Ex No:1 Develop application to display grayscale image using read and write operation

Date : . .2023

Aim :

To develop application to display grayscale image using read and write operation.

1.Read,Write and Save Image

Algorithm:

Step1:Upload the image from your local machine.

Step2:Get the filename of the uploaded image.

Step3:Read the uploaded image using OpenCV.

Step4:Display the image using OpenCV's cv2_imshow.

Step5:Specify the filename for the saved image.

Step6:Provide a download link for the saved image.

Code:

```
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow  # For displaying images in Colab
from google.colab import files  # For uploading and downloading files
uploaded = files.upload()
if len(uploaded) == 0:
    print("No files were uploaded.")
else:
    image_filename = list(uploaded.keys())[0]
    image = cv2.imread(image_filename)
    if image is None:
        print('Error: Could not open or read your image.')
    else:
        output_filename = 'output_image.jpg'
        cv2.imwrite(output_filename, image)
```

```
    height, width, _ = image.shape
    aspect_ratio = width / height
    plt.figure(figsize=(10 * aspect_ratio, 10))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title('Original Image')
    plt.axis('off')  # Turn off axis labels
    plt.show()
    saved_image = cv2.imread(output_filename)
    if saved_image is not None:
        print('Your image has been saved successfully as "output_image.jpg".')
    else:
        print('Error: Could not save your image.')
```

2.Convert to Gray Scale

Algorithm:

Step1:Upload the image from your local machine.

Step2:Get the filename of the uploaded image.

Step3:Read the uploaded image using OpenCV.

Step4: Convert the image to grayscale

Step5: Display the grayscale image

Code:

```
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow  # For displaying images in Colab
from google.colab import files  # For uploading and downloading files
uploaded = files.upload()
if len(uploaded) == 0:
    print("No files were uploaded.")
else:
    image_filename = list(uploaded.keys())[0]
```

```
    original_image = cv2.imread(image_filename)
    if original_image is None:
        print('Error: Could not open or read your image.')
    else:
        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()
```

3.Image Enhancement contrast bright

Algorithm:

Step1: Upload the image from your local machine.

Step2: Get the filename of the uploaded image.

Step3:Read the uploaded image using OpenCV.

Step4:Define contrast and brightness values (adjust these as needed).

Step5: Apply contrast and brightness adjustments.

Step6: Display the original and adjusted images

Code:

```
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow  # For displaying images in Colab
from google.colab import files  # For uploading and downloading files
uploaded = files.upload()
if len(uploaded) == 0:
    print("No files were uploaded.")
```

```
else:
    image_filename = list(uploaded.keys())[0]
    original_image = cv2.imread(image_filename)
    if original_image is None:
        print('Error: Could not open or read your image.')
    else:
        brightness_factor = 2.5
        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()
```

4.Image Resizing

Algorithm:

Step1: Upload the image from your local machine.

Step2: Get the filename of the uploaded image.

Step3:Read the uploaded image using OpenCV.

Step4:Define contrast and brightness values (adjust these as needed).

Step5: Image Resizing by the given input.

Step6: Display the original and adjusted images

Code:

```
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow  # For displaying images in Colab
from google.colab import files  # For uploading and downloading files
uploaded = files.upload()
if len(uploaded) == 0:
    print("No files were uploaded.")
else:
    image_filename = list(uploaded.keys())[0]
    original_image = cv2.imread(image_filename)
    if original_image is None:
        print('Error: Could not open or read your image.')
    else:
        # Prompt the user for the desired width and height
        new_width = int(input("Enter the desired width for resizing: "))
        new_height = int(input("Enter the desired height for resizing: "))
        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()
```

5.Image Negative

Algorithm:

Step1: Upload the image from your local machine.

Step2: Get the filename of the uploaded image.

Step3:Read the uploaded image using OpenCV.

Step4: Create the negative of the image

Step5: Display the negative image using OpenCV's cv2_imshow

Code:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

from google.colab.patches import cv2_imshow  # For displaying images in Colab

from google.colab import files  # For uploading and downloading files

uploaded = files.upload()

if len(uploaded) == 0:

   print("No files were uploaded.")

else:

   image_filename = list(uploaded.keys())[0]

   original_image = cv2.imread(image_filename)

   if original_image is None:

      print('Error: Could not open or read your image.')

   else:

      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()
```

6.1.Histogram Equalization

Algorithm:

Step1: Upload the image from your local machine.

Step2: Get the filename of the uploaded image.

Step3:Read the uploaded image using OpenCV.

Step4:Display the original image using OpenCV's cv2_imshow

Step5: Display the equalized image using OpenCV's cv2_imshow

Code:

```
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow  # For displaying images in Colab
from google.colab import files  # For uploading and downloading files
uploaded = files.upload()
if len(uploaded) == 0:
  print("No files were uploaded.")
else:
  image_filename = list(uploaded.keys())[0]
  img_bgr = cv2.imread(image_filename, 1)
  if img_bgr is None:
    print('Error: Could not open or read your image.')
  else:
    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()
```

6.2Histogram for Original image and Equalized image(Gray Scale)

Algorithm:

Step1: Upload the image from your local machine.

Step2: Get the filename of the uploaded image.

Step3:Read the uploaded image using OpenCV.

Step4:Display the original image using OpenCV's cv2_imshow

Step5: Display the equalized image using OpenCV's cv2_imshow

Code:

```
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow  # For displaying images in Colab
from google.colab import files  # For uploading and downloading files
uploaded = files.upload()
if len(uploaded) == 0:
    print("No files were uploaded.")
else:
    image_filename = list(uploaded.keys())[0]
    image = cv2.imread(image_filename, cv2.IMREAD_GRAYSCALE)
    if image is None:
        print('Error: Could not open or read your image.')
    else:
        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()
```

7.Addition and Subtraction of Image

Algorithm:

Step1: Upload the image from your local machine.

Step2: Get the filename of the uploaded image.

Step3: Upload the second image from your local machine.

Step4: Add the two images.

Step5: Subtract the second image from the first image.

Step6: Display the equalized image using OpenCV's cv2_imshow.

Code:

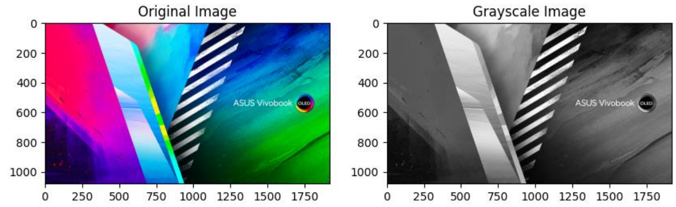
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow  # For displaying images in Colab
from google.colab import files  # For uploading and downloading files
uploaded1 = files.upload()
if len(uploaded1) == 0:
   print("No files were uploaded for Image 1.")
else:
   image_filename1 = list(uploaded1.keys())[0]
```

```
image1 = cv2.imread(image_filename1)
uploaded2 = files.upload()
if len(uploaded2) == 0:
    print("No files were uploaded for Image 2.")
else:
    image_filename2 = list(uploaded2.keys())[0]
    image2 = cv2.imread(image_filename2)
    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()
```

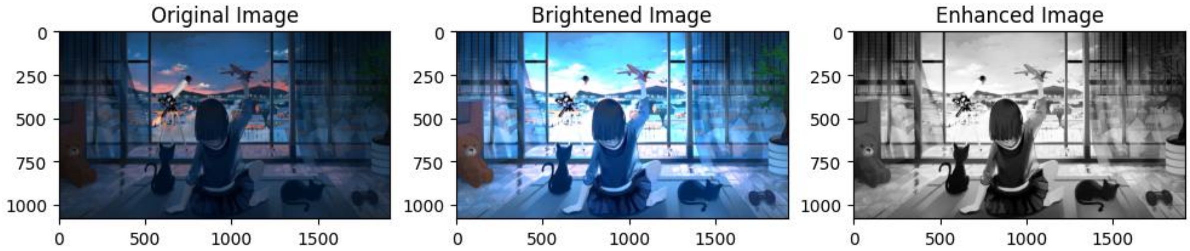
Output:

1.Read,Write and Save Image

2.Convert to Gray Scale

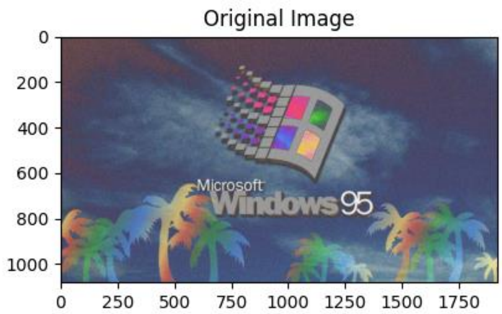


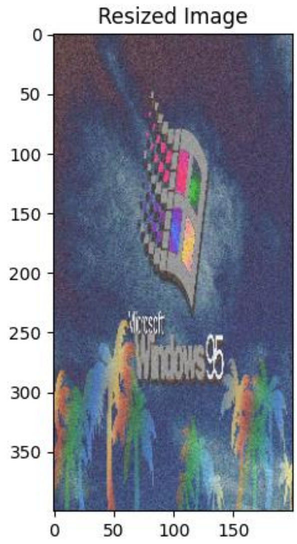
3.Image Enhancement contrast bright



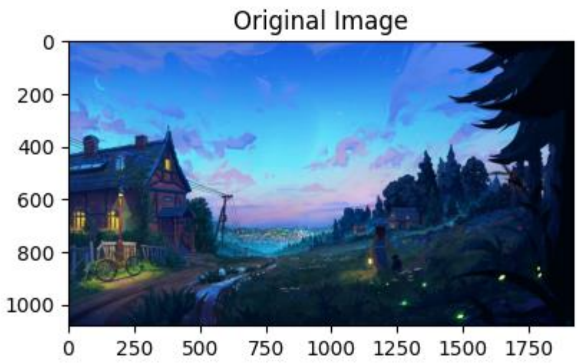
4.Image Resizing

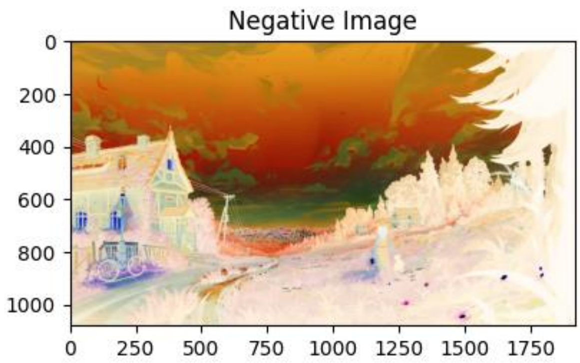
Enter the desired width: 200 Enter the desired height: 400



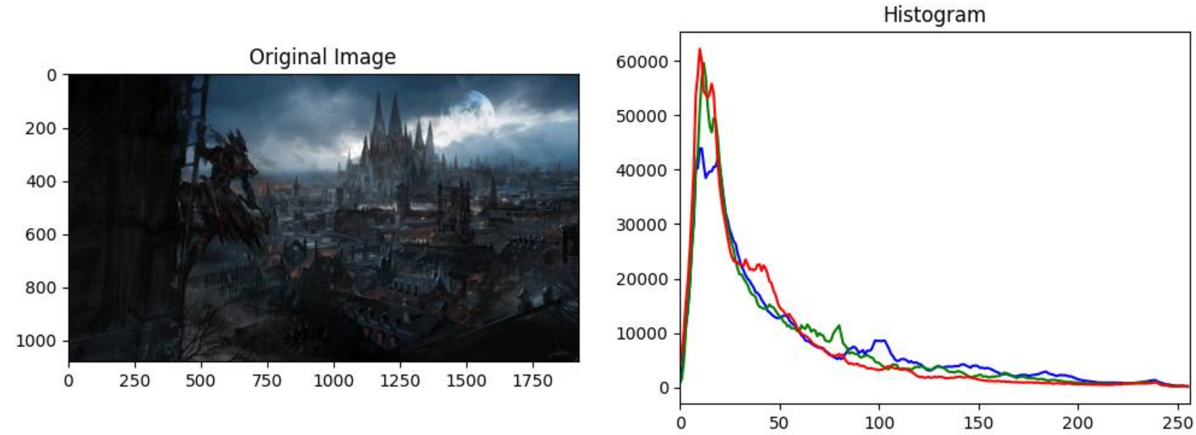


5.Image Negative

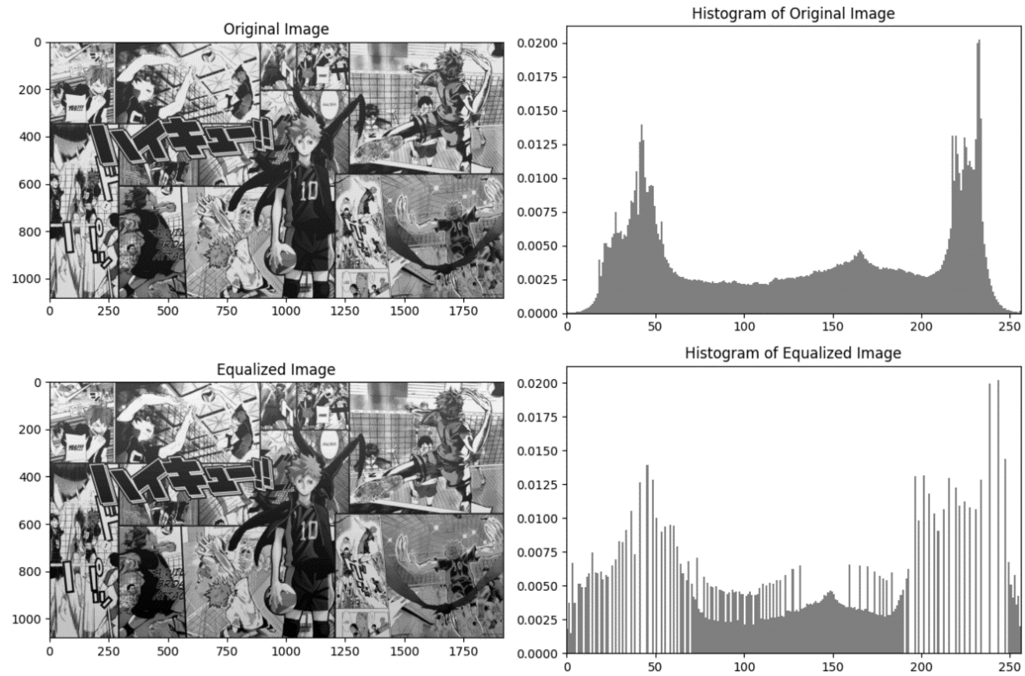




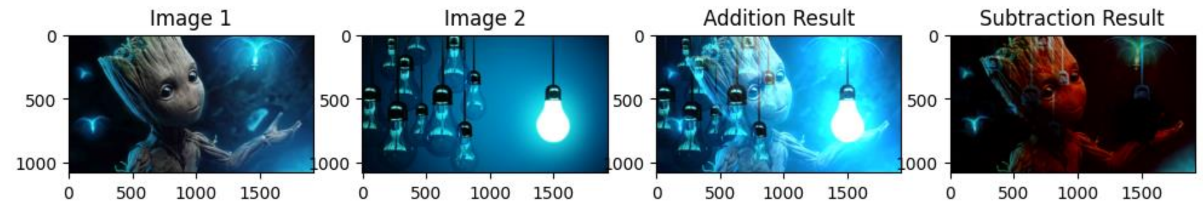
6.1.Histogram Equalization



6.2. Histogram for Original image and Equalized image(Gray Scale)



7.Addition and Subtraction of Image



Result:

The develop application to display grayscale image using read and write operation have been created successfully.

Ex No:2 **Adding and Removal Of Noise**

Date : . .2023

Aim :

To develop the code for adding the removal of noise.

Algorithm:

Step 1: Image Upload

- Use the files.upload() function to upload an image from the user's computer in a Jupyter Notebook or Google Colab environment.

Step 2: Image Preprocessing

- Read the uploaded image in color using OpenCV.
- Convert the color image to grayscale using cv2.cvtColor().

Step 3: Noise Addition

- Generate and add Gaussian noise to the grayscale image.
- Generate and add Impulse noise to the grayscale image.
- Generate and add Uniform noise to the grayscale image.

Step 4: Visualization

- Create subplots using Matplotlib for displaying images.
- Display the original color image, the images with different types of noise (Gaussian, Impulse, Uniform), and the combined images.

Step 5: Filter Application

- Apply Median filters to the images with noise.
- Apply an Average filter to one of the noisy images.
- Display the filtered images for comparison.

Code:

```
import numpy as np

import matplotlib.pyplot as plt

import cv2

from google.colab import files  # Import the 'files' module for Colab

uploaded = files.upload()
```

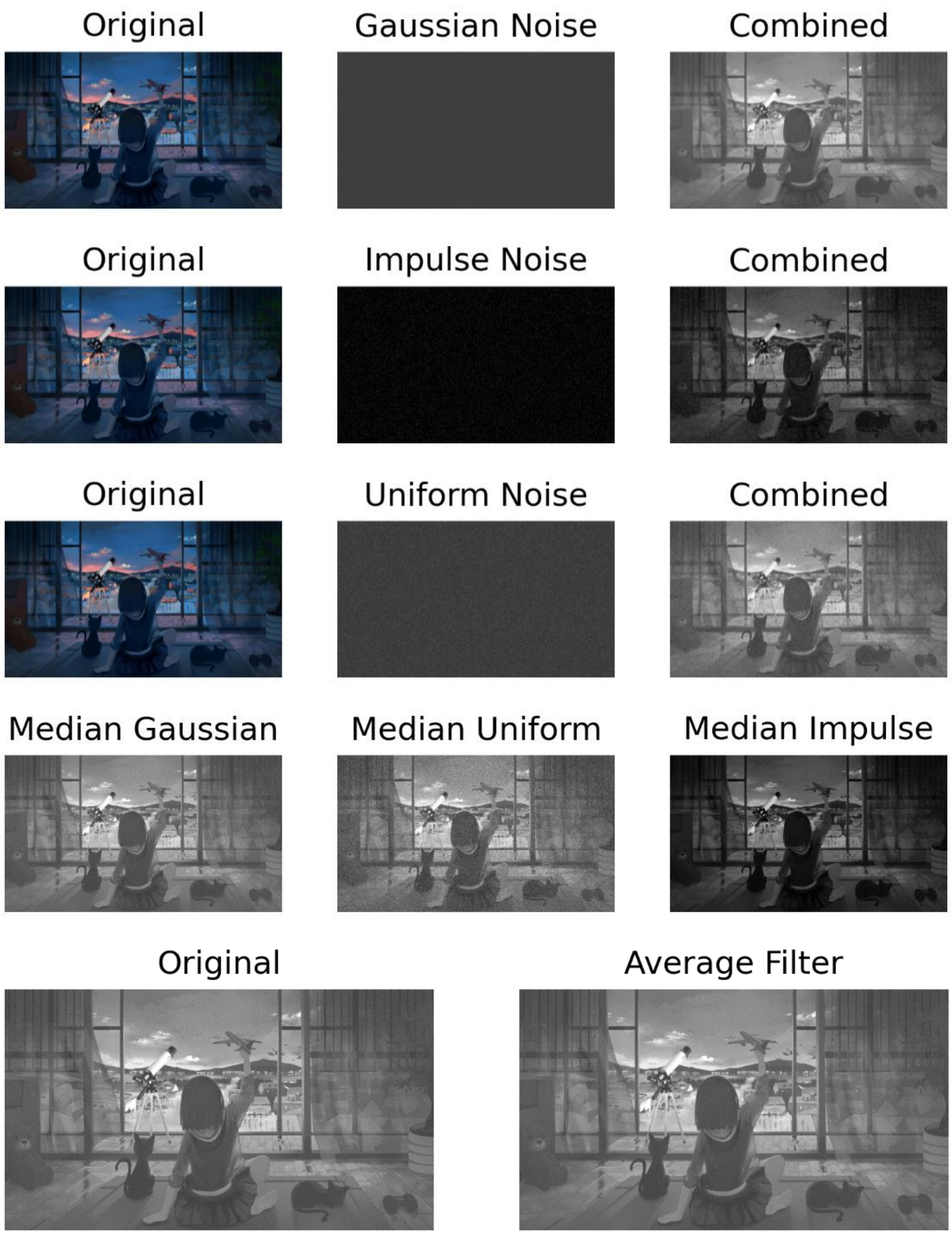
```
image_filename = list(uploaded.keys())[0]
img_color = cv2.imread(image_filename)
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
gauss_noise = np.zeros_like(img_gray, dtype=np.uint8)
cv2.randn(gauss_noise, 128, 20)
gauss_noise = (gauss_noise * 0.5).astype(np.uint8)
gn_img = cv2.add(img_gray, gauss_noise)
gauss_noise_color = cv2.cvtColor(gauss_noise, cv2.COLOR_GRAY2BGR)
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()
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
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)
imp_noise_color = cv2.cvtColor(imp_noise, cv2.COLOR_GRAY2BGR)
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()
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
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)
uni_noise_color = cv2.cvtColor(uni_noise, cv2.COLOR_GRAY2BGR)
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()
img_gray = cv2.cvtColor(img_color, cv2.COLOR_BGR2GRAY)
blurred1 = cv2.medianBlur(gn_img, 3)
blurred2 = cv2.medianBlur(un_img, 3)
blurred3 = cv2.medianBlur(in_img, 3)
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()
img_new = cv2.blur(gn_img, (3, 3))
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:

The develop the code for adding the removal of noise has been created successfully.

Ex No:3 **Edge Detection**
Date : . .2023

Aim :

To develop Edge Detection algorithm.
I) Canny edge detection
II) Sobel edge detection
III) Prewitt edge detection
Algorithm:

Step 1: Image Upload

- Use the files.upload() function to upload an image from the user's computer in a Jupyter Notebook or Google Colab environment.

Step 2: Image Processing

- Read the uploaded image in color using OpenCV.
- Convert the color image to grayscale using cv2.cvtColor().

Step 3: Edge Detection

- Apply Gaussian blur to the grayscale image using cv2.GaussianBlur().
- Apply Canny edge detection to the original color image using cv2.Canny().
- Apply Sobel edge detection to the blurred grayscale image in both the x and y directions using cv2.Sobel().
- Apply Prewitt edge detection using custom convolution kernels.

Step 4: Visualization

- Create subplots using Matplotlib for displaying images.
- Display the original color image and the results of different edge detection methods (Canny, Sobel, Prewitt).

Step 5: Display and User Interaction

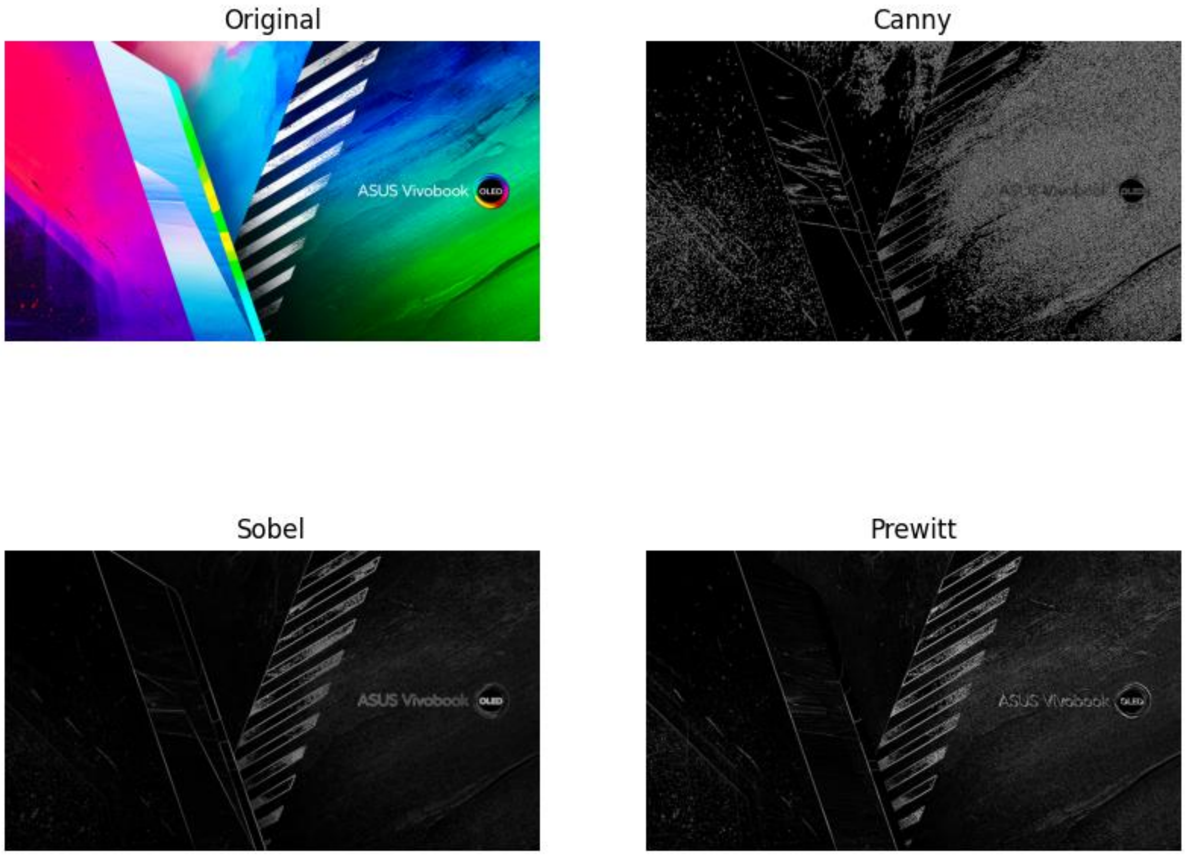
- Display the images with appropriate titles and labels.
- Show the plots using plt.show().

Code:

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

from google.colab import files
```

```
uploaded = files.upload()
image_filename = list(uploaded.keys())[0]
img = cv2.imread(image_filename)
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_gaussian = cv2.GaussianBlur(gray, (3, 3), 0)
img_canny = cv2.Canny(img, 100, 200)
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)
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
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:

The Edge Detection operation have been created successfully.

Ex No.: 4 **Perform morphological operations on an image**

Date : . .2023

Aim

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

i. Dilation
ii. Erosion
iii. Closing
iv. Opening

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:

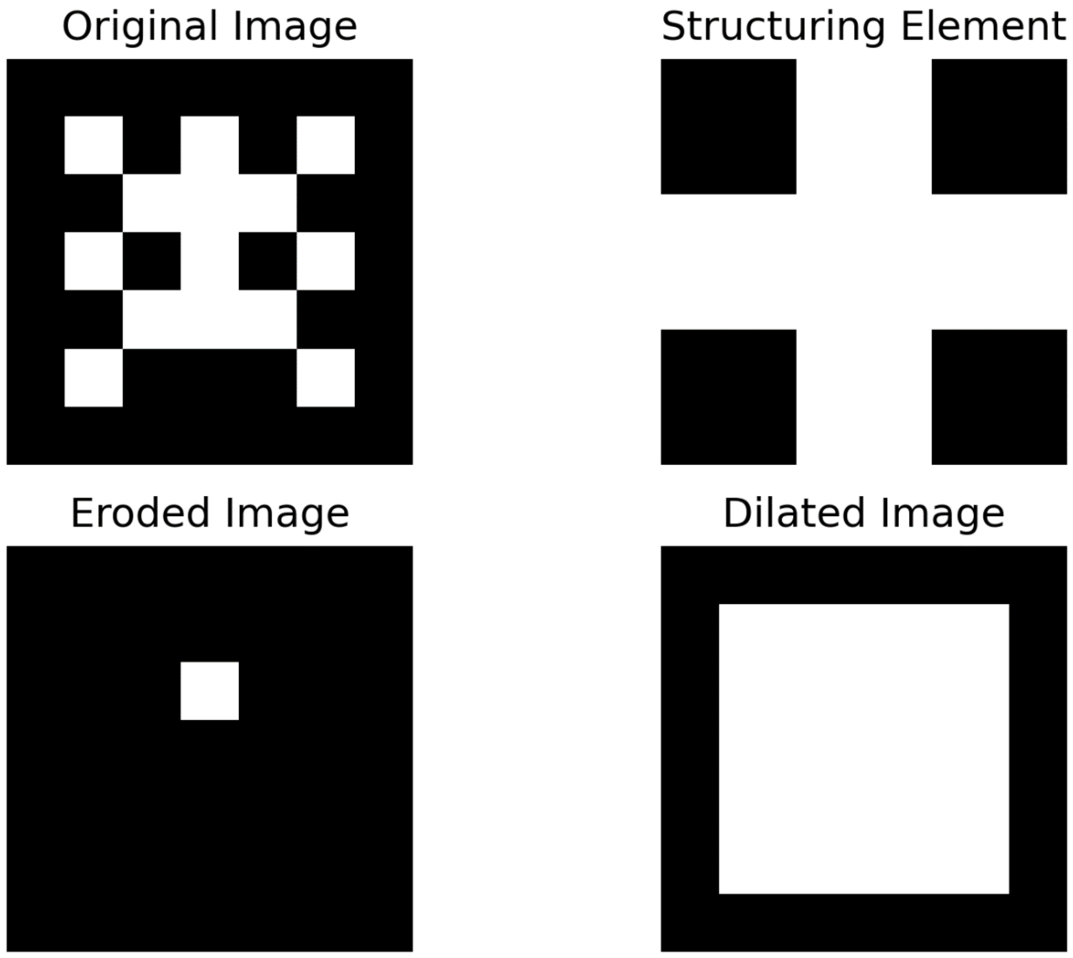
Dilation and Erosion:

```
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:

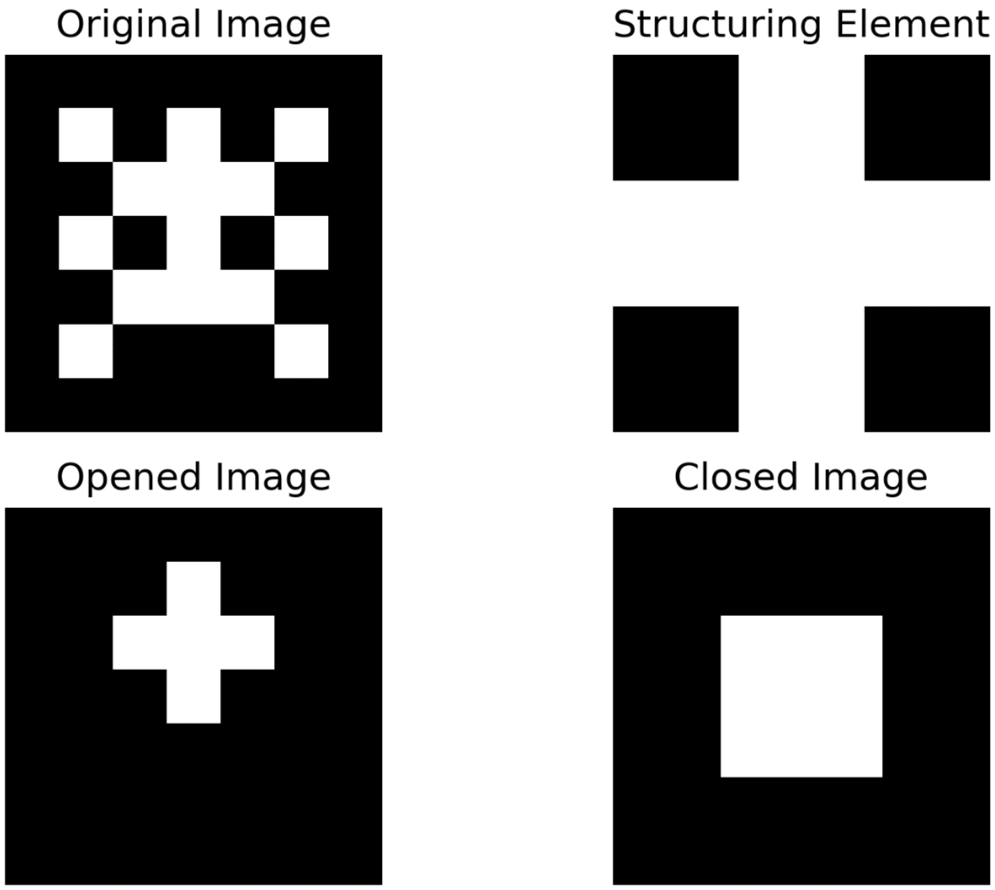


Opening and Closing:

```
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()
```

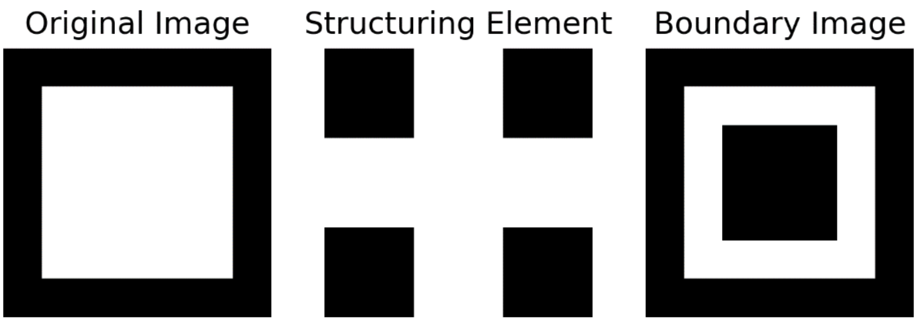
Output:



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 **Detect Lines In An Image Using Hough Transform**
Date: . .2023

Aim:

To detect lines in an image using hough transform

Algorithm Description:

Step 1 :Load the input image. Convert it to grayscale (if it's not already in grayscale) to simplify the computation. Apply Edge Detection:
Step 2: Apply an edge detection algorithm (e.g., Canny edge detector) to detect edges in the grayscale image.

Define the Hough Transform Parameters:

Step 3: Define the Hough Transform parameters, including:Resolution of the parameter space (rho and theta).Threshold for considering a point as part of a line.Minimum line length.Maximum gap between line segments.Initialize the Hough Accumulator:
Step 4:Create an accumulator array to represent the parameter space. The size of the array should be determined by the parameter space resolution.Voting in the Hough Space:
Step 5: For each edge point in the edge-detected image:Loop through a range of possible values of theta (usually from -90 to 90 degrees).Calculate the corresponding rho value for each (rho, theta) pair.
Step 6: Increment the accumulator array at the (rho, theta) position.

Find Local Maxima in the Accumulator:

Step 7: Identify local maxima in the accumulator space. These local maxima correspond to the detected lines.

Filter Detected Lines:

Step 8: Filter out detected lines based on the threshold, minimum line length, and maximum gap between line segments.Draw Detected Lines on the Original Image:
Step 9: For each detected line (rho, theta), convert them back to Cartesian coordinates.Draw the detected lines on the original image using the cv2.line function.Display or Save the Result:
Step 10: Display the original image with detected lines or save the result as needed.

Post-processing: You can perform additional post-processing steps on the detected lines, such as filtering based on angles or other criteria.

Code:

```
import cv2
import numpy as np
import math
from google.colab import files
from google.colab.patches import cv2_imshow
def hough_line(edge):
    theta = np.arange(0, 180, 1)
```

```
    cos = np.cos(np.deg2rad(theta))
    sin = np.sin(np.deg2rad(theta))
    rho_range = round(math.sqrt(edge.shape[0]**2 + edge.shape[1]**2))
    accumulator = np.zeros((2 * rho_range, len(theta)), dtype=np.uint8)
    edge_pixels = np.where(edge == 255)
    coordinates = list(zip(edge_pixels[0], edge_pixels[1]))
    for p in range(len(coordinates)):
        for t in range(len(theta)):
            rho = int(round(coordinates[p][1] * cos[t] + coordinates[p][0] * sin[t]))
            accumulator[rho, t] += 1
    return accumulator
# Upload an image file
uploaded = files.upload()
# Process the uploaded image
for filename in uploaded.keys():
    img = cv2.imread(filename)
    edges = cv2.Canny(img, 75, 150)
    accumulator = hough_line(edges)
    edge_pixels = np.where(accumulator > 120)
    coordinates = list(zip(edge_pixels[0], edge_pixels[1]))
    for i in range(0, len(coordinates)):
        a = np.cos(np.deg2rad(coordinates[i][1]))
        b = np.sin(np.deg2rad(coordinates[i][1]))
        x0 = a * coordinates[i][0]
        y0 = b * coordinates[i][0]
        x1 = int(x0 + 1000 * (-b))
        y1 = int(y0 + 1000 * (a))
        x2 = int(x0 - 1000 * (-b))
        y2 = int(y0 - 1000 * (a))
        cv2.line(img, (x1, y1), (x2, y2), (0, 255, 0), 1)
    cv2_imshow(img)
```

Output:

Result:

Thus the program to detect the lines in an image is being done using hough transform is executed successfully and verified.

Ex. No.:06 **Image Segmentation with Watershed Algorithm**
Date: . .2023

Aim:

To perform Image Segmentation with Watershed Algorithm

Algorithm:

Step 1: Load the Image

Step 2: Convert to Grayscale

Step 3: Apply Gaussian Blur

Step 4: Thresholding

Step 5: Morphological Operations (Optional)

Step 6: Sure Background and Sure Foreground

Step 7: Find the Unknown Region

Step 8: Label the Markers

Step 9: Apply the Watershed Algorithm

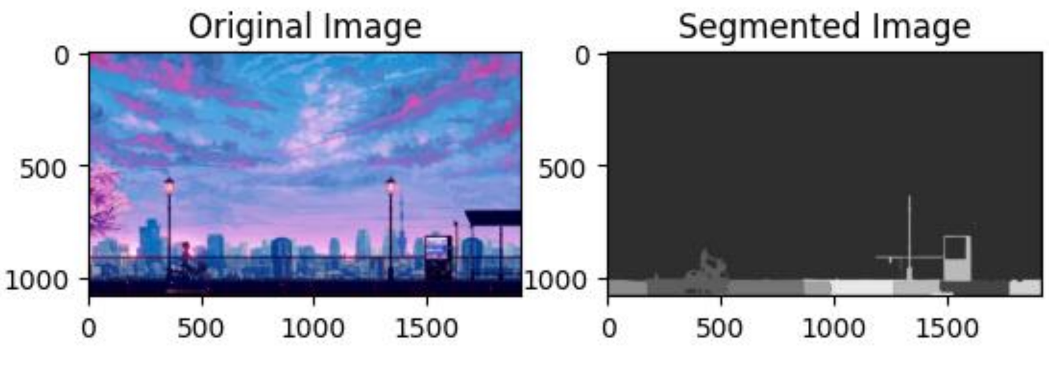
Step 10: Display the Segmented Image

Code:

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
from google.colab import files
# Upload an image file
uploaded = files.upload()
# Process the uploaded image
for filename in uploaded.keys():
    # Load the uploaded image
    image = cv2.imread(filename)
    # Convert the image to grayscale
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    # Apply thresholding to create a binary image
    ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)
    # Noise removal using morphological operations
    kernel = np.ones((3, 3), np.uint8)
```

```
opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)
# Sure background area
sure_bg = cv2.dilate(opening, kernel, iterations=3)
# Finding sure foreground area
dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)
sure_fg = np.uint8(sure_fg)
# Finding unknown region
unknown = cv2.subtract(sure_bg, sure_fg)
# Marker labelling
ret, markers = cv2.connectedComponents(sure_fg)
# Add 1 to all labels so that sure background is not 0, but 1
markers = markers + 1
# Mark the region of unknown with 0
markers[unknown == 255] = 0
# Apply watershed algorithm
markers = cv2.watershed(image, markers)
# Mark the segmented regions on the original image
image[markers == -1] = [0, 0, 255]  # Mark boundaries in red
# Display the result
plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Original Image')
plt.subplot(122), plt.imshow(markers, cmap='gray'), plt.title('Segmented Image')
plt.show()
```

Output:



Result:

The Image Segmentation with Watershed Algorithm have been created successfully.

Ex No:07 **3D Shape From Texture And 3d Object Detection**

DATE: . .2023

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:

Step 1: Convert the input texture image to grayscale.

Step 2: Apply gradient operations (Sobel filters) to calculate gradients in the x and y directions.

Step 3: Create a constant gradient component in the z-direction to represent surface depth.

Step 4: Combine the gradient components into a 3D vector for surface normals.

Step 5: Normalize the surface normals to ensure consistent magnitude.

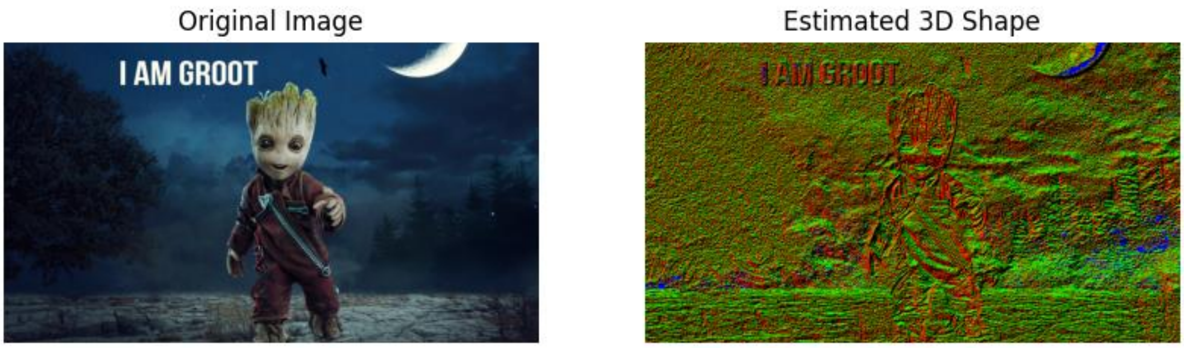
Step 6: Return the estimated 3D shape (surface normals).

Code:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files
def estimate_3d_shape_from_texture(texture_image):
    gray = cv2.cvtColor(texture_image, cv2.COLOR_BGR2GRAY)
    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)
    surface_normals = np.dstack((gradient_x, gradient_y, gradient_z))
    surface_normals /= np.linalg.norm(surface_normals, axis=-1, keepdims=True)
    return surface_normals
# Upload an image file
uploaded = files.upload()
# Process the uploaded image
for filename in uploaded.keys():
    # Load the uploaded image
    texture_image = cv2.imread(filename)
```

```
texture_image = cv2.cvtColor(texture_image, cv2.COLOR_BGR2RGB)
# Estimate 3D shape from the texture image
estimated_3d_shape = estimate_3d_shape_from_texture(texture_image)
# Display the original image and estimated 3D shape
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

Step 1: load the image in which you want to perform face and eye detection.

Step 2: convert the image to grayscale for efficient processing.

Step 3: create haar cascade classifiers for both face and eye detection.

Step 4: use the face cascade classifier to detect faces in the grayscale image, specifying scaling parameters (scale factor and minimum neighbors).

Step 5: iterate throughthe detected face regions.

Step 6: draw rectangles around the detected faces on the original color image.

Step 7: create regions of interest (roi) in both grayscale and color based on the face bounding box.

Step 8: use the eye cascade classifier to detect eyes within each face roi. Step 9: iterate through the detected eye regions within each face.

Step 10: draw rectangles around the detected eyes on the color face roi. Step 11: continue this process for all detected faces in the image.

Step 12: display the image with rectangles drawn around the detected faces and eyes. Step 13: optionally, save or analyze the results for further processing.

Step 14: the code effectively detects faces and eyes in the input image, highlighting them with rectangles.

Step 15: adjust the scaling parameters as needed to fine-tune the detection results.

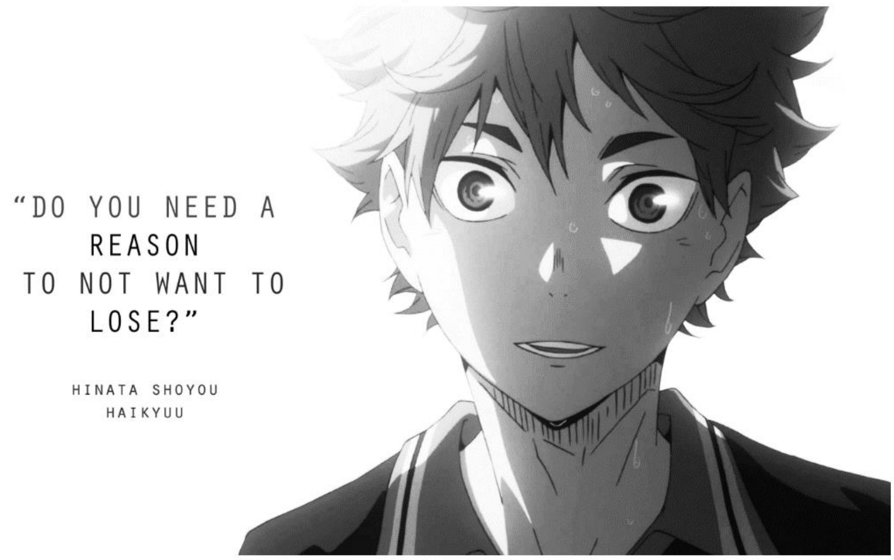
Code:

```
import cv2
import matplotlib.pyplot as plt
from google.colab import files
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_eye.xml')
# Upload an image file
uploaded = files.upload()
# Process the uploaded image
for filename in uploaded.keys():
    # Load the uploaded image
    img = cv2.imread(filename)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

```
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
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)
# Display the image with detected face and eyes
plt.figure(dpi=300)
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.axis("off")
plt.title('Face and Eye Detection')
plt.show()
```

Output:

Face and Eye Detection



Result:

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