EC7212 – COMPUTER VISION AND IMAGE

PROCESSING

ASSIGNMENT 02

NAME : SOVIS W. F. S. V

REG NO : EG/2020/4222

SEMESTER : 07

DATE : 27/06/2025

# Questions 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.

def add\_gaussian\_noise(image, mean=0, stddev=10):

    noise = np.random.normal(mean, stddev, image.shape) #Add Gaussian noise to a grayscale image.

    noisy\_image = np.clip(image + noise, 0, 255).astype(np.uint8)

    return noisy\_image

def apply\_otsu\_thresholding(image):

    thresh = threshold\_otsu(image) #Apply Otsu's thresholding on a grayscale image

    binary = (image >= thresh).astype(np.uint8) \* 255

    return binary, thresh

Figure1.1: Functions of Gaussian noise and Otsu thresholding

# Step 1: Create synthetic image with 2 objects and background

original = np.zeros((100, 100), dtype=np.uint8)

original[30:45, 30:45] = 75   # Object 1

original[50:70, 50:70] = 185    # Object 2

# Step 2: Add Gaussian noise

noisy = add\_gaussian\_noise(original)

# Step 3: Apply Otsu’s thresholding

binary\_otsu, otsu\_value = apply\_otsu\_thresholding(noisy)

Figure 1.2: Create synthetic image and apply to Gaussian and Otsu functions

plt.title("Original Synthetic Image")

plt.imshow(original, cmap='gray')

plt.axis('off')

plt.show()

plt.title("Noisy Image (Gaussian)")

plt.imshow(noisy, cmap='gray')

plt.axis('off')

plt.show()

plt.title(f"Otsu Thresholding (T={otsu\_value})")

plt.imshow(binary\_otsu, cmap='gray')

plt.axis('off')

plt.show()

Figure 1.3: Plot the outputs

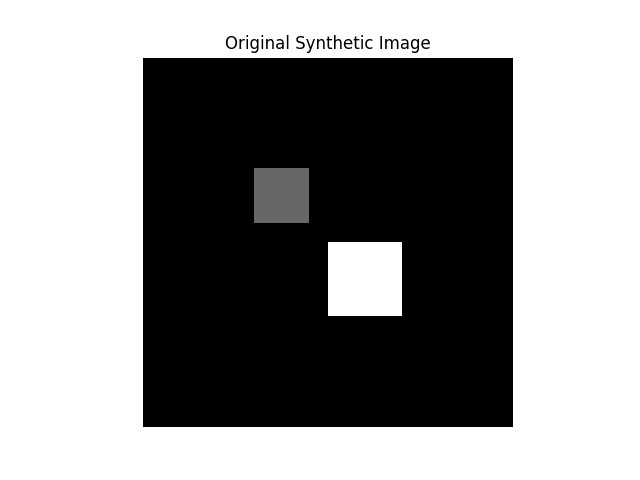


Figure 1. 4: Original synthetic image

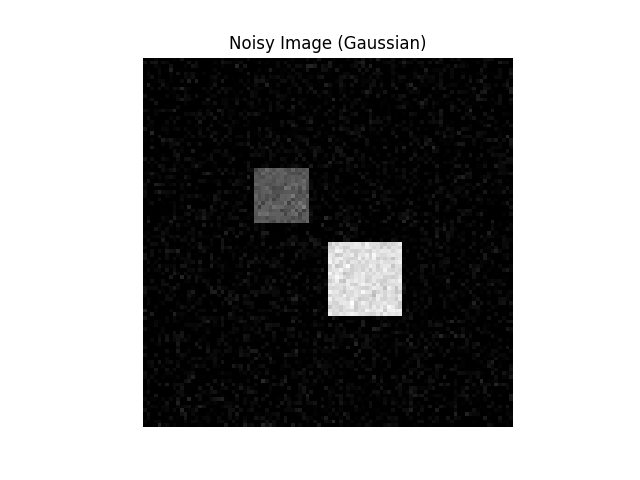


Figure 1.5: Gaussian noisy image

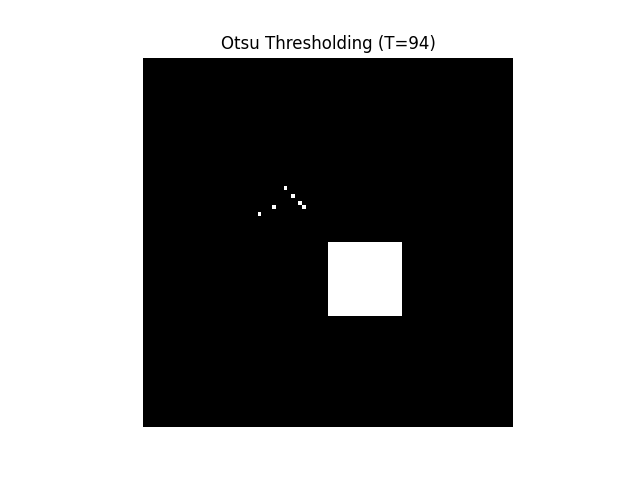


Figure 1.6: Otsu thresholding image

# Question 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.

def region\_growing(img, seeds, threshold=10):

    height, width = img.shape #Perform region growing segmentation.

    segmented = np.zeros\_like(img, dtype=np.uint8)

    visited = np.zeros\_like(img, dtype=bool)

    stack = list(seeds)

    for seed in seeds:

        segmented[seed] = 255

        visited[seed] = True

    while stack:

        x, y = stack.pop()

        for dx, dy in [(-1,0), (1,0), (0,-1), (0,1)]:

            nx, ny = x + dx, y + dy

            if 0 <= nx < height and 0 <= ny < width and not visited[nx, ny]:

                if abs(int(img[nx, ny]) - int(img[x, y])) <= threshold:

                    segmented[nx, ny] = 255

                    visited[nx, ny] = True

                    stack.append((nx, ny))

    return segmented

Figure 2.1: Region growing function

# Step 4: Region Growing with seeds from each object

seeds = [(20, 20), (50, 50)]  # Points inside both objects

region\_grown = region\_growing(original, seeds, threshold=10)

Figure 2.2: Applying to region growing function

plt.title("Region Growing Result")

plt.imshow(region\_grown, cmap='gray')

plt.axis('off')

plt.show()

Figure 2.3: Plot the outputs

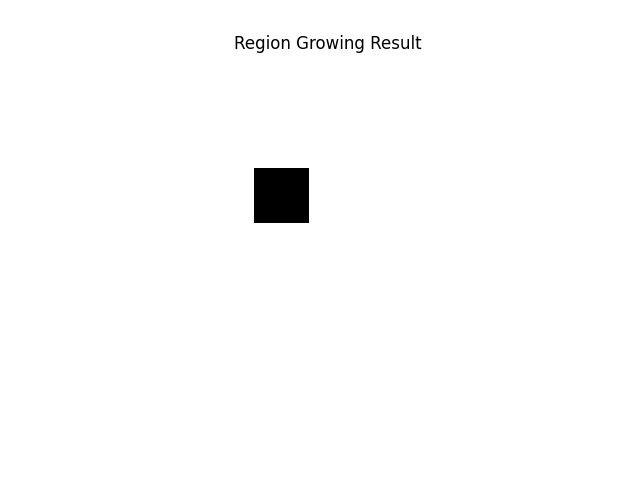


Figure 2.4: Region growing output image

GitHub link: <https://github.com/VishSeran/EC7212-Computer-Vision-and-Image-Processing-Take-Home-Assignment-2.git>