# Edge Detection

## 1 Introduction

This assignment dealt with the development of edge detection filters from scratch. By implementing each step manually, we enhanced our understanding of pixel-level image manipulation, a core concept in computer vision. The tasks involved converting color images to grayscale, applying Gaussian blur using convolution, implementing the Sobel operator, and detecting edges through thresholding. Additionally, I experimented with a custom edge detector to observe the effect of manual tweaks in the filtering pipeline.

## 2 Process Overview

The edge detection pipeline consisted of the following steps:

1. **Grayscale Conversion**
2. **Gaussian Blur using Convolution**
3. **Sobel Edge Detection**
4. **Thresholding**

These steps were use to build the standard Edge -detection using Sobel filter. Each of these steps is explained below along with their Python implementations.

### 2.1 Grayscale Conversion

To simplify computations and focus on intensity changes, the RGB image was converted to grayscale using a weighted sum of color channels based on human eye's perception of colored images.

```python
def to_grayscale(image):
    h, w = image.shape[0], image.shape[1]
    img = np.zeros((h, w), dtype=np.uint8)
    for i in range(h):
        for j in range(w):
            gray_value = 0.2989 * image[i, j, 0] + \
                         0.5870 * image[i, j, 1] + \
                         0.1140 * image[i, j, 2]
            img[i, j] = np.uint8(gray_value)
    return img
```

Listing 1: Grayscale Conversion

### 2.2 Gaussian Blur via Convolution

To reduce noise, a Gaussian kernel was applied using convolution. Padding was added to maintain image size.

```python
def convolution(image, kernel, show=False):
    h, w = image.shape
    pad = kernel.shape[0] // 2
    img_padded = np.zeros((h+2*pad, w+2*pad), dtype=np.float32)

    for i in range(h+2*pad):
        for j in range(w+2*pad):
            if pad <= i < h+pad and pad <= j < w+pad:
                img_padded[i, j] = image[i-pad, j-pad]

    img_conv = np.zeros((h, w), dtype=np.float32)
    for i in range(h):
        for j in range(w):
            for k in range(kernel.shape[0]):
                for l in range(kernel.shape[1]):
                    img_conv[i, j] += img_padded[i+k, j+l] * kernel[k, l]

    img_conv = np.clip(img_conv, 0, 255).astype(np.uint8)

    if show:
        plt.imshow(img_conv, cmap='gray')
        plt.title('Convoluted Image')
        plt.show()

    return img_conv
```

Listing 2: Convolution with Custom Kernel

### 2.3 Sobel Operator

The Sobel operator was used to compute intensity gradients in both the x and y directions. The magnitude of the gradients was then used for edge detection.

```python
def sobel_operator(image, show=True):
    kernel_x = np.array([[1, 0, -1],
                         [2, 0, -2],
                         [1, 0, -1]])
    kernel_y = np.array([[1, 2, 1],
                         [0, 0, 0],
                         [-1, -2, -1]])
    img_x = convolution(image, kernel_x).astype(np.float32)
    img_y = convolution(image, kernel_y).astype(np.float32)
    img_magnitude = np.sqrt(img_x**2 + img_y**2)
    img_magnitude = np.clip(img_magnitude, 0, 255).astype(np.uint8)

    if show:
        plt.imshow(img_magnitude, cmap='gray')
        plt.title('Sobel Filter')
        plt.axis('off')
        plt.show()

    return img_magnitude
```

Listing 3: Sobel Operator

### 2.4 Thresholding

A thresholding function is used to convert the gradient magnitude image into a binary edge map. A pixel is marked as an edge if its intensity is greater than a specified threshold. Different values of threshold were tested based on the result of the Sobel Operator.

## 3 Modified Edge Detection Filter

A custom filter was implemented to explore different visual results. The process included:

- Computing the average of RGB channels.
- Subtracting the grayscale image to highlight color differences.
- Applying a Gaussian-like kernel through convolution.
- Applying a simple threshold-based edge detection.

```python
def modified_edge_detector(image, threshhold):
    kernel = np.array([[1, 2, 1],
                       [2, 4, 2],
                       [1, 2, 1]], dtype=np.float32)
    kernel /= kernel.sum()

    img_avg = (image[:, :, 0] + image[:, :, 1] + image[:, :, 2]) / 3
    img_gray = to_grayscale(image)
    img = (img_avg - img_gray)

    plt.imshow(img, cmap='gray')
    plt.show()

    img1 = convolution(img, kernel, show=True)
    edge = Simple_edge_detector(img1, threshhold)

    plt.imshow(edge, cmap='gray')
    plt.title('Detected Edge')
    plt.show()
```

Listing 4: Modified Edge Detector

This approach highlights areas where color information differs significantly from grayscale, which is especially useful at colored object boundaries. The custom kernel applies a smoothing effect before edge detection, and the final threshold isolates the most prominent edges.

# 4 Results

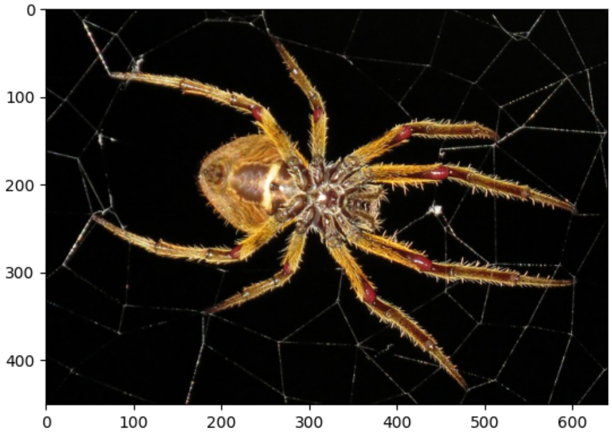
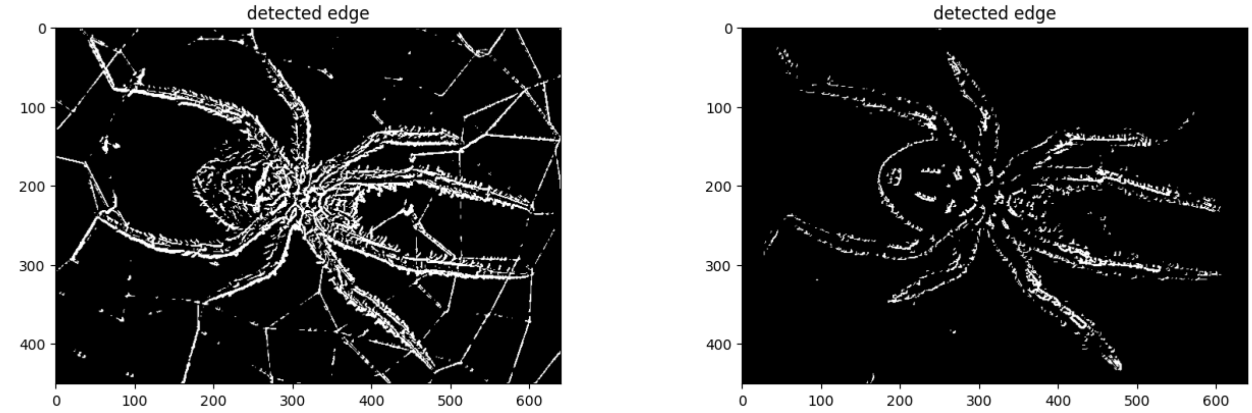


Figure 1: Image of a Spider



(a) Sobel Edge Detection

(b) Modified Edge Detection

Figure 2: Comparison: Sobel vs Modified Edge Detection