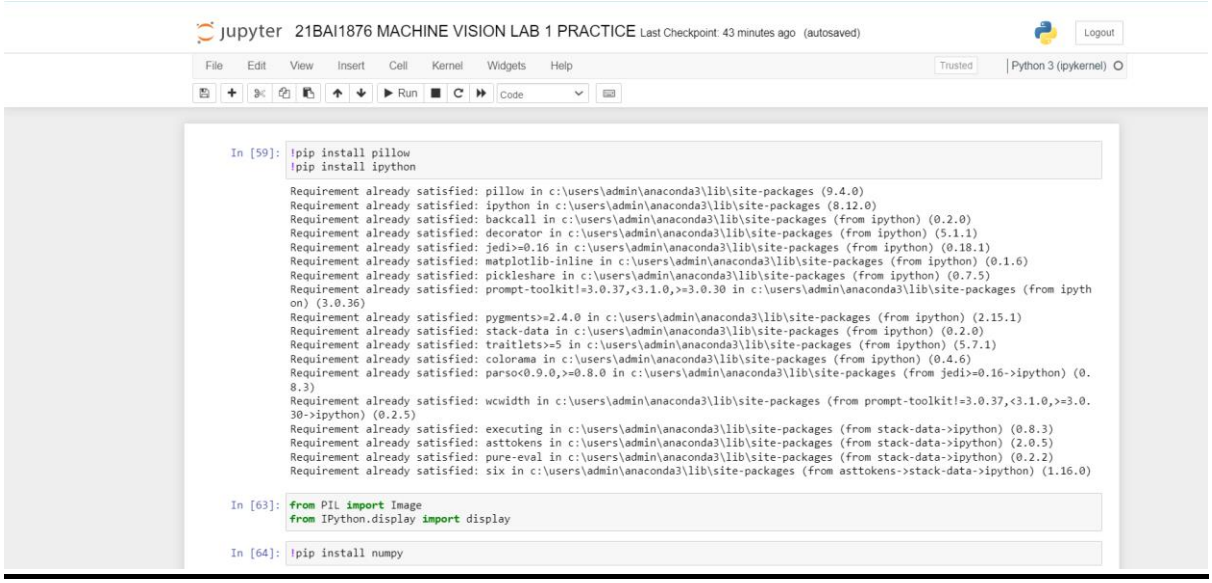
MACHINE VISION

LAB 1

NAME : OM SUBRATO DEY

REGISTER NUMBER: 21BAI1876

1. READ A BLACK AND WHITE IMAGE AND DISPLAY THE BLACK AND WHITE PIXELS OF THE IMAGE READ RESPECTIVELY. ALSO DISPLAY THE IMAGE IN BLACK AND WHITE CHANNELS SEPERATELY.



jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: 43 minutes ago (autosaved)

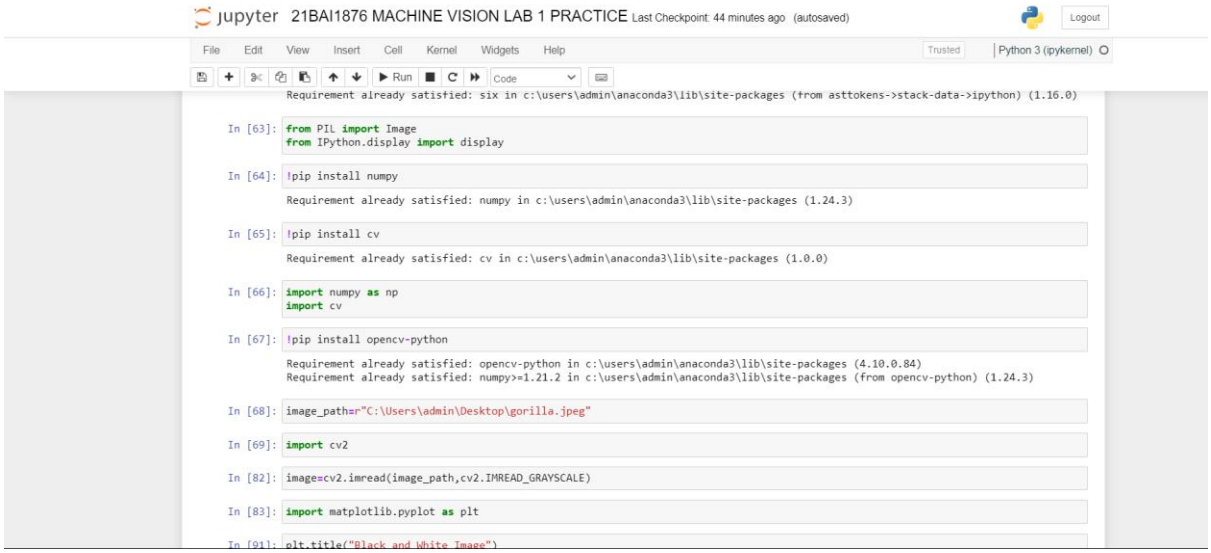
Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

```
In [59]: !pip install pillow
         !pip install ipython

Requirement already satisfied: pillow in c:\users\admin\anaconda3\lib\site-packages (9.4.0)
Requirement already satisfied: ipython in c:\users\admin\anaconda3\lib\site-packages (8.12.0)
Requirement already satisfied: backcall in c:\users\admin\anaconda3\lib\site-packages (from ipython) (0.2.0)
Requirement already satisfied: decorator in c:\users\admin\anaconda3\lib\site-packages (from ipython) (5.1.1)
Requirement already satisfied: jedi>=0.16 in c:\users\admin\anaconda3\lib\site-packages (from ipython) (0.18.1)
Requirement already satisfied: matplotlib-inline in c:\users\admin\anaconda3\lib\site-packages (from ipython) (0.1.6)
Requirement already satisfied: pickleshare in c:\users\admin\anaconda3\lib\site-packages (from ipython) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.37,<3.1.0,>=3.0.30 in c:\users\admin\anaconda3\lib\site-packages (from ipyth
on) (3.0.36)
Requirement already satisfied: pygments>=2.4.0 in c:\users\admin\anaconda3\lib\site-packages (from ipython) (2.15.1)
Requirement already satisfied: stack-data in c:\users\admin\anaconda3\lib\site-packages (from ipython) (0.2.0)
Requirement already satisfied: traitlets>=5 in c:\users\admin\anaconda3\lib\site-packages (from ipython) (5.7.1)
Requirement already satisfied: colorama in c:\users\admin\anaconda3\lib\site-packages (from ipython) (0.4.6)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in c:\users\admin\anaconda3\lib\site-packages (from jedi>=0.16->ipython) (0.
8.3)
Requirement already satisfied: wcwidth in c:\users\admin\anaconda3\lib\site-packages (from prompt-toolkit!=3.0.37,<3.1.0,>=3.0.
30->ipython) (0.2.5)
Requirement already satisfied: executing in c:\users\admin\anaconda3\lib\site-packages (from stack-data->ipython) (0.8.3)
Requirement already satisfied: asttokens in c:\users\admin\anaconda3\lib\site-packages (from stack-data->ipython) (2.0.5)
Requirement already satisfied: pure-eval in c:\users\admin\anaconda3\lib\site-packages (from stack-data->ipython) (0.2.2)
Requirement already satisfied: six in c:\users\admin\anaconda3\lib\site-packages (from asttokens->stack-data->ipython) (1.16.0)

In [63]: from PIL import Image
         from IPython.display import display

In [64]: !pip install numpy
```



jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: 44 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

```
Requirement already satisfied: six in c:\users\admin\anaconda3\lib\site-packages (from asttokens->stack-data->ipython) (1.16.0)

In [63]: from PIL import Image
         from IPython.display import display

In [64]: !pip install numpy

Requirement already satisfied: numpy in c:\users\admin\anaconda3\lib\site-packages (1.24.3)

In [65]: !pip install cv

Requirement already satisfied: cv in c:\users\admin\anaconda3\lib\site-packages (1.0.0)

In [66]: import numpy as np
         import cv

In [67]: !pip install opencv-python

Requirement already satisfied: opencv-python in c:\users\admin\anaconda3\lib\site-packages (4.10.0.84)
Requirement already satisfied: numpy>=1.21.2 in c:\users\admin\anaconda3\lib\site-packages (from opencv-python) (1.24.3)

In [68]: image_path=r"C:\Users\admin\Desktop\gorilla.jpeg"

In [69]: import cv2

In [82]: image=cv2.imread(image_path,cv2.IMREAD_GRAYSCALE)

In [83]: import matplotlib.pyplot as plt

In [91]: plt.title("Black and White Image")
```

jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [83]:
```
import matplotlib.pyplot as plt
```

In [91]:
```
plt.title("Black and White Image")
plt.axis('off')
plt.show()
```

In [85]:
```
black_pixels=np.argwhere(image==0)
white_pixels=np.argwhere(image==255)
```

In [86]:
```
print(f"Black pixels (total {black_pixels.shape[0]})")
print(black_pixels)
```

jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [85]:
```
black_pixels=np.argwhere(image==0)
white_pixels=np.argwhere(image==255)
```

In [86]:
```
print(f"Black pixels (total {black_pixels.shape[0]})")
print(black_pixels)
```

```
Black pixels (total 559)
[[  6  24]
 [ 20   6]
 [ 26  42]
 ...
 [220 219]
 [221  34]
 [223 206]]
```

In [87]:
```
print(f"\n white pixels (total {white_pixels.shape[0]})")
print(white_pixels)
```

```
 [ 94 202]
 [ 95 173]
 [ 95 202]
 [ 96 174]
 [ 96 201]
 [ 97 174]
 [ 97 201]
 [ 98 175]
 [ 98 202]
 [ 99 175]
 [ 99 202]
```

jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

```
[102 202]
[103 173]
[103 202]
```

In [77]:
```
ig=Image.open('gorilla.jpeg')
```

In [78]:
```
ig
```

Out[78]:

In [92]:
```
# Threshold the image to separate black and white pixels
_, black_channel = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
_, white_channel = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

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

jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: an hour ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [92]:
```
# Threshold the image to separate black and white pixels
_, black_channel = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY_INV)
_, white_channel = cv2.threshold(image, 127, 255, cv2.THRESH_BINARY)

# Display the original image
plt.subplot(1, 3, 1)
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Display the black channel
plt.subplot(1, 3, 2)
plt.imshow(black_channel, cmap='gray')
plt.title('Black Channel')
plt.axis('off')

# Display the white channel
plt.subplot(1, 3, 3)
plt.imshow(white_channel, cmap='gray')
plt.title('White Channel')
plt.axis('off')

plt.show()
```

2. READ A COLOUR IMAGE AND SEPARATE THE RED,BLUE AND GREEN CHANNELS.

jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: 2 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [95]:
```
"""2.READ A COLOUR IMAGE AND SEPARATE THE RED,BLUE AND GREEN CHANNELS."""
```

Out[95]:
```
'2.READ A COLOUR IMAGE AND SEPARATE THE RED,BLUE AND GREEN CHANNELS.'
```

In [96]:
```
image_path2=r"C:\Users\admin\Desktop\colourimage.jpeg"
```

In [97]:
```
img2=cv2.imread(image_path2)
```

In [102]:
```
# Convert the image from BGR to RGB format
image_rgb = cv2.cvtColor(img2, cv2.COLOR_BGR2RGB)

# Segregate the channels
red_channel = image_rgb.copy()
red_channel[:, :, 1] = 0  # Set green channel to 0
red_channel[:, :, 2] = 0  # Set blue channel to 0

green_channel = image_rgb.copy()
green_channel[:, :, 0] = 0  # Set red channel to 0
green_channel[:, :, 2] = 0  # Set blue channel to 0

blue_channel = image_rgb.copy()
blue_channel[:, :, 0] = 0  # Set red channel to 0
blue_channel[:, :, 1] = 0  # Set green channel to 0
```

In [103]:
```
# Display the images
plt.figure(figsize=(10, 7))

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

Out[103]:
```
(-0.5, 274.5, 182.5, -0.5)
```

jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [103]:
```
# Display the images
plt.figure(figsize=(10, 7))

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

Out[103]:
```
(-0.5, 274.5, 182.5, -0.5)
```

Original Image

In [104]:
```
plt.subplot(2, 2, 2)
plt.imshow(red_channel)
plt.title('Red Channel')
plt.axis('off')
```

Out[104]:
```
(-0.5, 274.5, 182.5, -0.5)
```

Red Channel

jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [104]:
```
plt.subplot(2, 2, 2)
plt.imshow(red_channel)
plt.title('Red Channel')
plt.axis('off')
```

Out[104]: (-0.5, 274.5, 182.5, -0.5)

Red Channel

In [105]:
```
plt.subplot(2, 2, 3)
plt.imshow(green_channel)
plt.title('Green Channel')
plt.axis('off')
```

Out[105]: (-0.5, 274.5, 182.5, -0.5)

Green Channel

jupyter 21BAI1876 MACHINE VISION LAB 1 PRACTICE Last Checkpoint: 3 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

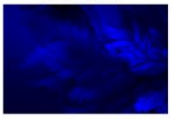
Trusted | Python 3 (ipykernel)

Run Code

In [106]:
```
plt.subplot(2, 2, 4)
plt.imshow(blue_channel)
plt.title('Blue Channel')
plt.axis('off')
```

Out[106]: (-0.5, 274.5, 182.5, -0.5)

Blue Channel



In [107]:
```
plt.tight_layout()
plt.show()
```

<Figure size 640x480 with 0 Axes>

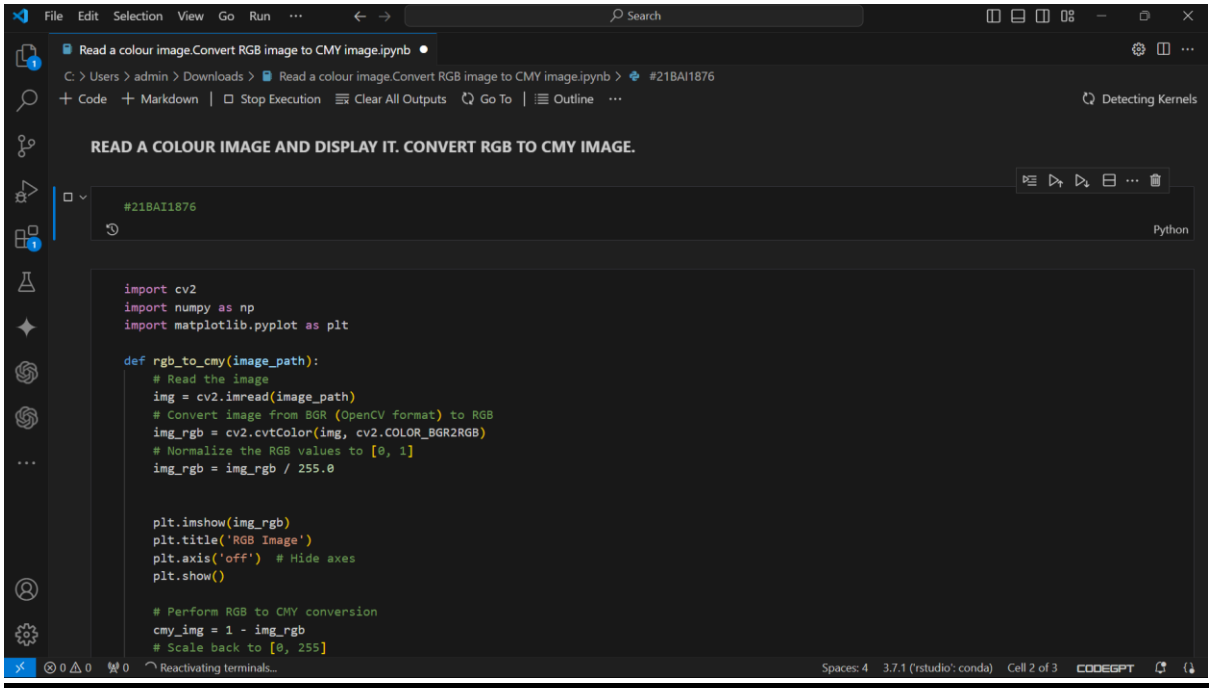
3. CONVERT A RGB IMAGE TO CMY IMAGE WHICH WILL BE THE NEGATIVE OF THE ORIGINAL IMAGE

CODE:

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

def rgb_to_cmy(image_path):
    # Read the image
    img = cv2.imread(image_path)
    # Convert image from BGR (OpenCV format) to RGB
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Normalize the RGB values to [0, 1]
    img_rgb = img_rgb / 255.0



    plt.imshow(img_rgb)
    plt.title('RGB Image')
    plt.axis('off')  # Hide axes
    plt.show()

    # Perform RGB to CMY conversion
    cmy_img = 1 - img_rgb
    # Scale back to [0, 255]
    cmy_img = (cmy_img * 255).astype(np.uint8)
    return cmy_img
```

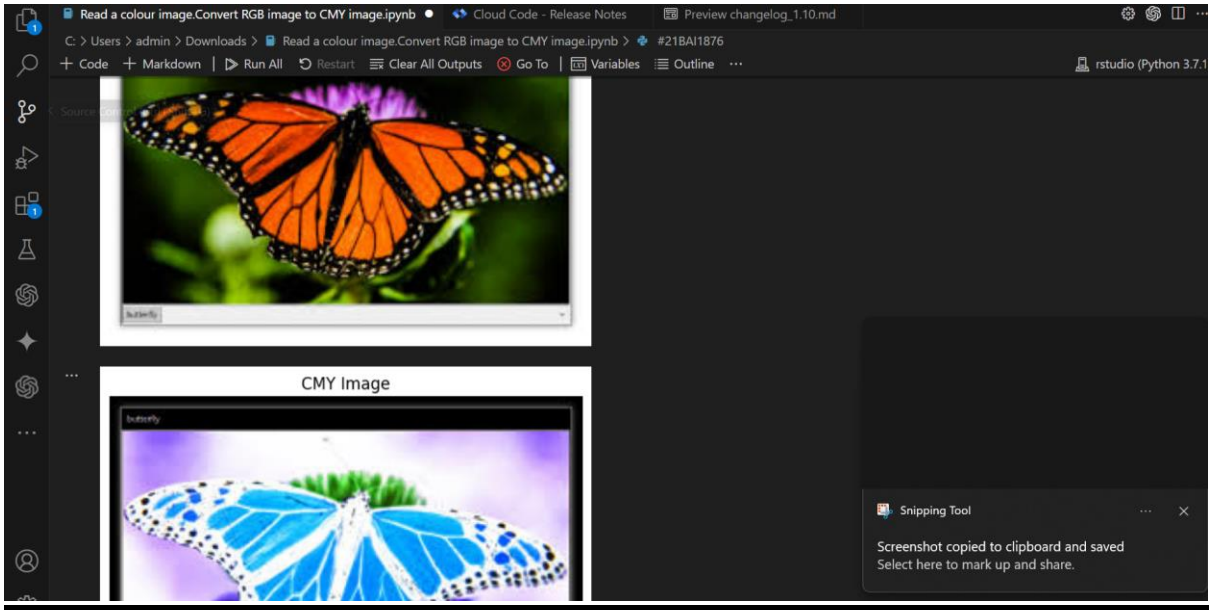
```
# Example usage
image_path = '/content/moth.jpeg'
cmy_image = rgb_to_cmy(image_path)

# Display the CMY image using Matplotlib
plt.imshow(cmy_image)
plt.title('CMY Image')
plt.axis('off')  # Hide axes
plt.show()

# Save the CMY image
cv2.imwrite('/content/moth_cmy.jpg', cv2.cvtColor(cmy_image,
cv2.COLOR_RGB2BGR))
```

OUTPUT:



```python
    # Normalize the RGB values to [0, 1]
    img_rgb = img_rgb / 255.0


    plt.imshow(img_rgb)
    plt.title('RGB Image')
    plt.axis('off')  # Hide axes
    plt.show()

    # Perform RGB to CMY conversion
    cmy_img = 1 - img_rgb
    # Scale back to [0, 255]
    cmy_img = (cmy_img * 255).astype(np.uint8)
    return cmy_img

# Example usage
image_path = '/content/moth.jpeg'
cmy_image = rgb_to_cmy(image_path)

# Display the CMY image using Matplotlib
plt.imshow(cmy_image)
plt.title('CMY Image')
plt.axis('off')  # Hide axes
plt.show()

# Save the CMY image
cv2.imwrite('/content/moth_cmy.jpg', cv2.cvtColor(cmy_image, cv2.COLOR_RGB2BGR))
```

[13] Python



4. And 5. CONVERT A RGB IMAGE TO HSI IMAGE AND THEN BACK TO RGB

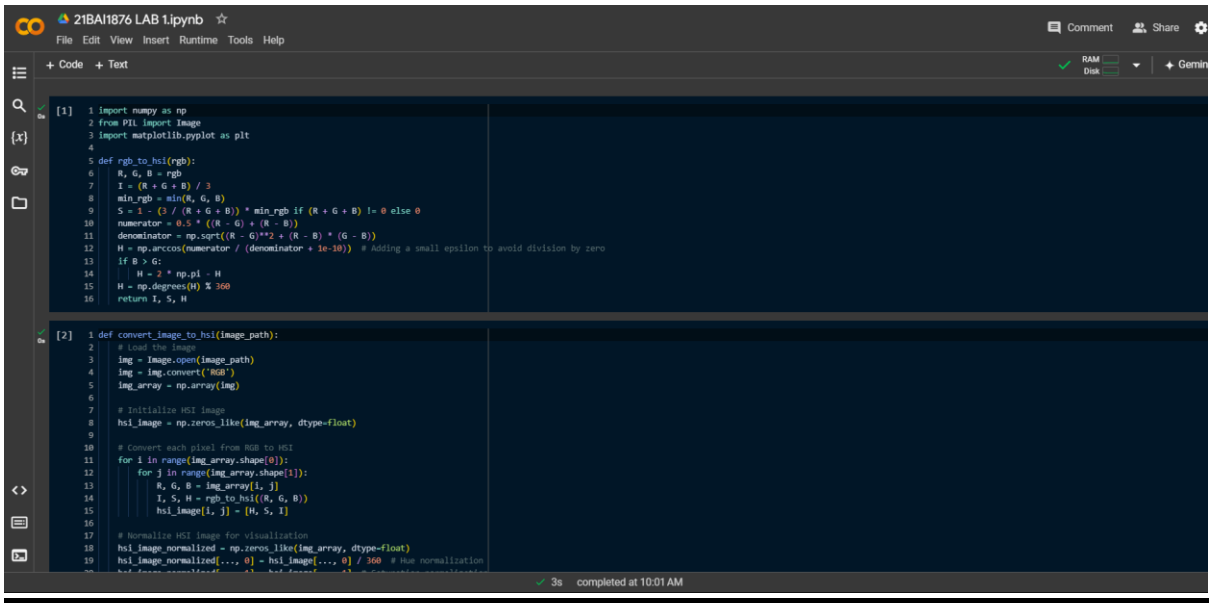
CODE:

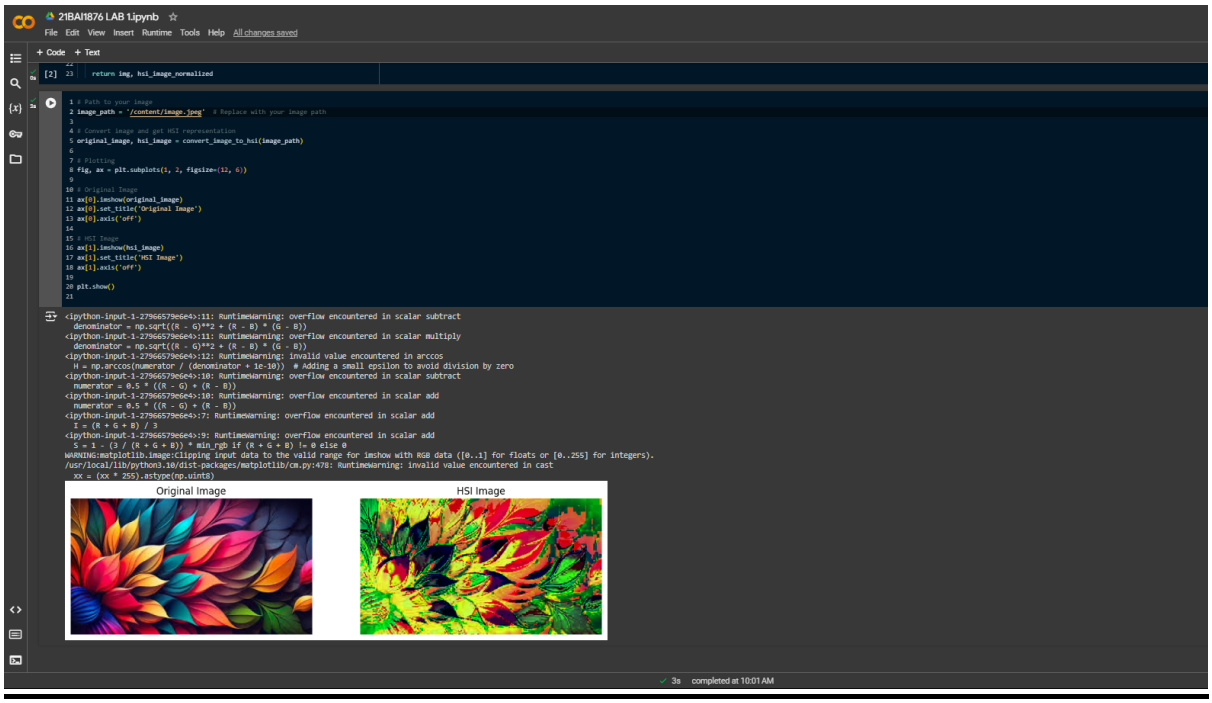
```
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt


def rgb_to_hsi(rgb):
    R, G, B = rgb
    I = (R + G + B) / 3
    min_rgb = min(R, G, B)
    S = 1 - (3 / (R + G + B)) * min_rgb if (R + G + B) != 0 else
0
    numerator = 0.5 * ((R - G) + (R - B))
    denominator = np.sqrt((R - G)**2 + (R - B) * (G - B))
    H = np.arccos(numerator / (denominator + 1e-10))  # Adding a
small epsilon to avoid division by zero
    if B > G:
        H = 2 * np.pi - H
    H = np.degrees(H) % 360
    return I, S, H


def convert_image_to_hsi(image_path):
    # Load the image
    img = Image.open(image_path)
    img = img.convert('RGB')
    img_array = np.array(img)

    # Initialize HSI image
```

```
    hsi_image = np.zeros_like(img_array, dtype=float)

    # Convert each pixel from RGB to HSI
    for i in range(img_array.shape[0]):
        for j in range(img_array.shape[1]):
            R, G, B = img_array[i, j]
            I, S, H = rgb_to_hsi((R, G, B))
            hsi_image[i, j] = [H, S, I]

    # Normalize HSI image for visualization
    hsi_image_normalized = np.zeros_like(img_array, dtype=float)
    hsi_image_normalized[..., 0] = hsi_image[..., 0] / 360  # Hue normalization
    hsi_image_normalized[..., 1] = hsi_image[..., 1]  # Saturation normalization
    hsi_image_normalized[..., 2] = hsi_image[..., 2] / 255  # Intensity normalization
    return img, hsi_image_normalized

# Path to your image
image_path = '/content/moth.jpeg'  # Replace with your image path

# Convert image and get HSI representation
original_image, hsi_image = convert_image_to_hsi(image_path)

# Plotting
fig, ax = plt.subplots(1, 2, figsize=(12, 6))

# Original Image
ax[0].imshow(original_image)
ax[0].set_title('Original Image')
```

```
ax[0].axis('off')
# HSI Image
ax[1].imshow(hsi_image)
ax[1].set_title('HSI Image')
ax[1].axis('off')
plt.show()
```

OUTPUT:





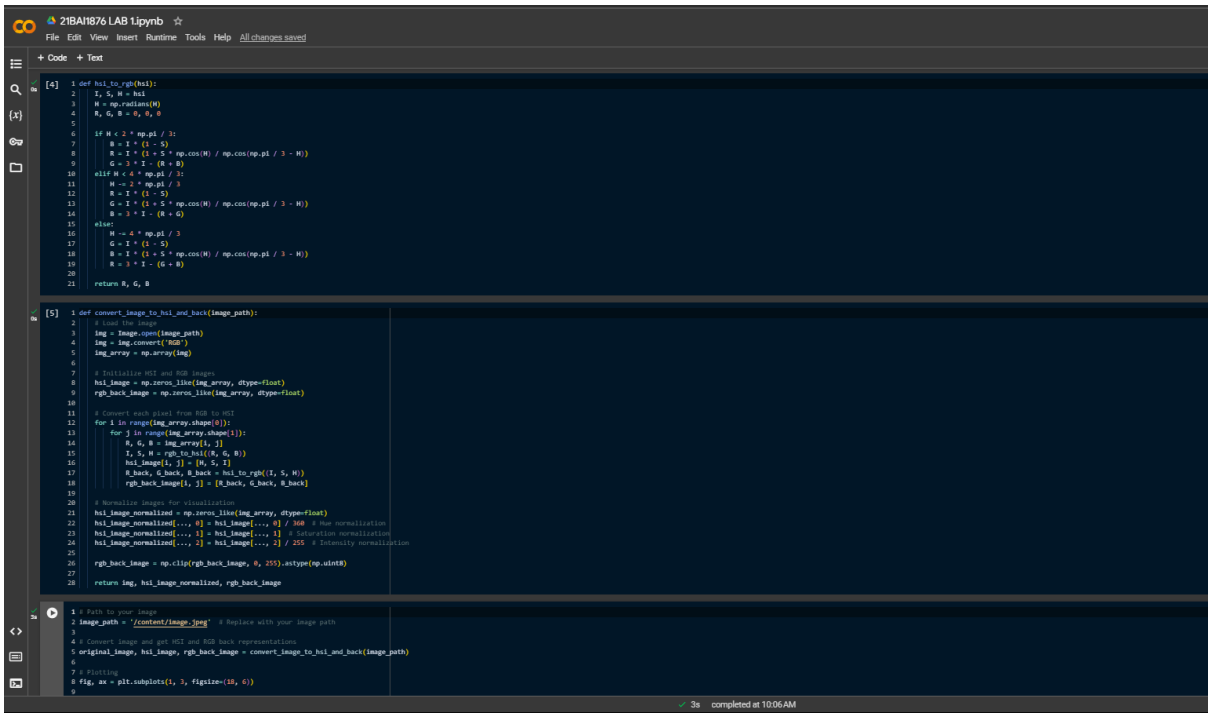
CODE FOR HIS TO RGB:

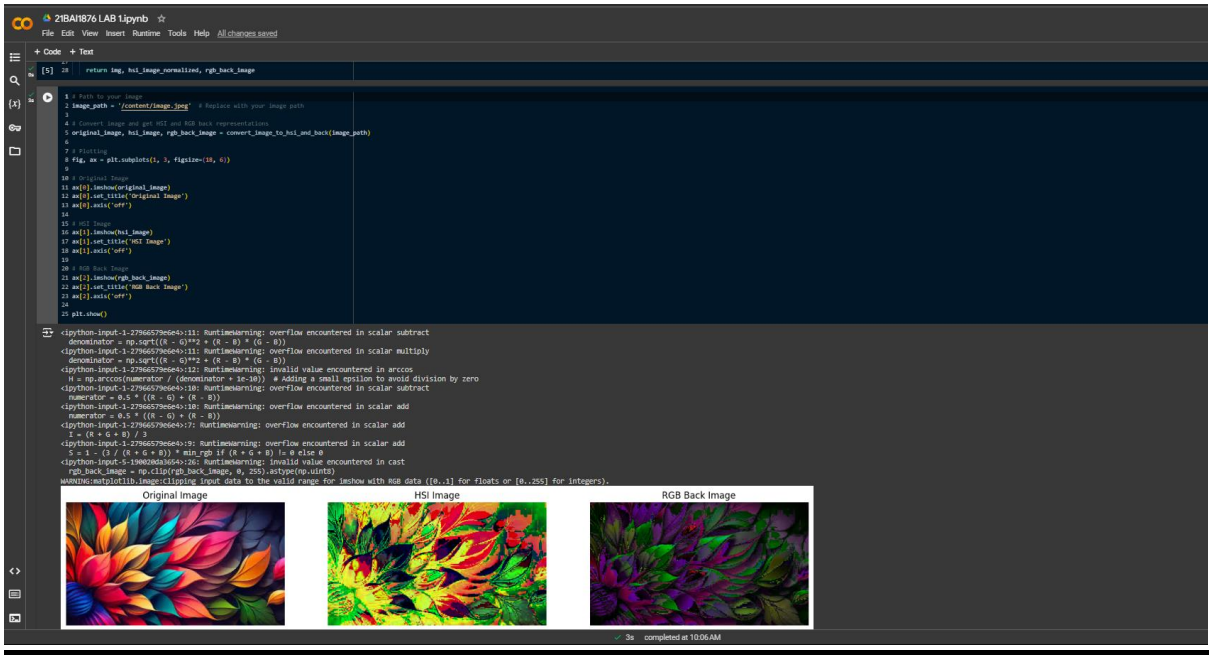
```
def hsi_to_rgb(hsi):
    I, S, H = hsi
    H = np.radians(H)
    R, G, B = 0, 0, 0

    if H < 2 * np.pi / 3:
        B = I * (1 - S)
        R = I * (1 + S * np.cos(H) / np.cos(np.pi / 3 - H))
        G = 3 * I - (R + B)
    elif H < 4 * np.pi / 3:
        H -= 2 * np.pi / 3
        R = I * (1 - S)
        G = I * (1 + S * np.cos(H) / np.cos(np.pi / 3 - H))
        B = 3 * I - (R + G)
    else:
        H -= 4 * np.pi / 3
        G = I * (1 - S)
        B = I * (1 + S * np.cos(H) / np.cos(np.pi / 3 - H))
        R = 3 * I - (G + B)

    return R, G, B

def convert_image_to_hsi_and_back(image_path):
    # Load the image
    img = Image.open(image_path)
    img = img.convert('RGB')
    img_array = np.array(img)
```

```
    # Initialize HSI and RGB images
    hsi_image = np.zeros_like(img_array, dtype=float)
    rgb_back_image = np.zeros_like(img_array, dtype=float)

    # Convert each pixel from RGB to HSI
    for i in range(img_array.shape[0]):
        for j in range(img_array.shape[1]):
            R, G, B = img_array[i, j]
            I, S, H = rgb_to_hsi((R, G, B))
            hsi_image[i, j] = [H, S, I]
            R_back, G_back, B_back = hsi_to_rgb((I, S, H))
            rgb_back_image[i, j] = [R_back, G_back, B_back]

    # Normalize images for visualization
    hsi_image_normalized = np.zeros_like(img_array, dtype=float)
    hsi_image_normalized[..., 0] = hsi_image[..., 0] / 360  #
Hue normalization
    hsi_image_normalized[..., 1] = hsi_image[..., 1]  #
Saturation normalization
    hsi_image_normalized[..., 2] = hsi_image[..., 2] / 255  #
Intensity normalization

    rgb_back_image = np.clip(rgb_back_image, 0,
255).astype(np.uint8)
    return img, hsi_image_normalized, rgb_back_image

# Path to your image
image_path = '/content/moth.jpeg'  # Replace with your image
path
```

```
# Convert image and get HSI and RGB back representations
original_image, hsi_image, rgb_back_image =
convert_image_to_hsi_and_back(image_path)


# Plotting
fig, ax = plt.subplots(1, 3, figsize=(18, 6))


# Original Image
ax[0].imshow(original_image)
ax[0].set_title('Original Image')
ax[0].axis('off')


# HSI Image
ax[1].imshow(hsi_image)
ax[1].set_title('HSI Image')
ax[1].axis('off')


# RGB Back Image
ax[2].imshow(rgb_back_image)
ax[2].set_title('RGB Back Image')
ax[2].axis('off')


plt.show()
```

OUTPUT:


21BAI1876 LAB 1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[4]
def hsi_to_rgb(hsi):
I, S, H = hsi
H = np.radians(H)
R, G, B = 0, 0, 0
if H < 2 * np.pi / 3:
B = I * (1 - S)
R = I * (1 + S * np.cos(H) / np.cos(np.pi / 3 - H))
G = 3 * I - (R + B)
elif H < 4 * np.pi / 3:
H -= 2 * np.pi / 3
R = I * (1 - S)
G = I * (1 + S * np.cos(H) / np.cos(np.pi / 3 - H))
B = 3 * I - (R + G)
else:
H -= 4 * np.pi / 3
G = I * (1 - S)
B = I * (1 + S * np.cos(H) / np.cos(np.pi / 3 - H))
R = 3 * I - (G + B)
return R, G, B
[5]
def convert_image_to_hsi_and_back(image_path):
# Load the image
img = Image.open(image_path)
img = img.convert('RGB')
img_array = np.array(img)
# Initialize HSI and RGB images
hsi_image = np.zeros_like(img_array, dtype=float)
rgb_back_image = np.zeros_like(img_array, dtype=float)
# Convert each pixel from RGB to HSI
for i in range(img_array.shape[0]):
for j in range(img_array.shape[1]):
R, G, B = img_array[i, j]
I, S, H = rgb_to_hsi((R, G, B))
hsi_image[i, j] = [H, S, I]
R_back, G_back, B_back = hsi_to_rgb((I, S, H))
rgb_back_image[i, j] = [R_back, G_back, B_back]
# Normalize images for visualization
hsi_image_normalized = np.zeros_like(img_array, dtype=float)
hsi_image_normalized[..., 0] = hsi_image[..., 0] / 360 # Hue normalization
hsi_image_normalized[..., 1] = hsi_image[..., 1] # Saturation normalization
hsi_image_normalized[..., 2] = hsi_image[..., 2] / 255 # Intensity normalization
rgb_back_image = np.clip(rgb_back_image, 0, 255).astype(np.uint8)
return img, hsi_image_normalized, rgb_back_image
# Path to your image
image_path = '/content/image.jpeg' # Replace with your image path
# Convert image and get HSI and RGB back representations
original_image, hsi_image, rgb_back_image = convert_image_to_hsi_and_back(image_path)
Plotting
fig, ax = plt.subplots(1, 3, figsize=(18, 6))
3s completed at 10:06 AM


21BAI1876 LAB 1.ipynb
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
[5] return img, hsi_image_normalized, rgb_back_image
# Path to your image
image_path = '/content/image.jpeg' # Replace with your image path
# Convert image and get HSI and RGB back representations
original_image, hsi_image, rgb_back_image = convert_image_to_hsi_and_back(image_path)
Plotting
fig, ax = plt.subplots(1, 3, figsize=(18, 6))
Original Image
ax[0].imshow(original_image)
ax[0].set_title('Original Image')
ax[0].axis('off')
HSI Image
ax[1].imshow(hsi_image)
ax[1].set_title('HSI Image')
ax[1].axis('off')
RGB Back Image
ax[2].imshow(rgb_back_image)
ax[2].set_title('RGB Back Image')
ax[2].axis('off')
plt.show()
<ipython-input-1-27966579e6e4>:11: RuntimeWarning: overflow encountered in scalar subtract
denominator = np.sqrt((R - G)**2 + (R - B) * (G - B))
<ipython-input-1-27966579e6e4>:11: RuntimeWarning: overflow encountered in scalar multiply
denominator = np.sqrt((R - G)**2 + (R - B) * (G - B))
<ipython-input-1-27966579e6e4>:12: RuntimeWarning: invalid value encountered in arccos
H = np.arccos(numerator / (denominator + 1e-10)) # Adding a small epsilon to avoid division by zero
<ipython-input-1-27966579e6e4>:10: RuntimeWarning: overflow encountered in scalar subtract
numerator = 0.5 * ((R - G) + (R - B))
<ipython-input-1-27966579e6e4>:10: RuntimeWarning: overflow encountered in scalar add
numerator = 0.5 * ((R - G) + (R - B))
<ipython-input-1-27966579e6e4>:7: RuntimeWarning: overflow encountered in scalar add
I = (R + G + B) / 3
<ipython-input-1-27966579e6e4>:9: RuntimeWarning: overflow encountered in scalar add
S = 1 - (3 / (R + G + B)) * min_rgb if (R + G + B) != 0 else 0
<ipython-input-5-190020d3d654>:26: RuntimeWarning: invalid value encountered in cast
rgb_back_image = np.clip(rgb_back_image, 0, 255).astype(np.uint8)
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Original Image
HSI Image
RGB Back Image
3s completed at 10:06 AM

MACHINE VISION LAB

Lab Ex 2: Gray Level Transformation – Image Negative, Gamma Correction, Log transform, Image Enhancement - Low pass/High Pass spatial Filtering – Gaussian Filters, Noise Filtering.

NAME : OM SUBRATO DEY

REG NO.: 21BAI1876

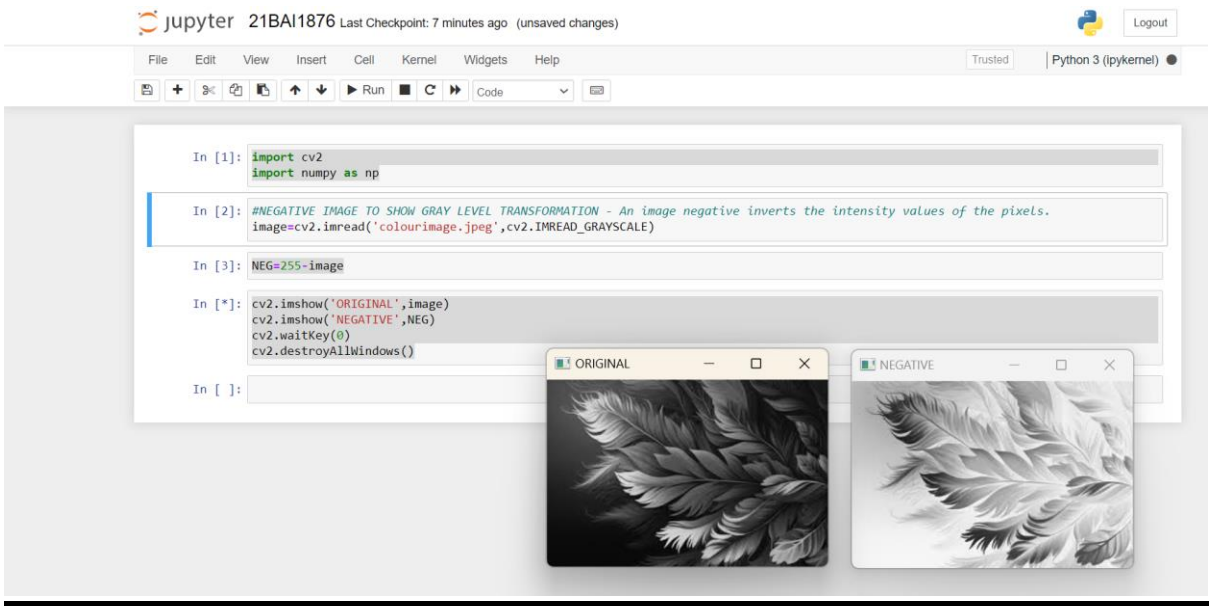
1. Gray Level Transformation – Image Negative

[An image negative inverts the intensity values of the pixels]

CODE:

```
import cv2

import numpy as np


#NEGATIVE IMAGE TO SHOW GRAY LEVEL TRANSFORMATION

image=cv2.imread('colourimage.jpeg',cv2.IMREAD_GRAYSCALE)

NEG=255-image

cv2.imshow('ORIGINAL',image)

cv2.imshow('NEGATIVE',NEG)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

OUTPUT:



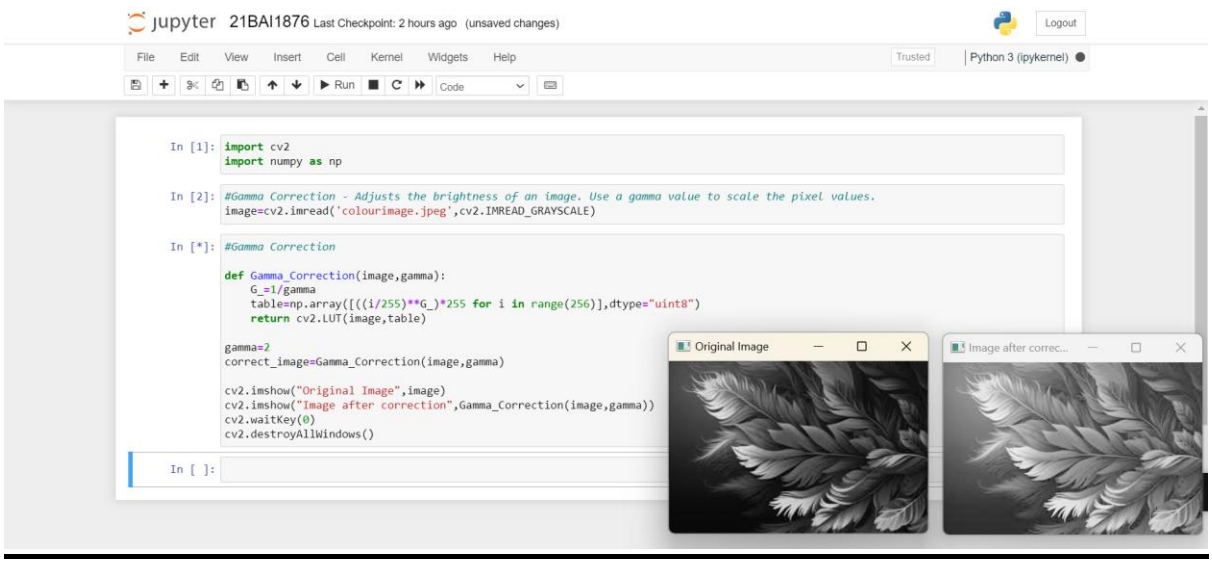
2. Gamma Correction – Modifying Brightness

[Adjusts the brightness of an image. Use a gamma value to scale the pixel values.]

CODE:

```
import cv2
import numpy as np
#reading image
image=cv2.imread('colourimage.jpeg',cv2.IMREAD_GRAYSCALE)


def Gamma_Correction(image,gamma):
    G_=1/gamma
    table=np.array([((i/255)**G_)*255 for i in range(256)],dtype="uint8")
    return cv2.LUT(image,table)


gamma=2
correct_image=Gamma_Correction(image,gamma)


cv2.imshow("Original Image",image)
cv2.imshow("Image after correction",Gamma_Correction(image,gamma))
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT:


jupyter 21BAI1876 Last Checkpoint: 2 hours ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [1]:
import cv2
import numpy as np
In [2]:
#Gamma Correction - Adjusts the brightness of an image. Use a gamma value to scale the pixel values.
image=cv2.imread('colourimage.jpeg',cv2.IMREAD_GRAYSCALE)
In [*]:
#Gamma Correction
def Gamma_Correction(image,gamma):
G_=1/gamma
table=np.array([((i/255)**G_)*255 for i in range(256)],dtype="uint8")
return cv2.LUT(image,table)
gamma=2
correct_image=Gamma_Correction(image,gamma)
cv2.imshow("Original Image",image)
cv2.imshow("Image after correction",Gamma_Correction(image,gamma))
cv2.waitKey(0)
cv2.destroyAllWindows()
In [ ]:
Original Image
Image after correc...

3. Log transform – For better dynamic range

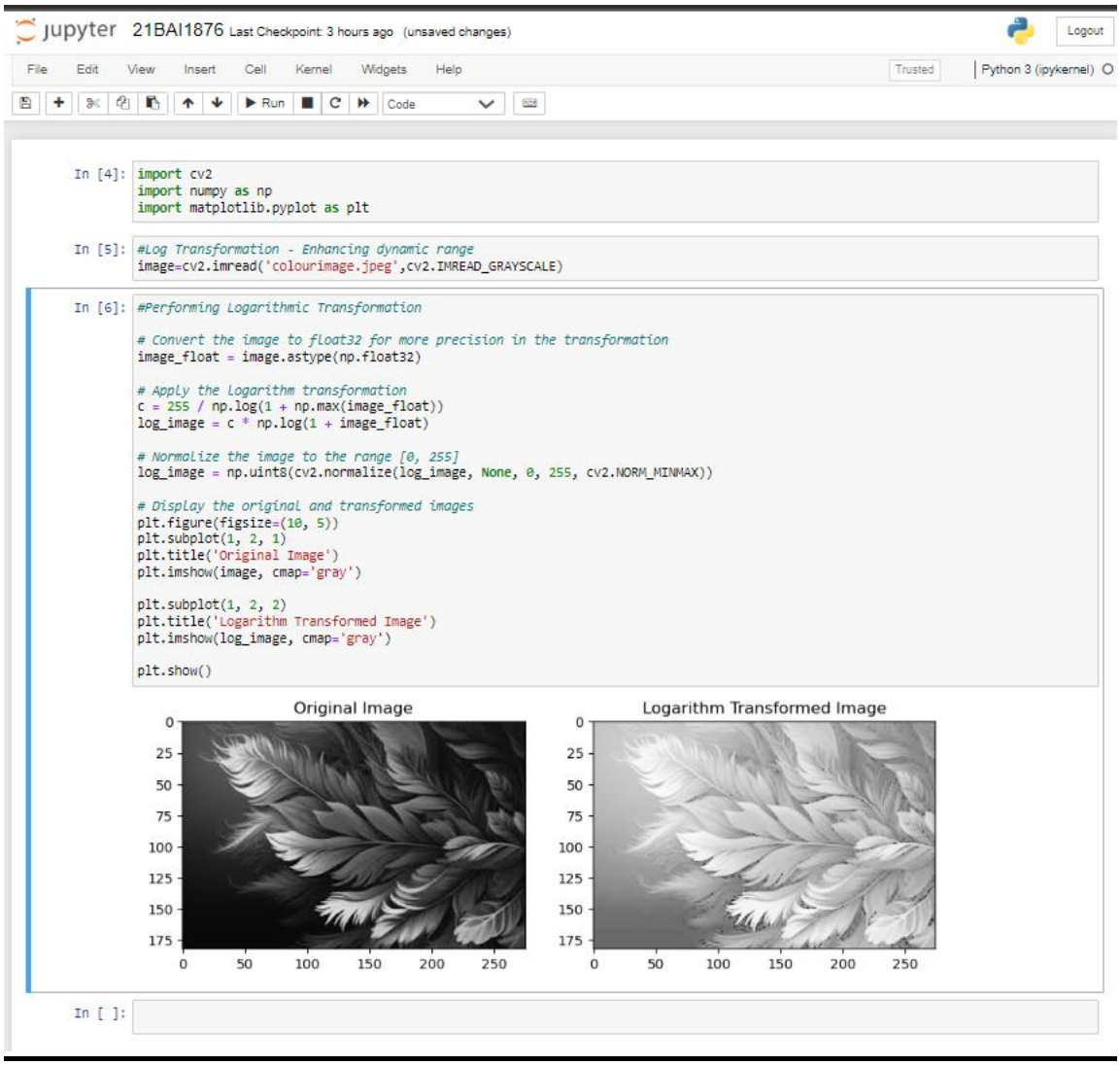
[Enhances the dynamic range of an image.]

CODE:

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

#Log Transformation - Enhancing dynamic range
image=cv2.imread('colourimage.jpeg',cv2.IMREAD_GRAYSCALE)

#Performing Logarithmic Transformation
# Convert the image to float32 for more precision in the transformation
image_float = image.astype(np.float32)

# Apply the logarithm transformation
c = 255 / np.log(1 + np.max(image_float))
log_image = c * np.log(1 + image_float)

# Normalize the image to the range [0, 255]
log_image = np.uint8(cv2.normalize(log_image, None, 0, 255, cv2.NORM_MINMAX))

# Display the original and transformed images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.title('Original Image')
```

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


plt.subplot(1, 2, 2)
plt.title('Logarithm Transformed Image')
plt.imshow(log_image, cmap='gray')
plt.show()
```

OUTPUT:



4. Image Enhancement - Low Pass/High Pass Spatial Filtering

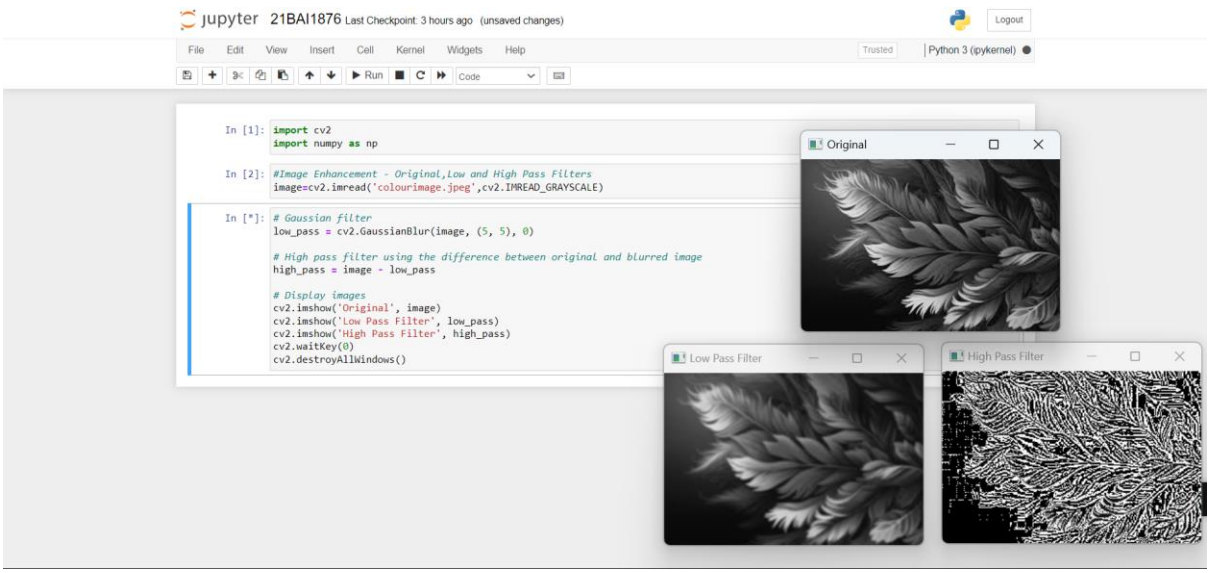
[Low Pass Filtering: Smooths the image, reduces high-frequency noise. High Pass Filtering: Enhances edges and fine details. Gaussian Filters are commonly used for both.]

CODE:

```
import cv2

import numpy as np

#Image Enhancement - Original,Low and High Pass Filters

image=cv2.imread('colourimage.jpeg',cv2.IMREAD_GRAYSCALE)

# Gaussian filter

low_pass = cv2.GaussianBlur(image, (5, 5), 0)



# High pass filter using the difference between original and blurred image

high_pass = image - low_pass



# Display images

cv2.imshow('Original', image)

cv2.imshow('Low Pass Filter', low_pass)

cv2.imshow('High Pass Filter', high_pass)

cv2.waitKey(0)
```

cv2.destroyAllWindows()

OUTPUT:



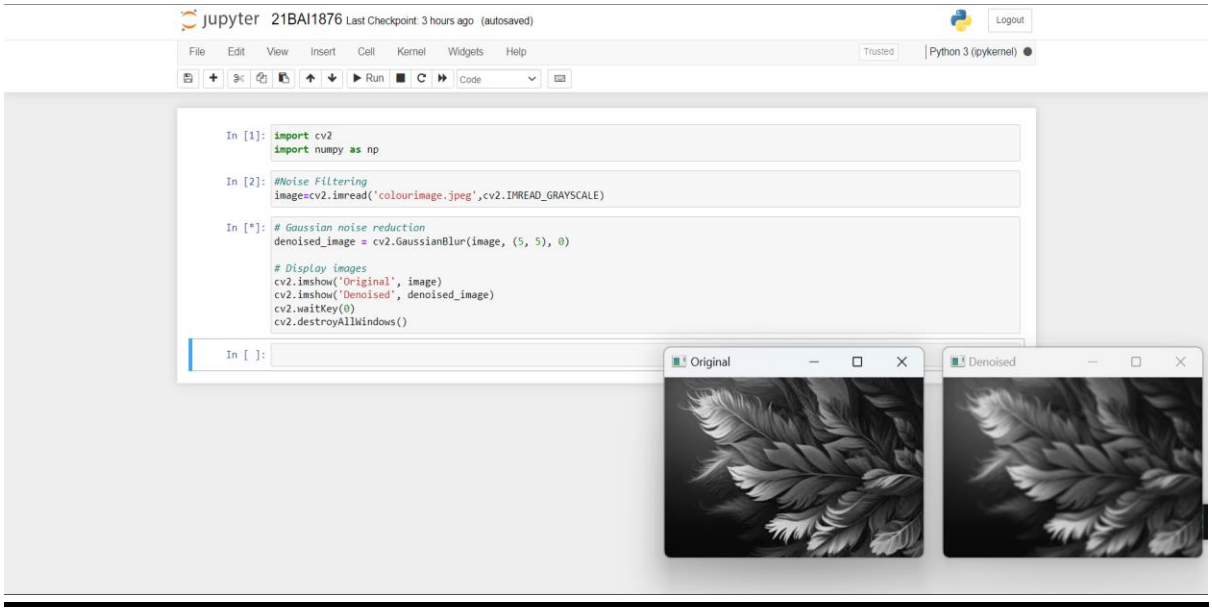
5. Noise Filtering – Gaussian Filters

[Gaussian filters are effective for reducing noise.]

CODE:

```
import cv2

import numpy as np

#Noise Filtering

image=cv2.imread('colourimage.jpeg',cv2.IMREAD_GRAYSCALE)

# Gaussian noise reduction

denoised_image = cv2.GaussianBlur(image, (5, 5), 0)

# Display images

cv2.imshow('Original', image)

cv2.imshow('Denoised', denoised_image)

cv2.waitKey(0)

cv2.destroyAllWindows()
```

OUTPUT:


jupyter 21BAI1876 Last Checkpoint: 3 hours ago (autosaved)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3 (ipykernel)
Run
Code
In [1]:
import cv2
import numpy as np
In [2]:
#Noise Filtering
image=cv2.imread('colourimage.jpeg',cv2.IMREAD_GRAYSCALE)
In [*]:
# Gaussian noise reduction
denoised_image = cv2.GaussianBlur(image, (5, 5), 0)
# Display images
cv2.imshow('Original', image)
cv2.imshow('Denoised', denoised_image)
cv2.waitKey(0)
cv2.destroyAllWindows()
In [ ]:
Original
Denoised

MACHINE VISION

LAB 3 : HISTOGRAM EQUALIZATION

NAME : OM SUBRATO DEY

REGISTER NUMBER: 21BAI1876

FACULTY : AMUTHA S

CODE:

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

#As per given data on board
numbers = [10,15,5,6,21,37,41,17,54,65,27,33,3,64,2]

# Define the bins for the categories
divisions_along_x_axis = [0, 9, 19, 29, 39, 49, 59, 69, 79]

# Categorize the random numbers into the bins
counts, _ = np.histogram(numbers, bins=divisions_along_x_axis)

# Plotting the histogram using matplotlib
plt.hist(numbers, bins=divisions_along_x_axis,
edgecolor='black')
plt.title('Numbers Histogram')
plt.xlabel('Divisions along X Axis')
plt.ylabel('Frequency')
plt.xticks(bins)

# Plotting the line graph using matplotlib
plt.figure(figsize=(10, 6))
plt.plot(bin_centers, counts, marker='o', linestyle='-',
color='r')
plt.title('Numbers Line Graph')
plt.xlabel('Number Ranges')
plt.ylabel('Frequency')
```

OUTPUT:

jupyter 21BAI1876_LAB_3_HISTOGRAM_PLOTTING Last Checkpoint: 21 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

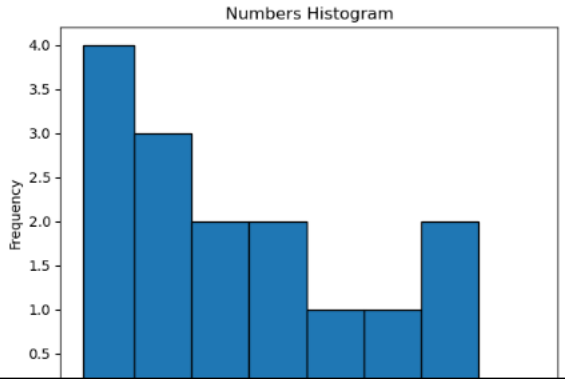
Run Code

In [19]:
```
import matplotlib.pyplot as plt
import numpy as np
import cv2
```

In [20]:
```
numbers = [10,15,5,6,21,37,41,17,54,65,27,33,3,64,2]

# Define the bins for the categories
divisions_along_x_axis = [0, 9, 19, 29, 39, 49, 59, 69, 79]

# Categorize the random numbers into the bins
counts, _ = np.histogram(numbers, bins=divisions_along_x_axis)
```

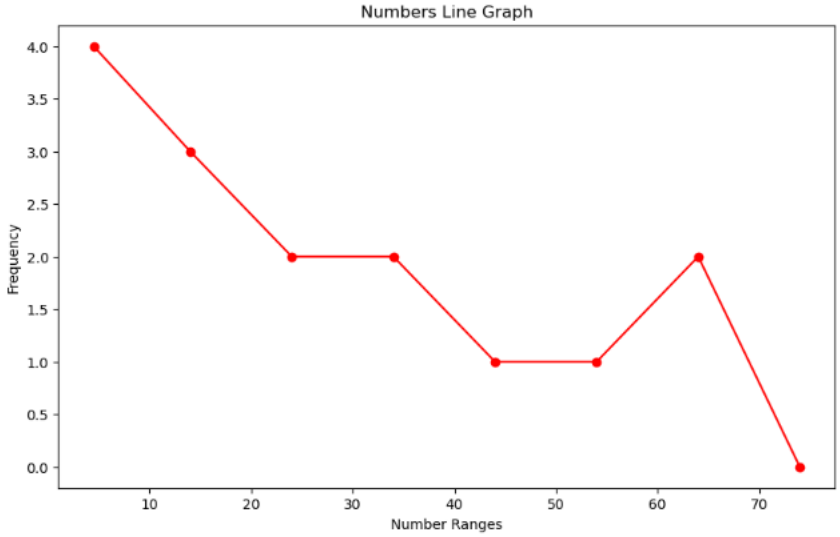
In [21]:
```
# Plotting the histogram using matplotlib
plt.hist(numbers, bins=divisions_along_x_axis, edgecolor='black')
plt.title('Numbers Histogram')
plt.xlabel('Divisions along X Axis')
plt.ylabel('Frequency')
plt.xticks(bins)
```

Out[21]:
```
([<matplotlib.axis.XTick at 0x214b3e209d0>,
  <matplotlib.axis.XTick at 0x214b3e20550>,
  <matplotlib.axis.XTick at 0x214b3e05950>,
  <matplotlib.axis.XTick at 0x214b40a39d0>,
  <matplotlib.axis.XTick at 0x214b40a2fd0>,
  <matplotlib.axis.XTick at 0x214b40af4d0>,
  <matplotlib.axis.XTick at 0x214b40b5610>,
  <matplotlib.axis.XTick at 0x214b40b7850>,
  <matplotlib.axis.XTick at 0x214b40b9ad0>],
 [Text(0, 0, '0'),
  Text(9, 0, '9'),
  Text(19, 0, '19'),
  Text(29, 0, '29'),
  Text(39, 0, '39'),
  Text(49, 0, '49'),
  Text(59, 0, '59'),
  Text(69, 0, '69'),
  Text(79, 0, '79')])
```



In [29]:
```
# Plotting the line graph using matplotlib
plt.figure(figsize=(10, 6))
plt.plot(bin_centers, counts, marker='o', linestyle='-', color='r')
plt.title('Numbers Line Graph')
plt.xlabel('Number Ranges')
plt.ylabel('Frequency')
```

Out[29]: Text(0, 0.5, 'Frequency')



In [ ]:

MACHINE VISION

LAB 4: Median filter, Min filter, Max filter

NAME : OM SUBRATO DEY

REGISTER NUMBER : 21BAI1876

1. Median Filter:

CODE:

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

# Load the image
image_path = 'OMPHOTO.jpeg'  # Replace with your image path
image = cv2.imread(image_path, cv2.IMREAD_COLOR)

# Convert image to RGB (OpenCV loads images in BGR by default)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Apply median filter
# kernel_size should be an odd integer (e.g., 3, 5, 7)
kernel_size = 5
filtered_image = cv2.medianBlur(image_rgb, kernel_size)

# Display the original and filtered images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Filtered Image')
plt.imshow(filtered_image)
```

```
plt.axis('off')
plt.show()
```

OUTPUT:

jupyter 21BAI1876_LAB_4 Last Checkpoint: 11 minutes ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

In [1]:
```
#MEDIAN FILTER
```

In [2]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
```

In [3]:
```
# Load the image
image_path = 'OMPHOTO.jpeg'  # Replace with your image path
image = cv2.imread(image_path, cv2.IMREAD_COLOR)
```

In [4]:
```
# Convert image to RGB (OpenCV loads images in BGR by default)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

In [5]:
```
# Apply median filter
# kernel_size should be an odd integer (e.g., 3, 5, 7)
kernel_size = 5
filtered_image = cv2.medianBlur(image_rgb, kernel_size)
```

In [6]:
```
# Display the original and filtered images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Filtered Image')
plt.imshow(filtered_image)
plt.axis('off')

plt.show()
```

2. Min Filter:

CODE:

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


def min_filter(image, kernel_size):
    # Pad the image to handle border effects
    pad_size = kernel_size // 2
    padded_image = np.pad(image, ((pad_size, pad_size), (pad_size, pad_size), (0, 0)), mode='edge')


    # Output image
    filtered_image = np.zeros_like(image)


    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # Extract the neighborhood
            neighborhood = padded_image[i:i+kernel_size, j:j+kernel_size]
            # Apply min filter
            filtered_image[i, j] = np.min(neighborhood, axis=(0, 1))


    return filtered_image


# Load the image
image_path = 'OMPHOTO.jpeg'  # Replace with your image path
image = cv2.imread(image_path, cv2.IMREAD_COLOR)
```

```
# Convert image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Apply min filter
kernel_size = 5
min_filtered_image = min_filter(image_rgb, kernel_size)

# Display the original and min filtered images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Min Filtered Image')
plt.imshow(min_filtered_image)
plt.axis('off')

plt.show()
```

OUTPUT:

jupyter 21BAI1876_LAB_4 Last Checkpoint: 12 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

Run

In [7]:
```
# MIN FILTER
```

In [8]:
```
def min_filter(image, kernel_size):
    # Pad the image to handle border effects
    pad_size = kernel_size // 2
    padded_image = np.pad(image, ((pad_size, pad_size), (pad_size, pad_size), (0, 0)), mode='edge')

    # Output image
    filtered_image = np.zeros_like(image)

    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # Extract the neighborhood
            neighborhood = padded_image[i:i+kernel_size, j:j+kernel_size]
            # Apply min filter
            filtered_image[i, j] = np.min(neighborhood, axis=(0, 1))

    return filtered_image
```

In [9]:
```
# Load the image
image_path = 'OMPHOTO.jpeg'  # Replace with your image path
image = cv2.imread(image_path, cv2.IMREAD_COLOR)

# Convert image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

In [10]:
```
# Apply min filter
kernel_size = 5
min_filtered_image = min_filter(image_rgb, kernel_size)
```

In [11]:
```
# Display the original and min filtered images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Min Filtered Image')
plt.imshow(min_filtered_image)
plt.axis('off')

plt.show()
```

3. Max Filter

CODE:

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


def max_filter(image, kernel_size):
    # Pad the image to handle border effects
    pad_size = kernel_size // 2
    padded_image = np.pad(image, ((pad_size, pad_size), (pad_size, pad_size), (0, 0)), mode='edge')


    # Output image
    filtered_image = np.zeros_like(image)


    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # Extract the neighborhood
            neighborhood = padded_image[i:i+kernel_size, j:j+kernel_size]
            # Apply max filter
            filtered_image[i, j] = np.max(neighborhood, axis=(0, 1))


    return filtered_image


# Load the image
image_path = 'OMPHOTO.jpeg'  # Replace with your image path
image = cv2.imread(image_path, cv2.IMREAD_COLOR)
```

```
# Convert image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# Apply max filter
kernel_size = 5
max_filtered_image = max_filter(image_rgb, kernel_size)

# Display the original and max filtered images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Max Filtered Image')
plt.imshow(max_filtered_image)
plt.axis('off')

plt.show()
```

OUTPUT:

jupyter 21BAI1876_LAB_4 Last Checkpoint: a minute ago (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help

Trusted | Python 3 (ipykernel)

Run Code

```
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            # Extract the neighborhood
            neighborhood = padded_image[i:i+kernel_size, j:j+kernel_size]
            # Apply max filter
            filtered_image[i, j] = np.max(neighborhood, axis=(0, 1))

    return filtered_image
```

In [15]:
```
# Load the image
image_path = 'OMPHOTO.jpeg'  # Replace with your image path
image = cv2.imread(image_path, cv2.IMREAD_COLOR)

# Convert image to RGB
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

In [16]:
```
# Apply max filter
kernel_size = 5
max_filtered_image = max_filter(image_rgb, kernel_size)
```

In [17]:
```
# Apply max filter
kernel_size = 5
max_filtered_image = max_filter(image_rgb, kernel_size)

# Display the original and max filtered images
plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)
plt.title('Original Image')
plt.imshow(image_rgb)
plt.axis('off')

plt.subplot(1, 2, 2)
plt.title('Max Filtered Image')
plt.imshow(max_filtered_image)
plt.axis('off')

plt.show()
```

In [ ]:

MACHINE VISION LAB 5

Ex. Edge Detection Using Sobel, Robert and Prewitt.

NAME : OM SUBRATO DEY

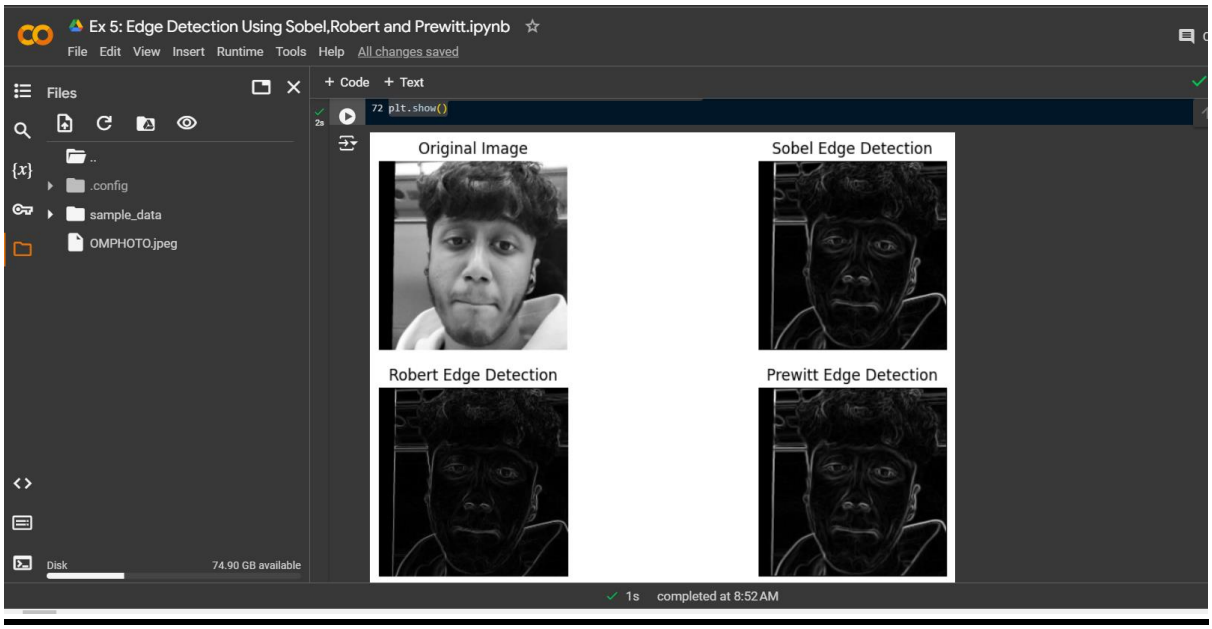
REGISTER NUMBER : 21BAI1876

CODE:

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

# Load the image
img = cv2.imread('/content/OMPHOTO.jpeg', 0)

# Define the Sobel operators
sobel_x = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
sobel_y = np.array([[-1, -2, -1],[0, 0, 0],[1, 2, 1]])

# Define the Robert operators
robert_x = np.array([[1, 0],[0, -1]])
robert_y = np.array([[0, 1],[-1, 0]])

# Define the Prewitt operators
prewitt_x = np.array([[-1, 0, 1],[-1, 0, 1],[-1, 0, 1]])
prewitt_y = np.array([[-1, -1, -1],[0, 0, 0],[1, 1, 1]])

# Function to apply the operators
def apply_operator(img, operator):
    rows, cols = img.shape
    result = np.zeros((rows, cols))
    operator_rows, operator_cols = operator.shape
    for i in range(rows - operator_rows + 1):
        for j in range(cols - operator_cols + 1):
            sum = 0
            for k in range(operator_rows):
                for l in range(operator_cols):
```

```
                    sum += img[i + k, j + l] * operator[k, l]
            result[i + 1, j + 1] = sum
    return result

# Apply the Sobel operators
sobel_x_img = apply_operator(img, sobel_x)
sobel_y_img = apply_operator(img, sobel_y)
sobel_img = np.sqrt(sobel_x_img**2 + sobel_y_img**2)

# Apply the Robert operators
robert_x_img = apply_operator(img, robert_x)
robert_y_img = apply_operator(img, robert_y)
robert_img = np.sqrt(robert_x_img**2 + robert_y_img**2)

# Apply the Prewitt operators
prewitt_x_img = apply_operator(img, prewitt_x)
prewitt_y_img = apply_operator(img, prewitt_y)
prewitt_img = np.sqrt(prewitt_x_img**2 + prewitt_y_img**2)

# Display the results
plt.figure(figsize=(10, 6))
plt.subplot(2, 2, 1), plt.imshow(img, cmap='gray')
plt.title('Original Image'), plt.axis('off')
plt.subplot(2, 2, 2), plt.imshow(sobel_img, cmap='gray')
plt.title('Sobel Edge Detection'), plt.axis('off')
plt.subplot(2, 2, 3), plt.imshow(robert_img, cmap='gray')
plt.title('Robert Edge Detection'), plt.axis('off')
plt.subplot(2, 2, 4), plt.imshow(prewitt_img, cmap='gray')
plt.title('Prewitt Edge Detection'), plt.axis('off')
plt.show()
```

OUTPUT:


Ex 5: Edge Detection Using Sobel,Robert and Prewitt.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
+ Code + Text
72 plt.show()
.config
sample_data
OMPHOTO.jpeg
Disk 74.90 GB available
Original Image
Sobel Edge Detection
Robert Edge Detection
Prewitt Edge Detection
1s completed at 8:52 AM

MACHINE VISION LAB 6

Ex. Image Segmentation (Thresholding and Region Growing)

NAME : OM SUBRATO DEY

REGISTER NUMBER : 21BAI1876

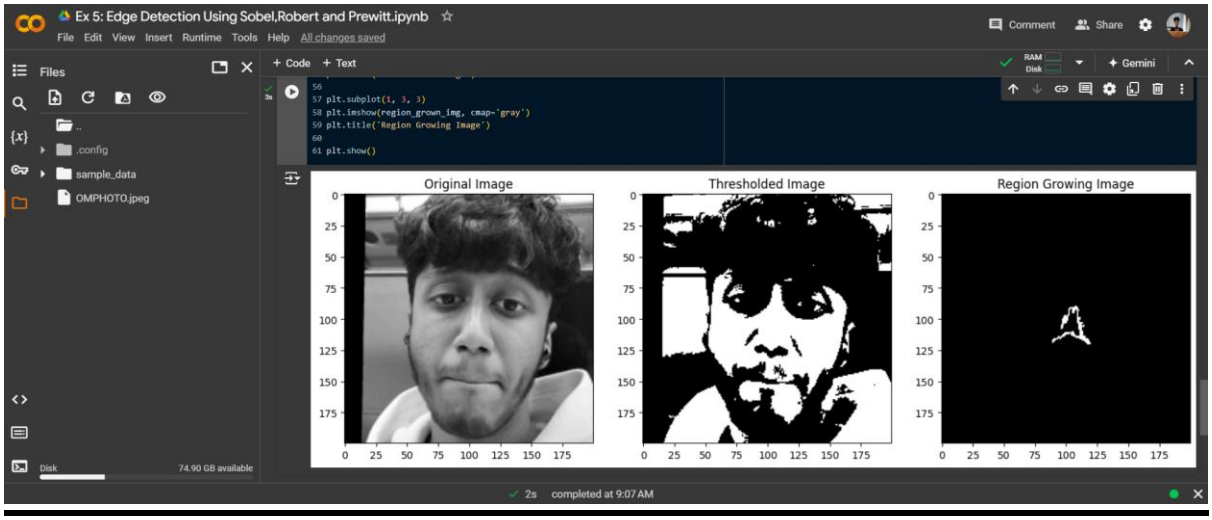
CODE:

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

# Load the image in grayscale
image_path = '/content/OMPHOTO.jpeg'
img = cv2.imread(image_path, 0)  # 0 for grayscale

# Thresholding (binary segmentation)
_, thresh_img = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)

# Region Growing function
def region_growing(img, seed, threshold=10):
    height, width = img.shape
    segmented_img = np.zeros_like(img)
    visited = np.zeros_like(img, dtype=bool)

    # Initialize the region with the seed point
    region_pixels = [seed]
    region_intensity = img[seed]
    segmented_img[seed] = 255
    visited[seed] = True

    # Define the 4-connectivity (up, down, left, right)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]

    while region_pixels:
        current_pixel = region_pixels.pop(0)
```

```
        for direction in directions:
            x = current_pixel[0] + direction[0]
            y = current_pixel[1] + direction[1]

            if 0 <= x < height and 0 <= y < width and not
visited[x, y]:
                if abs(int(img[x, y]) - int(region_intensity)) <
threshold:
                    segmented_img[x, y] = 255
                    visited[x, y] = True
                    region_pixels.append((x, y))

    return segmented_img

# Seed point for region growing (you can adjust this)
seed_point = (100, 100)  # Example: manually set a seed
region_grown_img = region_growing(img, seed_point, threshold=10)

# Plot original, thresholded, and region growing images
plt.figure(figsize=(15, 5))

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

plt.subplot(1, 3, 2)
plt.imshow(thresh_img, cmap='gray')
plt.title('Thresholded Image')

plt.subplot(1, 3, 3)
plt.imshow(region_grown_img, cmap='gray')
```

```
plt.title('Region Growing Image')
plt.show()
```

OUTPUT:



COLAB NOTEBOOK LINK:

https://colab.research.google.com/drive/1QThx-6z5IP0Ux9SDSUZPih0YkHE_zjPl?usp=sharing

MACHINE VISION LAB 7 AND LAB 8

COLAB NOTEBOOK LINK:

[https://colab.research.google.com/drive/1KLBZ139Cd7m0XIt4JCfKxF__rQf9TCfj?usp=sharing]

NAME : OM SUBRATO DEY

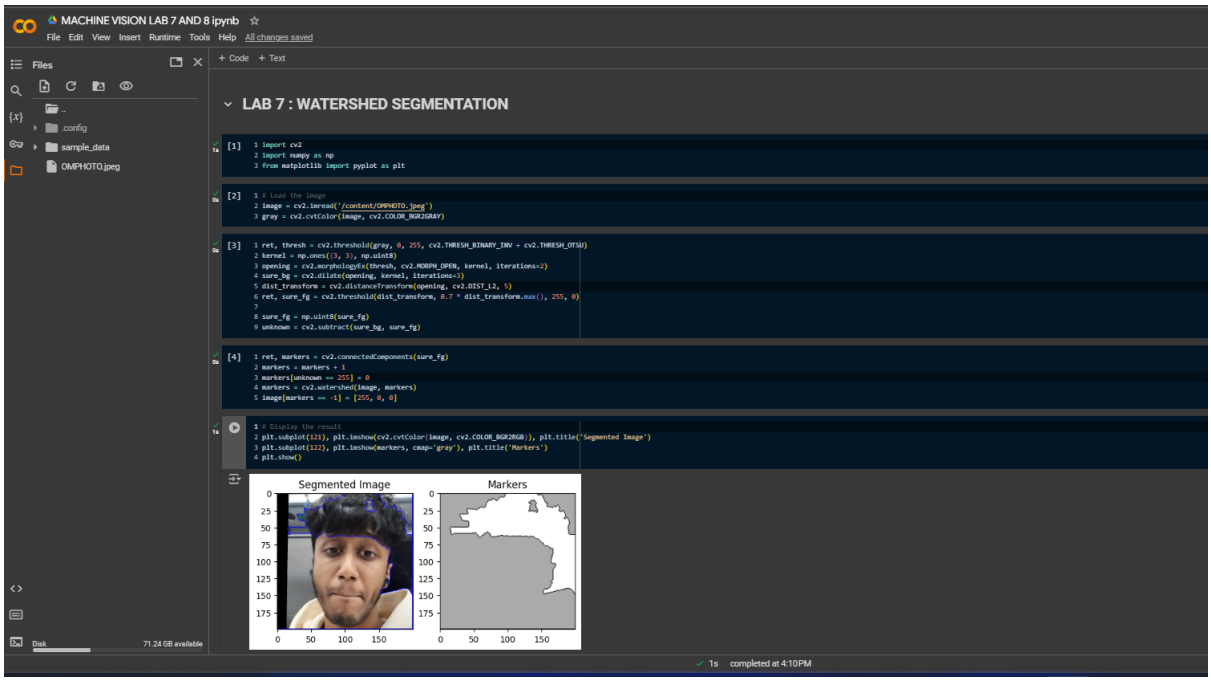
REGISTER NO. 21BAI1876

LAB 7: WATERSHED SEGMENTATION

CODE:

```
import cv2

import numpy as np

from matplotlib import pyplot as plt


# Load the image

image = cv2.imread('/content/OMPHOTO.jpeg')

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


ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

kernel = np.ones((3, 3), np.uint8)

opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)

sure_bg = cv2.dilate(opening, kernel, iterations=3)

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)

unknown = cv2.subtract(sure_bg, sure_fg)


ret, markers = cv2.connectedComponents(sure_fg)

markers = markers + 1

markers[unknown == 255] = 0

markers = cv2.watershed(image, markers)

image[markers == -1] = [255, 0, 0]


# Display the result

plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)),
plt.title('Segmented Image')

plt.subplot(122), plt.imshow(markers, cmap='gray'), plt.title('Markers')

plt.show()
```

OUTPUT:


MACHINE VISION LAB 7 AND 8.ipynb
File Edit View Insert Runtime Tools Help All changes saved
Files
.config
sample_data
OMPHOTO.jpeg
Disk 71.24 GB available
+ Code + Text
LAB 7 : WATERSHED SEGMENTATION
[1] 1 import cv2
2 import numpy as np
3 from matplotlib import pyplot as plt
[2] 1 # Load the image
2 image = cv2.imread('/content/OMPHOTO.jpeg')
3 gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
[3] 1 ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)
2 kernel = np.ones((3, 3), np.uint8)
3 opening = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel, iterations=2)
4 sure_bg = cv2.dilate(opening, kernel, iterations=3)
5 dist_transform = cv2.distanceTransform(opening, cv2.DIST_L2, 5)
6 ret, sure_fg = cv2.threshold(dist_transform, 0.7 * dist_transform.max(), 255, 0)
7
8 sure_fg = np.uint8(sure_fg)
9 unknown = cv2.subtract(sure_bg, sure_fg)
[4] 1 ret, markers = cv2.connectedComponents(sure_fg)
2 markers = markers + 1
3 markers[unknown == 255] = 0
4 markers = cv2.watershed(image, markers)
5 image[markers == -1] = [255, 0, 0]
1 # Display the result
2 plt.subplot(121), plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB)), plt.title('Segmented Image')
3 plt.subplot(122), plt.imshow(markers, cmap='gray'), plt.title('Markers')
4 plt.show()
Segmented Image
Markers
1s completed at 4:10PM

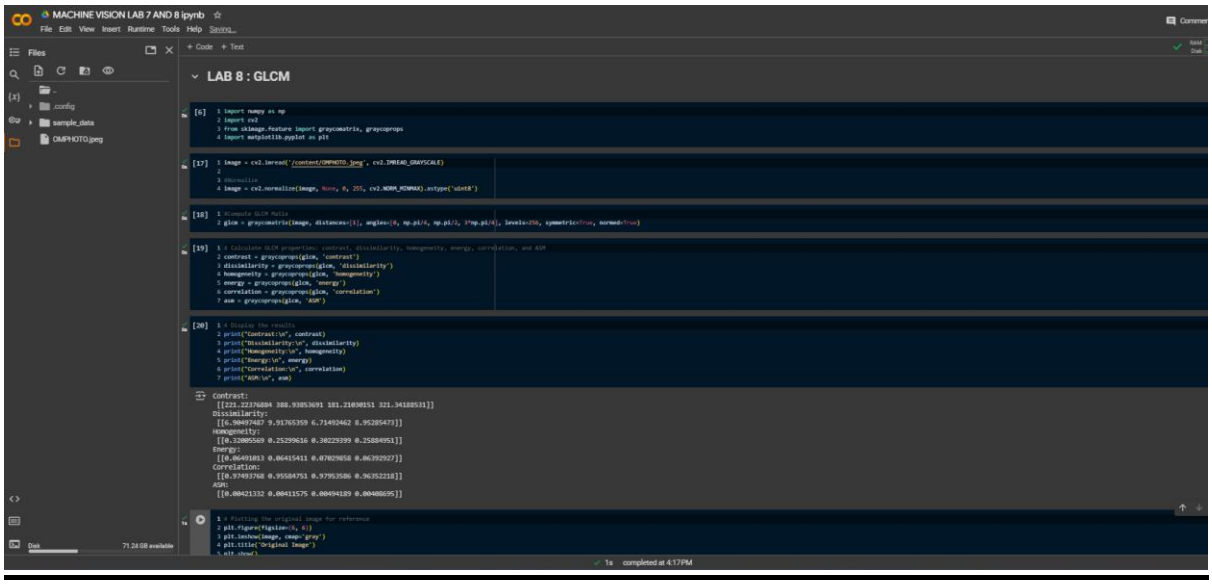
LAB 8: GLCM (Gray Level Co-occurrence Matrix)

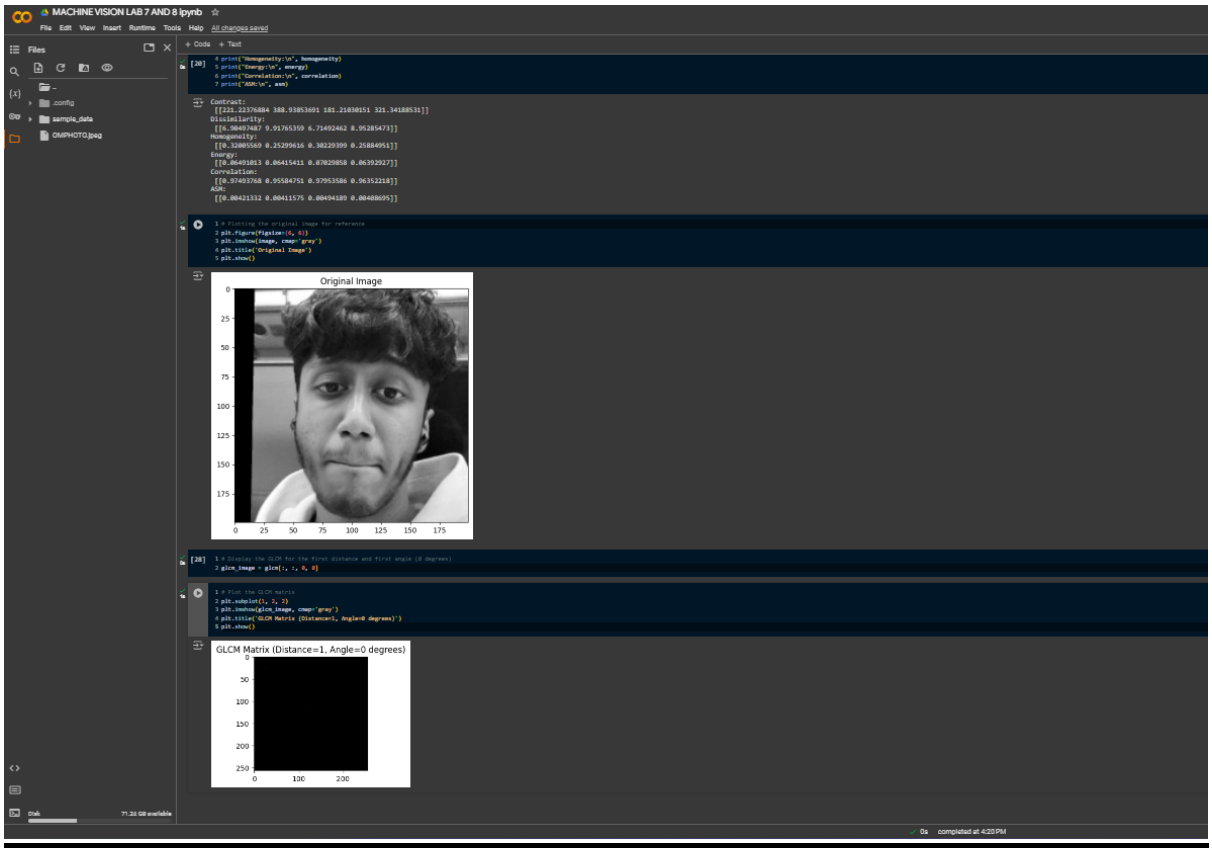
CODE:

```
import numpy as np

import cv2

from skimage.feature import graycomatrix, graycoprops

import matplotlib.pyplot as plt


image = cv2.imread('/content/OMPHOTO.jpeg', cv2.IMREAD_GRAYSCALE)


#Normalize

image = cv2.normalize(image, None, 0, 255, cv2.NORM_MINMAX).astype('uint8')


#Compute GLCM Matix

glcm = graycomatrix(image, distances=[1], angles=[0, np.pi/4, np.pi/2,
3*np.pi/4], levels=256, symmetric=True, normed=True)


# Calculate GLCM properties: contrast, dissimilarity, homogeneity, energy,
correlation, and ASM

contrast = graycoprops(glcm, 'contrast')

dissimilarity = graycoprops(glcm, 'dissimilarity')

homogeneity = graycoprops(glcm, 'homogeneity')

energy = graycoprops(glcm, 'energy')

correlation = graycoprops(glcm, 'correlation')

asm = graycoprops(glcm, 'ASM')


# Display the results

print("Contrast:\n", contrast)

print("Dissimilarity:\n", dissimilarity)

print("Homogeneity:\n", homogeneity)

print("Energy:\n", energy)

print("Correlation:\n", correlation)

print("ASM:\n", asm)
```

```
# Plotting the original image for reference
plt.figure(figsize=(6, 6))
plt.imshow(image, cmap='gray')
plt.title('Original Image')
plt.show()

# Display the GLCM for the first distance and first angle (0 degrees)
glcm_image = glcm[:, :, 0, 0]

# Plot the GLCM matrix
plt.subplot(1, 2, 2)
plt.imshow(glcm_image, cmap='gray')
plt.title('GLCM Matrix (Distance=1, Angle=0 degrees)')
plt.show()
```

OUTPUT:


MACHINE VISION LAB 7 AND 8.ipynb
LAB 8 : GLCM
Contrast:
[[221.22376884 388.93053691 181.21030151 321.34188531]]
Dissimilarity:
[[6.90497487 9.91765359 6.71492462 8.95285473]]
Homogeneity:
[[0.32005569 0.25299616 0.38229399 0.25884951]]
Energy:
[[0.06491813 0.06415411 0.07029858 0.06392927]]
Correlation:
[[0.97493768 0.95584751 0.97953586 0.96352218]]
ASM:
[[0.00421332 0.00411575 0.00494189 0.00408695]]


MACHINE VISION LAB 7 AND 8.ipynb
Original Image
GLCM Matrix (Distance=1, Angle=0 degrees)