EC7212 – COMPUTER VISION AND IMAGE PROCESSING

ASSIGNMENT 02

NAME :DISSANAYAKE D.K.R.C.K.

REG NO : EG/2020/3910

SEMESTER : 07

DATE : 27/06/2025

Table of Contents

1 Introduction

Image segmentation is a fundamental process in computer vision that involves partitioning an image into meaningful regions, often to isolate objects from the background or to simplify image analysis. In this assignment, two classical segmentation techniques were implemented using Python.

1.1 Otsu's Thresholding on a Noisy Image

This task involves generating a synthetic grayscale image containing two objects and a background, each with distinct intensity values. Gaussian noise is added to simulate real-world imperfections, and Otsu's thresholding is applied to automatically determine an optimal threshold that separates foreground and background.

1.2 Region Growing Segmentation

In this task, a region-growing algorithm is used to segment an object starting from a user-defined seed point. Neighboring pixels are added to the region based on their intensity similarity to the seed, effectively grouping connected pixels with similar values.

Both tasks were implemented in Jupyter Notebooks using Python libraries such as NumPy, OpenCV, and Matplotlib. The assignment provides hands-on experience with both global (Otsu's) and local (region growing) segmentation techniques, highlighting their strengths and limitations in different contexts.

2 Methodology

The implementation of this assignment was carried out in a structured and modular approach using Python. Each image segmentation task was developed and tested separately in its own Jupyter Notebook file to ensure clarity and ease of experimentation. The project followed a consistent methodology across both tasks.

2.1 Development Environment

The project was organized in a clearly defined folder structure, separating source code (`notebooks/`), output images (`outputs/`), and supporting files (`README.md`, `requirements.txt`). All programs were written using Python 3, utilizing the following key libraries.

- **NumPy** for numerical operations and array manipulation
- **OpenCV (cv2)** for image processing and thresholding
- **Matplotlib** for displaying and analyzing results
- **Jupyter Notebook** for step-by-step coding, testing, and result visualization

2.2 Folder Structure

```
image_segmentation_project/
│
├── notebooks/                          # Task notebooks
│   ├── task1_otsu_thresholding.ipynb
│   └── task2_region_growing.ipynb
│
├── outputs/                            # Generated output images
│   ├── task1/
│   └── task2/
│
├── requirements.txt                    # Python dependencies
└── README.md                           # Project documentation
```

2.3 Image Input

2.3.1 Otsu's Thresholding on a Noisy Image

A synthetic grayscale image was created programmatically, containing two objects and a background with three distinct intensity values.

Gaussian noise was added to simulate a real-world noisy environment.

2.3.2 Region Growing Segmentation

The noisy image (or a clean synthetic version) was used as the input to test the robustness of the region-growing segmentation.

All images were processed as grayscale to simplify thresholding and region analysis.

2.4 Result Visualization and Saving

At each stage of processing, results were visualized using **Matplotlib** for easy comparison.

Intermediate outputs (e.g., noisy image, thresholded mask, segmented regions) were saved to their corresponding `outputs/` subfolder.

Proper titles and axis settings were applied to make the visualizations clean and interpretable.

Each image was saved with descriptive filenames for documentation and future reference.

3 Implementation

3.1 Consider an image with 2 objects and a total of 3-pixel values. Add Gaussian noise to the image. Implement and test Otsu's Algorithm.

- Generate a synthetic grayscale image with 3 intensity levels
- Add Gaussian noise to the image
- Apply Otsu's thresholding to separate foreground and background
- Save and visualize the thresholded result

Code and Results

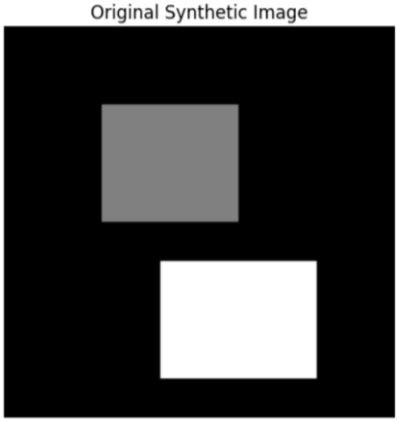
Task 1 : Consider an image with 2 objects and a total of 3-pixel values (1 for each object and one for the background). Add Gaussian noise to the image. Implement and test Otsu's algorithm with this image.

Import Required Libraries

In [1]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
```

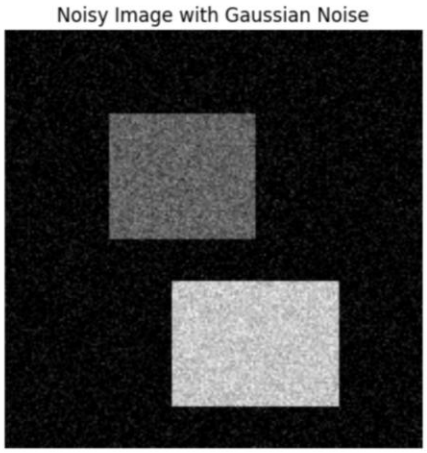
Create Synthetic Image with 2 Objects

In [2]:
```
def create_synthetic_image(shape=(200, 200)):
    """
    Create a 3-level grayscale image with 2 square objects and background.
    Pixel values: 0 (background), 100 (object1), 200 (object2)
    """
    img = np.zeros(shape, dtype=np.uint8)

    # Draw first object (value 100)
    img[40:100, 50:120] = 100

    # Draw second object (value 200)
    img[120:180, 80:160] = 200

    return img

synthetic_img = create_synthetic_image()

plt.imshow(synthetic_img, cmap='gray')
plt.title("Original Synthetic Image")
plt.axis('off')
plt.show()
```



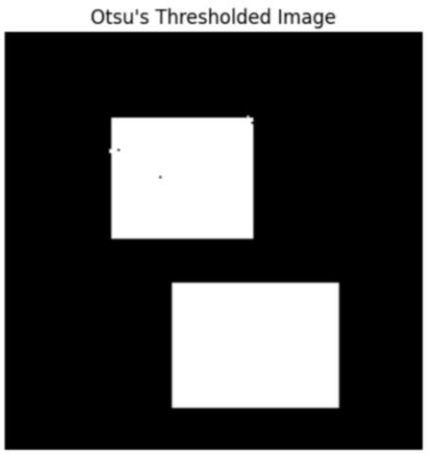
Add Gaussian Noise

In [3]:
```python
def add_gaussian_noise(image, mean=0, std=20):
    noise = np.random.normal(mean, std, image.shape)
    noisy_image = image + noise
    noisy_image = np.clip(noisy_image, 0, 255)  # Keep in range
    return noisy_image.astype(np.uint8)

noisy_img = add_gaussian_noise(synthetic_img)

plt.imshow(noisy_img, cmap='gray')
plt.title("Noisy Image with Gaussian Noise")
plt.axis('off')
plt.show()
```



Apply Otsu's Thresholding

In [4]:
```python
# Apply Gaussian blur before thresholding (optional but recommended)
blurred = cv2.GaussianBlur(noisy_img, (5, 5), 0)

# Otsu's thresholding
_, otsu_thresh = cv2.threshold(blurred, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

plt.imshow(otsu_thresh, cmap='gray')
plt.title("Otsu's Thresholded Image")
plt.axis('off')
plt.show()
```



Save Outputs

In [5]:
```python
output_dir = '../outputs/task1'
os.makedirs(output_dir, exist_ok=True)

cv2.imwrite(os.path.join(output_dir, 'synthetic_image.png'), synthetic_img)
cv2.imwrite(os.path.join(output_dir, 'noisy_image.png'), noisy_img)
cv2.imwrite(os.path.join(output_dir, 'otsu_result.png'), otsu_thresh)

print("Images saved in:", output_dir)
```

```
Images saved in: ../outputs/task1
```

3.2 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. Start from a user-defined seed point inside an object

- Grow the region by including neighboring pixels within a defined intensity range
- Return a binary mask representing the segmented object
- Adjustable threshold and seed position

Code and Results

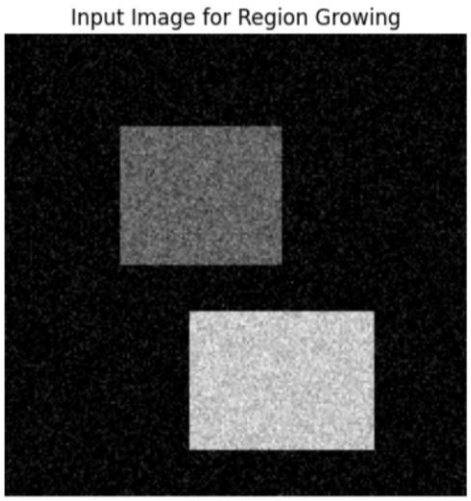
Task 2 : Implement a region-growing technique for image segmentation. The basic idea is to start from a set of points inside the object of interest (foreground), denoted as seeds, and recursively add neighboring pixels as long as they are in a pre-defined range of the pixel values of the seeds.

Import Required Libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import cv2
import os
from collections import deque
```

Load Image (Use Noisy Image from Task 1)

In [2]:
```
image_path = '../outputs/task1/noisy_image.png'  # Reuse output from Task 1
img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

plt.imshow(img, cmap='gray')
plt.title("Input Image for Region Growing")
plt.axis('off')
plt.show()
```

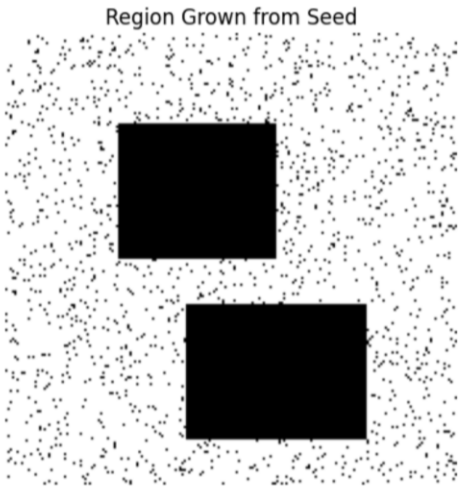


Region Growing Algorithm

In [89]:
```python
def region_growing(image, seed, threshold=10):
    """
    Perform region growing from the seed point using intensity similarity.
    :param image: Grayscale image
    :param seed: (x, y) coordinates
    :param threshold: Max difference from seed value to allow inclusion
    :return: Binary mask of the segmented region
    """
    h, w = image.shape
    seed_value = image[seed[1], seed[0]]  # (x, y) format
    visited = np.zeros_like(image, dtype=bool)
    mask = np.zeros_like(image, dtype=np.uint8)

    q = deque()
    q.append(seed)

    while q:
        x, y = q.popleft()
        if visited[y, x]:
            continue

        visited[y, x] = True
        if abs(int(image[y, x]) - int(seed_value)) <= threshold:
            mask[y, x] = 255
            for dx in [-1, 0, 1]:
                for dy in [-1, 0, 1]:
                    nx, ny = x + dx, y + dy
                    if 0 <= nx < w and 0 <= ny < h and not visited[ny, nx]:
                        q.append((nx, ny))

    return mask
```

Run Region Growing from a Seed

In [90]:
```python
seed_point = (4, 0)  # (x, y) coordinate inside object
threshold = 30  # Intensity threshold for region growing

segmented_mask = region_growing(img, seed=seed_point, threshold=threshold)

plt.imshow(segmented_mask, cmap='gray')
plt.title("Region Grown from Seed")
plt.axis('off')
plt.show()
```



Save the Result

In [91]:
```python
output_dir = '../outputs/task2'
os.makedirs(output_dir, exist_ok=True)

cv2.imwrite(os.path.join(output_dir, 'region_grown.png'), segmented_mask)
print("Region-growing result saved to:", output_dir)
```

```
Region-growing result saved to: ../outputs/task2
```

4 Github Repository

You can visit and check the Full project from this Link

https://github.com/chathuradissanayake/Image-Segmentation-Project-Python.git