ADDITIONAL EXERCISES – IMAGE PROCESSING

Ex. 1 Median Blur

Aim

To apply a **median filter** in the spatial domain to a given grayscale image, effectively reducing noise—especially **salt-and-pepper noise**—while preserving sharp edges and details.

Theoretical Concept

The median blur is a **non-linear filtering** technique used for image smoothing and noise reduction. It works by replacing the central pixel in a defined neighborhood (called the **kernel** or window) with the **median** value of all the pixels in that neighborhood.

Unlike an average (or mean) filter, which replaces the pixel with the average value, the median filter is highly effective against **impulsive noise**, such as salt-and-pepper noise. Salt-and-pepper noise consists of random dark (pepper) or bright (salt) pixels that are significantly different from their surroundings.

Here's how it works:

1. A kernel (e.g., a 3x3 or 5x5 square) slides over the image.
2. For each pixel, the values of all pixels within the kernel are collected.
3. These values are then sorted in numerical order.
4. The middle value (the median) is selected from the sorted list.
5. The original pixel's value is replaced with this median.

Because the median is a robust statistical measure, it is not significantly affected by extreme outlier values (the noisy pixels), allowing the filter to completely remove the noise without averaging it into the surrounding pixels. This is a key advantage, as it helps to **preserve edges** and other fine details better than a simple blurring filter. .

Implementation

```
import cv2

import numpy as np

from matplotlib import pyplot as plt


# Load the image

img =
cv2.imread(r"E:\study5sem\lab\3d085e3ae6a03dd4ffda19bb7eb27738.jpg")


# Convert from BGR to RGB for correct display in matplotlib

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


# Apply the median blur filter

# The second parameter is the kernel size, which must be a positive odd integer.

median = cv2.medianBlur(img, 5)


# Convert the blurred image to RGB as well

median_rgb = cv2.cvtColor(median, cv2.COLOR_BGR2RGB)


# Display the original and blurred images side-by-side

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


# Original Image

plt.subplot(1, 2, 1)
```

```
plt.imshow(img_rgb)
plt.title('Original')
plt.xticks([]), plt.yticks([])

# Median Blurred Image
plt.subplot(1, 2, 2)
plt.imshow(median_rgb)
plt.title('Median Blurred')
plt.xticks([]), plt.yticks([])

plt.show()
```

Sample Input and Output

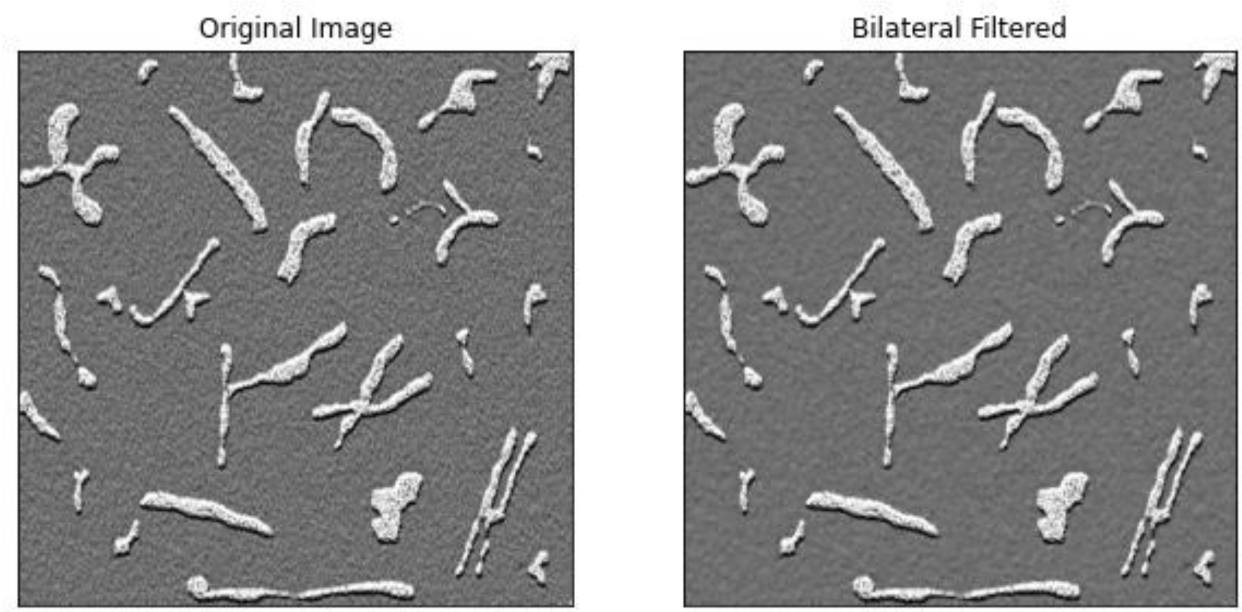
Bilateral filter

Implementation

```
import cv2

import numpy as np

from matplotlib import pyplot as plt


# Load the image

img =
cv2.imread(r"C:\Users\Intel\Downloads\DIP3E_Problem_Figures\CH10_Problem_
Figures\FigP1036.tif")


# Convert from BGR to RGB for correct display in matplotlib

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


# Apply the bilateral filter

bilateral = cv2.bilateralFilter(img, 9, 75, 75)


# Convert the filtered image to RGB for display

bilateral_rgb = cv2.cvtColor(bilateral, cv2.COLOR_BGR2RGB)


# Display the original and filtered images

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


# Original Image
```

```
plt.subplot(1, 2, 1)
plt.imshow(img_rgb)
plt.title('Original Image')
plt.xticks([]), plt.yticks([])

# Bilateral Filtered Image
plt.subplot(1, 2, 2)
plt.imshow(bilateral_rgb)
plt.title('Bilateral Filtered')
plt.xticks([]), plt.yticks([])

plt.show()
```

Sample Input and Output

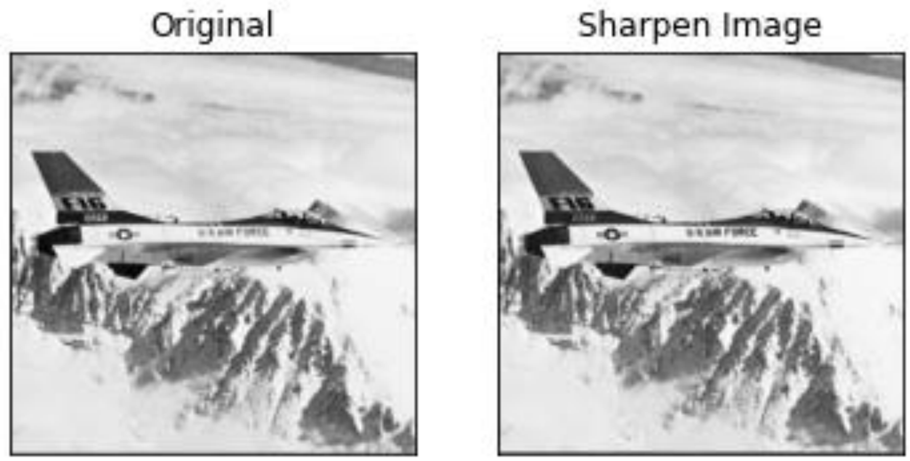


LAPLACIAN BASED SHARPENING

Implementation

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

img = cv2.imread(r"E:\study5sem\lab\airplane.bmp", 1)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
laplacian = cv2.Laplacian(gray, cv2.CV_64F)
laplacian = cv2.convertScaleAbs(laplacian)
sharpen_img = cv2.addWeighted(gray, 1, laplacian, -0.5, 0)

plt.subplot(121)
plt.imshow(gray, cmap='gray')
plt.title('Original')
plt.xticks([]), plt.yticks([])

plt.subplot(122)
plt.imshow(sharpen_img, cmap='gray')
plt.title('Sharpen Image')
plt.xticks([]), plt.yticks([])

plt.show()
```

Sample Input and Output

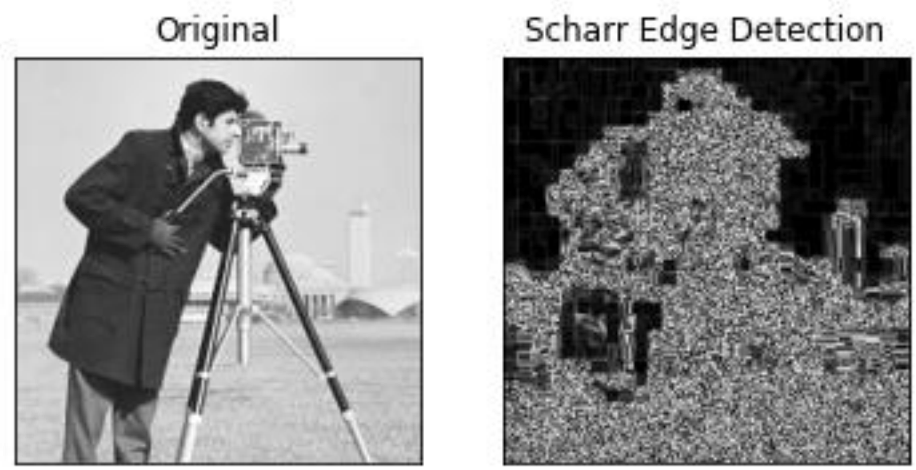


Ex. 2 **SCHARR EDGE DETECTION**

Implementation

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

img = cv.imread( "E:\study5sem\lab\images (1).jpg", 0)
scharrx = cv.Scharr(img, cv.CV_64F, 1, 0)
scharry = cv.Scharr(img, cv.CV_64F, 0, 1)
scharr = np.hypot(scharrx, scharry)
scharr = np.uint8(np.absolute(scharr))

plt.subplot(121)
plt.imshow(img, cmap='gray')
plt.title('Original')
plt.xticks([]), plt.yticks([])

plt.subplot(122)
plt.imshow(scharr, cmap='gray')
plt.title('Scharr Edge Detection')
plt.xticks([]), plt.yticks([])

plt.show()
```

Sample Input and Output

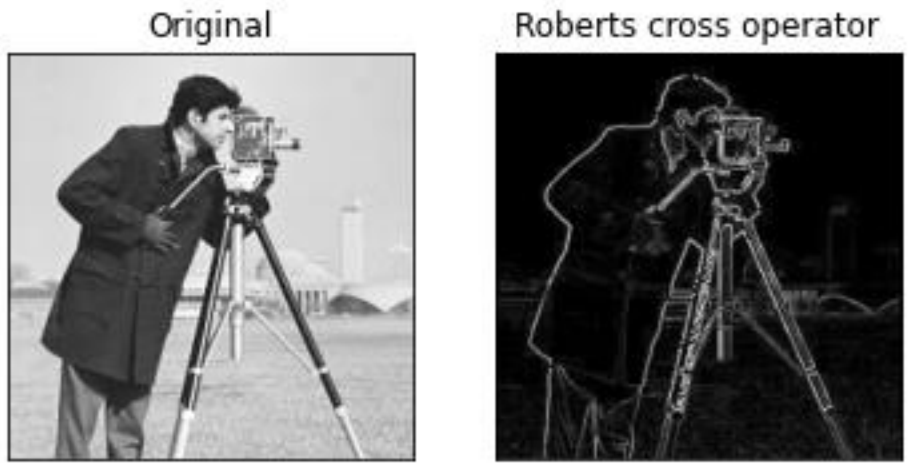


ROBERTS CROSS OPERATOR

Implementation

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

img = cv.imread("E:\study5sem\lab\images (1).jpg", 0)

kernel_roberts_x = np.array([[1, 0], [0, -1]], dtype=np.float32)
kernel_roberts_y = np.array([[0, 1], [-1, 0]], dtype=np.float32)

roberts_x = cv.filter2D(img, cv.CV_64F, kernel_roberts_x)
roberts_y = cv.filter2D(img, cv.CV_64F, kernel_roberts_y)

roberts = np.sqrt(np.square(roberts_x) + np.square(roberts_y))
roberts = np.uint8(np.absolute(roberts))

plt.subplot(121), plt.imshow(img, cmap='gray')
plt.title("Original"), plt.xticks([]), plt.yticks([])

plt.subplot(122), plt.imshow(roberts, cmap='gray')
plt.title("Roberts cross operator"), plt.xticks([]), plt.yticks([])

plt.show()
```

Sample Input and Output

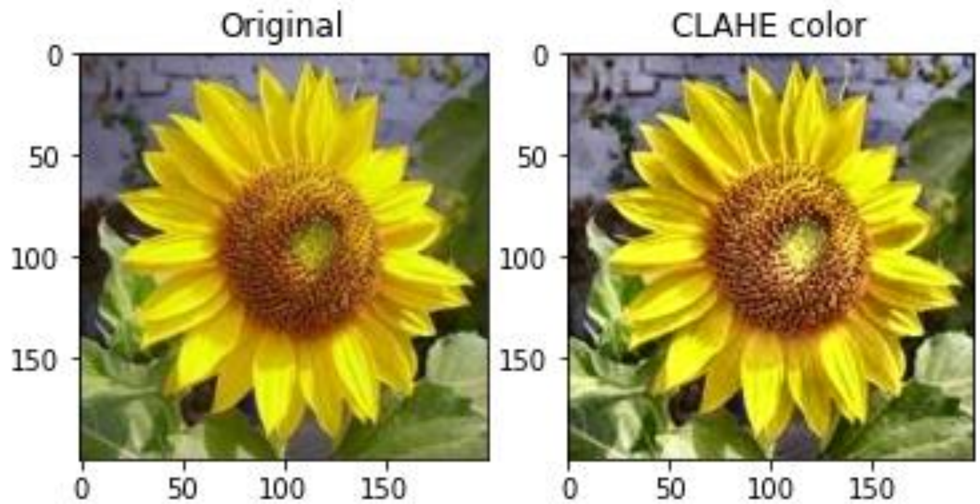


Ex. 3 **Adaptive Histogram Equalization(CLAHE)**

Implementation

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("E:\study5sem\lab\download (1).jpg")

ycrcb = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb)

clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
ycrcb[:, :, 0] = clahe.apply(ycrcb[:, :, 0])

clahe_color = cv2.cvtColor(ycrcb, cv2.COLOR_YCrCb2BGR)

plt.subplot(1, 2, 1)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title("Original")

plt.subplot(1, 2, 2)
plt.imshow(cv2.cvtColor(clahe_color, cv2.COLOR_BGR2RGB))
plt.title("CLAHE color")

plt.show()
```

Sample Input and Output



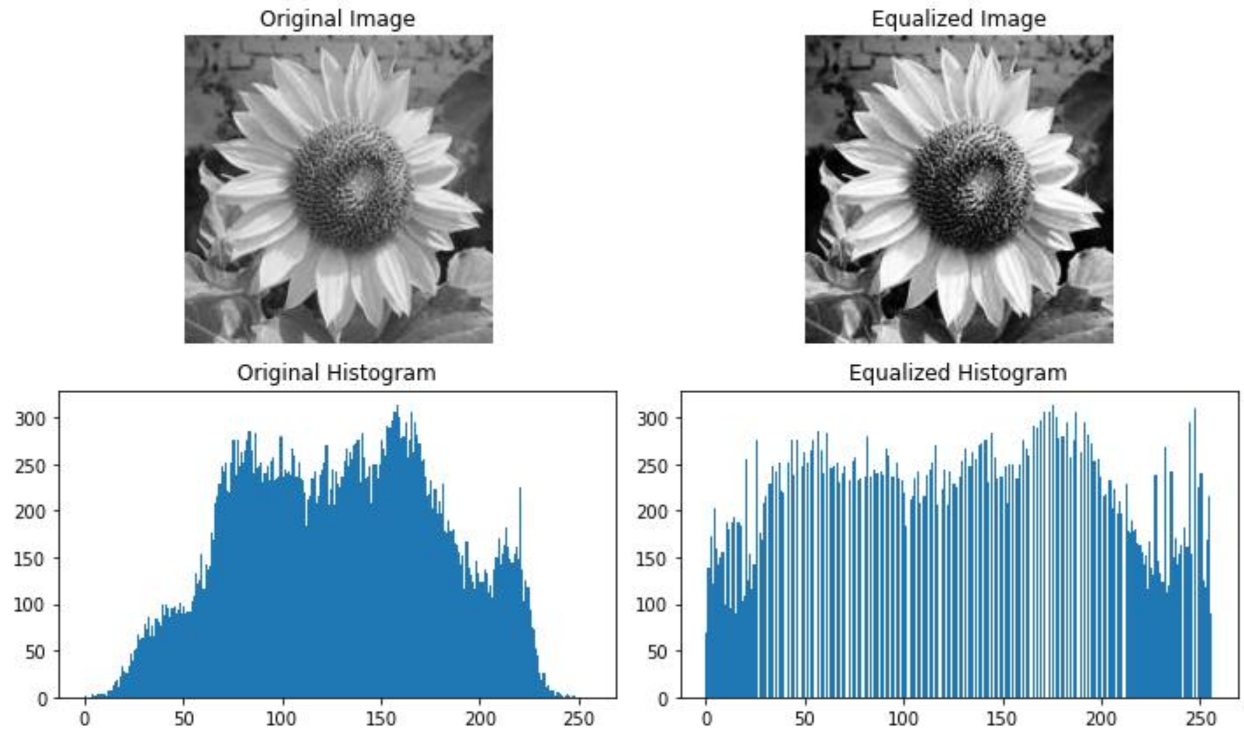
Synthetic Image

Implementation

```
import cv2
import matplotlib.pyplot as plt

# Read the image in grayscale
img = cv2.imread("E:\study5sem\lab\download (1).jpg", 0)

# Apply histogram equalization
equalized = cv2.equalizeHist(img)

# Create a figure
plt.figure(figsize=(10, 6))

# Original image
plt.subplot(2, 2, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Image")
plt.axis("off")

# Equalized image
plt.subplot(2, 2, 2)
plt.imshow(equalized, cmap='gray')
plt.title("Equalized Image")
```

```
plt.axis("off")

# Histogram of original image
plt.subplot(2, 2, 3)
plt.hist(img.ravel(), 256, [0, 256])
plt.title("Original Histogram")

# Histogram of equalized image
plt.subplot(2, 2, 4)
plt.hist(equalized.ravel(), 256, [0, 256])
plt.title("Equalized Histogram")

# Adjust layout and show
plt.tight_layout()
plt.show()
```

Sample Input and Output



Adaptive histogram equalization (AHE)

Implementation

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


def adaptive_hist_equalization(img, tile_size=(8, 8)):

    # Get image shape

    h, w = img.shape

    th, tw = tile_size


    # Output image

    out = np.zeros_like(img)


    # Process each tile

    for i in range(0, h, th):

        for j in range(0, w, tw):

            # Define tile region

            tile = img[i:min(i+th, h), j:min(j+tw, w)]


            # Apply histogram equalization on the tile

            equalized_tile = cv2.equalizeHist(tile)


            # Put back into output
```

```
            out[i:min(i+th, h), j:min(j+tw, w)] = equalized_tile

    return out



# Read grayscale image
# Make sure "ship.jpg" is in the same directory as the script
img = cv2.imread("E:\study5sem\lab\download (1).jpg", 0)

# Apply AHE
ahe_img = adaptive_hist_equalization(img, tile_size=(32, 32))

# Apply CLAHE (for comparison)
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
clahe_img = clahe.apply(img)

# Show results using matplotlib
plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original')
plt.axis('off')
```

```
plt.subplot(1, 3, 2)
plt.imshow(ahe_img, cmap='gray')
plt.title('AHE (Adaptive Histogram Equalization)')
plt.axis('off')

plt.subplot(1, 3, 3)
plt.imshow(clahe_img, cmap='gray')
plt.title('CLAHE')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Sample Input and Output

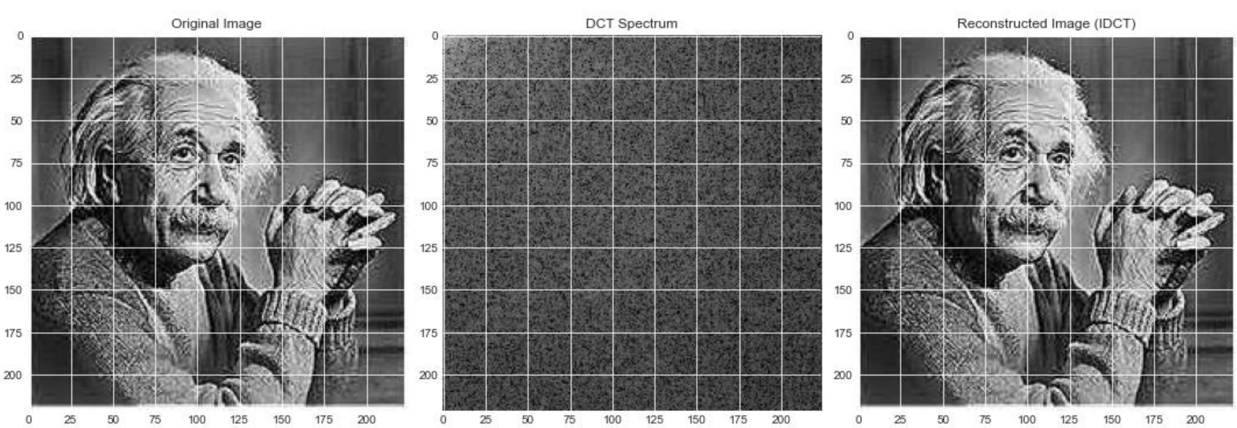
Ex. 6 **Agglomerative clustering**

Implementation

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

# Load grayscale image
img = cv2.imread("E:\study5sem\lab\download.jpg", 0)

# Convert to float32 for DCT
img_float = np.float32(img)

# Apply DCT
dct = cv2.dct(img_float)

# Apply Inverse DCT
idct = cv2.idct(dct)

# Plot results
plt.figure(figsize=(15, 8))

plt.subplot(131)
plt.imshow(img, cmap="gray")
plt.title("Original Image")
```

```
plt.subplot(132)
plt.imshow(np.log(1 + np.abs(dct)), cmap="gray")
plt.title("DCT Spectrum")

plt.subplot(133)
plt.imshow(idct, cmap="gray")
plt.title("Reconstructed Image (IDCT)")

plt.tight_layout()
plt.show()
```

Sample Input and Output



GrabCut Segmentation

Implementation

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Load the image

img = cv2.imread("E:\study5sem\lab\download (1).jpg")


# Convert the image to RGB for matplotlib display

img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)


# Create a blank mask with the same dimensions as the image

mask = np.zeros(img.shape[:2], np.uint8)


# Initialize two arrays for the background and foreground models

bgdModel = np.zeros((1, 65), np.float64)

fgdModel = np.zeros((1, 65), np.float64)


# Define a rectangle around the object you want to segment

rect = (50, 50, img.shape[1] - 100, img.shape[0] - 100)


# Apply the GrabCut algorithm

cv2.grabCut(img, mask, rect, bgdModel, fgdModel, 5, cv2.GC_INIT_WITH_RECT)
```

```
# Create a new mask to get only the foreground pixels
new_mask = np.where((mask == 2) | (mask == 0), 0, 1).astype('uint8')

# Create the segmented image by multiplying the original image with the new
mask
segmented_img = img_rgb * new_mask[:, :, np.newaxis]

# --- Display the results using plt.subplot() ---
plt.figure(figsize=(12, 6))

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

# GrabCut Segmentation
plt.subplot(1, 2, 2)
plt.imshow(segmented_img)
plt.title("GrabCut Segmentation")
plt.axis('off')

plt.tight_layout()
plt.show()
```

Sample Input and Output

Thresholding

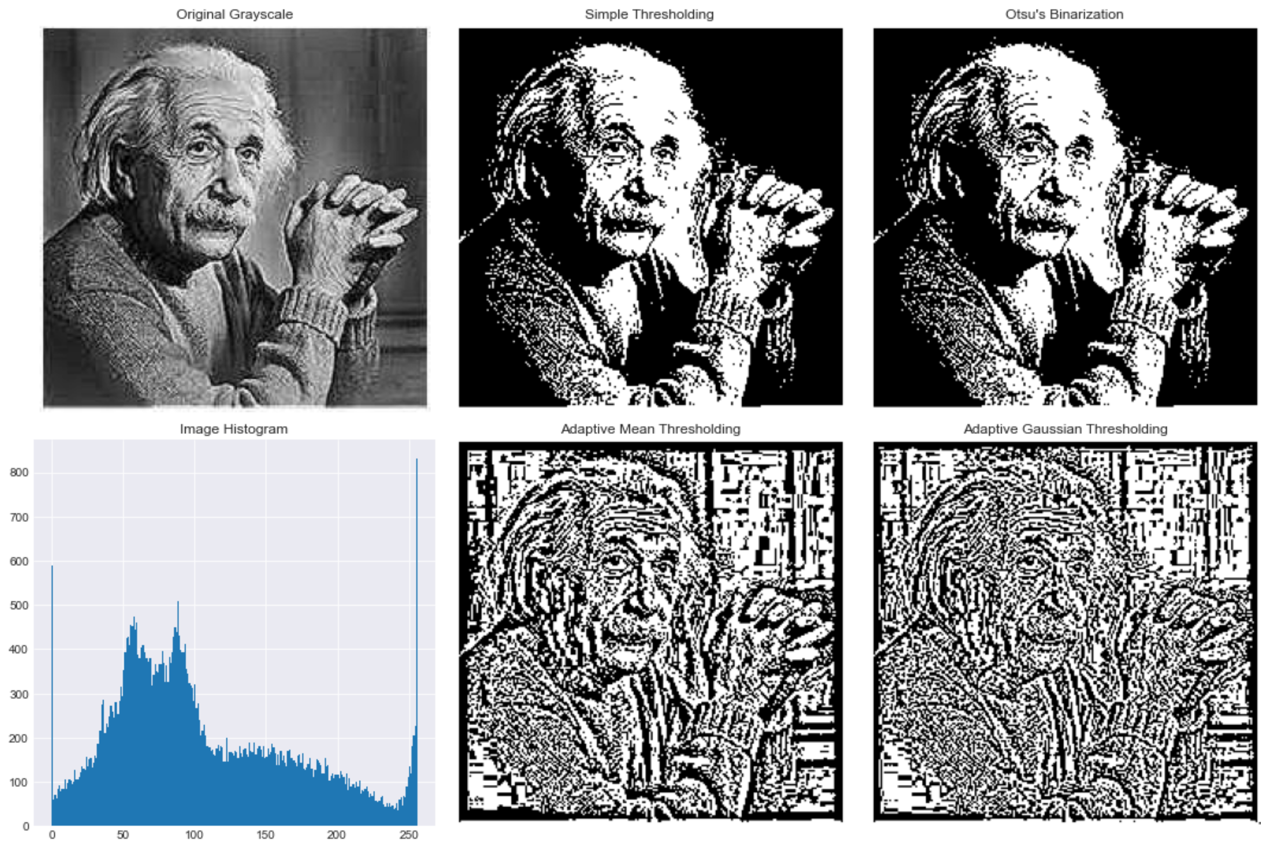
Implementation

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Load a grayscale image

# Make sure "road.jpg" exists in the same directory.

img = cv2.imread("E:\study5sem\lab\download.jpg", 0)


# --- Simple Thresholding ---

# A global threshold value is chosen manually

ret, simple_thresh = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)


# --- Adaptive Thresholding ---

# The threshold is calculated for small neighborhoods

adaptive_mean = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_MEAN_C, cv2.THRESH_BINARY, 11, 2)

adaptive_gaussian = cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 11, 2)


# --- Otsu's Binarization ---

# The threshold is automatically calculated from the image histogram
```

```
ret, otsu_thresh = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)

# --- Display the results ---
plt.figure(figsize=(15, 10))

plt.subplot(2, 3, 1)
plt.imshow(img, cmap='gray')
plt.title("Original Grayscale")
plt.axis('off')

plt.subplot(2, 3, 2)
plt.imshow(simple_thresh, cmap='gray')
plt.title("Simple Thresholding")
plt.axis('off')

plt.subplot(2, 3, 3)
plt.imshow(otsu_thresh, cmap='gray')
plt.title("Otsu's Binarization")
plt.axis('off')

plt.subplot(2, 3, 4)
plt.hist(img.ravel(), 256, [0, 256])
plt.title("Image Histogram")
```

```
plt.subplot(2, 3, 5)
plt.imshow(adaptive_mean, cmap='gray')
plt.title("Adaptive Mean Thresholding")
plt.axis('off')

plt.subplot(2, 3, 6)
plt.imshow(adaptive_gaussian, cmap='gray')
plt.title("Adaptive Gaussian Thresholding")
plt.axis('off')

plt.tight_layout()
plt.show()
```

Sample Input and Output



Ex. 7 **Morphological(opening & closing)**

Implementation

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Load image (assuming 'image.jpg' exists in the script's directory)

# The '0' argument loads the image in grayscale

img = cv2.imread("E:\study5sem\lab\jimage.png", 0)


# Create a kernel (structuring element) for morphological operations

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


# Perform morphological opening

# Opening removes small objects from the foreground (e.g., small holes or specks
of noise)

opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)


# Perform morphological closing

# Closing fills small holes in the foreground objects

closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)


# Display the images using Matplotlib

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

```
# Original Image
plt.subplot(1, 3, 1)
plt.imshow(img, cmap='gray')
plt.title('Original')
plt.axis('off')

# Opening Operation
plt.subplot(1, 3, 2)
plt.imshow(opening, cmap='gray')
plt.title('Opening')
plt.axis('off')

# Closing Operation
plt.subplot(1, 3, 3)
plt.imshow(closing, cmap='gray')
plt.title('Closing')
plt.axis('off')

plt.tight_layout()
plt.show()
```

Sample Input and Output

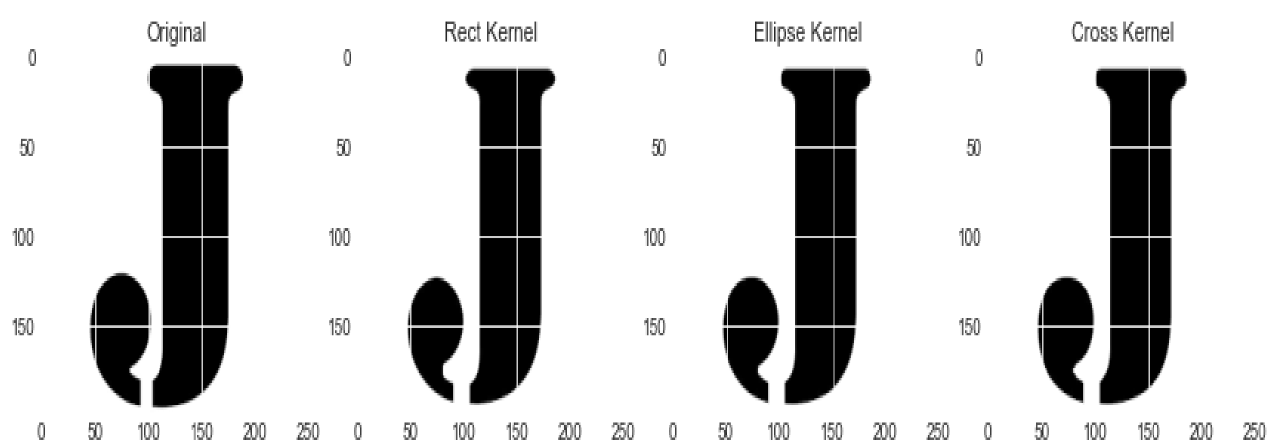
Morphologica(rectangle,ellipse,cross kernel)

Implementation

```
import cv2

import numpy as np

import matplotlib.pyplot as plt


# Load grayscale image

img = cv2.imread("E:\study5sem\lab\jimage.png", 0)


# Different filters

rect_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (5,5))

ellipse_kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))

cross_kernel = cv2.getStructuringElement(cv2.MORPH_CROSS, (5,5))
```

```
# Apply dilation with different kernels
dilate_rect = cv2.dilate(img, rect_kernel, iterations=1)
dilate_ellipse = cv2.dilate(img, ellipse_kernel, iterations=1)
dilate_cross = cv2.dilate(img, cross_kernel, iterations=1)

# Show results
plt.figure(figsize=(12,6))

plt.subplot(141), plt.imshow(img, cmap='gray'), plt.title("Original")
plt.subplot(142), plt.imshow(dilate_rect, cmap='gray'), plt.title("Rect Kernel")
plt.subplot(143), plt.imshow(dilate_ellipse, cmap='gray'), plt.title("Ellipse Kernel")
plt.subplot(144), plt.imshow(dilate_cross, cmap='gray'), plt.title("Cross Kernel")

plt.tight_layout()
plt.show()
```

Sample Input and Output



Ex. 8 **shi-tomasi corner**

Implementation

```
import numpy as np
import cv2 as cv
from matplotlib import pyplot as plt

# Read the image
img = cv.imread("E:\study5sem\lab\images (2).jpg")

# Convert to grayscale
gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

# Detect corners using Shi-Tomasi method
corners = cv.goodFeaturesToTrack(gray, maxCorners=100, qualityLevel=0.01,
minDistance=10)
corners = np.int0(corners)

# Copy image to draw detected corners
shi_img = img.copy()

# Draw circles at detected corners
for i in corners:
    x, y = i.ravel()
    cv.circle(shi_img, (x, y), 3, (0, 255, 0), -1)
```

```
# Display using matplotlib
plt.subplot(121)
plt.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
plt.title("Original Image")
plt.axis("off")

plt.subplot(122)
plt.imshow(cv.cvtColor(shi_img, cv.COLOR_BGR2RGB))
plt.title("Shi-Tomasi Corners")
plt.axis("off")

plt.show()
```

Sample Input and Output

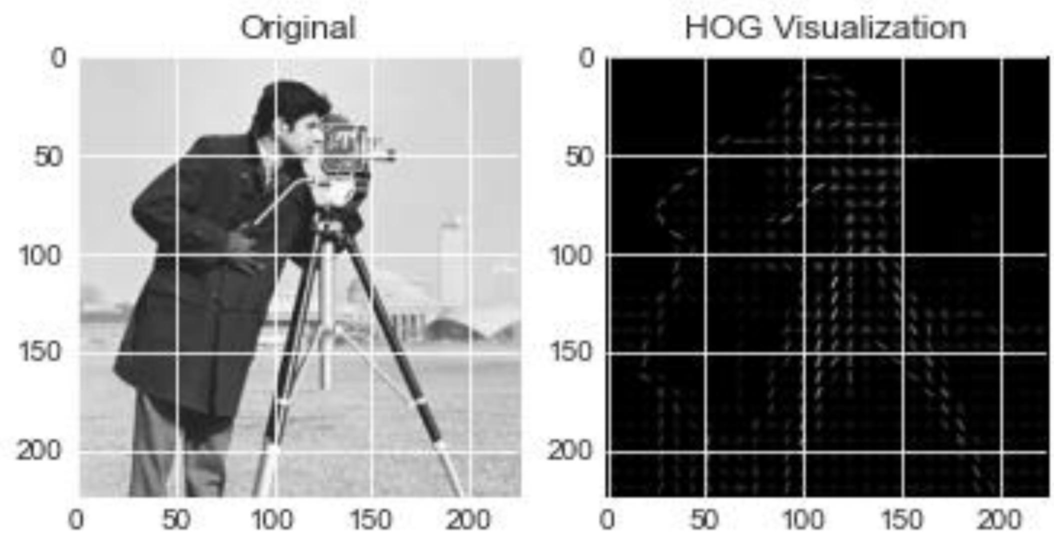
HOG

Implementation

```
import cv2

import matplotlib.pyplot as plt

from skimage.feature import hog


# Load and convert to grayscale

img = cv2.imread("E:\study5sem\lab\images (2).jpg")

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


# Compute HOG features + visualization

features, hog_image = hog(gray,

                    orientations=9,

                    pixels_per_cell=(8, 8),

                    cells_per_block=(2, 2),

                    visualize=True)


print("HOG feature vector length:", len(features))


# Show visualization

plt.subplot(121)

plt.title("Original")

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

```
plt.subplot(122)
plt.title("HOG Visualization")
plt.imshow(hog_image, cmap='gray')
plt.show()
```

Sample Input and Output



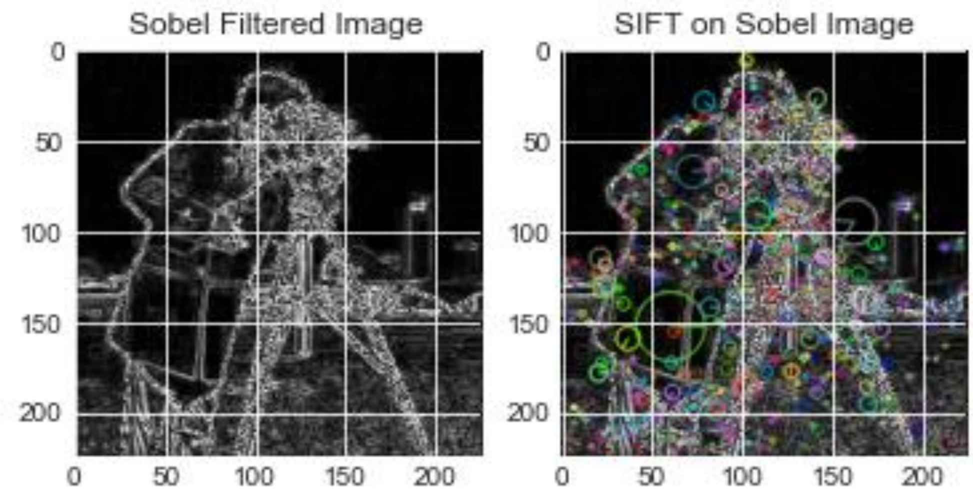
SIFT on Sobel Image

Implementation

```
import cv2 as cv

import numpy as np

from matplotlib import pyplot as plt


# Read image

img = cv.imread("E:\study5sem\lab\images (2).jpg")

gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)


# Step 1: Apply Sobel filter (edge detection)

sobelx = cv.Sobel(gray, cv.CV_64F, 1, 0, ksize=3)  # X direction

sobely = cv.Sobel(gray, cv.CV_64F, 0, 1, ksize=3)  # Y direction

sobel = cv.magnitude(sobelx, sobely)             # Combine


sobel = np.uint8(np.absolute(sobel))  # convert back to uint8


# Step 2: Apply SIFT on filtered image

sift = cv.SIFT_create()

kp = sift.detect(sobel, None)


# Step 3: Draw keypoints

output = cv.drawKeypoints(sobel, kp, None,
flags=cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
```

```
# Show results
plt.subplot(121), plt.imshow(sobel, cmap='gray'), plt.title("Sobel Filtered Image")
plt.subplot(122), plt.imshow(output), plt.title("SIFT on Sobel Image")
plt.show()
```

Sample Input and Output



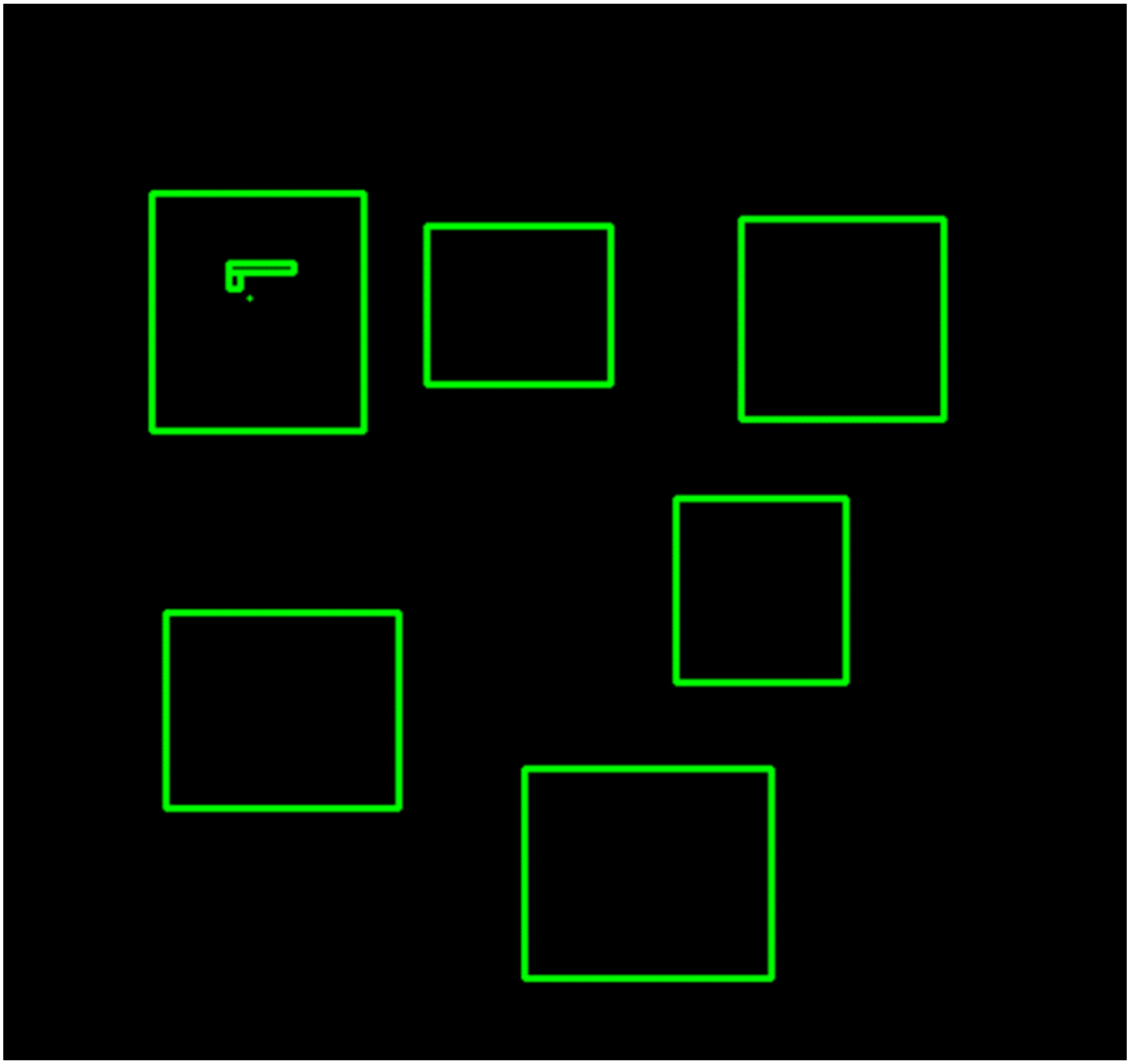
Ex. 9 **Mouse as a paint brush (rectangle)**

Implementation

```
import cv2

import numpy as np


# Initialize variables

drawing = False  # True if mouse is pressed

ix, iy = -1, -1  # Initial coordinates


# Mouse callback function

def draw_rectangle(event, x, y, flags, param):

    global ix, iy, drawing, img


    if event == cv2.EVENT_LBUTTONDOWN:

        drawing = True

        ix, iy = x, y


    elif event == cv2.EVENT_MOUSEMOVE:

        if drawing:

            img_copy = img.copy()

            cv2.rectangle(img_copy, (ix, iy), (x, y), (0, 255, 0), 2)

            cv2.imshow('image', img_copy)


    elif event == cv2.EVENT_LBUTTONUP:
```

```
        drawing = False
        cv2.rectangle(img, (ix, iy), (x, y), (0, 255, 0), 2)

# Create a black image
img = np.zeros((512, 512, 3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_rectangle)

while True:
    cv2.imshow('image', img)
    if cv2.waitKey(1) & 0xFF == 27:  # Press ESC to exit
        break

cv2.destroyAllWindows()
```

Sample Input and Output



Mouse as a paint brush (text)

Implementation

```
import cv2
import numpy as np


def draw_text(event, x, y, flags, param):
    if event == cv2.EVENT_LBUTTONDOWN:   # Left click
        font = cv2.FONT_HERSHEY_SIMPLEX
        cv2.putText(img, "Hi", (x, y), font, 1, (0, 255, 255), 2, cv2.LINE_AA)


img = np.zeros((512,512,3), np.uint8)
cv2.namedWindow('image')
cv2.setMouseCallback('image', draw_text)


while(1):
    cv2.imshow('image', img)
    if cv2.waitKey(20) & 0xFF == 27:   # ESC to exit
        break


cv2.destroyAllWindows()
```

Sample Input and Output