EE7204: COMPUTER VISION AND IMAGE PROCESSING

TAKE HOME ASSIGNMENT - 02

NAME	: LAKSHAN D.P.W.K.
REG. NO	: EG/2019/3649
SEMESTER	: 07
DATE	: 21/04/2024

Contents

List of Figures

1 GitHub Link

https://github.com/KalinduLakshan/EE7204_EG_2019_3649_TakeHomeAssignment_02.git

2 Coding Answers

2.1 Consider an image with 2 objects and a total of 3-pixel values (1 for each object and one for the background). Add Gaussian noise to the image. Implement and test Otsu's algorithm with this image.

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


def add_gaussian_noise(img, mean=0, sigma=25):
    """Adds Gaussian noise to an image."""
    gauss = np.random.normal(mean, sigma, img.shape).astype(np.float32)
    noisy_img = cv.add(img.astype(np.float32), gauss)
    noisy_img = np.clip(noisy_img, 0, 255).astype(np.uint8)
    return noisy_img


def apply_otsus_threshold(img):
    """Applies Otsu's thresholding to an image."""
    _, thresh_img = cv.threshold(img, 0, 255, cv.THRESH_BINARY +
cv.THRESH_OTSU)
    return thresh_img


# Create an image with a black background
height, width = 200, 200
image = np.zeros((height, width), dtype=np.uint8)

# Draw a gray rectangle
cv.rectangle(image, (50, 50), (150, 120), 100, -1)  # Gray value

# Draw a white square
cv.rectangle(image, (70, 130), (130, 190), 200, -1)  # White value

# Generate a noisy version of the image
noisy_image = add_gaussian_noise(image)

# Apply Otsu's thresholding
thresholded_image = apply_otsus_threshold(noisy_image)

# Display the results
fig, axes = plt.subplots(1, 3, figsize=(15, 5))
axes[0].imshow(image, cmap='gray')
axes[0].set_title('Original Image')
axes[0].axis('off')

axes[1].imshow(noisy_image, cmap='gray')
axes[1].set_title('Noisy Image')
```

```
axes[1].axis('off')

axes[2].imshow(thresholded_image, cmap='gray')
axes[2].set_title('Thresholded Image')
axes[2].axis('off')

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

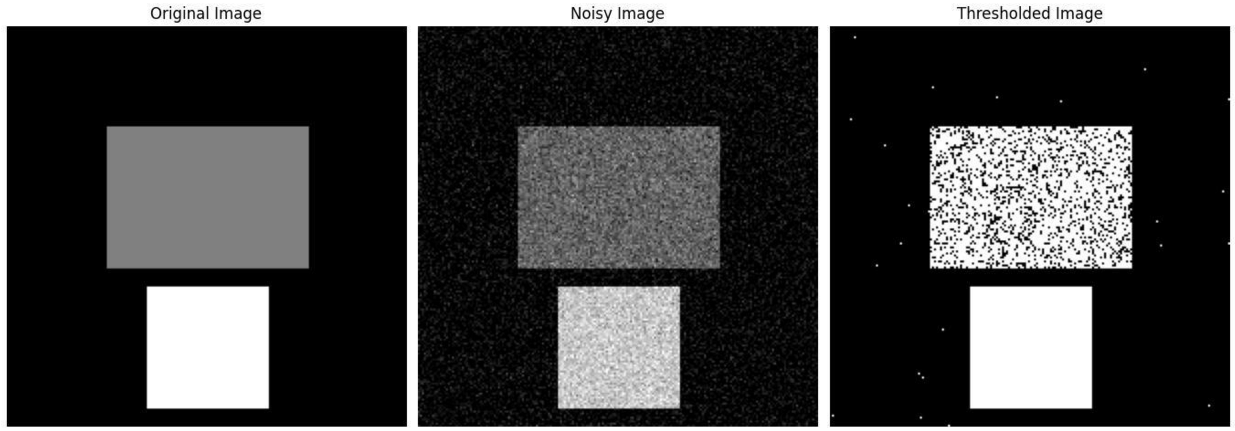
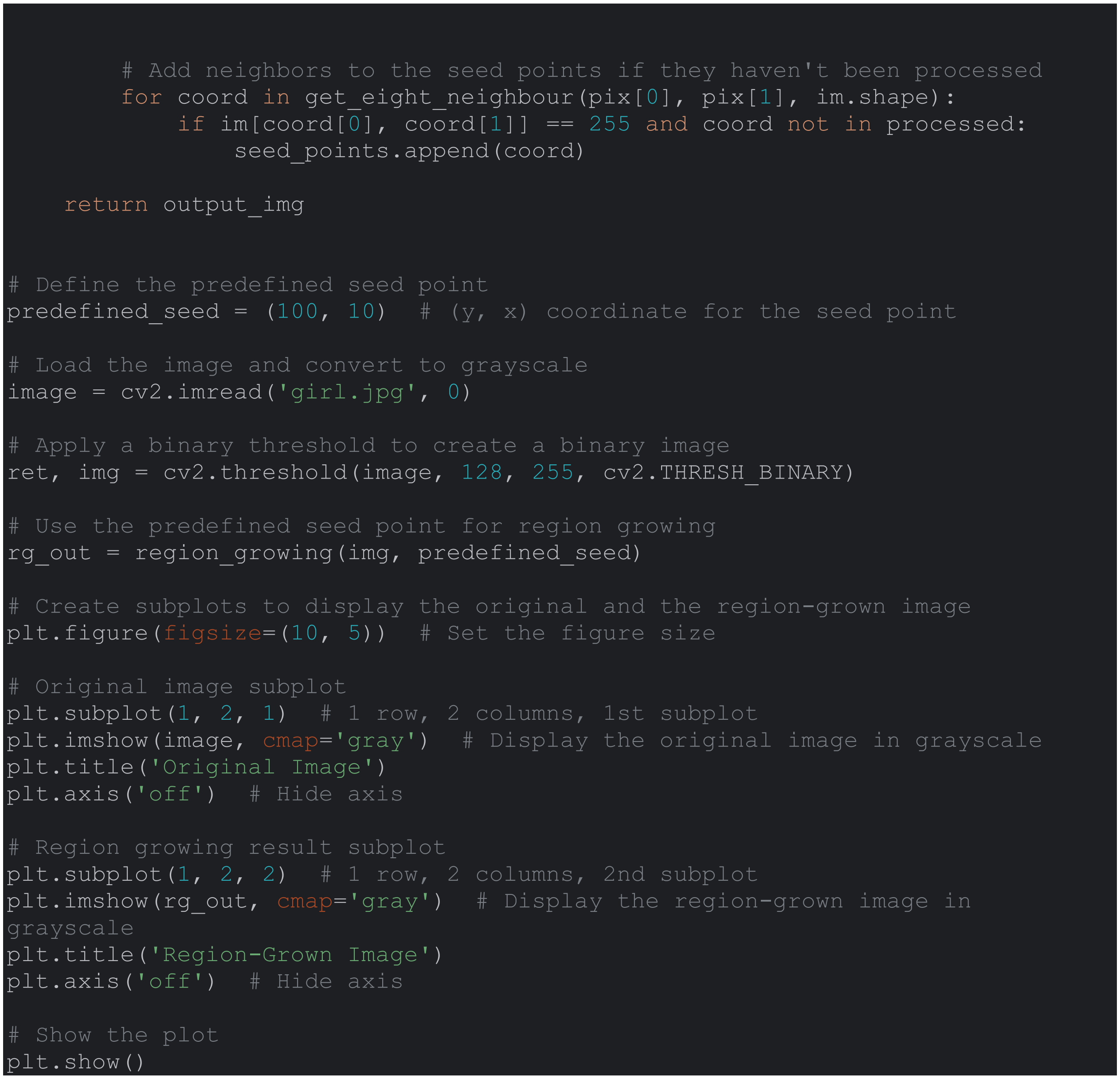


FIGURE 2.1: ORIGINAL IMAGE, GAUSSIAN NOISE ADDED IMAGE AND IMAGE AFTER PERFORMING OTSU'S ALGORITHM

2.2 Implement a region-growing technique for image segmentation. The basic idea is to start from a set of points inside the object of interest (foreground), denoted as seeds, and recursively add neighboring pixels as long as they are in a pre-defined range of the pixel values of the seeds.

```
# Import the necessary libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt


def get_eight_neighbour(x, y, shape):
    # Returns the coordinates of the 8-neighbors of a point within a given
shape
    output = []
    max_x = shape[1] - 1
    max_y = shape[0] - 1

    # Define the eight possible neighbors and constrain them within the image
boundaries.
    neighbors = [
        (x - 1, y - 1), (x, y - 1), (x + 1, y - 1),  # Top row
        (x - 1, y),                     (x + 1, y),   # Middle row (left and
right)
        (x - 1, y + 1), (x, y + 1), (x + 1, y + 1)  # Bottom row
    ]

    for nx, ny in neighbors:
        # Constrain the coordinates within the image boundaries
        output_x = min(max(nx, 0), max_x)
        output_y = min(max(ny, 0), max_y)
        output.append((output_x, output_y))

    return output


def region_growing(im, seed):
    # Performs region growing from a given seed point
    seed_points = [seed]  # Start with the initial seed point
    output_img = np.zeros_like(im)  # Output image initialized to zeros
(black)
    processed = set()  # A set to track processed points

    while seed_points:
        pix = seed_points.pop(0)  # Get the next seed point to process
        if pix in processed:
            continue  # If already processed, skip

        output_img[pix[0], pix[1]] = 255  # Mark the point as white in the
output image
        processed.add(pix)  # Add to the set of processed points
```



```
        # Add neighbors to the seed points if they haven't been processed
        for coord in get_eight_neighbour(pix[0], pix[1], im.shape):
            if im[coord[0], coord[1]] == 255 and coord not in processed:
                seed_points.append(coord)

    return output_img


# Define the predefined seed point
predefined_seed = (100, 10)  # (y, x) coordinate for the seed point

# Load the image and convert to grayscale
image = cv2.imread('girl.jpg', 0)

# Apply a binary threshold to create a binary image
ret, img = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)

# Use the predefined seed point for region growing
rg_out = region_growing(img, predefined_seed)

# Create subplots to display the original and the region-grown image
plt.figure(figsize=(10, 5))  # Set the figure size

# Original image subplot
plt.subplot(1, 2, 1)  # 1 row, 2 columns, 1st subplot
plt.imshow(image, cmap='gray')  # Display the original image in grayscale
plt.title('Original Image')
plt.axis('off')  # Hide axis

# Region growing result subplot
plt.subplot(1, 2, 2)  # 1 row, 2 columns, 2nd subplot
plt.imshow(rg_out, cmap='gray')  # Display the region-grown image in
grayscale
plt.title('Region-Grown Image')
plt.axis('off')  # Hide axis

# Show the plot
plt.show()
```

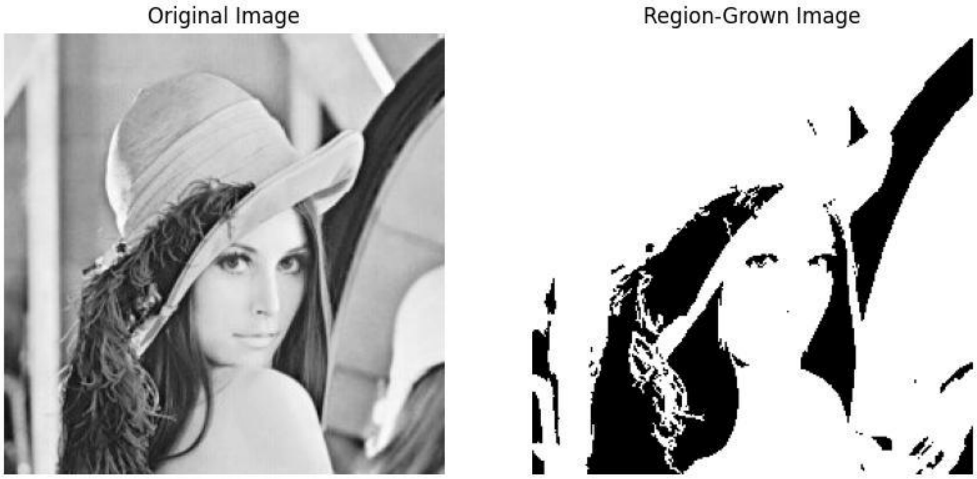


FIGURE 2.2: ORIGINAL IMAGE AND THE REGIONAL-GROWN IMAGE