# Task Report

**Name:** Hifza Sethi

**Roll No:** 221349

**Semester:** 7th

**Subject:** Digital Image Processing (DIP)

**Topic:** Thresholding, Adaptive Thresholding, Seed Selection in Segmentation, and Bit-Plane Slicing for Data Hiding

## Table of Contents

## 1. Introduction

Digital Image Processing (DIP) deals with analyzing and modifying digital images using computer algorithms.

In this task, I implemented and studied **thresholding techniques**, **segmentation seed selection**, and **bit-plane slicing** for secure message hiding.

These techniques are widely used in **object detection, document processing, medical image analysis, and image security**.

## 2. Thresholding

### 2.1 What is Thresholding?

Thresholding is a process that converts a grayscale image into a **binary image** (only black and white).

It works by choosing a cutoff value, called a **threshold (T)**.

Pixels brighter than T are turned white (255), and others are turned black (0).

Formula:

[

g(x, y) =

\begin{cases}

255, & \text{if } f(x, y) > T \

0, & \text{otherwise}

\end{cases}

]

This makes it easy to separate the **object (foreground)** from the **background**.

## 2.2 Types of Thresholding

### *(a) Global Thresholding*

In this method, a **fixed threshold value** is used for the entire image.

It works well when lighting conditions are even.

**Python Example:**

```
_, th_global = cv2.threshold(img, 127, 255, cv2.THRESH_BINARY)
```

### *(b) Otsu's Thresholding*

When an image has varying brightness, a fixed threshold may fail.

**Otsu's method** automatically calculates the best threshold value by minimizing the difference between the object and background intensities.

**Python Example:**

```
_, th_otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
```

### *(c) Adaptive Thresholding*

When lighting changes across the image (like shadows), **adaptive thresholding** divides the image into small regions and applies a different threshold for each region.

There are two common types:

- **Adaptive Mean Thresholding** — uses the average of neighboring pixels.

- **Adaptive Gaussian Thresholding** — uses a weighted average for smoother results.

**Python Example:**

```
th_adaptive = cv2.adaptiveThreshold(img, 255,
                                    cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                    cv2.THRESH_BINARY, 11, 2)
```

### 2.3 Results and Observations

- **Global Thresholding** works well for clear, evenly lit images.
- **Otsu's Thresholding** automatically selects an optimal threshold.
- **Adaptive Gaussian Thresholding** gives the best results when the image has uneven lighting or shadows.

In my experiment, adaptive thresholding successfully highlighted text even in bright and dark regions.

## 3. Seed Selection in Segmentation

Segmentation means dividing an image into meaningful parts — like separating an object from its background.

### 3.1 Region Growing and Seeds

In the **region growing** method, segmentation starts from **seed points**.

The algorithm expands these regions by adding neighboring pixels that are similar in intensity.

### 3.2 How to Select a Good Seed

1. **Manual Selection:**

The user picks a pixel that belongs to the object.

2. **Automatic Selection:**
    a. Choose the pixel with **maximum intensity** for bright objects.
    b. Or select one **near the object center**.
    c. For multiple objects, use multiple seeds.

### 3.3 Importance

Choosing the correct seed ensures that the region grows within the actual object area.

Incorrect seeds can lead to wrong segmentation results.

Thus, the success of segmentation largely depends on **accurate seed selection**.

## 4. Bit-Plane Slicing for Message Hiding

### 4.1 Concept

A grayscale image contains 8 bits per pixel.

Each bit forms a **bit-plane**, from the most significant (bit 7) to the least significant (bit 0).

The **least significant bit (LSB)** plane carries very small visual information, so we can **use it to hide secret data** without noticeable changes to the image.

This method is known as **image steganography**.

### 4.2 Implementation Steps

1. **Create a Grayscale Image**

```
img = np.ones((100, 300), dtype=np.uint8) * 255
cv2.putText(img, "HELLO AI", (30, 60), cv2.FONT_HERSHEY_SIMPLEX, 1.5,
(0), 3)
```

2. **Convert Message to Binary**

```
message = "Hi friend! I finished my assignment :)"
bits = ''.join([format(ord(c), '08b') for c in message])
```

3. **Hide the Message**

```
secret_img = img.copy().flatten()
for i in range(len(bits)):
    secret_img[i] = (secret_img[i] & 254) | int(bits[i])
secret_img = secret_img.reshape(img.shape)
cv2.imwrite("F:/python/AI/dip/task3/secret_image.png", secret_img)
```

4. **Decode the Message**

```
flat = secret_img.flatten()
recovered_bits = [str(flat[i] & 1) for i in range(len(bits))]
recovered_message = ''.join([chr(int(''.join(recovered_bits[i:i+8]),
2))
                             for i in range(0, len(recovered_bits),
8)])
print("Hidden Message:", recovered_message)
```

### 4.3 Observation

The secret image looks identical to the original because changes in the least significant bits are invisible to the human eye.

However, the hidden message can be recovered perfectly, proving that **data can be securely transmitted inside an image**.

## 5. Conclusion and Insight

This task helped me understand that image processing is not just about pictures .It's about **data and structure** within each pixel.

- I learned that **thresholding** separates objects based on pixel intensity.
- **Otsu's and adaptive thresholding** can automatically adjust to brightness changes.
- I understood that **seed selection** directly affects segmentation accuracy.
- Finally, **bit-plane slicing** taught me that images can also be used to **safely hide and share confidential messages**.

This task improved my logical thinking and made me appreciate how mathematical operations on pixels can extract meaning and even hide information — showing the true power of digital image processing.

## 6. References

1. Gonzalez, R. C., & Woods, R. E. (2018). *Digital Image Processing* (4th Edition). Pearson.
2. OpenCV Official Documentation: https://docs.opencv.org
3. Python Image Processing Tutorials – GeeksforGeeks, OpenCV.org