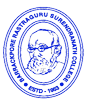
**Barrackpore Rastraguru Surendranath College**

**B.Sc. (Hons.) in Computer Science**

**Semester-VI**

****

**B.Sc. Computer Science Honours Semester-VI Examination 2023**

Name : LABANI DAS

College Roll No : 2001055

Semester : SEM-VI

WBSU Reg. No. : 1032021100062

Subject : Computer Science Hons. (CMSA)

Paper Code : CMSADSE05P

Paper Name : Digital Image Processing Assignments

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Program No.** | **Program Name** | **Page No.** |
| 01. | Read and display an image | 3 |
| 02. | Write down the dimensions of an image | 4 |
| 03. | Display the results of height and width reduction of an image to 30% of its current height and width | 5 |
| 04. | Display the color negative and grayscale negative of a color image | 6 |
| 05. | Display the results of flipping an image vertically and horizontally. Also rotate the image by 45 degrees and 90 degrees. | 8 |
| 06. | Display the results of down-sampling using ‘face’ and ‘rose’ image | 10 |
| 07. | Display the effects of quantization using ‘face’ and ‘rose’ image | 11 |
| 08. | Calculate mean value of a grayscale image. Convert the grayscale image to a binary image using the mean value as the threshold. Use ‘house’ image to show the results. | 13 |
| 09. | Display the smoothing effects of 3×3 and 5×5 mean filters using ‘man’ image. | 14 |
| 10. | Display the results of salt-and-pepper noise reduction by using median filter on ‘Circuit\_board\_noisy’ image | 16 |
| 11. | . Display the results of edge detection by Prewitt operator, using ‘house’ image | 18 |
| 12. | . Display the histograms of low-contrast, medium-constrast, and high-constrast images | 20 |

**WRITE A PYTHON PROGRAM TO :**

**PROGRAM NUMBER: 1**

Read and display an image.

**Program code:**

# importing cv2

import cv2

# Using cv2.imread() method

# Using 0 to read image in grayscale mode

img = cv2.imread("images/Catching.JPG", 0)

# Displaying the image

cv2.imshow('image', img)

# Using 1 to read image in color mode

img = cv2.imread("images/Catching.jpg", 1)

# Displaying the image

cv2.imshow('image1', img)

# Wait for the user to press a key

cv2.waitKey(0)

# Close all windows

cv2.destroyAllWindows()

**Program output:**

****

*color mode grayscale mode*

**PROGRAM NUMBER: 2**

Write down the dimensions of an image

**Program code:**

#Display Dimensions of image

# importing cv2

import cv2

img = cv2.imread("images/Catching.jpg")

# get dimensions of image

dimensions = img.shape

# height, width, number of channels in image

height = img.shape[0]

width = img.shape[1]

channels = img.shape[2]

print('Image Dimension    : ',dimensions)

print('Image Height       : ',height)

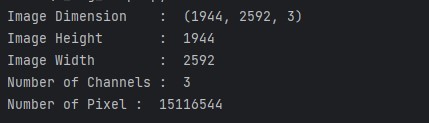
print('Image Width        : ',width)

print('Number of Channels : ',channels)

print('Number of Pixel : ',img.size)

**Program output:**

****

****

*Image dimensions*

**PROGRAM NUMBER:3**

Display the results of height and width reduction of an image to 30% of its current height and width

**Program code:**

# importing cv2

import cv2

img = cv2.imread("images/Catching.jpg")

print('Original Dimensions : ',img.shape)

scale\_percent = 30 # percent of original size

width = int(img.shape[1] \* scale\_percent / 100)

height = int(img.shape[0] \* scale\_percent / 100)

dim = (width, height)

# resize image

resized = cv2.resize(img, dim, interpolation = cv2.INTER\_AREA)

print('Resized Dimensions : ',resized.shape)

cv2.imshow("Resized image", resized)

# Wait for the user to press a key

cv2.waitKey(0)

# Close all windows

cv2.destroyAllWindows()

**Program output:**

****

****

*Original size reduced size*

**PROGRAM NUMBER:4**

Display the color negative and grayscale negative of a color image.

**Program code:**

# importing cv2

import cv2

import matplotlib.pyplot as plt

img = cv2.imread("images/Catching.jpg")

gray= cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

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

colored\_negative = abs(255-img)

gray\_negative = abs(255-gray)

imgs = [img, gray, colored\_negative, gray\_negative]

title = ['coloured', 'gray', 'coloured-negative', 'gray-negative']

plt.subplot(2, 2, 1)

plt.title(title[0])

plt.imshow(imgs[0])

plt.xticks([])

plt.yticks([])

plt.subplot(2, 2, 2)

plt.title(title[1])

plt.imshow(imgs[1], cmap='gray')

plt.xticks([])

plt.yticks([])

plt.subplot(2, 2, 3)

plt.title(title[2])

plt.imshow(imgs[2])

plt.xticks([])

plt.yticks([])

plt.subplot(2, 2, 4)

plt.title(title[3])

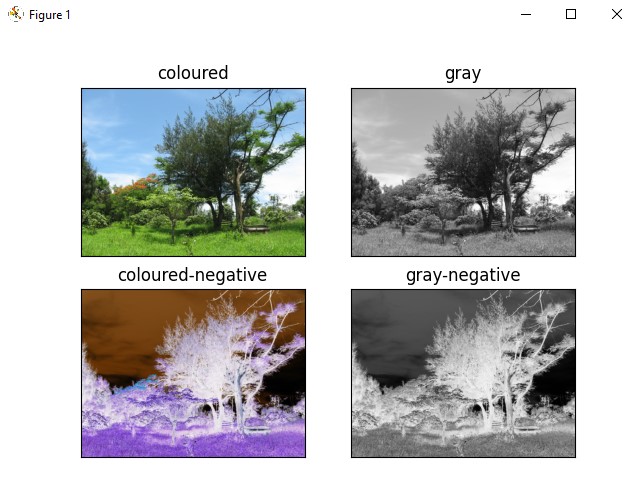
plt.imshow(imgs[3], cmap='gray')

plt.xticks([])

plt.yticks([])

plt.show()

**Program output:**



**PROGRAM NUMBER:5**

Display the results of flipping an image vertically and horizontally. Also rotate the image by 45 degrees and 90 degrees.

**Program code:**

# importing cv2

import cv2

import matplotlib.pyplot as plt

from PIL import Image

img = Image.open("images/Catching.jpg")

hori\_flippedImage = img.transpose(Image.FLIP\_LEFT\_RIGHT)

Vert\_flippedImage = img.transpose(Image.FLIP\_TOP\_BOTTOM)

degree\_flippedImage = img.rotate(45)

degree90\_flippedImage = img.transpose(Image.ROTATE\_90)

imgs = [img, hori\_flippedImage, Vert\_flippedImage, degree\_flippedImage, degree90\_flippedImage]

title = ['Original', 'horizontally\_flipped', 'Vertically\_flipped', 'specifying degree', '90degree\_flipped']

plt.subplot(2, 3, 1)

plt.title(title[0])

plt.imshow(imgs[0])

plt.xticks([])

plt.yticks([])

plt.subplot(2, 3, 2)

plt.title(title[1])

plt.imshow(imgs[1])

plt.xticks([])

plt.yticks([])

plt.subplot(2, 3, 3)

plt.title(title[2])

plt.imshow(imgs[2])

plt.xticks([])

plt.yticks([])

plt.subplot(2, 3, 4)

plt.title(title[3])

plt.imshow(imgs[3])

plt.xticks([])

plt.yticks([])

plt.subplot(2, 3, 5)

plt.title(title[4])

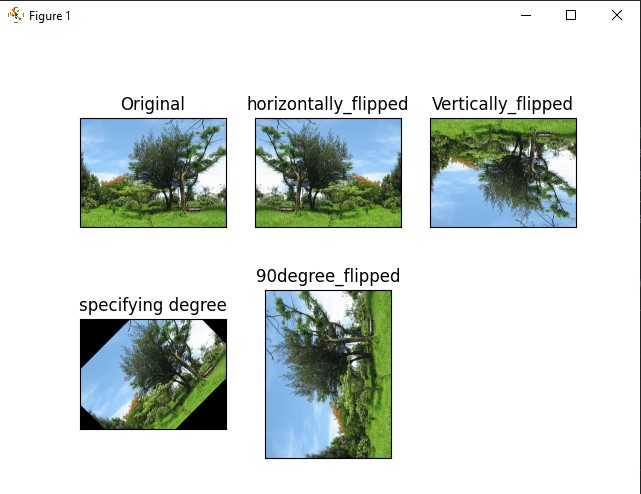
plt.imshow(imgs[4])

plt.xticks([])

plt.yticks([])

plt.show()

**Program output:**

****

**PROGRAM NUMBER:6**

Display the results of down-sampling using ‘face’ and ‘rose’ image

**Program code:**

from PIL import Image

def downsample\_image(image\_path, output\_path, factor):

    # Open the image

    image = Image.open(image\_path)

    # Calculate the new width and height based on the downsampling factor

    width = image.width // factor

    height = image.height // factor

    # Downsample the image using the 'ANTIALIAS' filter for better quality

    downscaled\_image = image.resize((width, height), Image.ANTIALIAS)

    downscaled\_image.show()

def show\_downsampled\_image(file\_name):

    input\_image\_path = f"Images/{file\_name}"

    output\_image\_path = f"6/{file\_name.split('.')[0]}\_downsampled.jpg"

    downsampling\_factor = 2

    #  downsample and save image

    downsample\_image(input\_image\_path, output\_image\_path, downsampling\_factor)

# face image

show\_downsampled\_image("face.jpg")

# rose image

show\_downsampled\_image("rose-512.bmp")

**Program output:**



****

**



****



**PROGRAM NUMBER:7**

Display the effects of quantization using ‘face’ and ‘rose’ image.

**Program code:**

from PIL import Image

def quantize\_image(image\_path, output\_path, num\_colors):

    # Open the image

    image = Image.open(image\_path)

    # Apply quantization by reducing the number of colors

    quantized\_image = image.quantize(num\_colors)

    #show image

    quantized\_image.show()

def showQuantizedimage(file\_name):

    input\_image\_path = f"images/{file\_name}"

    output\_image\_path = f"7/{file\_name.split('.')[0]}\_quantized.jpg"

    num\_colors = 20

    quantize\_image(input\_image\_path, output\_image\_path, num\_colors)

# face image

showQuantizedimage("face.jpg")

# rose image

showQuantizedimage("rose-512.bmp")

**Program output:**

****

*Original image quantized image*

**

****

*Original image quantized image*

**PROGRAM NUMBER:8**

Calculate mean value of a grayscale image. Convert the grayscale image to a binary image using the mean value as the threshold. Use ‘house’ image to show the results.

**Program code:**

import numpy as np

from PIL import Image

# Open the image and convert it to grayscale

image = Image.open('images/house.jpg')

# Convert the image to a NumPy array

image\_array = np.array(image)

# Calculate the mean value

mean\_value = np.mean(image\_array)

# show mean value

print("Mean value:", mean\_value)

# Create a binary image by applying the threshold

# if image\_array >= mean\_value TRUE value assigned to that pixel is 255

# if image\_array >= mean\_value FALSE value assigned to that pixel is 0

binary\_image = np.where(image\_array >= mean\_value, 255, 0).astype(np.uint8)

# Convert the NumPy array back to an image

binary\_image = Image.fromarray(binary\_image)

# show the binary image

binary\_image.show()

**Program output:**

****

*Original image binary image*

**PROGRAM NUMBER:9**

Display the smoothing effects of 3×3 and 5×5 mean filters using ‘man’ image.

**Program code:**

import cv2

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

def apply\_mean\_filter(image\_path, kernel\_size, title):

    # Open the image using Pillow

    image = Image.open(image\_path)

    # Convert the image to a NumPy array

    image\_array = np.array(image)

    # Apply the mean filter

    filtered\_image\_array = cv2.blur(image\_array, (kernel\_size, kernel\_size))

    # convert image array to image

    filtered\_image = Image.fromarray(filtered\_image\_array)

    # Display the image with the specified title

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

    plt.title(title)

    plt.axis('off')

    plt.show()

# man image path

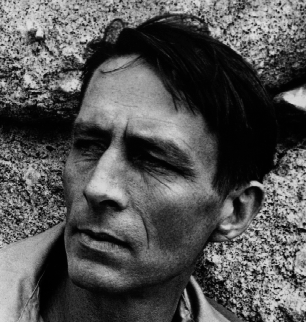
input\_image\_path = 'images/man.gif'

kernel\_size\_3x3 = 3

kernel\_size\_5x5 = 5

apply\_mean\_filter(input\_image\_path, kernel\_size\_3x3, "smoothning using 3x3 mean filter")

apply\_mean\_filter(input\_image\_path, kernel\_size\_5x5,"smoothning using 5x5 mean filter ")

**Program output:**

*original image*

****

*blurred imahs using 3x3 smoothning mean filter*

****

*blurred imahs using 5x5 smoothning mean filter*

**PROGRAM NUMBER:10**

Display the results of salt-and-pepper noise reduction by using median filter on ‘Circuit\_board\_noisy’ image.

**Program code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def reduce\_salt\_and\_pepper\_noise(image\_path,  kernel\_size):

    # Load the image

    image = cv2.imread(image\_path)

    # Add salt-and-pepper noise to the image

    noisy\_image = add\_salt\_and\_pepper\_noise(image)

    # Apply median filter for noise reduction

    # The Median blur operation is similar to the other averaging methods.

    # Here, the central element of the image is replaced by the median of all the pixels in the kernel area.

    denoised\_image = cv2.medianBlur(noisy\_image, kernel\_size)

    # Display the original, noisy, and denoised images

    titles = ['Original Image', 'Noisy Image', 'Denoised Image']

    images = [image, noisy\_image, denoised\_image]

    plt.figure(figsize=(10, 5))

    for i in range(3):

        plt.subplot(1, 3, i + 1)

        plt.imshow(cv2.cvtColor(images[i], cv2.COLOR\_BGR2RGB))

        plt.title(titles[i])

        plt.axis('off')

    plt.tight\_layout()

    plt.show()

def add\_salt\_and\_pepper\_noise(image, noise\_prob=0.05):

    # Generate a mask for salt-and-pepper noise

    mask = np.random.choice([0, 1, 2], size=image.shape[:2], p=[1 - noise\_prob, noise\_prob / 2, noise\_prob / 2])

    # Add salt-and-pepper noise to the image

    noisy\_image = np.copy(image)

    noisy\_image[mask == 1] = 255  # Salt noise

    noisy\_image[mask == 2] = 0  # Pepper noise

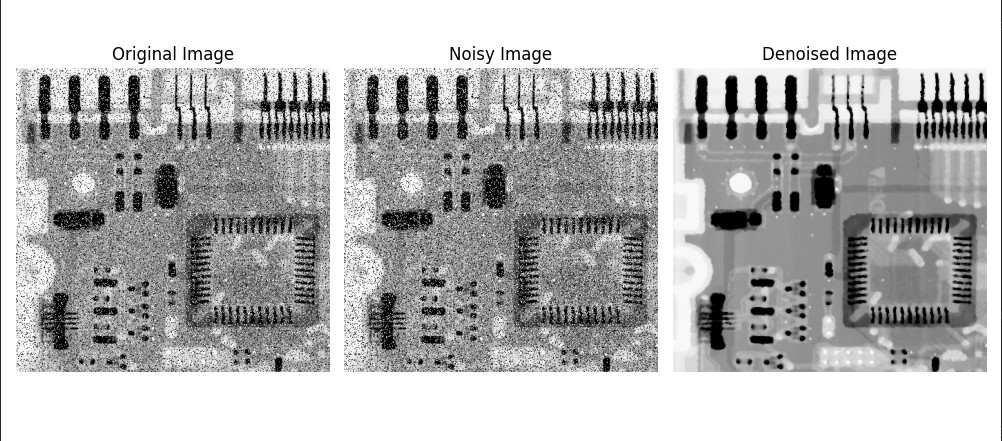
    return noisy\_image

# image path

input\_image\_path = 'images/Circuit\_board\_noisy.tif'

kernel\_size = 5

reduce\_salt\_and\_pepper\_noise(input\_image\_path,  kernel\_size)

**Program output:**

**PROGRAM NUMBER:11**

Display the results of edge detection by Prewitt operator, using ‘house’ image.

**Program code:**

import cv2

import numpy as np

import matplotlib.pyplot as plt

def apply\_prewitt\_operator(image\_path):

    # Load the image

    image = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

    # Apply the Prewitt operator

    gradient\_x = cv2.Sobel(image, cv2.CV\_64F, 1, 0, ksize=3)

    gradient\_y = cv2.Sobel(image, cv2.CV\_64F, 0, 1, ksize=3)

    # Combine the gradients to obtain the edge magnitude

    gradient\_magnitude = np.sqrt(np.square(gradient\_x) + np.square(gradient\_y))

    # Normalize the gradient magnitude to the range [0, 255]

    gradient\_magnitude\_normalized = cv2.normalize(gradient\_magnitude, None, 0, 255, cv2.NORM\_MINMAX, dtype=cv2.CV\_8U)

    # Display the original and edge-detected images

    titles = ['Original Image', 'Prewitt Edge Detection']

    images = [image, gradient\_magnitude\_normalized]

    plt.figure(figsize=(10, 5))

    for i in range(2):

        plt.subplot(1, 2, i + 1)

        plt.imshow(images[i], cmap='gray')

        plt.title(titles[i])

        plt.axis('off')

    plt.tight\_layout()

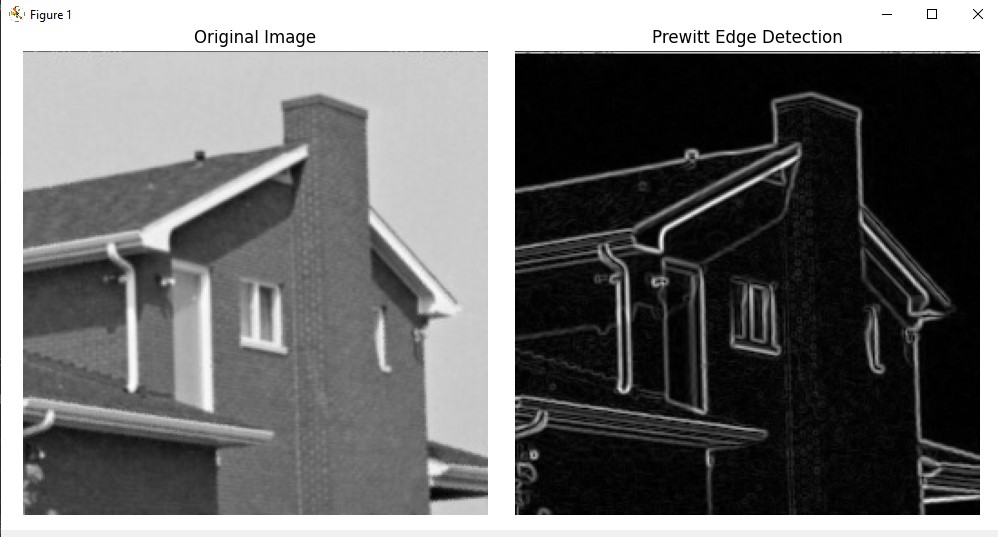
    plt.show()

# image path

input\_image\_path = 'images/house.jpg'

apply\_prewitt\_operator(input\_image\_path)

**Program output:**



**PROGRAM NUMBER:12**

Display the histograms of low-contrast, medium-constrast, and high-constrast images

**Program code:**

import cv2

import matplotlib.pyplot as plt

def display\_histogram(image\_path):

    # Load the image

    image = cv2.imread(image\_path, 0)

    # Calculate the histogram

    histogram = cv2.calcHist([image], [0], None, [256], [0, 256])

    # Display the histogram

    plt.figure(figsize=(8, 5))

    plt.plot(histogram, color='black')

    plt.title('Histogram')

    plt.xlabel('Pixel Value')

    plt.ylabel('Frequency')

    plt.xlim([0, 256])

    plt.show()

# Example usage

low\_contrast\_image\_path = 'images/Einstein low contrast.tif'

medium\_contrast\_image\_path = 'images/Einstein med contrast.tif'

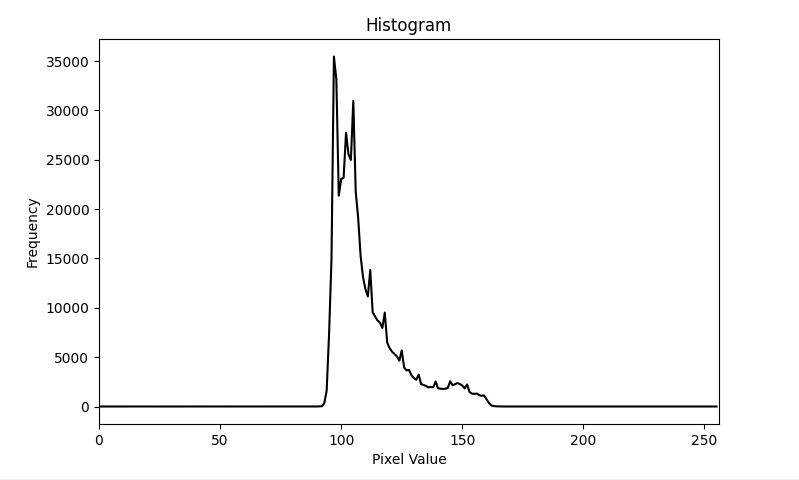
high\_contrast\_image\_path = 'images/Einstein high contrast.tif'

display\_histogram(low\_contrast\_image\_path)

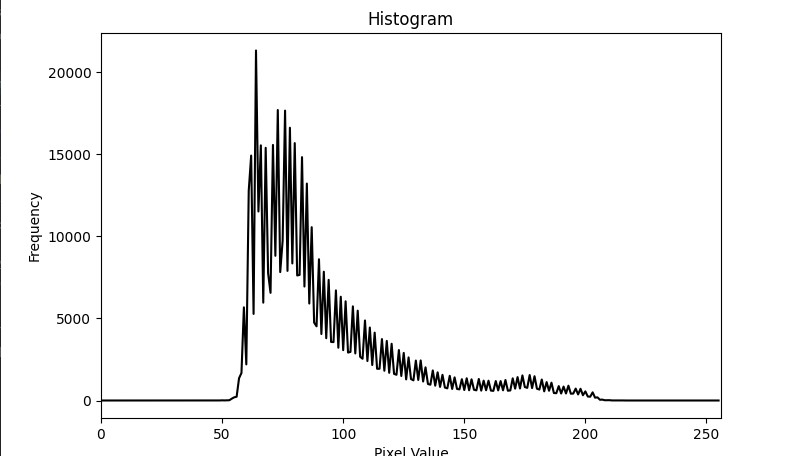
display\_histogram(medium\_contrast\_image\_path)

display\_histogram(high\_contrast\_image\_path)

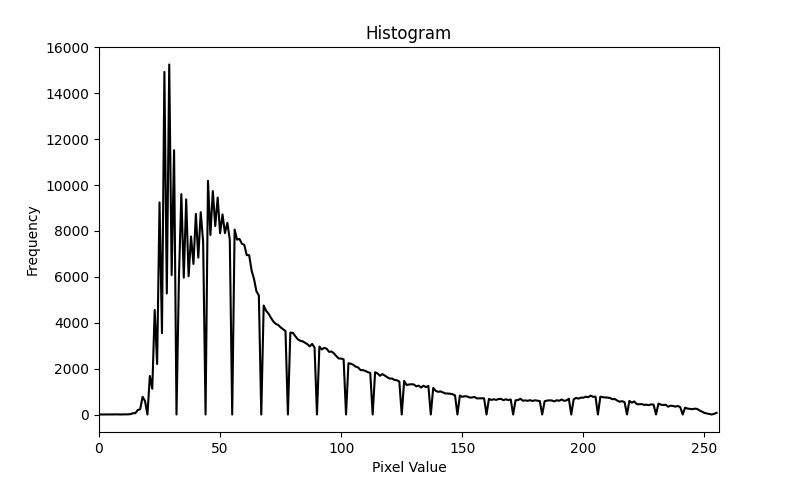
**Program output:**

**Einstein low contrast.tif**

*Low contrast image histogram*

**

*Einstein med contrast.tifmedium contrast image histogam*

**Einstein high contrast.tif***high contrast image histogam*