Image processing & Machine Vision Assignment Report

Assignment - I

MPSM Pathirana

D/ENG/22/0061/EE

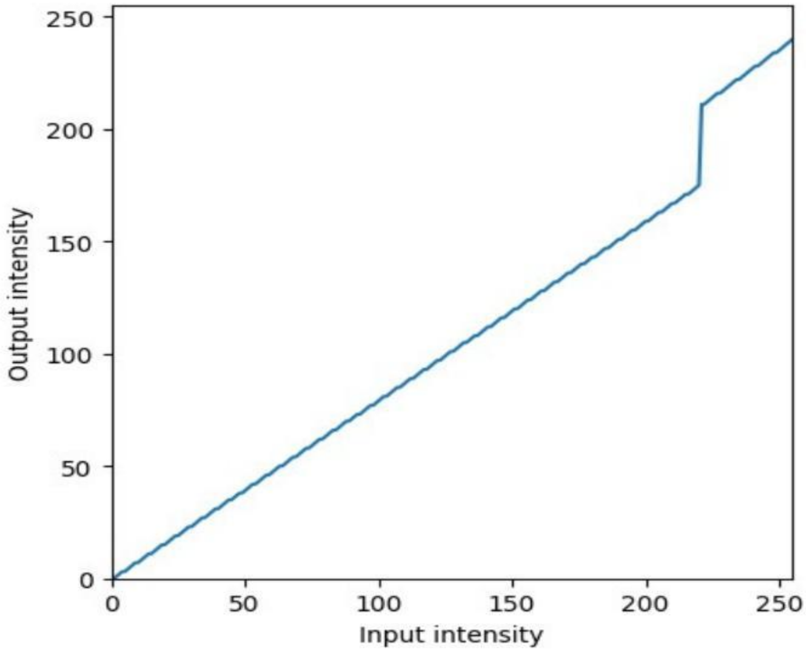
Department of Electrical, Electronic and Telecommunication Engineering

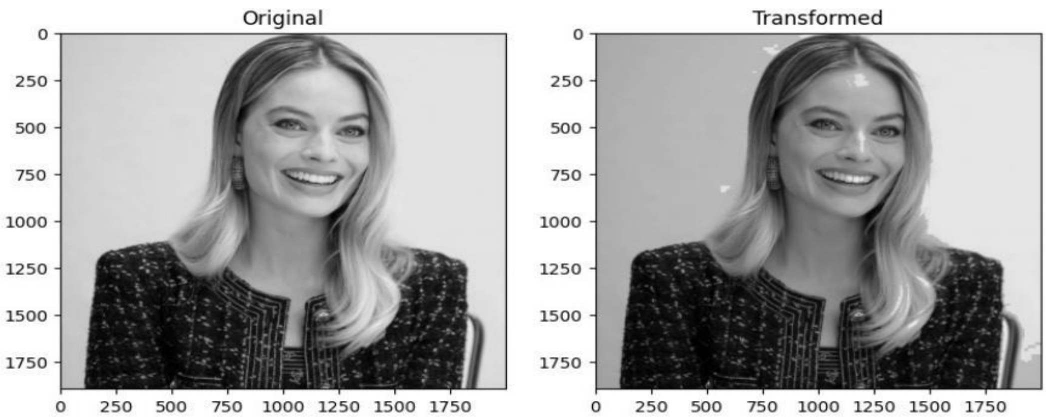
Content

Question 01

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

c = np.array([(220, 175), (220, 210), (255, 240)])

t1 = np.linspace(0, c[0, 1], c[0, 0]+1).astype('uint8')
print(len(t1))
t2 = np.linspace(c[0, 1] + 1, c[1, 0]-c[0, 0], c[1, 0]-c[0,
0]).astype('uint8')
print(len(t2))
t3 = np.linspace(c[1, 1] + 1, 240, 255 - c[1, 0]).astype('uint8')
print(len(t3))

transform = np.concatenate((t1, t2, t3), axis=0).astype('uint8')
print(len(transform))

fig, ax = plt.subplots()
ax.plot(transform)
ax.set_xlabel(r'Input intensity')
ax.set_ylabel('Output intensity')
ax.set_xlim(0, 255)
ax.set_ylim(0, 255)
ax.set_aspect('equal')
plt.show()

img_orig = cv.imread('margot_golden_gray.jpg', cv.IMREAD_GRAYSCALE)
image_transformed = cv.LUT(img_orig, transform)

fig, ax = plt.subplots(1, 2, figsize=(10, 20))
ax[0].imshow(img_orig, cmap="gray")
ax[0].set_title("Original")
ax[1].imshow(image_transformed, cmap="gray")
ax[1].set_title("Transformed")
plt.show()
```

```
221
0
35
256
```





Discussion

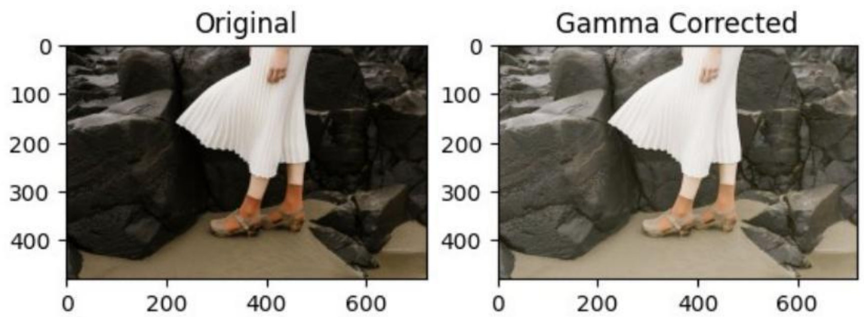
Intensity transformation plays a crucial role in adjusting the overall brightness, contrast, and other adjustment of the images. Also, intensity transformation is the fundamental process in image processing that can modifying the pixel values. I attached the code and results.

Question 02

a.

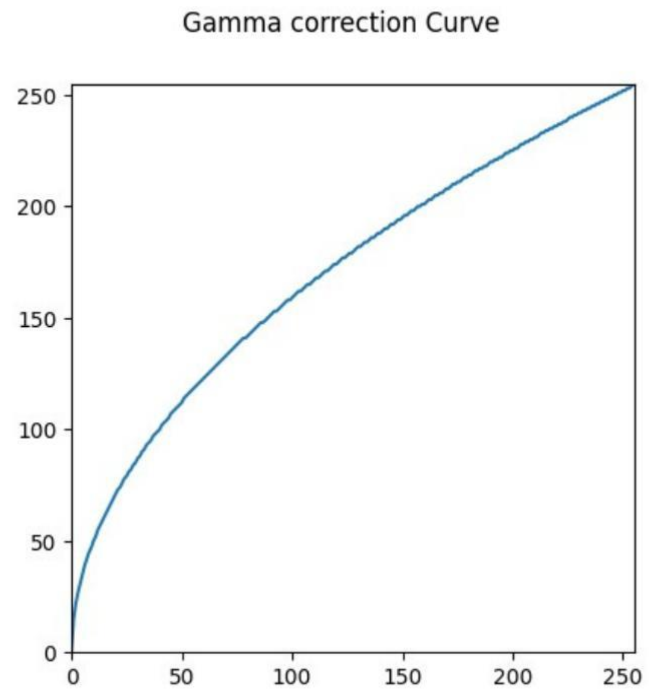
```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img_orig = cv.imread('highlights_and_shadows.jpg', cv.IMREAD_COLOR)

gamma = .5

table = np.array(
    [(i/255.0)**(gamma)*255.0 for i in np.arange(0,
256)]).astype('uint8')

img_gamma = cv.LUT(img_orig, table)
img_orig = cv.cvtColor(img_orig, cv.COLOR_BGR2RGB)
img_gamma = cv.cvtColor(img_gamma, cv.COLOR_BGR2RGB)

f, axarr = plt.subplots(1, 2)
axarr[0].imshow(img_orig)
axarr[0].set_title('Original')
axarr[1].imshow(img_gamma)
axarr[1].set_title('Gamma Corrected')
```

```
Text(0.5, 1.0, 'Gamma Corrected')
```




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

img_orig = cv.imread('highlights_and_shadows.jpg', cv.IMREAD_COLOR)

gamma = .5

table = np.array(
    [(i/255.0)**(gamma)*255.0 for i in np.arange(0,
256)]).astype('uint8')

img_gamma = cv.LUT(img_orig, table)
```

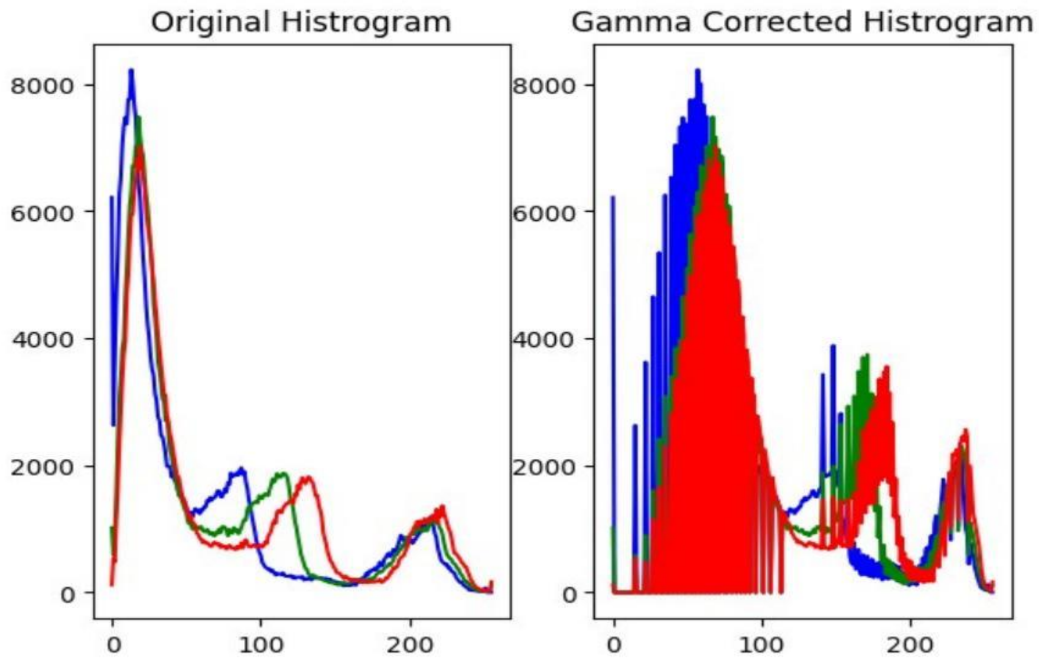
```
img_orig = cv.cvtColor(img_orig, cv.COLOR_BGR2RGB)
img_gamma = cv.cvtColor(img_gamma, cv.COLOR_BGR2RGB)

fig, axarr = plt.subplots()
axarr.plot(table)
axarr.set_xlim(0, 255)
axarr.set_ylim(0, 255)
axarr.set_aspect('equal')

plt.suptitle("Gamma correction Value Curve")
plt.show()
```



b.

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

img = cv.imread('highlights_and_shadows.jpg', cv.IMREAD_COLOR)

gamma = .5

table = np.array(
    [(i/255.0)**(gamma)*255.0 for i in np.arange(0,
256)]).astype('uint8')
```

```python
img_gamma = cv.LUT(img_orig, table)
img_orig = cv.cvtColor(img_orig, cv.COLOR_BGR2RGB)
img_gamma = cv.cvtColor(img_gamma, cv.COLOR_BGR2RGB)

f, axarr = plt.subplots(1, 2)
color = ('b', 'g', 'r')

for i, c in enumerate(color):
    hist_orig = cv.calcHist([img_orig], [i], None, [256], [0, 256])
    axarr[0].plot(hist_orig, color=c)
    axarr[0].set_title('Original Histrogram')
    hist_gamma = cv.calcHist([img_gamma], [i], None, [256], [0, 256])
    axarr[1].plot(hist_gamma, color=c)
    axarr[1].set_title('Gamma Corrected Histrogram')
```



Discussion

Histograms show a visual representation of the distribution of pixel intensities in an image. Original Histogram color spaces reflects the pixel intensities in the L channel. Indicate how common each intensities level is within the image. After applying gamma correction alter the relationship between the input and output pixel intensities this one affected to overall brightness and contrast.

Question 03

a.

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

def f(x, a, sigma):
    return np.minimum(x + a * 128 * np.exp(-(x - 128) ** 2 / (2 *
sigma ** 2)), 255)

image = cv.imread("spider.png")
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

saturation_plane = hsv_image[:, :, 1]

a = 0.5
sigma = 70

modified_saturation = f(saturation_plane, a, sigma)

hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)
modified_image = cv.cvtColor(hsv_image, cv.COLOR_HSV2BGR)
hue_plane, saturation_plane, value_plane =
cv.split(cv.cvtColor(modified_image, cv.COLOR_BGR2HSV))

fig, ax = plt.subplots(1, 3, figsize=(10,5), sharey = True)

ax[0].imshow(hue_plane, cmap='gray')
ax[0].set_title('Hue Plane')
ax[0].axis('off')

ax[1].imshow(saturation_plane, cmap='gray')
ax[1].set_title('Saturation Plane')
ax[1].axis('off')

ax[2].imshow(value_plane, cmap='gray')
ax[2].set_title('Value Plane')
ax[2].axis('off')
```

```
(-0.5, 779.5, 437.5, -0.5)
```

b.

```python
import cv as cv
import numpy as np
```

```python
import matplotlib.pyplot as plt

def f(x, a, sigma):
    return np.minimum(x + a * 128 * np.exp(-(x - 128) ** 2 / (2 *
sigma ** 2)), 255)

image = cv.imread("spider.png")

hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
saturation_plane = hsv_image[:, :, 1]

a = 0.5
sigma = 70

modified_saturation = f(saturation_plane, a, sigma)

hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)
modified_image = cv.cvtColor(hsv_image, cv.COLOR_HSV2BGR)

fig, ax = plt.subplots(1, 2, figsize=(10, 20))
ax[0].imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[0].axis('off')
ax[1].imshow(cv.cvtColor(modified_image, cv.COLOR_BGR2RGB))
ax[1].set_title('Modified Image')
ax[1].axis('off')
plt.show()
```

c.

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

def f(x, a, sigma):
    return np.minimum(x + a * 128 * np.exp(-(x - 128) ** 2 / (2 *
sigma ** 2)), 255)

image = cv.imread("spider.png")
```

```
hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)
modified_image = cv.merge([hue_plane, hsv_image[:, :, 1],
value_plane])
final_modified_image = cv.cvtColor(modified_image, cv.COLOR_HSV2BGR)

fig, ax = plt.subplots(1, 2, figsize=(10, 20))
ax[0].imshow(cv.cvtColor(image, cv.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[0].axis('off')
ax[1].imshow(cv.cvtColor(final_modified_image, cv.COLOR_BGR2RGB))
ax[1].set_title(f'Final Modified Image (a={a})')
ax[1].axis('off')
plt.show()
```

d.

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

def f(x, a, sigma):
    return np.minimum(x + a * 128 * np.exp(-(x - 128) ** 2 / (2 *
sigma ** 2)), 255)

original_image = cv.imread("spider.png")

hsv_image = cv.cvtColor(original_image, cv.COLOR_BGR2HSV)

saturation_plane = hsv_image[:, :, 1]

vibrance_factor = 1.5
vibrance_enhanced_image = original_image.copy()
vibrance_enhanced_image[:, :, 1] = np.clip(vibrance_factor *
saturation_plane, 0, 255)

a = 0.5
sigma = 70

modified_saturation = f(saturation_plane, a, sigma)
hsv_image[:, :, 1] = modified_saturation.astype(np.uint8)
```

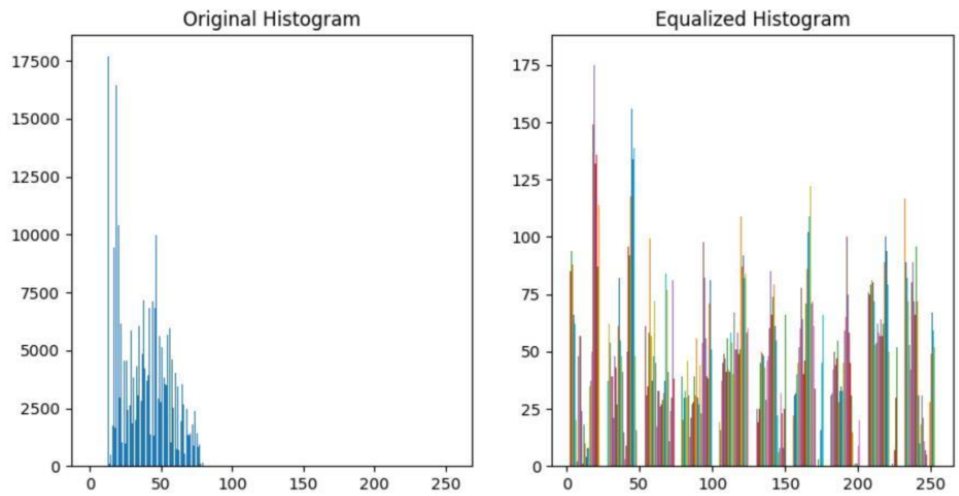
```
intensity_transformed_image = cv.cvtColor(hsv_image, cv.COLOR_HSV2BGR)

fig, ax = plt.subplots(1, 3, figsize=(10,5), sharey = True)

ax[0].imshow(cv.cvtColor(original_image, cv.COLOR_BGR2RGB))
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(cv.cvtColor(vibrance_enhanced_image, cv.COLOR_BGR2RGB))
ax[1].set_title('Vibrance Enhanced')
ax[1].axis('off')

ax[2].imshow(cv.cvtColor(intensity_transformed_image,
cv.COLOR_BGR2RGB)),
ax[2].set_title('Intensity Transformed')
ax[2].axis('off')
plt.show()
```

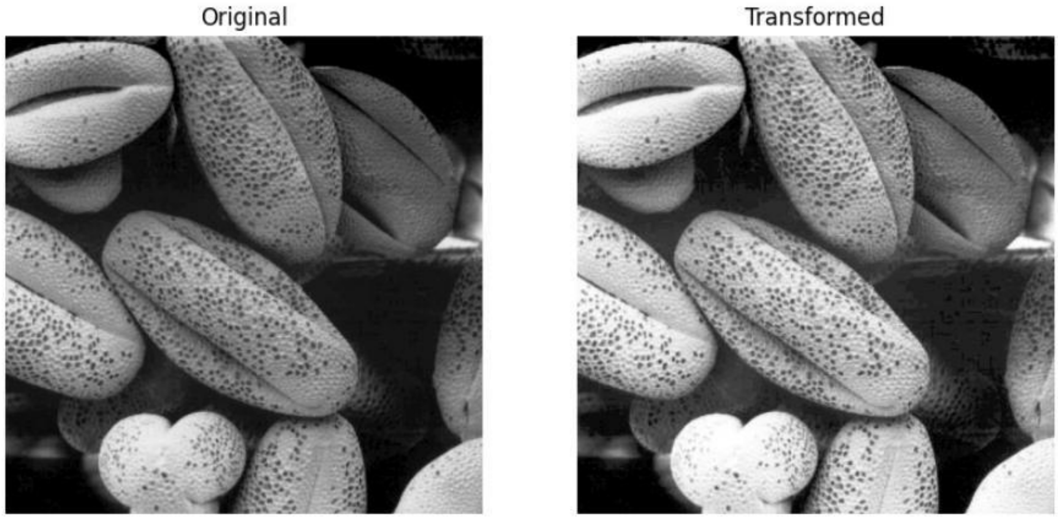
Discussion

Under this question we consider image enhancement process after we increase the vibrance of a image by applying intensity transformation to the saturation, HUE, HSV planes. Considering these three planes we focus directed towards enhancing and bring out more vivid colors. The intensity transformation involves a parameter ‘a’ is adjusted to achieve a pleasing vibrance enhancement.

Question 04

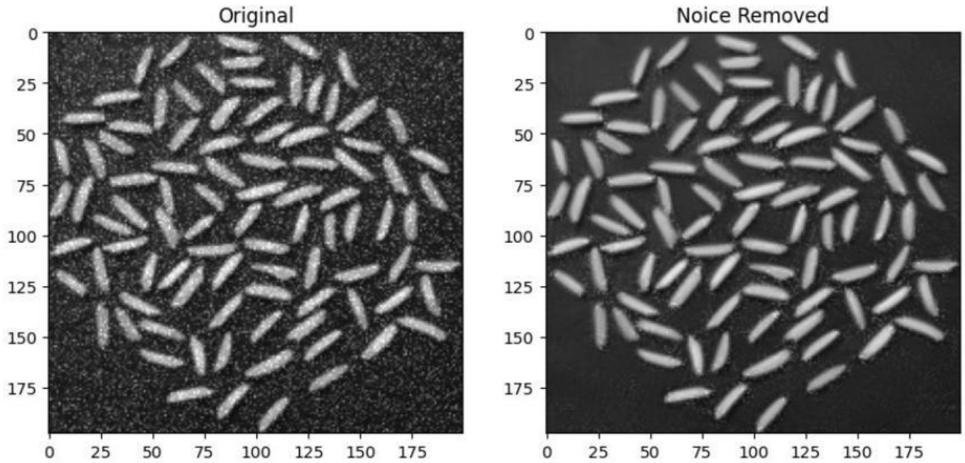
```python
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np

#not changed
im = cv.imread('shells.tif', cv.IMREAD_GRAYSCALE)
assert im is not None

plt.figure(figsize = [10, 5])
plt.subplot(1, 2, 1)
plt.gca().set_title('Original Histogram')
h = np.zeros(256)
h = [np.sum(im==i) for i in range (256)]
plt.bar(range(256), h)

plt.subplot(1, 2, 2)
plt.gca().set_title('Equalized Histogram')
eh = cv.equalizeHist(im)
plt.hist(eh)
plt.show()

fig, ax= plt.subplots(1,2, figsize=(10,20))
ax[0].imshow(im, cmap="gray")
ax[0].set_title('Original')
ax[0].axis('off')

ax[1].imshow(eh, cmap="gray")
ax[1].set_title('Transformed')
ax[1].axis('off')
plt.show()
```



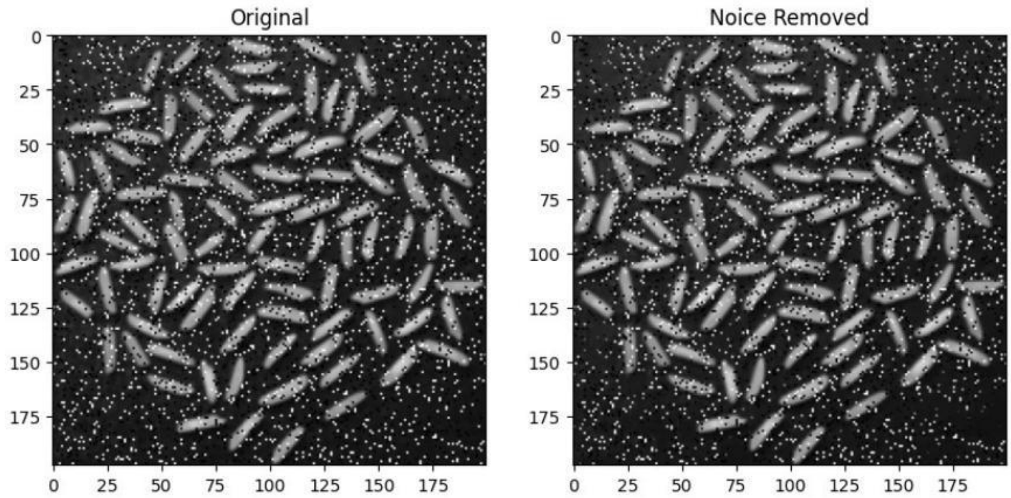

Original
Transformed

Question 05

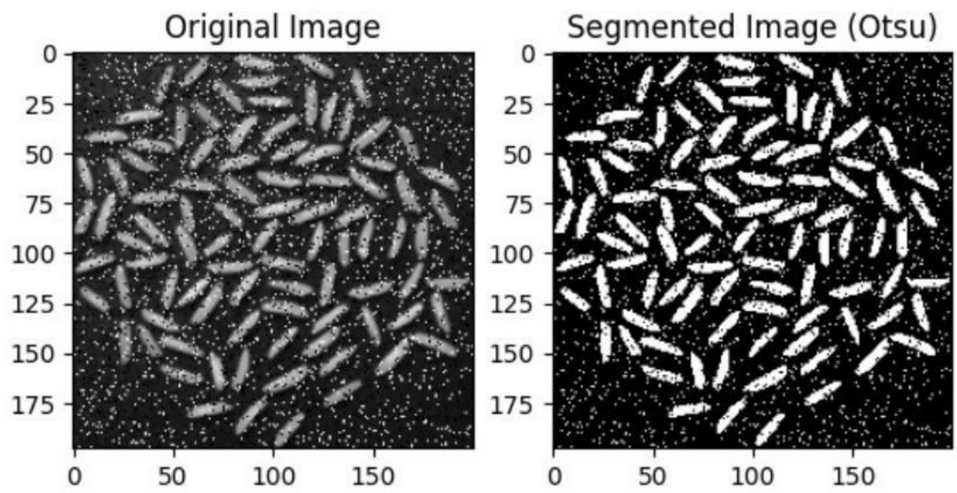
a.
```python
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np

image = cv.imread("rice_gaussian_noise.png")

dst = cv.fastNlMeansDenoisingColored(image, None, 20, 20, 7, 15)

fig, ax = plt.subplots(1, 2, figsize=(10, 20))
ax[0].imshow(image, cmap="gray")
ax[0].set_title("Original")
ax[1].imshow(dst, cmap="gray")
ax[1].set_title("Noice Removed")
plt.show()
```



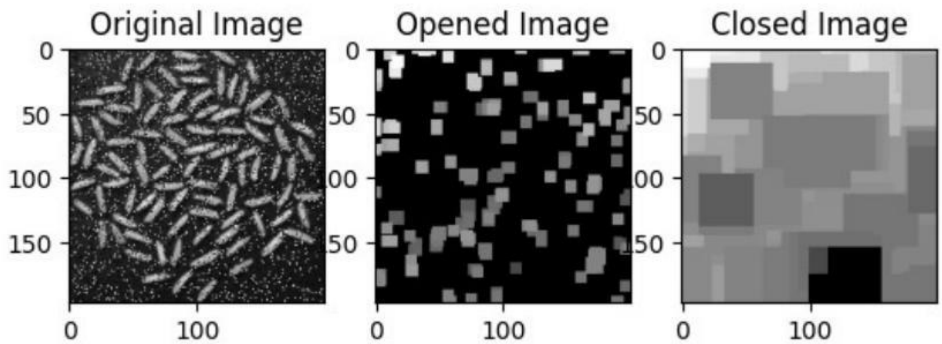
b.
```python
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np

image = cv.imread("rice_salt_pepper_noise.png")

dst = cv.fastNlMeansDenoisingColored(image, None, 20, 20, 7, 15)

fig, ax = plt.subplots(1, 2, figsize=(10, 20))
ax[0].imshow(image, cmap="gray")
ax[0].set_title("Original")
ax[1].imshow(dst, cmap="gray")
ax[1].set_title("Noice Removed")
plt.show()
```



c.
```
import cv2 as cv
import matplotlib.pyplot as plt

# Read the grayscale image
image = cv.imread("rice_salt_pepper_noise.png", cv.IMREAD_GRAYSCALE)

# Apply Otsu's thresholding
_, binary_image = cv.threshold(image, 0, 255, cv.THRESH_BINARY +
cv.THRESH_OTSU)

# Display the original and segmented images
plt.subplot(121), plt.imshow(image, cmap='gray'), plt.title('Original
Image')
plt.subplot(122), plt.imshow(binary_image, cmap='gray'),
plt.title('Segmented Image (Otsu)')
plt.show()
```



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

# Read the binary image
image = cv.imread("rice_salt_pepper_noise.png", cv.IMREAD_GRAYSCALE)

# Apply morphological operations to remove small objects (opening)
kernel_open = np.ones((5, 5), np.uint8)
image_opened = cv.morphologyEx(image, cv.MORPH_OPEN, kernel_open,
iterations=2)

# Apply morphological operations to fill holes (closing)
kernel_close = np.ones((5, 5), np.uint8)
image_closed = cv.morphologyEx(image_opened, cv.MORPH_CLOSE,
kernel_close, iterations=10)

# Display the original, opened, and closed images
plt.subplot(131), plt.imshow(image, cmap='gray'), plt.title('Original
Image')
plt.subplot(132), plt.imshow(image_opened, cmap='gray'),
plt.title('Opened Image')
plt.subplot(133), plt.imshow(image_closed, cmap='gray'),
plt.title('Closed Image')
plt.show()
```



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

im = cv.imread('rice_gaussian_noise.png', cv.IMREAD_GRAYSCALE)

denoised_im = cv.fastNlMeansDenoising(im, None, h=28,
searchWindowSize=10)

_, segmented_image = cv.threshold(denoised_im, 0, 255,
cv.THRESH_BINARY + cv.THRESH_OTSU)
```

```
kernel = cv.getStructuringElement(cv.MORPH_ELLIPSE, (5, 5))

closed_image = cv.morphologyEx(segmented_image, cv.MORPH_CLOSE,
kernel)

opened_image = cv.morphologyEx(closed_image, cv.MORPH_OPEN, kernel)

num_labels, labels = cv.connectedComponents(opened_image)

num_rice_grains = num_labels - 1

print("Number of rice grains:", num_rice_grains)

Number of rice grains: 68
```

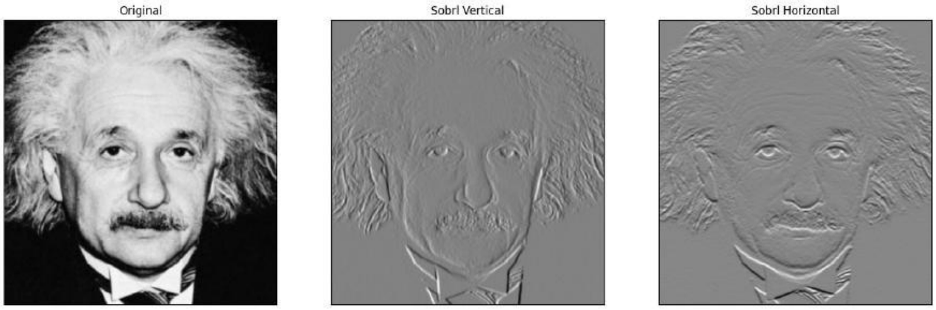
Discussion

Otsu’s method applied to segment the images, disfurnishing rice grains from the background. Morphological operations refine the segmentation by eliminating small artifacts and filling in gaps.

Question 06

a.
```
import cv as cv
import numpy as np
from matplotlib import pyplot as plt

img = cv.imread('einstein.png', cv.IMREAD_GRAYSCALE)

kernal = np.ones((11,11),np.float32)/121
imgc =cv. filter2D(img,-1,kernal)

sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]],
dtype=np.float32)
sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]],
dtype=np.float32)

sobel_x = cv.filter2D(img, cv.CV_64F, sobel_kernel_x)
sobel_y = cv.filter2D(img, cv.CV_64F, sobel_kernel_y)


fig,axes = plt.subplots(1,3, sharex='all', sharey='all',
figsize=(18,18))
axes[0].imshow(img, cmap='gray')
axes[0].set_title('Original')
axes[0].set_xticks([]), axes[0].set_yticks([])

axes[1].imshow(sobel_x, cmap='gray')
axes[1].set_title('Sobrl Vertical')
axes[1].set_xticks([]), axes[1].set_yticks([])

axes[2].imshow(sobel_y, cmap='gray')
axes[2].set_title('Sobrl Horizontal')
axes[2].set_xticks([]), axes[0].set_yticks([])
```

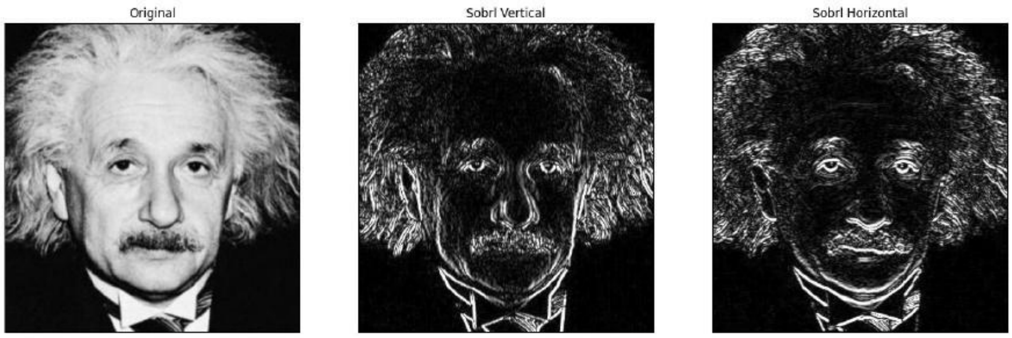
```
([], [])
```



b.
```
import cv2 as cv
import numpy as np
```

```python
# Load an image from file
image = cv.imread('einstein.png', cv.IMREAD_GRAYSCALE)

# Check if the image is loaded successfully
if image is None:
    print("Error: Unable to load the image.")
    exit()

# Define the Sobel filter kernels
sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]],
dtype=np.float32)
sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]],
dtype=np.float32)

# Apply the Sobel filters using filter2D
sobel_x = cv.filter2D(image, cv.CV_64F, sobel_kernel_x)
sobel_y = cv.filter2D(image, cv.CV_64F, sobel_kernel_y)

# Convert the results to uint8 for visualization
sobel_x = cv.convertScaleAbs(sobel_x)
sobel_y = cv.convertScaleAbs(sobel_y)

fig,axes = plt.subplots(1,3, sharex='all', sharey='all',
figsize=(18,18))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original')
axes[0].set_xticks([]), axes[0].set_yticks([])

axes[1].imshow(sobel_x, cmap='gray')
axes[1].set_title('Sobrl Vertical')
axes[1].set_xticks([]), axes[1].set_yticks([])

axes[2].imshow(sobel_y, cmap='gray')
axes[2].set_title('Sobrl Horizontal')
axes[2].set_xticks([]), axes[0].set_yticks([])
```

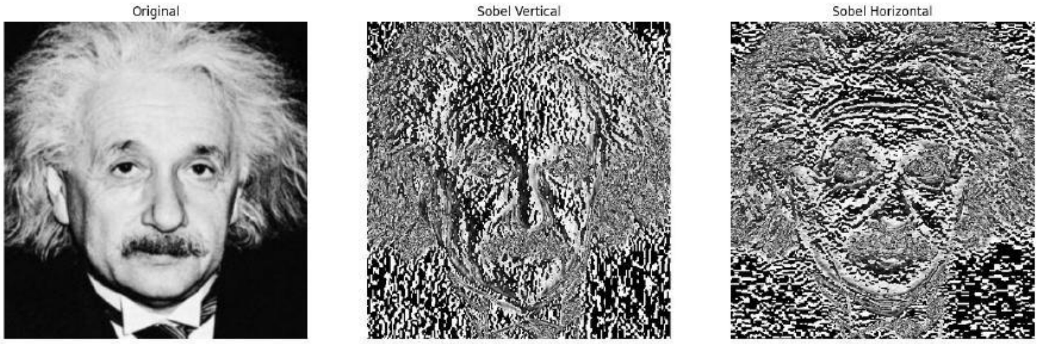
```
([], [])
```



c.

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

def sobel_filter(image):

    sobel_kernel_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    sobel_kernel_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    sobel_x = convolve2d(image, sobel_kernel_x)
    sobel_y = convolve2d(image, sobel_kernel_y)


    gradient_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
    sobel_x = np.abs(sobel_x).astype(np.uint8)
    sobel_y = np.abs(sobel_y).astype(np.uint8)
    gradient_magnitude = gradient_magnitude.astype(np.uint8)

    return sobel_x, sobel_y, gradient_magnitude

def convolve2d(image, kernel):
    height, width = image.shape
    k_height, k_width = kernel.shape

    pad_height = k_height // 2
    pad_width = k_width // 2
    padded_image = np.pad(image, ((pad_height, pad_height),
(pad_width, pad_width)), mode='edge')

    result = np.zeros_like(image)
    for i in range(height):
        for j in range(width):
            result[i, j] = np.sum(padded_image[i:i+k_height,
j:j+k_width] * kernel)

    return result

image = cv.imread('einstein.png', cv.IMREAD_GRAYSCALE)

if len(image.shape) > 2:
    image = cv.cvtColor(image, cv.COLOR_BGR2GRAY)

sobel_x, sobel_y, gradient_magnitude = sobel_filter(image)

fig, axes = plt.subplots(1, 3, sharex='all', sharey='all',
figsize=(18, 18))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original')
axes[0].axis('off')
```

```
axes[1].imshow(sobel_x, cmap='gray')
axes[1].set_title('Sobel Vertical')
axes[1].axis('off')

axes[2].imshow(sobel_y, cmap='gray')
axes[2].set_title('Sobel Horizontal')
axes[2].axis('off')

(-0.5, 363.5, 379.5, -0.5)
```



Discussion

Sobel filtering is a crucial technique in image processing that emphasizes intensity changes for edge detection. In the context of Figure 6, three approaches are employed. First, with the existing filter2D function, Sobel filtering may be completed efficiently and rapidly. Second, a custom Sobel filter implementation offers an interactive understanding of the underlying computations.

Question 07

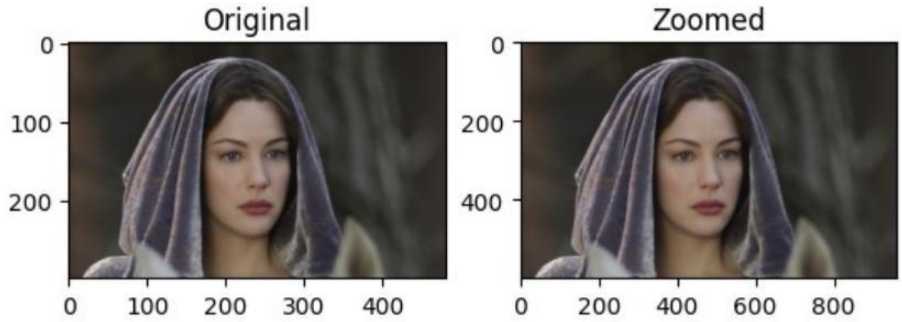
```python
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img = cv.imread('im01small.png')

scale = 4
rows = int(img.shape[0] * scale)
cols = int(img.shape[1] * scale)
```

```python
zoomed = np.zeros((rows, cols, img.shape[2]), dtype=np.uint8)
for i in range(0, rows):
    for j in range(0, cols):
        zoomed[i, j] = img[int(i/scale), int(j/scale)]

fig, ax = plt.subplots(1, 2)
ax[0].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
ax[0].set_title('Original')
ax[1].imshow(cv.cvtColor(zoomed, cv.COLOR_BGR2RGB))
ax[1].set_title('Zoomed')
```

```
Text(0.5, 1.0, 'Zoomed')
```


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

img = cv.imread('im02small.png')

scale = 2
rows = int(img.shape[0] * scale)
cols = int(img.shape[1] * scale)

zoomed = np.zeros((rows, cols, img.shape[2]), dtype=np.uint8)
for i in range(0, rows):
    for j in range(0, cols):
        zoomed[i, j] = img[int(i/scale), int(j/scale)]

fig, ax = plt.subplots(1, 2)
ax[0].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
ax[0].set_title('Original')
ax[1].imshow(cv.cvtColor(zoomed, cv.COLOR_BGR2RGB))
ax[1].set_title('Zoomed')
```

```
Text(0.5, 1.0, 'Zoomed')
```




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

img = cv.imread('im03small.png')

scale = 3
rows = int(img.shape[0] * scale)
cols = int(img.shape[1] * scale)

zoomed = np.zeros((rows, cols, img.shape[2]), dtype=np.uint8)
for i in range(0, rows):
    for j in range(0, cols):
        zoomed[i, j] = img[int(i/scale), int(j/scale)]

fig, ax = plt.subplots(1, 2)
ax[0].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
ax[0].set_title('Original')
ax[1].imshow(cv.cvtColor(zoomed, cv.COLOR_BGR2RGB))
ax[1].set_title('Zoomed')
```

```
Text(0.5, 1.0, 'Zoomed')
```

Question 08

a.

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

image = cv.imread('daisy.jpg')

mask = np.zeros(image.shape[:2], np.uint8)
background = np.zeros((1,65), np.float64)

rect = (20, 20, 550, 550)

cv.grabCut(image, mask, rect, None, None, 5, cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask == cv.GC_FGD) | (mask == cv.GC_PR_FGD), 1,
0).astype('uint8')

foreground = cv.bitwise_and(image, image, mask=mask2)
background = cv.bitwise_and(image, image, mask=1 - mask2)

segmentation_mask = np.where(mask2[:, :, np.newaxis] == 1, 255,
0).astype('uint8')

fig, ax = plt.subplots(1, 4, figsize=(10,10), sharey = True)

ax[0].imshow(image[:,:,::-1]),
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(mask)
ax[1].set_title('Segmentation Mask')
ax[1].axis('off')

ax[2].imshow(background[:,:,::-1]),
ax[2].set_title('Background Image')
ax[2].axis('off')

ax[3].imshow(foreground[:,:,::-1])
ax[3].set_title('Foreground Image')
ax[3].axis('off')

(-0.5, 560.5, 840.5, -0.5)
```

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

image = cv.imread('daisy.jpg')

mask = np.zeros(image.shape[:2], np.uint8)
background = np.zeros((1,65), np.float64)

rect = (20, 20, 550, 550)

cv.grabCut(image, mask, rect, None, None, 5, cv.GC_INIT_WITH_RECT)

mask2 = np.where((mask == cv.GC_FGD) | (mask == cv.GC_PR_FGD), 1,
0).astype('uint8')

foreground = cv.bitwise_and(image, image, mask=mask2)
background = cv.bitwise_and(image, image, mask=1 - mask2)
segmentation_mask = np.where(mask2[:, :, np.newaxis] == 1, 255,
0).astype('uint8')

blurred_bg = cv.GaussianBlur(background, (31, 31), 0)
enhanced_img = cv.addWeighted(foreground, 1, blurred_bg, 0.8, 0)

fig, ax = plt.subplots(1, 2, figsize=(12,6), sharey = True)

ax[0].imshow(image[:,:,::-1])
ax[0].set_title('Original Image')
ax[0].axis('off')

ax[1].imshow(enhanced_img[:,:,::-1])
ax[1].set_title('Enhanced Image')
ax[1].axis('off')

(-0.5, 560.5, 840.5, -0.5)
```

c. The improved image's darker backdrop, which extends over the margin of the flower, is mostly the outcome of a Gaussian blur applied to the background. The image is first divided into foreground (flower) and background using Grab Cut. The background is then smoothed using a Gaussian blur with a (15, 15) kernel. The backdrop appears darker because of this smoothing effect, which averages pixel values. The final improved image is produced by combining the sharp foreground with the blurred backdrop. The degree of blurring and the ensuing darkness in the backdrop can be altered by varying certain parameters, such as the kernel size.

GitHub Link

https://github.com/Sandeepa0/Image-Processing.git