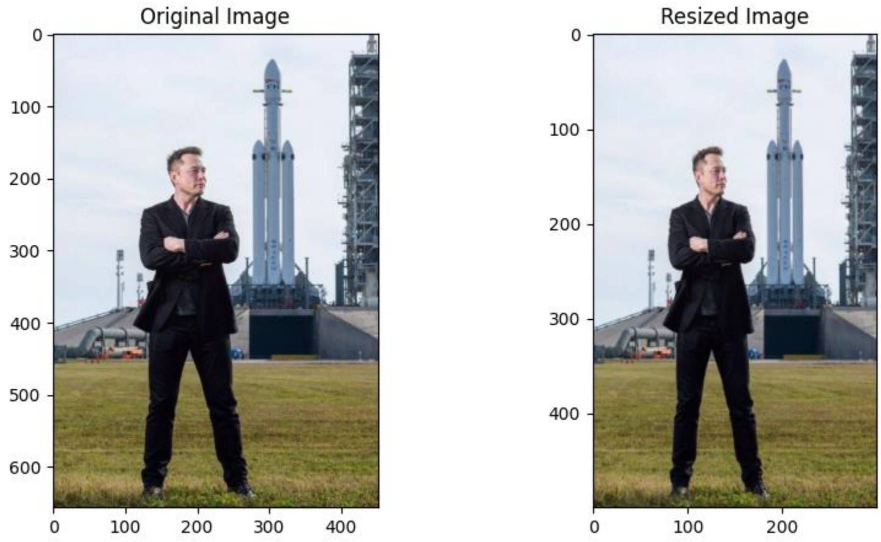


4) IMAGE RESIZING

CODE:

```
import cv2
import matplotlib.pyplot as plt
original_image = cv2.imread('/content/sample_data/elon.jpg')
new_width = 300
new_height = 500
resized_image = cv2.resize(original_image, (new_width, new_height))
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(resized_image, cv2.COLOR_BGR2RGB))
plt.title('Resized Image')
plt.show()
```

OUTPUT:

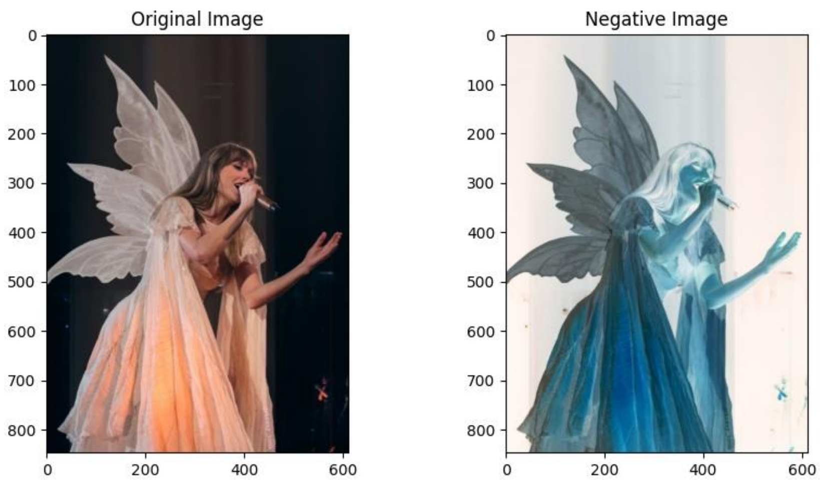


5) NEGATIVE IMAGE

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
original_image = cv2.imread('/content/sample_data/Taylor swift.jpg')
negative_image = 255 - original_image
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(original_image, cv2.COLOR_BGR2RGB))
plt.title('Original Image')
plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(negative_image, cv2.COLOR_BGR2RGB))
plt.title('Negative Image')
plt.show()
```

OUTPUT:

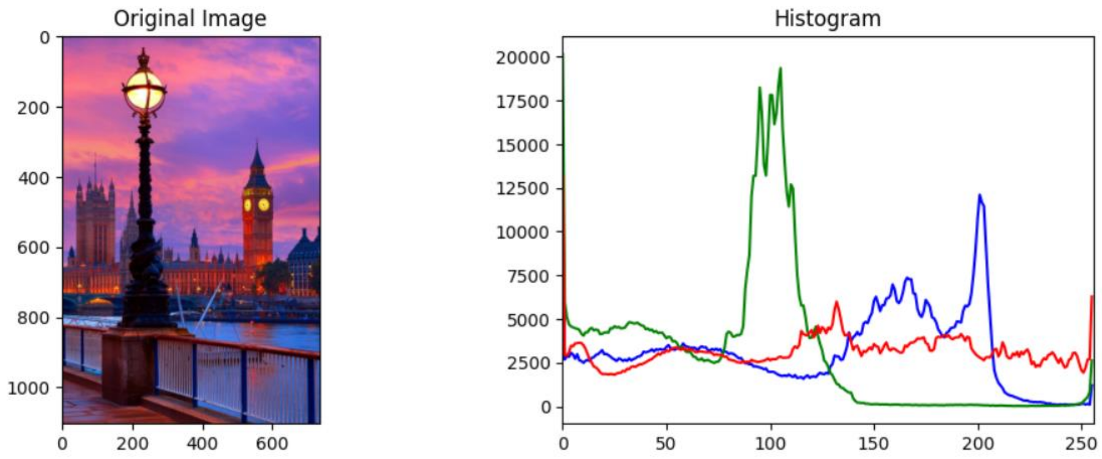


6) HISTOGRAM EQUALIZATION

CODE:

```
import cv2
import matplotlib.pyplot as plt
img_bgr = cv2.imread('/content/sample_data/180 London Captions That'll Inspire You To Plan an Epic Trip.jpeg', 1)
fig, axs = plt.subplots(1, 2, figsize=(12, 4))
axs[0].imshow(cv2.cvtColor(img_bgr, cv2.COLOR_BGR2RGB))
axs[0].set_title('Original Image')
color = ('b', 'g', 'r')
for i, col in enumerate(color):
    histr = cv2.calcHist([img_bgr], [i], None, [256], [0, 256])
    axs[1].plot(histr, color=col)
axs[1].set_title('Histogram')
axs[1].set_xlim([0, 256])
plt.show()
```

OUTPUT:

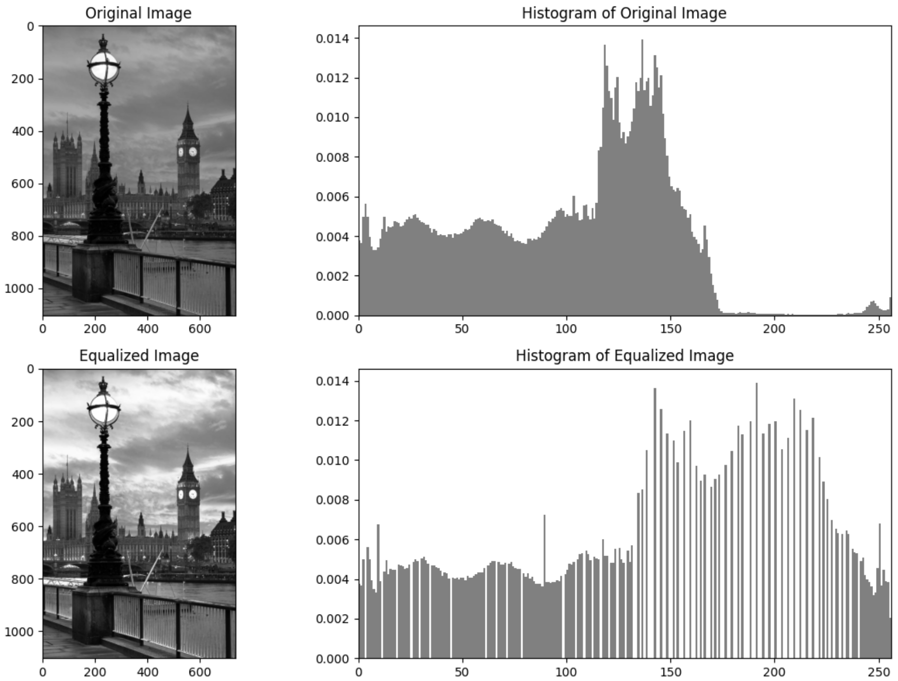


HISTOGRAM FOR ORIGINAL IMAGE AND EQUALIZED IMAGE (Gray Scale):

CODE:

```
import cv2
import matplotlib.pyplot as plt
image = cv2.imread('/content/sample_data/LONDON.jpeg', cv2.IMREAD_GRAYSCALE)
equalized_image = cv2.equalizeHist(image)
fig, axs = plt.subplots(2, 2, figsize=(12, 8))
axs[0, 0].imshow(image, cmap='gray')
axs[0, 0].set_title('Original Image')
axs[0, 1].hist(image.ravel(), bins=256, range=(0, 256), density=True, color='gray')
axs[0, 1].set_title('Histogram of Original Image')
axs[0, 1].set_xlim([0, 256])
axs[1, 0].imshow(equalized_image, cmap='gray')
axs[1, 0].set_title('Equalized Image')
axs[1, 1].hist(equalized_image.ravel(), bins=256, range=(0, 256), density=True, color='gray')
axs[1, 1].set_title('Histogram of Equalized Image')
axs[1, 1].set_xlim([0, 256])
plt.tight_layout()
plt.show()
```

OUTPUT:

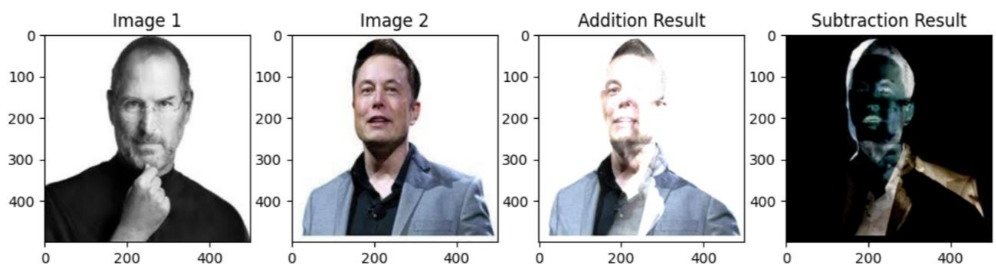


7) ADDITION AND SUBTRACTION OF A IMAGE

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
image1 = cv2.imread('/content/sample_data/steve jobs.jpg')
image2 = cv2.imread('/content/sample_data/elon musk (1).jpg')
if image1.shape != image2.shape:
   raise ValueError("Both images should be in the same dimensions")
addition_result = cv2.add(image1, image2)
subtraction_result = cv2.subtract(image1, image2)
plt.figure(figsize=(12, 4))
plt.subplot(1, 4, 1)
plt.imshow(cv2.cvtColor(image1, cv2.COLOR_BGR2RGB))
plt.title('Image 1')
plt.subplot(1, 4, 2)
plt.imshow(cv2.cvtColor(image2, cv2.COLOR_BGR2RGB))
plt.title('Image 2')
plt.subplot(1, 4, 3)
plt.imshow(cv2.cvtColor(addition_result, cv2.COLOR_BGR2RGB))
plt.title('Addition Result')
plt.subplot(1, 4, 4)
plt.imshow(cv2.cvtColor(subtraction_result, cv2.COLOR_BGR2RGB))
plt.title('Subtraction Result')
plt.show()
```

OUTPUT:



RESULT:

Thus the program executed successfully for the image processing operations and the outputs are verified.

EX NO:02 **ADDING AND REMOVAL OF NOISE**

AIM:

To assess the efficacy of spatial domain filters (Average, Gaussian, Median) in mitigating Gaussian and Salt and Pepper noise in digital images.

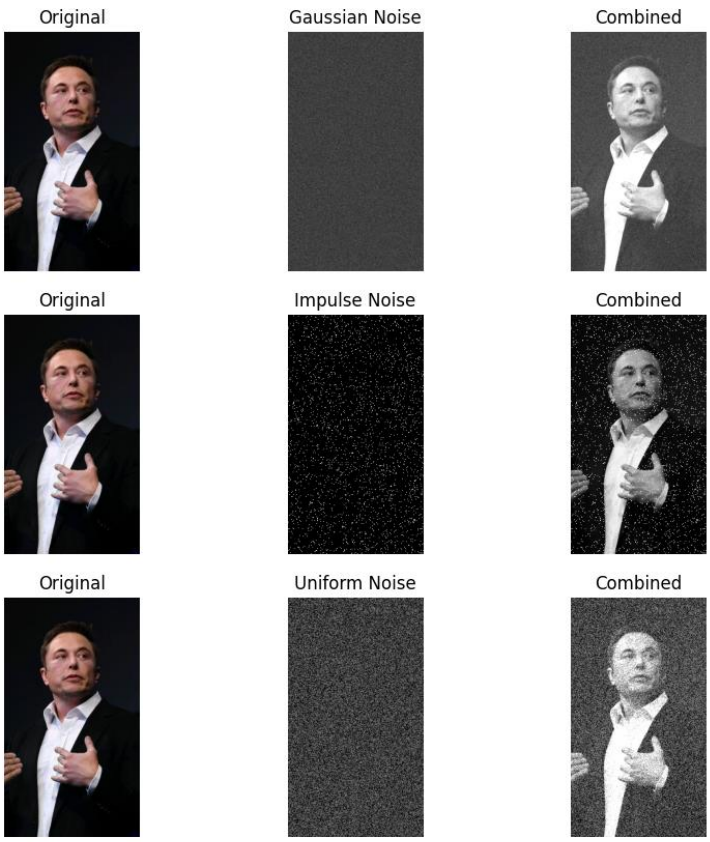
CODE:

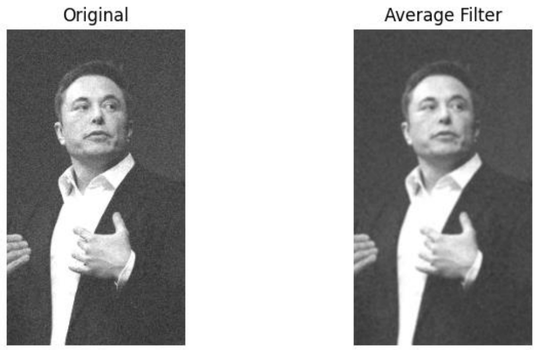
```
import numpy as np
import matplotlib.pyplot as plt
import cv2

# Read the image in color
img_color = cv2.imread("/content/sample_data/elon musk.jpg")

# Convert the color image to grayscale for processing
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

# Adding Gaussian noise
gauss_noise = np.zeros_like(img_gray, dtype=np.uint8)
cv2.randn(gauss_noise, 128, 20)
gauss_noise = (gauss_noise * 0.5).astype(np.uint8)
gn_img = cv2.add(img_gray, gauss_noise)

# Convert the noise to color for display
gauss_noise_color = cv2.cvtColor(gauss_noise, cv2.COLOR_GRAY2BGR)

# Display images for Gaussian noise
fig = plt.figure(dpi=300)
fig.add_subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Original")

fig.add_subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(gauss_noise_color, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Gaussian Noise")

fig.add_subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(gn_img, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Combined")

plt.show()

# Reset variables for next noise type
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

# Adding Impulse noise
imp_noise = np.zeros_like(img_gray, dtype=np.uint8)
cv2.randu(imp_noise, 0, 255)
imp_noise = cv2.threshold(imp_noise, 245, 255, cv2.THRESH_BINARY)[1]
in_img = cv2.add(img_gray, imp_noise)

# Convert the noise to color for display
imp_noise_color = cv2.cvtColor(imp_noise, cv2.COLOR_GRAY2BGR)
```

```
# Display images for Impulse noise
fig = plt.figure(dpi=300)
fig.add_subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Original")

fig.add_subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(imp_noise_color, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Impulse Noise")

fig.add_subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(in_img, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Combined")

plt.show()

# Reset variables for next noise type
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

# Adding Uniform noise
uni_noise = np.zeros_like(img_gray, dtype=np.uint8)
cv2.randu(uni_noise, 0, 255)
uni_noise = (uni_noise * 0.5).astype(np.uint8)
un_img = cv2.add(img_gray, uni_noise)

# Convert the noise to color for display
uni_noise_color = cv2.cvtColor(uni_noise, cv2.COLOR_GRAY2BGR)

# Display images for Uniform noise
fig = plt.figure(dpi=300)
fig.add_subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(img_color, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Original")

fig.add_subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(uni_noise_color, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Uniform Noise")

fig.add_subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(un_img, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Combined")

plt.show()

# Reset variables for next filter type
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)

# Applying Median filters
blurred1 = cv2.medianBlur(gn_img, 3)
blurred2 = cv2.medianBlur(un_img, 3)
blurred3 = cv2.medianBlur(in_img, 3)

# Display images for Median filters
fig = plt.figure(dpi=300)
```

```
fig.add_subplot(1, 3, 1)
plt.imshow(cv2.cvtColor(blurred1, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Median Gaussian")

fig.add_subplot(1, 3, 2)
plt.imshow(cv2.cvtColor(blurred2, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Median Uniform")

fig.add_subplot(1, 3, 3)
plt.imshow(cv2.cvtColor(blurred3, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Median Impulse")

plt.show()

# Applying Average filter
img_new = cv2.blur(gn_img, (3, 3))

# Display images for Average filter
fig = plt.figure(dpi=300)
fig.add_subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(gn_img, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Original")
fig.add_subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(img_new, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title("Average Filter")
plt.show()
```

OUTPUT:





RESULT:

Thus the program executed successfully and the outputs are verified.

EX.NO:03 **EDGE DETECTION**

AIM:

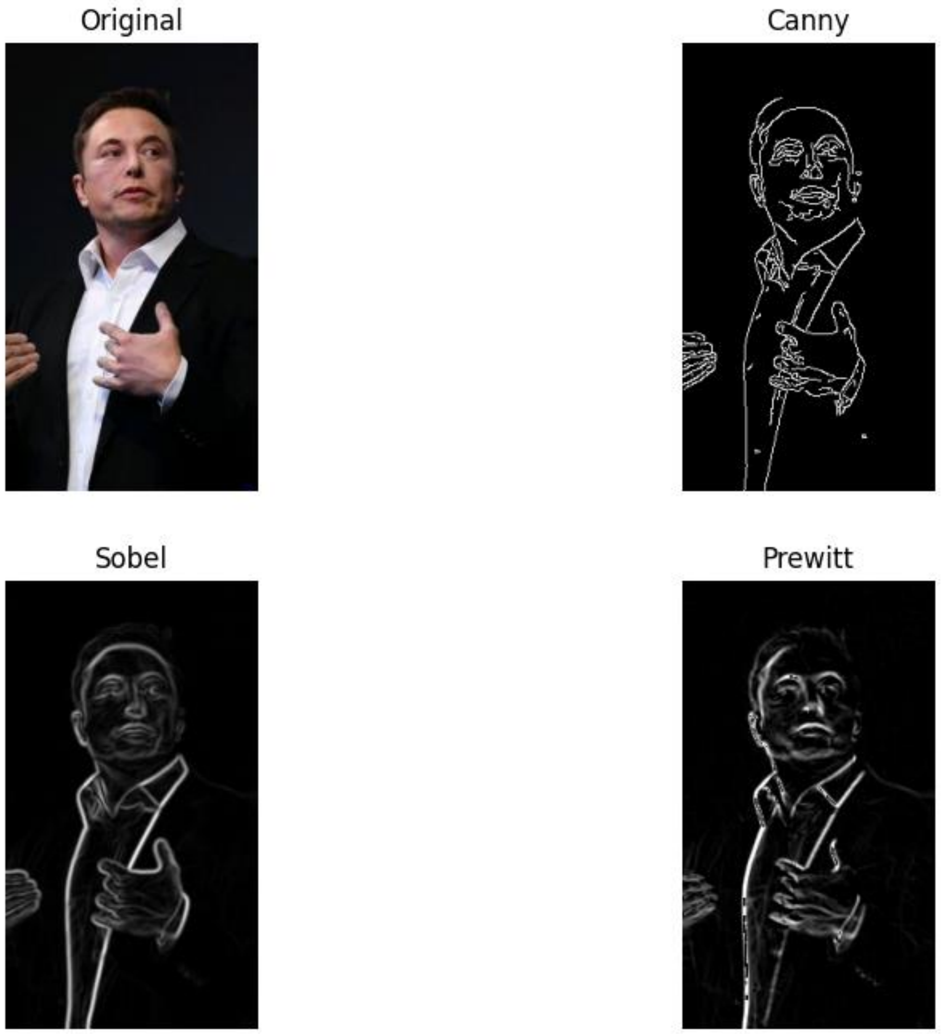
To explore and compare the performance of popular edge detection algorithms, namely **Sobel, Prewitt, Roberts, and Canny**, in identifying edges within digital images and assess their suitability for various applications.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Read the image
img = cv2.imread('/content/sample_data/elon musk.jpg')

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Apply Gaussian Blur
img_gaussian = cv2.GaussianBlur(gray, (3, 3), 0)

# Canny edge detection
img_canny = cv2.Canny(img, 100, 200)

# Sobel edge detection
img_sobelx = cv2.Sobel(img_gaussian, cv2.CV_64F, 1, 0, ksize=5)
img_sobely = cv2.Sobel(img_gaussian, cv2.CV_64F, 0, 1, ksize=5)
img_sobel = np.sqrt(img_sobelx**2 + img_sobely**2)

# Prewitt edge detection
kernelx = np.array([[1, 1, 1], [0, 0, 0], [-1, -1, -1]])
kernely = np.array([[-1, 0, 1], [-1, 0, 1], [-1, 0, 1]])
img_prewittx = cv2.filter2D(img_gaussian, -1, kernelx)
img_prewitty = cv2.filter2D(img_gaussian, -1, kernely)
img_prewitt = img_prewittx + img_prewitty

# Plotting
fig, axs = plt.subplots(2, 2, figsize=(10, 8))

axs[0, 0].imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
axs[0, 0].set_title('Original')
axs[0, 0].axis('off')

axs[0, 1].imshow(img_canny, cmap='gray')
axs[0, 1].set_title('Canny')
axs[0, 1].axis('off')

axs[1, 0].imshow(img_sobel, cmap='gray')
axs[1, 0].set_title('Sobel')
axs[1, 0].axis('off')

axs[1, 1].imshow(img_prewitt, cmap='gray')
axs[1, 1].set_title('Prewitt')
```

```
axs[1, 1].axis('off')
plt.show()
```

OUTPUT:



RESULT:

Thus the program executed successfully and the outputs are verified.

EX NO: 4 **EROSION, DILATION, OPENING, AND CLOSING**

AIM:
The aim of this exercise is to apply morphological operations to binary images.

ALGORITHM

1. Define the input binary image and the structuring element (SE).
2. Initialize result matrices for dilation, erosion, opening, and closing.
3. Implement the erosion operation:
   - Iterate through the image pixels, excluding the border.
   - For each pixel, find the minimum value in the SE-neighborhood.
   - Update the corresponding pixel in the erosion result matrix.
4. Implement the dilation operation:
   - Iterate through the image pixels, excluding the border.
   - For each pixel, find the maximum value in the SE-neighborhood.
   - Update the corresponding pixel in the dilation result matrix.
5. Implement opening using erosion and dilation:
   - Erode the input image using the SE to obtain the eroded image.
   - Dilate the eroded image using the same SE to get the opened image.
   - The opened image is the result of the opening operation.
6. Implement closing using dilation and erosion:
   - Dilate the input image using the SE to obtain the dilated image.
   - Erode the dilated image using the same SE to get the closed image.
   - The closed image is the result of the closing operation.
7. To perform boundary extraction using erosion:
   - Erode the input image using the SE to obtain the eroded image.
   - Subtract the eroded image from the original image to get the boundary image.
8. To perform boundary extraction using dilation:
   - Dilate the input image using the SE to obtain the dilated image.
   - Subtract the original image from the dilated image to get the boundary image.

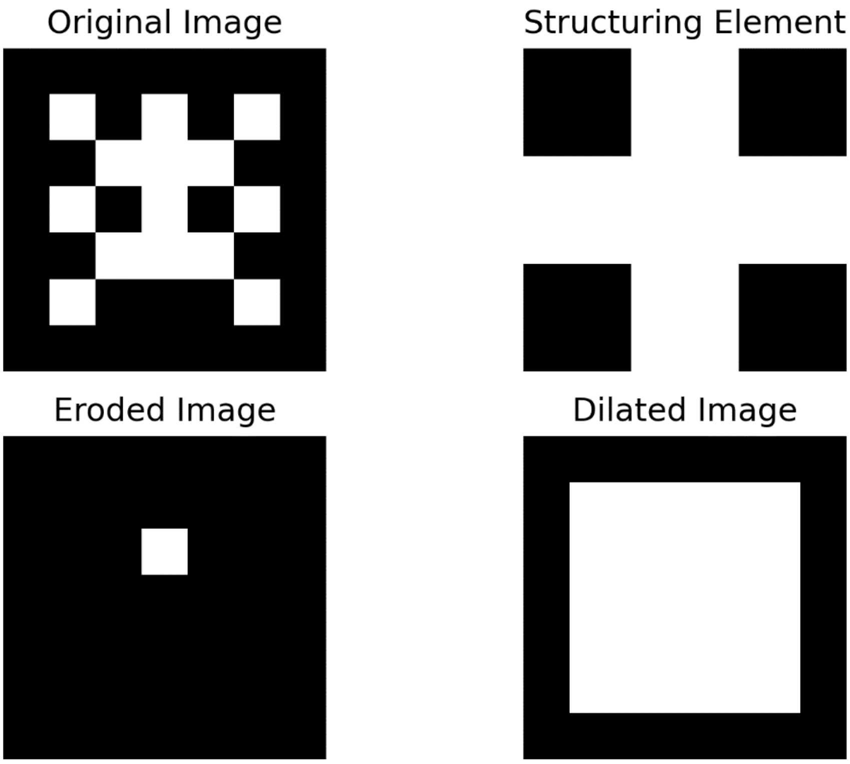
CODE:

DILATION AND EROSION:

```
import matplotlib.pyplot as plt
def erosion(image, se):
    m, n = len(image), len(image[0])
    result = [[0 for _ in range(n)] for _ in range(m)]
    for i in range(1, m - 1):
        for j in range(1, n - 1):
            min_val = 255
            for k in range(-1, 2):
                for l in range(-1, 2):
                    if se[k + 1][l + 1] == 255:
                        min_val = min(min_val, image[i + k][j + l])
            result[i][j] = min_val
    return result
def dilation(image, se):
    m, n = len(image), len(image[0])
    result = [[0 for _ in range(n)] for _ in range(m)]
    for i in range(1, m - 1):
```

```
        for j in range(1, n - 1):
            max_val = 0
            for k in range(-1, 2):
                for l in range(-1, 2):
                    if se[k + 1][l + 1] == 255:
                        max_val = max(max_val, image[i + k][j + l])
            result[i][j] = max_val
    return result
image = [
    [0, 0, 0, 0, 0, 0, 0],
    [0, 255, 0, 255, 0, 255, 0],
    [0, 0, 255, 255, 255, 0, 0],
    [0, 255, 0, 255, 0, 255, 0],
    [0, 0, 255, 255, 255, 0, 0],
    [0, 255, 0, 0, 0, 255, 0],
    [0, 0, 0, 0, 0, 0, 0]
]
se = [
    [0, 255, 0],
    [255, 255, 255],
    [0, 255, 0]
]
eroded_image = erosion(image, se)
dilated_image = dilation(image, se)
fig = plt.figure(dpi=300)
fig.add_subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.axis("off")
plt.title("Original Image")
fig.add_subplot(2, 2, 2)
plt.imshow(se, cmap='gray')
plt.axis("off")
plt.title("Structuring Element")
fig.add_subplot(2, 2, 3)
plt.imshow(eroded_image, cmap='gray')
plt.axis("off")
plt.title("Eroded Image")
fig.add_subplot(2, 2, 4)
plt.imshow(dilated_image, cmap='gray')
plt.axis("off")
plt.title("Dilated Image")
plt.show()
```

OUTPUT:

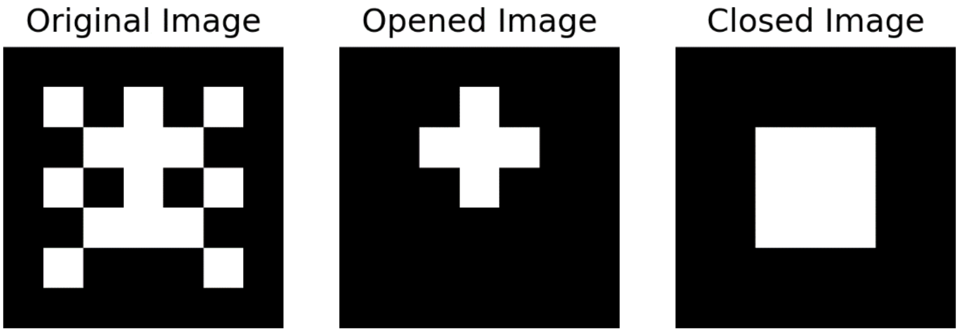


OPENING AND CLOSING:

```
import matplotlib.pyplot as plt
def opening(image, se):
    eroded = erosion(image, se)
    opened = dilation(eroded, se)
    return opened
def closing(image, se):
    dilated = dilation(image, se)
    closed = erosion(dilated, se)
    return closed
image = [
    [0, 0, 0, 0, 0, 0, 0],
    [0, 255, 0, 255, 0, 255, 0],
    [0, 0, 255, 255, 255, 0, 0],
    [0, 255, 0, 255, 0, 255, 0],
    [0, 0, 255, 255, 255, 0, 0],
    [0, 255, 0, 0, 0, 255, 0],
    [0, 0, 0, 0, 0, 0, 0]
]
se = [
    [0, 255, 0],
    [255, 255, 255],
    [0, 255, 0]
]
opened_image = opening(image, se)
closed_image = closing(image, se)
fig = plt.figure(dpi=300)
fig.add_subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
```

```
plt.axis("off")
plt.title("Original Image")
fig.add_subplot(1, 3, 2)
plt.imshow(opened_image, cmap='gray')
plt.axis("off")
plt.title("Opened Image")
fig.add_subplot(1, 3, 3)
plt.imshow(closed_image, cmap='gray')
plt.axis("off")
plt.title("Closed Image")
plt.show()
```

OUTPUT:

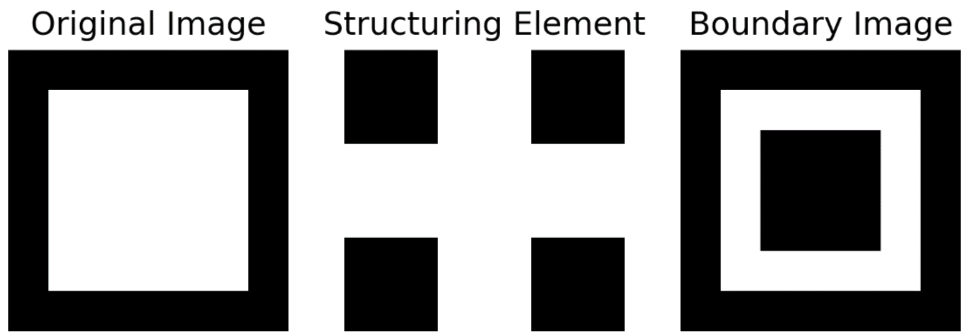


BOUNDARY EXTRACTION:

```
import numpy as np
# Original image and structuring element
image = [
   [0, 0, 0, 0, 0, 0, 0],
   [0, 255, 255, 255, 255, 255, 0],
   [0, 255, 255, 255, 255, 255, 0],
   [0, 255, 255, 255, 255, 255, 0],
   [0, 255, 255, 255, 255, 255, 0],
   [0, 255, 255, 255, 255, 255, 0],
   [0, 0, 0, 0, 0, 0, 0]
]
se = [
   [0, 255, 0],
   [255, 255, 255],
   [0, 255, 0]
]
# Compute erosion
eroded_image = erosion(image, se)
# Compute boundary by subtracting eroded image from original image
boundary_image = np.subtract(image, eroded_image)
# Display the boundary image
fig = plt.figure(dpi=300)
fig.add_subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.axis("off")
plt.title("Original Image")
fig.add_subplot(1, 3, 2)
```

```
plt.imshow(se, cmap='gray')
plt.axis("off")
plt.title("Structuring Element")
fig.add_subplot(1, 3, 3)
plt.imshow(boundary_image, cmap='gray')
plt.axis("off")
plt.title("Boundary Image")
plt.show()
```

OUTPUT:



RESULT:

The application of morphological operations, including dilation, erosion, opening, closing, and boundary extraction, to binary images is successfully done and output is verified.

EX NO: 05 **LINE DETECTION USING HOUGH METHOD**

AIM:

Detect and visualize lines in an image using the Canny edge detection and Hough transform techniques.

ALGORITHM:

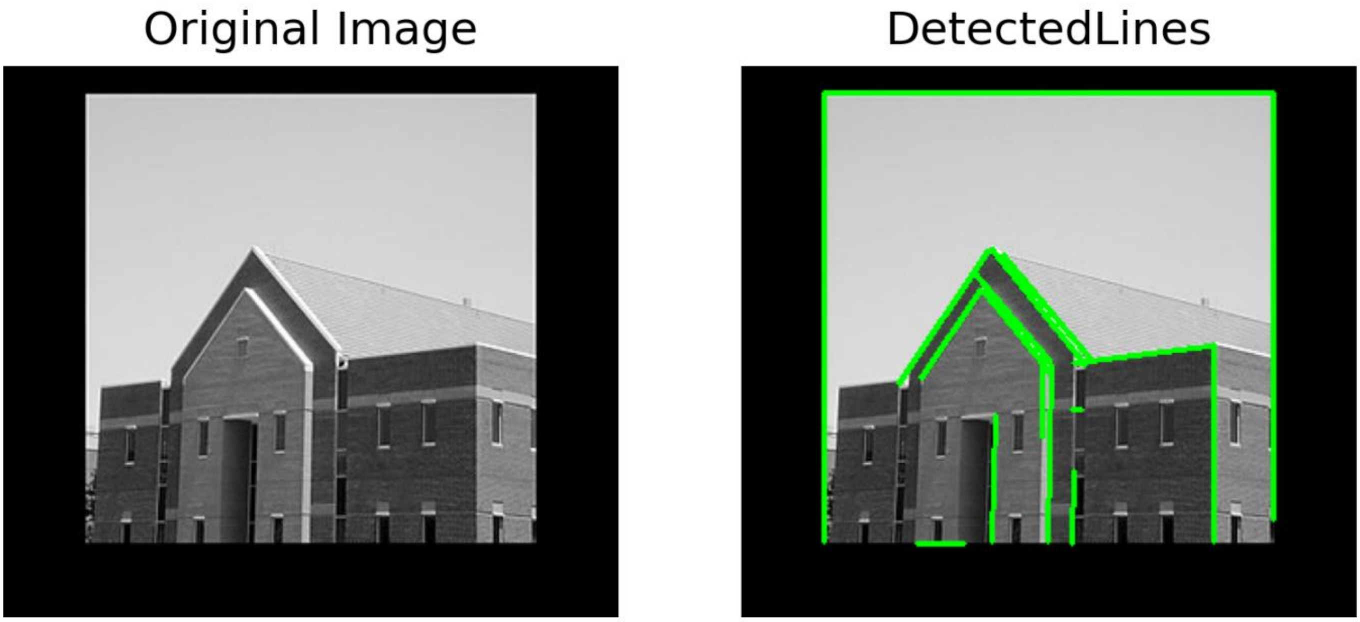
1. Read the input image.
2. Convert the image to grayscale.
3. Apply the Canny edge detection algorithm to the grayscale image.
4. Use the HoughLinesP method to detect lines in the edge image.
5. Specify parameters for the Hough transform, including distance and angle resolutions, a threshold for minimum votes, minimum line length, and maximum gap between lines.
6. Iterate through the detected lines and extract their endpoints.
7. Draw the detected lines on the original image in green.
8. Maintain a list to store the endpoints of the detected lines.
9. Display the original image with detected lines.
10. Output the result showing the original image with highlighted detected lines.

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read image
image = cv2.imread('/content/sample_data/line.jpg')
image=cv2.cvtColor(image,cv2.COLOR_BGR2RGB)
fig=plt.figure(dpi=300)
fig.add_subplot(1,2,1)
plt.imshow(image)
plt.axis("off")
plt.title('Original Image')
(x,y)=image.shape[:2]
# Convert image to grayscale
img=cv2.imread('/content/sample_data/line.jpg',0)
gray = cv2.cvtColor(img,cv2.COLOR_BGR2RGB)
# Use canny edge detection
edges = cv2.Canny(gray,x,y,apertureSize=3)
# Apply HoughLinesP method to directly obtain line end points
lines_list =[]
lines = cv2.HoughLinesP(
    edges, # Input edge image
    1, # Distance resolution in pixels
    np.pi/180, # Angle resolution in radians
    threshold=50, # Min number of votes for valid line
    minLineLength=5, # Min allowed length of line
    maxLineGap=10 # Max allowed gap between line for joining them
    )
# Iterate over points
for points in lines:
  # Extracted points nested in the list
  x1,y1,x2,y2=points[0]
  # Draw the lines joing the points
```

```
  # On the original image
  cv2.line(image,(x1,y1),(x2,y2),(0,255,0),2)
  # Maintain a simples lookup list for points
  lines_list.append([(x1,y1),(x2,y2)])
fig.add_subplot(1,2,2)
plt.imshow(image)
plt.axis("off")
plt.title('DetectedLines')
```

OUTPUT:



Result

Thus, the line detection using hough method is successfully written and output is verified.

EX NO: 06 IMAGE SEGMENTATION WITH WATERSHED ALGORITHM

AIM:

To Segment objects in an image using the Watershed algorithm to distinguish between the foreground and background regions.

ALGORITHM:

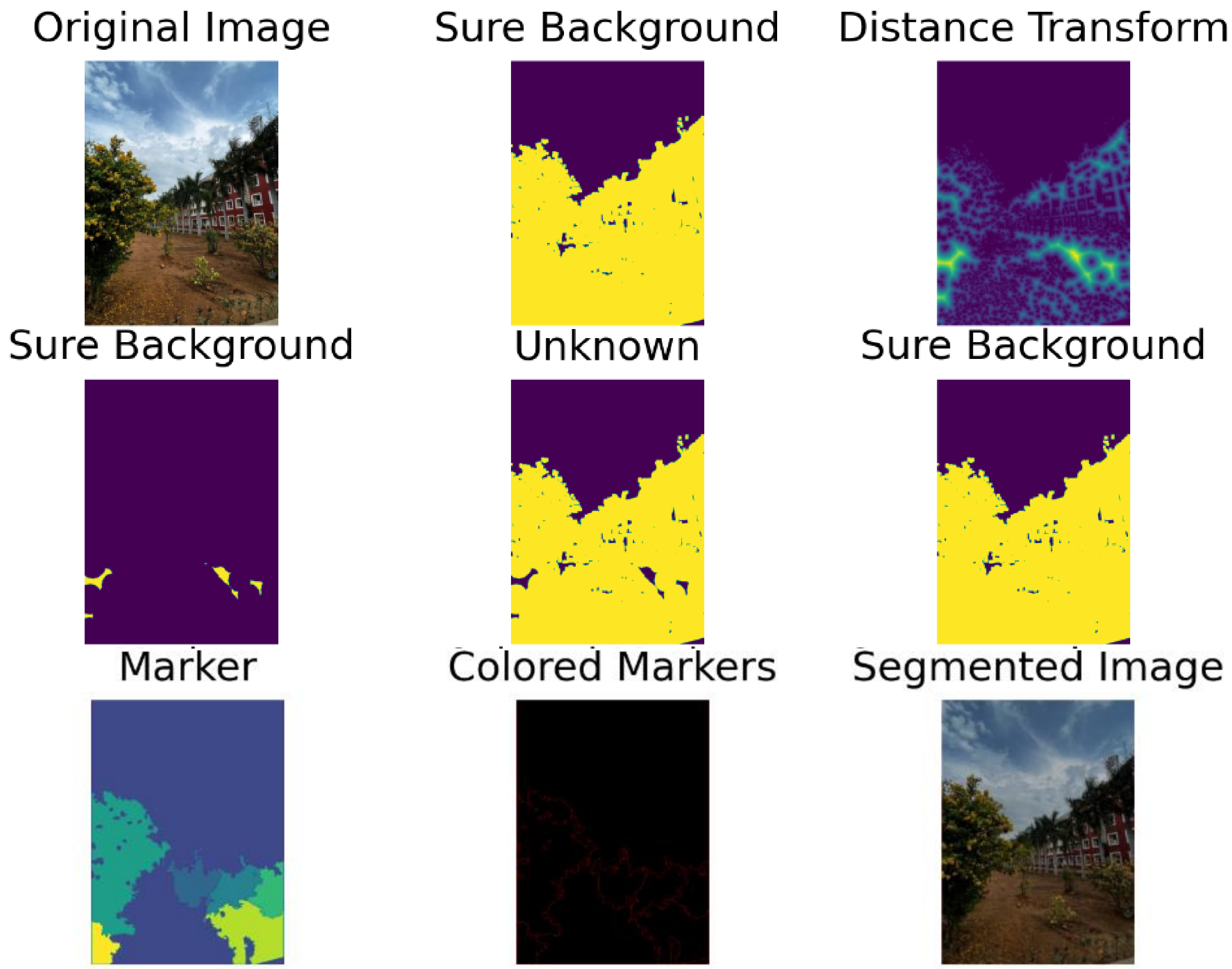
1. Load the input image.
2. Convert the image to grayscale and ensure it's in RGB format for visualization.
3. Apply Otsu's thresholding to obtain a binary image, emphasizing object boundaries.
4. Perform morphological opening to remove noise in the binary image.
5. Identify the sure background region by dilation.
6. Calculate the distance transform of the binary image to find the sure foreground.
7. Create a marker image, and mark sure foreground and unknown regions.
8. Apply the Watershed algorithm using the marker image to segment objects.
9. Apply colormap to the markers, highlighting object boundaries.
10. Generate the segmented image by overlaying the colored markers on the original image.

CODE:

```
#IMAGE SEGMENTATION WITH WATERSHED ALGORITHM
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Load the image
image = cv2.imread('/content/sample_data/VCET2.jpg')
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
image=cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
fig=plt.figure(dpi=300)
fig.add_subplot(3,3,1)
plt.imshow(image)
plt.axis("off")
plt.title('Original Image')
ret, threshold = cv2.threshold(gray, 0, 255,  cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
# Perform morphological opening to remove noise
kernel = np.ones((3, 3), np.uint8)
opening = cv2.morphologyEx(threshold, cv2.MORPH_OPEN, kernel, iterations=2)
# Find background region
sure_bg = cv2.dilate(opening, kernel, iterations=3)
# Find foreground region
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)
# Create marker image
sure_fg = np.uint8(sure_fg)
unknown = cv2.subtract(sure_bg, sure_fg)
# Apply Watershed algorithm
ret, markers = cv2.connectedComponents(sure_fg)
markers = markers + 1
```

```
markers[unknown == 255] = 0
cv2.watershed(image, markers)
# Apply colormap to the markers
colored_markers = np.zeros_like(image)
colored_markers[markers == -1] = [255, 0, 0]  # Boundaries in blue color
# Display the segmented image
segmented_image = cv2.addWeighted(image, 0.7, colored_markers, 0.3, 0)
fig.add_subplot(3,3,2)
plt.imshow(sure_bg)
plt.axis("off")
plt.title('Sure Background')

fig.add_subplot(3,3,3)
plt.imshow(dist_transform)
plt.axis("off")
plt.title('Distance Transform')
fig.add_subplot(3,3,4)

plt.imshow(sure_fg)
plt.axis("off")
plt.title('Sure Background')
fig.add_subplot(3,3,5)

plt.imshow(unknown)
plt.axis("off")
plt.title('Unknown')
fig.add_subplot(3,3,6)

plt.imshow(sure_bg)
plt.axis("off")
plt.title('Sure Background')
fig.add_subplot(3,3,7)

plt.imshow(markers)
plt.axis("off")
plt.title('Marker')
fig.add_subplot(3,3,8)

plt.imshow(colored_markers)
plt.axis("off")
plt.title('Colored Markers')
fig.add_subplot(3,3,9)

plt.imshow(segmented_image)
plt.axis("off")
plt.title('Segmented Image')
plt.show()
```

OUTPUT:



RESULT:

The code segments objects in the image using the Watershed algorithm, showing the sure background, distance transform, sure foreground, and the final segmented image with object boundaries marked in red.

EX NO: 7 3D SHAPE FROM TEXTURE AND 3D OBJECT DETECTION

AIM:

Estimate the 3D shape of an object from a texture image using a basic gradient-based method (Shape-from-Shading) and visualize the estimated surface normals.

3D SHAPE FROM TEXTURE

ALGORITHM:

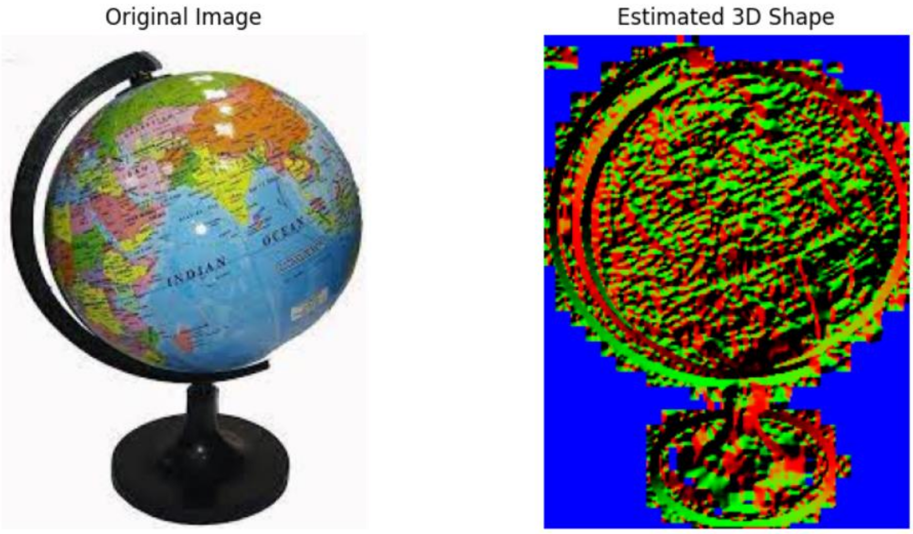
1. Convert the input texture image to grayscale.
2. Apply gradient operations (Sobel filters) to calculate gradients in the x and y directions.
3. Create a constant gradient component in the z-direction to represent surface depth.
4. Combine the gradient components into a 3D vector for surface normals.
5. Normalize the surface normals to ensure consistent magnitude.
6. Return the estimated 3D shape (surface normals).

CODE:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def estimate_3d_shape_from_texture(texture_image):
    # Convert the texture image to grayscale
    gray = cv2.cvtColor(texture_image, cv2.COLOR_BGR2GRAY)
    # Apply a simple gradient-based method (Shape-from-Shading)
    # Note: This is a very basic example and may not work well in many cases
    gradient_x = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=5)
    gradient_y = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=5)
    gradient_z = np.ones_like(gradient_x)

    # Calculate the surface normals
    surface_normals = np.dstack((gradient_x, gradient_y, gradient_z))
    surface_normals /= np.linalg.norm(surface_normals, axis=-1, keepdims=True)
    return surface_normals

# Example usage
if __name__ == "__main__":
    # Load a texture image (replace 'globe.jpg' with your own texture)
    texture_image = cv2.imread('/content/sample_data/globe.jpg')
    texture_image = cv2.cvtColor(texture_image, cv2.COLOR_BGR2RGB)

    # Estimate the 3D shape
    estimated_3d_shape = estimate_3d_shape_from_texture(texture_image)

    # Display the result
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(texture_image)
    plt.axis("off")
    plt.title('Original Image')
```

```
plt.subplot(1, 2, 2)
plt.imshow(estimated_3d_shape)
plt.axis("off")
plt.title('Estimated 3D Shape')

plt.show()
```

OUTPUT:



3D OBJECT DETECTION

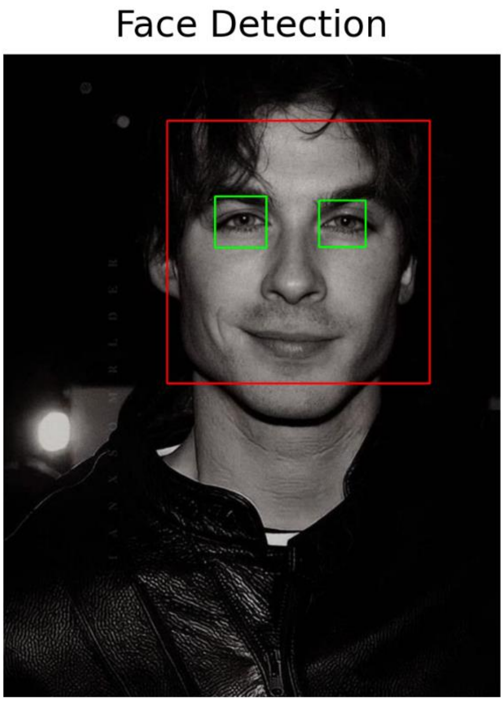
ALGORITHM

1. Load the image in which you want to perform face and eye detection.
2. Convert the image to grayscale for efficient processing.
3. Create Haar Cascade classifiers for both face and eye detection.
4. Use the face cascade classifier to detect faces in the grayscale image, specifying scaling parameters (scale factor and minimum neighbors).
5. Iterate through the detected face regions.
6. Draw rectangles around the detected faces on the original color image.
7. Create regions of interest (ROI) in both grayscale and color based on the face bounding box.
8. Use the eye cascade classifier to detect eyes within each face ROI.
9. Iterate through the detected eye regions within each face.
10. Draw rectangles around the detected eyes on the color face ROI.
11. Continue this process for all detected faces in the image.
12. Display the image with rectangles drawn around the detected faces and eyes.
13. Optionally, save or analyze the results for further processing.
14. The code effectively detects faces and eyes in the input image, highlighting them with rectangles.
15. Adjust the scaling parameters as needed to fine-tune the detection results.

CODE:

```
import numpy as np
import cv2
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
img = cv2.imread("/content/sample_data/Mr.Lan Somerhalder.jpg")
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
#faces = face_cascade.detectMultiScale(gray)
for (x,y,w,h) in faces:
   cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
   roi_gray = gray[y:y+h, x:x+w]
   roi_color = img[y:y+h, x:x+w]
   eyes = eye_cascade.detectMultiScale(roi_gray)
   for (ex,ey,ew,eh) in eyes:
       cv2.rectangle(roi_color,(ex,ey),(ex+ew,ey+eh),(0,255,0),2)
fig=plt.figure(dpi=300)
fig.add_subplot(1,2,1)
plt.imshow(img)
plt.axis("off")
plt.title('Face Detection')
```

OUTPUT:



RESULT:

Thus, 3D Shape from texture and 3D Object is successfully done.