# 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):
    image = np.full((height, width), val_bg, dtype=np.uint8)
    cv2.rectangle(image, (width//8, height//8), (width//2, height//2), val_obj1,
    -1)
    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):
    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)
    return noisy_image.astype(np.uint8)

def display_and_save_results(original, noisy, otsu_img, otsu_thresh_val,
    output_dir):
    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()
    plt.savefig(os.path.join(output_dir, "task1_comparison.png"))
    plt.show()

    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)

    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__":
    output_directory = "results/task-1"
    synthetic_img = create_synthetic_image()
    noisy_img = add_gaussian_noise(synthetic_img.copy(), sigma=30)
    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
    }") # in my case 104.00
    display_and_save_results(
        synthetic_img, noisy_img, otsu_thresholded_img, threshold_value,
    output_directory
    )
```

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 104.00.

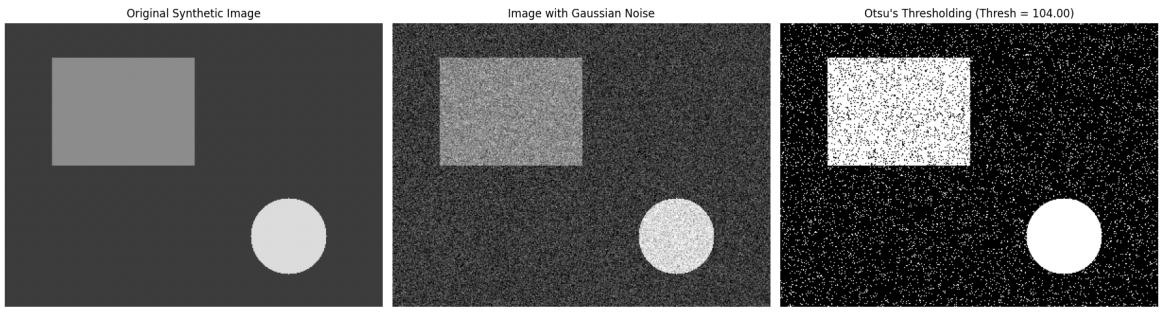


Figure 1: Comparison: (Left) Original synthetic image, (Middle) Image with Gaussian noise, (Right) Result of Otsu's thresholding.

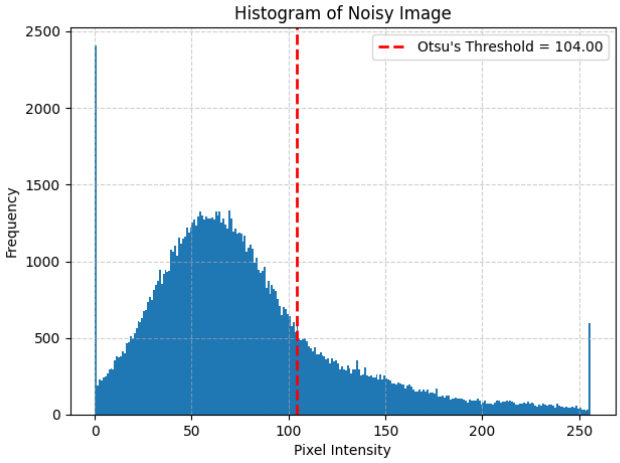


Figure 2: Histogram of the noisy image, with the calculated Otsu's threshold marked by the red dashed line.

## 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):

    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)

    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

    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]
                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):
    os.makedirs(output_dir, exist_ok=True)

    overlay = cv2.cvtColor(original, cv2.COLOR_GRAY2BGR)
    overlay[segmented == 255] = [0, 0, 255] # Mark segmented area in red
    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 Brain MRI")
    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()

    cv2.imwrite(os.path.join(output_dir, f"brain_mri_original.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 on Brain MRI ---")
    output_directory = "results/task-2"
    image_path = "input/brain_image.jpg"

    # 1. Load the brain MRI image
    try:
        brain_img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
        if brain_img is None:
            raise FileNotFoundError(f"Image not found at path: {image_path}")
    except FileNotFoundError as e:
```

```
        print(e)
        exit()
    except Exception as e:
        print(f"An error occurred while loading the image: {e}")
        exit()

    # --- Test Case 1: Segmenting White Matter ---
    seeds_white_matter = [(215, 225)]
    threshold_white_matter = 15

    print(f"\nSegmenting White Matter with seeds: {seeds_white_matter}")
    segmented_mask_wm = region_growing(brain_img.copy(), seeds_white_matter,
    threshold_val=threshold_white_matter)
    display_and_save_segmentation(brain_img, segmented_mask_wm,
    seeds_white_matter, output_directory, "white_matter")

    # --- Test Case 2: Segmenting Gray Matter ---
    seeds_gray_matter = [(150, 150)]
    threshold_gray_matter = 14

    print(f"\nSegmenting Gray Matter with seeds: {seeds_gray_matter}")
    segmented_mask_gm = region_growing(brain_img.copy(), seeds_gray_matter,
    threshold_val=threshold_gray_matter)
    display_and_save_segmentation(brain_img, segmented_mask_gm, seeds_gray_matter
    , output_directory, "gray_matter")
```

Listing 2: Region Growing Implementation

**Results and Analysis:**

The region growing algorithm was applied to a T1-weighted brain MRI scan to segment specific tissue types. The primary challenge was selecting an appropriate seed point and fine-tuning the intensity ‘threshold_val‘ to accurately capture the desired region without "leaking" into adjacent tissues.

Multiple test cases were run to segment both white and gray matter. The analysis below focuses on segmenting the gray matter, which required careful parameter tuning.

**Parameter Tuning for Gray Matter Segmentation:** Several thresholds were tested for a seed point placed within the cortical gray matter.

- **Thresholds 10 & 13:** These were too strict, resulting in an "under-segmentation" where the region failed to grow and connect properly, missing large parts of the target tissue.
- **Thresholds 15, 16, & 20:** These were too lenient, causing the region to "leak" significantly into the brighter white matter, leading to "over-segmentation" and loss of accuracy.
- **Threshold 14:** This value was found to provide the optimal balance. It was lenient enough to capture the natural intensity variations within the gray matter, allowing the region to grow along the complex cortical folds, while being strict enough to maintain a clear boundary with the adjacent white matter.

The final, optimized result using ‘threshold_val = 14‘ is presented below.


Figure 3: Optimal segmentation of gray matter using a seed point and a threshold of 14. From left to right: Original MRI, binary mask of the segmented region, and an overlay showing the segmented area (red) and seed point (green).


Figure 4: Optimal segmentation of gray matter using a seed point and a threshold of 13. From left to right: Original MRI, binary mask of the segmented region, and an overlay showing the segmented area (red) and seed point (green).

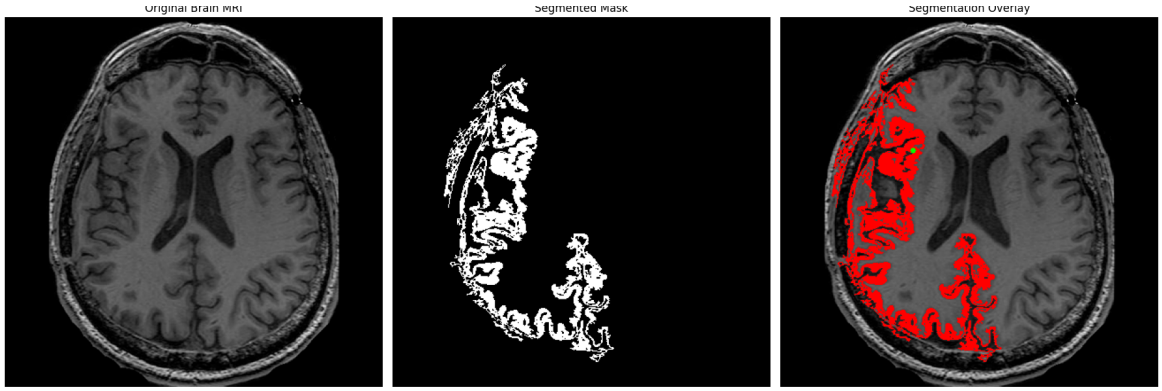


Figure 5: Optimal segmentation of gray matter using a seed point and a threshold of 15. From left to right: Original MRI, binary mask of the segmented region, and an overlay showing the segmented area (red) and seed point (green).

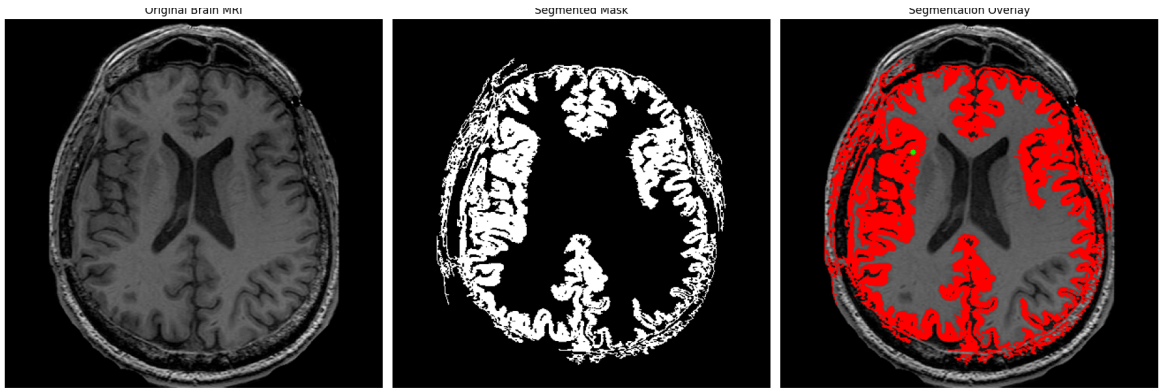


Figure 6: Optimal segmentation of gray matter using a seed point and a threshold of 20. From left to right: Original MRI, binary mask of the segmented region, and an overlay showing the segmented area (red) and seed point (green).

**White Matter Segmentation:** The algorithm was also successfully applied to segment the brighter white matter tissue using a different seed point and a tuned threshold, as shown below.


Figure 7: Segmentation of the white matter using a different seed point and threshold.

The results demonstrate that region growing is a powerful, yet sensitive, segmentation technique. Its success is highly dependent on the careful selection of seed points and the homogeneity threshold, which often requires empirical tuning for specific applications like medical image analysis.