EC7212 – COMPUTER VISION AND IMAGE PROCESSING

ASSIGNMENT 02

NAME : ABEYSEKARA P.K.
REG NO : EG/2020/3799
SEMESTER : 07
DATE : 27/06/2025

GitHub Repository Link
https://github.com/PasanAbeysekara/image-Segmentation-Techniques

Task Implementations and Results

Task 1: Otsu's Algorithm on a Noisy Synthetic Image

Code Implementation: The Python script for this task :

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

def create_synthetic_image(height=300, width=400, val_bg=60, val_obj1=140,
    val_obj2=220):
    """Creates a synthetic image with two objects and a background."""
    image = np.full((height, width), val_bg, dtype=np.uint8)

    # Object 1: Rectangle
    cv2.rectangle(image, (width//8, height//8), (width//2, height//2), val_obj1,
    -1)

    # Object 2: Circle
    cv2.circle(image, (width*3//4, height*3//4), width//10, val_obj2, -1)

    return image

def add_gaussian_noise(image, mean=0, sigma=30):
    """Adds Gaussian noise to an image."""
    row, col = image.shape
    gauss = np.random.normal(mean, sigma, (row, col))
    noisy_image = image.astype(np.float32) + gauss
    noisy_image = np.clip(noisy_image, 0, 255) # Clip values to be in 0-255 range
    return noisy_image.astype(np.uint8)

def display_and_save_results(original, noisy, otsu_img, otsu_thresh_val,
    output_dir):
    """Displays and saves the final image comparison."""
    os.makedirs(output_dir, exist_ok=True)

    plt.figure(figsize=(18, 6))

    plt.subplot(1, 3, 1)
    plt.imshow(original, cmap='gray', vmin=0, vmax=255)
    plt.title("Original Synthetic Image")
    plt.axis('off')

    plt.subplot(1, 3, 2)
    plt.imshow(noisy, cmap='gray', vmin=0, vmax=255)
    plt.title("Image with Gaussian Noise")
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.imshow(otsu_img, cmap='gray', vmin=0, vmax=255)
    plt.title(f"Otsu's Thresholding (Thresh = {otsu_thresh_val:.2f})")
    plt.axis('off')

    plt.tight_layout()
    # Save the composite figure for the report
    plt.savefig(os.path.join(output_dir, "task1_comparison.png"))
    plt.show()

    # Save individual images
```

```
    cv2.imwrite(os.path.join(output_dir, "synthetic_original.png"), original)
    cv2.imwrite(os.path.join(output_dir, "synthetic_noisy.png"), noisy)
    cv2.imwrite(os.path.join(output_dir, "synthetic_otsu_thresholded.png"),
    otsu_img)

    # Also save the histogram of the noisy image
    plt.figure(figsize=(7, 5))
    plt.hist(noisy.ravel(), 256, [0, 256])
    plt.title("Histogram of Noisy Image")
    plt.xlabel("Pixel Intensity")
    plt.ylabel("Frequency")
    plt.axvline(x=otsu_thresh_val, color='r', linestyle='dashed', linewidth=2,
    label=f"Otsu's Threshold = {otsu_thresh_val:.2f}")
    plt.legend()
    plt.grid(True, linestyle='--', alpha=0.6)
    plt.savefig(os.path.join(output_dir, "synthetic_noisy_histogram.png"))
    plt.show()


if __name__ == "__main__":
    print("--- Task 1: Otsu's Algorithm ---")
    output_directory = "results/task-1"

    # 1. Create a synthetic image with 3 distinct pixel values
    synthetic_img = create_synthetic_image()

    # 2. Add Gaussian noise to the image
    noisy_img = add_gaussian_noise(synthetic_img.copy(), sigma=30)

    # 3. Implement and test Otsu's algorithm
    # Using cv2.THRESH_OTSU automatically finds the optimal threshold
    threshold_value, otsu_thresholded_img = cv2.threshold(
        noisy_img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU
    )

    print(f"Optimal threshold value determined by Otsu's method: {threshold_value
    }")

    # 4. Display and save all results
    display_and_save_results(
        synthetic_img, noisy_img, otsu_thresholded_img, threshold_value,
    output_directory
    )

    print("\nTask 1 Completed.")
```

Listing 1: Otsu's Algorithm Implementation and Test

Results:

The process involved creating a synthetic image with three distinct intensity levels, adding Gaussian noise, and then applying Otsu's thresholding. The algorithm successfully found an optimal threshold to separate the foreground objects from the background, despite the noise. The histogram of the noisy image clearly shows the overlapping distributions that Otsu's method is designed to handle. The determined threshold value was found to be [INSERT THRESHOLD VALUE FROM YOUR SCRIPT OUTPUT].

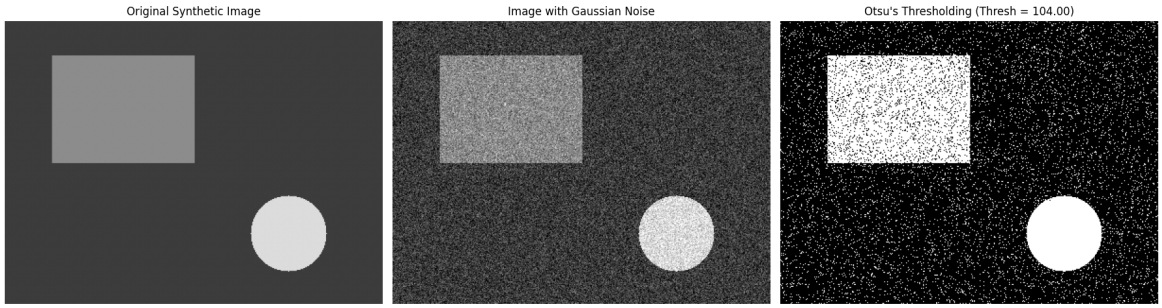

Figure 1: Comparison: (Left) Original synthetic image, (Middle) Image with Gaussian noise, (Right) Result of Otsu's thresholding.

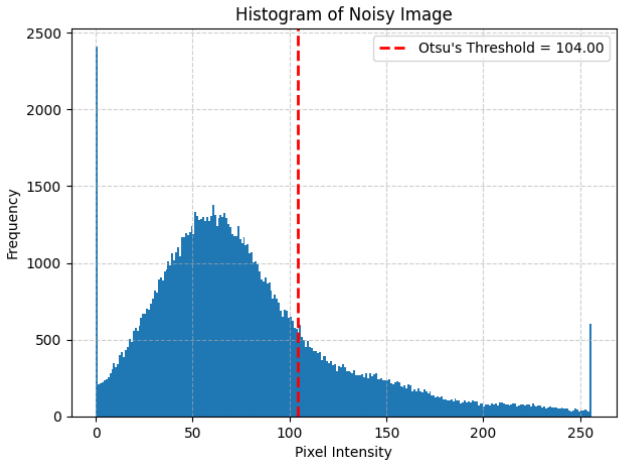

Figure 2: Histogram of the noisy image, with the calculated Otsu's threshold marked by the red dashed line.

Task 2: Region Growing Segmentation

Code Implementation:

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

def region_growing(image, seeds, threshold_val, connectivity=8):
    """
    Performs region growing segmentation iteratively.
    Args:
        image: Input grayscale image.
        seeds: A list of (y, x) tuples for seed points.
        threshold_val: Max absolute intensity difference from the seed average.
        connectivity: 4 or 8 for neighbor connectivity.
    Returns:
        Segmented image (binary mask).
    """
    if image is None: raise ValueError("Input image is None.")
    if not seeds: raise ValueError("Seed points must be provided.")
```

```
    height, width = image.shape[:2]
    segmented_mask = np.zeros_like(image, dtype=np.uint8)

    # Calculate average intensity of initial seeds
    seed_intensities = [image[y, x] for y, x in seeds if 0 <= y < height and 0 <=
     x < width]
    if not seed_intensities:
        print("Warning: All seed points are outside image bounds.")
        return segmented_mask

    avg_seed_intensity = np.mean(seed_intensities)
    print(f"Average seed intensity: {avg_seed_intensity:.2f}")
    print(f"Intensity difference threshold: {threshold_val}")

    points_to_process = []
    for seed_y, seed_x in seeds:
        if 0 <= seed_y < height and 0 <= seed_x < width:
            if segmented_mask[seed_y, seed_x] == 0:
                points_to_process.append((seed_y, seed_x))
                segmented_mask[seed_y, seed_x] = 255

    # Define neighbors
    if connectivity == 8:
        neighbors = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0)
    , (1, 1)]
    elif connectivity == 4:
        neighbors = [(-1, 0), (0, -1), (0, 1), (1, 0)]
    else:
        raise ValueError("Connectivity must be 4 or 8.")

    head = 0
    while head < len(points_to_process):
        curr_y, curr_x = points_to_process[head]
        head += 1

        for dy, dx in neighbors:
            ny, nx = curr_y + dy, curr_x + dx

            if 0 <= ny < height and 0 <= nx < width and segmented_mask[ny, nx] ==
     0:
                pixel_intensity = image[ny, nx]
                # Compare neighbor's intensity to the average intensity of the
    initial seeds
                if abs(int(pixel_intensity) - int(avg_seed_intensity)) <=
    threshold_val:
                    segmented_mask[ny, nx] = 255
                    points_to_process.append((ny, nx))

    return segmented_mask

def display_and_save_segmentation(original, segmented, seeds, output_dir,
    filename_suffix):
    """Displays and saves the segmentation result."""
    os.makedirs(output_dir, exist_ok=True)

    # Create an overlay image for visualization
    overlay = cv2.cvtColor(original, cv2.COLOR_GRAY2BGR)
    overlay[segmented == 255] = [0, 0, 255] # Mark segmented area in blue
    for y, x in seeds:
        cv2.circle(overlay, (x, y), 3, (0, 255, 0), -1) # Mark seeds in green

    plt.figure(figsize=(18, 6))
    plt.subplot(1, 3, 1)
    plt.imshow(original, cmap='gray')
    plt.title("Original Test Image")
    plt.axis('off')
```

```
    plt.subplot(1, 3, 2)
    plt.imshow(segmented, cmap='gray')
    plt.title("Segmented Mask")
    plt.axis('off')

    plt.subplot(1, 3, 3)
    plt.imshow(cv2.cvtColor(overlay, cv2.COLOR_BGR2RGB))
    plt.title("Segmentation Overlay")
    plt.axis('off')

    plt.tight_layout()
    plt.savefig(os.path.join(output_dir, f"segmentation_results_{filename_suffix
    }.png"))
    plt.show()

    # Save individual images
    cv2.imwrite(os.path.join(output_dir, f"segmentation_test_image.png"),
    original)
    cv2.imwrite(os.path.join(output_dir, f"segmented_mask_{filename_suffix}.png")
    , segmented)
    cv2.imwrite(os.path.join(output_dir, f"segmentation_overlay_{filename_suffix
    }.png"), overlay)


if __name__ == "__main__":
    print("--- Task 2: Region Growing Segmentation ---")
    output_directory = "results/task-2"

    # Create a test image
    height, width = 300, 300
    test_img = np.full((height, width), 40, dtype=np.uint8) # Background
    # Object 1 (darker)
    cv2.rectangle(test_img, (50, 50), (120, 120), 100, -1)
    # Object 2 (brighter)
    cv2.ellipse(test_img, (200, 200), (60, 40), 0, 0, 360, 180, -1)
    # Add noise
    noise = np.random.normal(0, 8, test_img.shape).astype(np.int16)
    test_img = np.clip(test_img.astype(np.int16) + noise, 0, 255).astype(np.uint8
    )

    # --- Test Case 1: Segmenting the rectangle ---
    seeds_rect = [(80, 80)]
    threshold_rect = 20 # Max intensity difference allowed
    print(f"\nSegmenting rectangle with seeds: {seeds_rect}")
    segmented_mask_rect = region_growing(test_img.copy(), seeds_rect,
    threshold_val=threshold_rect)
    display_and_save_segmentation(test_img, segmented_mask_rect, seeds_rect,
    output_directory, "rectangle")

    # --- Test Case 2: Segmenting the ellipse ---
    seeds_ellipse = [(200, 200)]
    threshold_ellipse = 25 # Threshold might need to be different
    print(f"\nSegmenting ellipse with seeds: {seeds_ellipse}")
    segmented_mask_ellipse = region_growing(test_img.copy(), seeds_ellipse,
    threshold_val=threshold_ellipse)
    display_and_save_segmentation(test_img, segmented_mask_ellipse, seeds_ellipse
    , output_directory, "ellipse")

    print("\nTask 2 Completed.")
```

Listing 2: Region Growing Implementation

Results:

A region growing algorithm was implemented to segment objects based on pixel intensity homogeneity starting from seed points. The algorithm was tested on a synthetic image containing a rectangle and an ellipse with different brightness levels, plus noise. Two test cases were run: one with a seed in the rectangle and one with a seed in the ellipse. The results show that the algorithm successfully identified and segmented the respective objects by growing from the initial seeds, demonstrating its effectiveness for segmentation when prior knowledge (the seed points) is available.

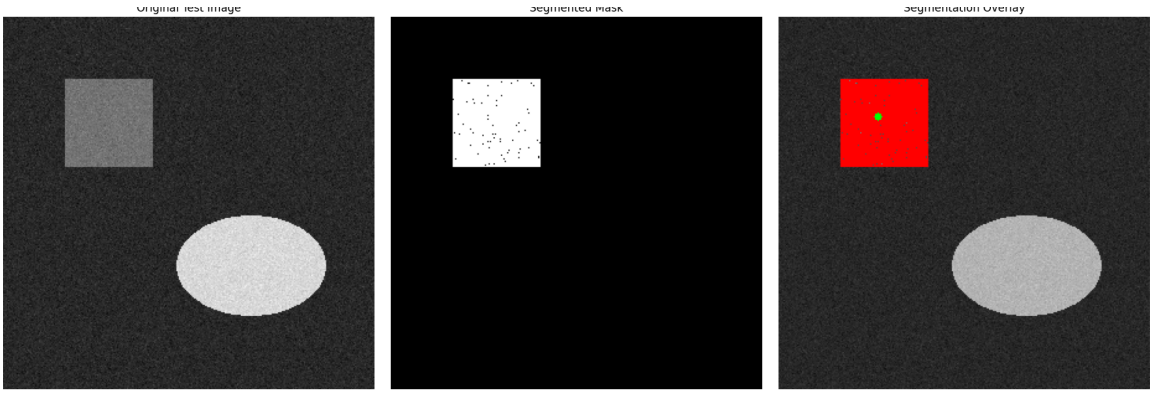

Figure 3: Segmentation of the rectangle. From left to right: Original image, binary mask of the segmented region, and an overlay showing the segmented area (blue) and seed point (green).

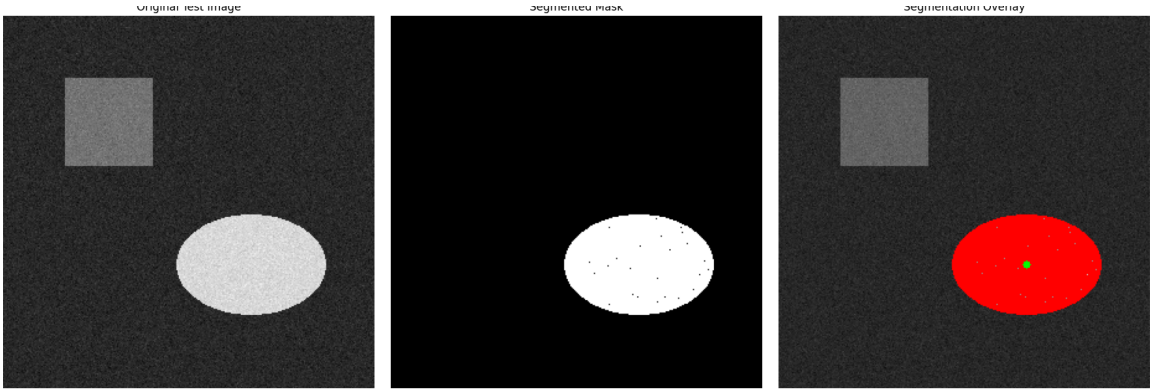

Figure 4: Segmentation of the ellipse, demonstrating the algorithm's ability to segment a different object with a new seed point and threshold.