Assignment 4

Paolo Deidda, Raffaele Perri

March 26, 2025

Problem 1 [5 points]

A synthetic color image 1 was created where:

- The red channel had a strong transition from bright to dark on the left side.
- The green channel had the opposite transition from dark to bright.
- The blue channel was kept constant.

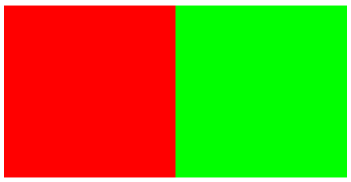

Figure 1: Synthetic image

Applying the Sobel operator to each channel individually and summing the results led to the cancellation of gradients, demonstrating the issue.

Implementation

The Sobel operator was applied separately to the R, G, and B channels, and their gradients were summed.

```
# Blue channel (index 0) remains 0 throughout.

# Define the Sobel kernels for x and y directions
gx = np.array([[-1, 0, 1],
               [-2, 0, 2],
               [-1, 0, 1]], dtype=np.float32)
gy = gx.T # swaps rows and columns of gx

# Function to compute the x and y gradients using
def compute_gradients(channel):
    grad_x = cv2.filter2D(channel.astype(np.float32), -1, gx)
    grad_y = cv2.filter2D(channel.astype(np.float32), -1, gy)
    return grad_x, grad_y
```

```

# Split the image into B, G, R channels (apparently OpenCV uses BGR order)
B, G, R = cv2.split(img)

# Compute gradients for each channel
Rx, Ry = compute_gradients(R)
Gx, Gy = compute_gradients(G)
Bx, By = compute_gradients(B)  # This will be zero everywhere
```

Results

Figure 2 shows the computed gradients for the synthetic image. The first image represents the gradient in the x-direction, the second image shows the gradient in the y-direction, and the third image depicts the overall gradient magnitude.

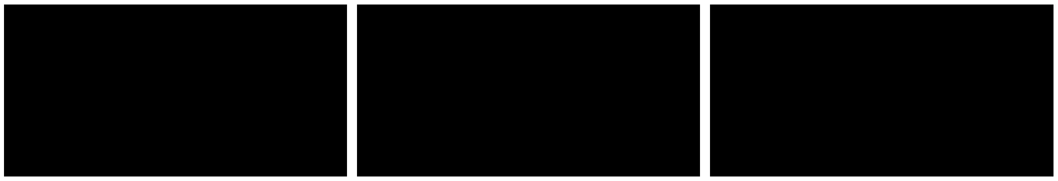

Figure 2: Computed gradients: (left) Gradient X, (middle) Gradient Y, (right) Gradient Magnitude

Conclusion

We can see there is no broder detection; the images appear as one solid color. This happens because when computing the gradient across channels, the strong increase in one channel can be canceled by a strong decrease in another. In this case, the red channel transitions from strong to zero, while the green channel transitions from zero to strong. When summed, these opposing gradients effectively nullify each other, resulting in a misleading overall gradient calculation.

This demonstrates the issue with summing individual color gradients. The results show that edges can be present in a color image while the summed gradient remains zero. Therefore we need an alternative gradient computation methods in color image processing.

Problem 2 [5 points]

The implementation follows these steps:

1. **Image Preprocessing:**
    - Load the input image.
    - Convert it to grayscale if necessary.
2. **Applying Otsu's Thresholding:**
    - Compute the optimal threshold automatically using Otsu's method.
    - Apply the threshold to generate a binary image.
3. **Visualization and Output:**
    - Display both the original and thresholded images.
    - Save the resulting binary image.

The function `otsu_thresholding()` converts the image to grayscale if it is not already and then applies Otsu's thresholding using `cv2.threshold()`.
The key component of the function is:

```
_, binary_image = cv2.threshold(gray_image, 0, 255, cv2.THRESH_BINARY +
cv2.THRESH_OTSU)
```

Here, Otsu's method automatically determines the threshold value by analyzing the image's histogram and maximizing the inter-class variance.
Applying Otsu's method to "houses.pgm" effectively separates the objects in the image from the background. The resulting binary image highlights structural details such as edges of buildings and other prominent objects.

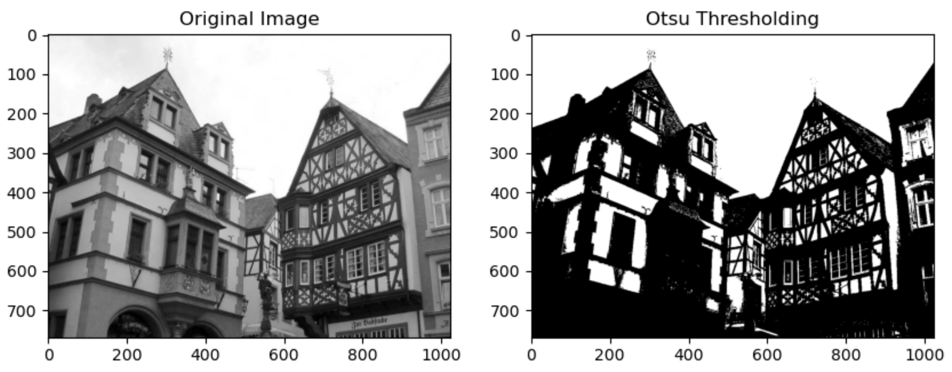

Figure 3: Otsu method.

The implementation successfully applies Otsu's method to "houses.pgm," demonstrating its ability to segment objects efficiently.

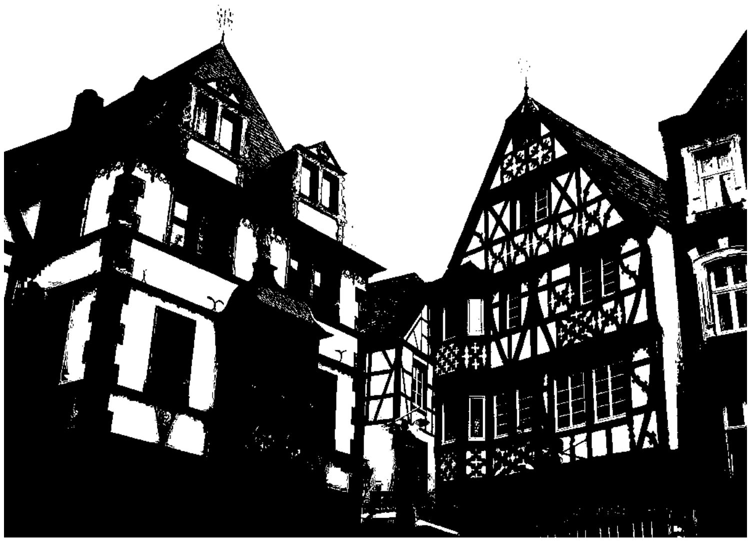

Figure 4: Final image.

Future improvements could involve adaptive thresholding techniques for cases where global thresholding is insufficient due to uneven lighting conditions.