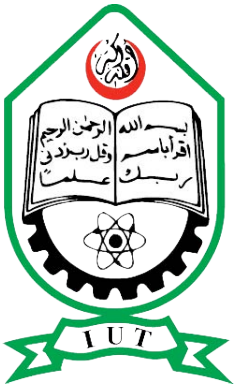


Department of Computer Science and Engineering

Islamic University of Technology

CSE 4733: Digital Image Processing

Assignment 01
Canny Edge Detector

K. M. Abesh Ahsan

200041225

Section 2

April 6, 2025

Contents

1 Introduction

The **Canny edge detector** is an edge detection operator that uses a multi-stage algorithm to detect a wide range of edges in images. It was developed by John F. Canny in 1986. He also produced a computational theory of edge detection explaining why the technique works [4].

In this report, the Canny edge detection algorithm has been implemented using the resources and explanations provided in [1, 2, 3, 4].

My implementation is available at: https://github.com/abeshahsan/Assignment---Canny-Edge-Detector

1.1 Why Canny?

The Canny edge detector was developed to address key limitations of simpler methods like the Sobel filter, which only computes gradient magnitude and direction. The motivation stemmed from three main goals:

Noise Reduction: Sobel is sensitive to noise, as it directly computes gradients without smoothing.

Edge Thinning: Sobel produces thick edges, while Canny uses non-maximum suppression to yield single-pixel-wide edges.

Edge Connectivity: Sobel relies on a single threshold, often creating fragmented edges. Canny employs hysteresis thresholding (dual thresholds) to link weak edges to strong ones, improving continuity.

1.2 Why Canny over Sobel?

Sobel alone lacks noise suppression, generates thick edges, and struggles with false positives/negatives due to basic thresholding. Canny integrates Gaussian smoothing, edge thinning, and intelligent thresholding, making it more robust, precise, and reliable for real-world applications where edge quality is critical.

1.3 Canny Edge Detection Algorithm

The Canny edge detection algorithm is a multi-step process that involves several stages to ensure accurate and reliable edge detection. The main steps of the Canny edge detection algorithm are as follows:

1. Noise Reduction
2. Gradient Calculation
3. Non-maximum Suppression
4. Double Threshold
5. Edge Tracking by Hysteresis

In the subsequent sections, we will delve into each of these steps in detail.

To illustrate the Canny edge detection algorithm, we will use the image shown in **??** as an example. The result of applying the algorithm is depicted in **??**.

2 Grayscale Conversion

The Canny Edge Detection algorithm operates on grayscale images. This is because edge detection is primarily concerned with changes in intensity. Converting a color image to grayscale simplifies the computation and focuses on the luminance information, which is crucial for detecting edges.

The effect after converting a color image to grayscale is shown in **??**.

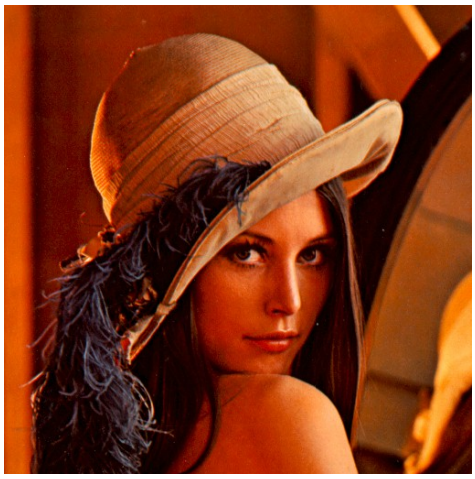
3 Noise Reduction

The method of *Canny edge detection* is solely based on gradients, and the gradient calculation is very sensitive to noise. So, it is necessary to reduce noise in the image before applying the edge detection algorithm. There are several methods to reduce noise in an image. The most common method is to apply a *Gaussian filter* to the image. It is a low-pass filter that removes high-frequency noise from the image; i.e., it smoothens the image.

(a) Original Image

(b) Edge Detected Image

Figure 1: Comparison of Original and Edge Detected Images using Canny Edge Detection Algorithm



(a) Original Color Image

(b) Grayscale Image

Figure 2: Output of Converting a Color Image to Grayscale

$$H(x,y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1}$$

Where, $H(x,y)$ is the Gaussian filter, σ is the standard deviation of the Gaussian distribution, and x and y are the coordinates of the filter. The Gaussian filter is applied to the image using convolution.

After applying the Gaussian filter, the image becomes less noisy, and the edges become more pronounced. This is illustrated in **??**.

4 Edge Detection

Edge detection is a fundamental task in image processing and computer vision, aimed at identifying points in an image where the intensity changes sharply. These points typically correspond to object boundaries and other significant features within the image.

The gradient of an image is a vector that has both magnitude and direction. The magnitude indicates the strength of the edge, while the direction indicates the orientation of the edge (to be more precise, the shape of the object).

4.1 Sobel Operator

The *Sobel operator* is a widely used edge detection technique that employs two convolution kernels to estimate the gradient of the image intensity. One kernel is used to calculate the gradient in the x-direction, while the other is used

(a) Original Grayscale Image

(b) Image after Applying Gaussian Filter (5×5)

Figure 3: Effect of Noise Reduction on an Image

to calculate the gradient in the y-direction.

Sobel Operator in the x-direction The convolution kernel for the *Sobel* operator in the x-direction is defined as:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \tag{2}$$

Sobel Operator in the y-direction The convolution kernel for the *Sobel* operator in the y-direction is defined as:

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{3}$$

4.2 Magnitude and Direction of the Gradient

The gradient magnitude and direction are essential for understanding the strength and orientation of edges in an image. These can be computed using the gradients in the x and y directions, G_x and G_y, respectively.

Gradient Magnitude The magnitude of the gradient can be calculated using two different norms:

- **L1 Norm:**

$$\text{Magnitude}_{L1} = |G_x| + |G_y| \tag{4}$$

- **L2 Norm:**

$$\text{Magnitude}_{L2} = \sqrt{G_x^2 + G_y^2} \tag{5}$$

Gradient Direction The direction of the gradient indicates the orientation of the edge and is computed as follows:

$$\text{Direction} = \arctan\left(\frac{G_y}{G_x}\right) \tag{6}$$

The gradient direction is typically measured in radians and can be converted to degrees if needed.

The result of applying the Sobel operator to an image is shown in **??**.

It is important to note in **??** that the gradient direction can have both positive and negative values. Consequently, when visualizing the image, negative values are automatically truncated.

(a) Gradient Magnitude

(b) Gradient Direction

Figure 4: Gradient Magnitude and Direction using Sobel Operator

5 Non-Maximum Suppression

Non-Maximum Suppression (NMS) is a technique used to thin out the edges detected in the gradient image, ensuring that only the most significant edges are retained. This step is essential for refining the edge map and removing spurious responses, which helps in accurately identifying the true edges in an image.

Technically, NMS examines the gradient magnitude and direction at each pixel, comparing it with neighboring pixels along the gradient direction. If the current pixel's magnitude is not the maximum, it is suppressed (set to zero); otherwise, it is retained. This ensures that only local maxima are preserved, resulting in a thinned edge map.

In simpler terms, NMS sharpens blurry, thick edges by keeping only the highest points along an edge direction, like tracing an outline with a fine-tipped pen instead of a broad marker.

The process of Non-Maximum Suppression can be broken down into the following steps:

1. Compute the gradient direction for each pixel in the gradient image using **??**.
2. For each pixel, compare its gradient magnitude with the magnitudes of the two neighboring pixels along the gradient direction.
3. If the pixel's magnitude is not the maximum among the three, set it to zero; otherwise, retain its value.

See in **??** for an example of how NMS works. The cell enclosed by red-dashed border is the current pixel that is being processed. The two neighboring pixels with black-dashed borders are the pixels that falls in the gradient direction of the current pixel. In **??** the current pixel's gradient direction is pi. And in **??** the current pixel's gradient direction is $5pi/4$. **??** shows the case when the current pixel is a local maximum. In this case, the current pixel is retained. So, no suppression is done. **??** shows the case when the current pixel is not a local maximum. In this case, the current pixel is suppressed. So, the pixel is set to zero.

The result of applying non-maximum suppression to the gradient magnitude image is shown in **??**.

6 Double Thresholding

Following non-maximum suppression, the edge pixels provide a clearer representation of true edges in the image. Nonetheless, some edge pixels might still appear due to noise and color differences. To eliminate these false positives, it is essential to discard edge pixels with low gradient values while retaining those with high gradient values.

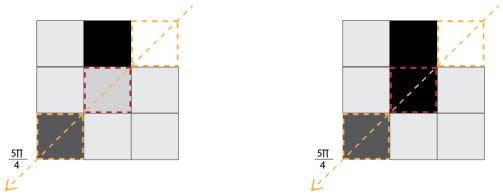
This is achieved through a process called *double thresholding*.

The Canny edge detector uses double thresholding to classify edge pixels into three categories: strong edges, weak edges, and non-edge pixels. The process is as follows:

1. **Strong Edges:** Pixels with gradient magnitudes higher than a high threshold are classified as strong edges.
2. **Non-Edge Pixels:** Pixels with gradient magnitudes lower than a low threshold are classified as non-edge pixels.

(a1) Before NMS (a2) After NMS

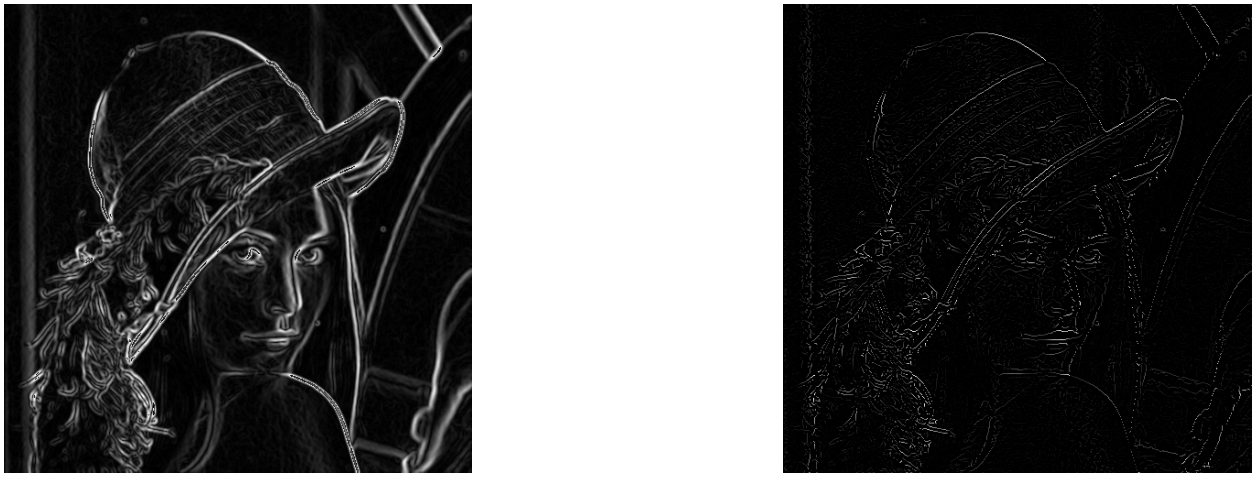
(a) No Suppression



(b1) Before NMS (b2) After NMS

(b) Suppression

Figure 5: Visualization of Non-Maximum Suppression



(a) Gradient Magnitude (b) After Non-Maximum Suppression

Figure 6: Effect of Non-Maximum Suppression on Gradient Magnitude

3. **Weak Edges:** Pixels with gradient magnitudes between the high and low thresholds are classified as weak edges.

The high and low thresholds are determined based on the gradient magnitude distribution in the image. Typically, the high threshold is set to a value that retains only strong edges, while the low threshold is set to a fraction of the high threshold.

In my version of implementation, I used ratios for the threshold — T_{high} and T_{high} — to determine the high and low thresholds. The ratios are provided as input parameters to the function. Then the high and low thresholds are calculated as follows:

$$T_{\text{high}} = r_{\text{max}} \cdot r_{\text{high\_ratio}}$$
$$T_{\text{low}} = T_{\text{high}} \cdot r_{\text{low\_ratio}}$$

The *strong pixels* are assigned a value of 255, the *weak pixels* are assigned a value lower than 255 (e.g., 100), and the *non-edge pixels* are assigned a value of 0.

The images before and after double thresholding are shown in **??**.

(a) After Non-Maximum Suppression

(b) After Double Thresholding

Figure 7: Effect of Double Thresholding

7 Hysteresis

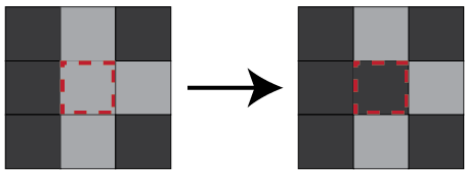
The final step in the Canny edge detection algorithm is edge linking through a process called *hysteresis*. This step aims to connect weak edge pixels to strong edge pixels to form continuous and complete edges in the image.

What hysteresis does is, it helps to eliminate false edges by considering the connectivity of weak edges to strong edges. The idea is that if a weak edge pixel is connected to a strong edge pixel, it should be retained as part of the edge. Conversely, if a weak edge pixel is not connected to any strong edge pixels, it should be discarded.

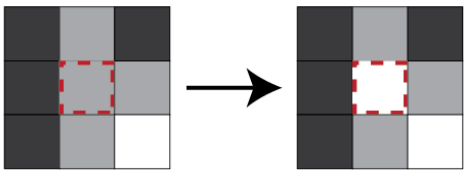
The hysteresis process involves the following steps:

1. Examine a weak edge pixel and its 8-connected neighborhood pixels.
2. Identify if there is at least one strong edge pixel in the neighborhood.
3. If a strong edge pixel is found, preserve the weak edge pixel as part of the edge, i.e., convert it to a strong edge pixel.
4. Repeat the process for neighboring weak edge pixels to ensure continuity.

The visualization of the hysteresis process is shown in **??**. In the first case, the weak edge pixel has no strong edge neighbors, so it is set to 0. In the second case, the weak edge pixel has (at least one) strong edge neighbors, so it is set to 255.



(a) Weak Pixel with No Strong Neighbors (Set to 0)



(b) Weak Pixel with Strong Neighbors (Set to 255)

Figure 8: Effect of Hysteresis on Weak Edge Pixels

The result of the hysteresis process is a binary image with strong edges that are connected to weak edges, forming complete edges in the image, as shown in **??**.

8 Results

8.1 Some Results Using Different Images

The Canny Edge Detector was applied to various images, demonstrating its effectiveness in edge detection. The results are shown in **??** to **??**. Each image is presented in grayscale and with the edges detected using the Canny method.

(a) After Double Thresholding

(b) After Hysteresis

Figure 9: Effect of Hysteresis

(a) Lenna Grayscale

(b) Lenna Edge Detected

Figure 10: Canny Edge Detection Results for Different Images (Part 1)

(c) Road Grayscale

(d) Road Edge Detected

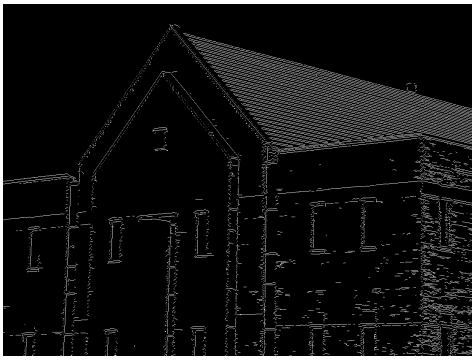
Figure 10: Canny Edge Detection Results for Different Images (Part 2)

(e) Van Grayscale

(f) Van Edge Detected

Figure 10: Canny Edge Detection Results for Different Images (Part 3)

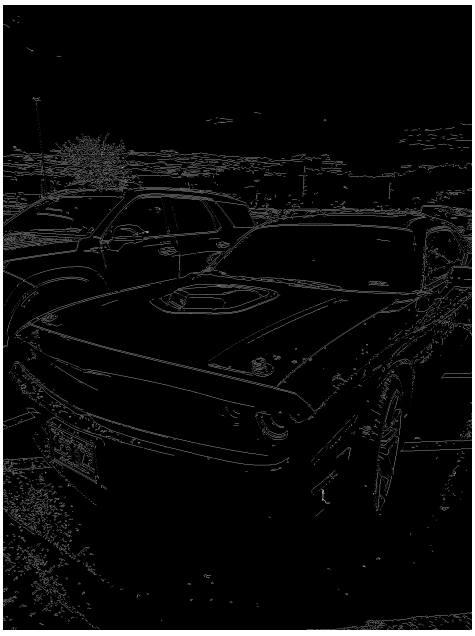
(g) Building Grayscale



(h) Building Edge Detected

Figure 10: Canny Edge Detection Results for Different Images (Part 4)

(i) Car Grayscale

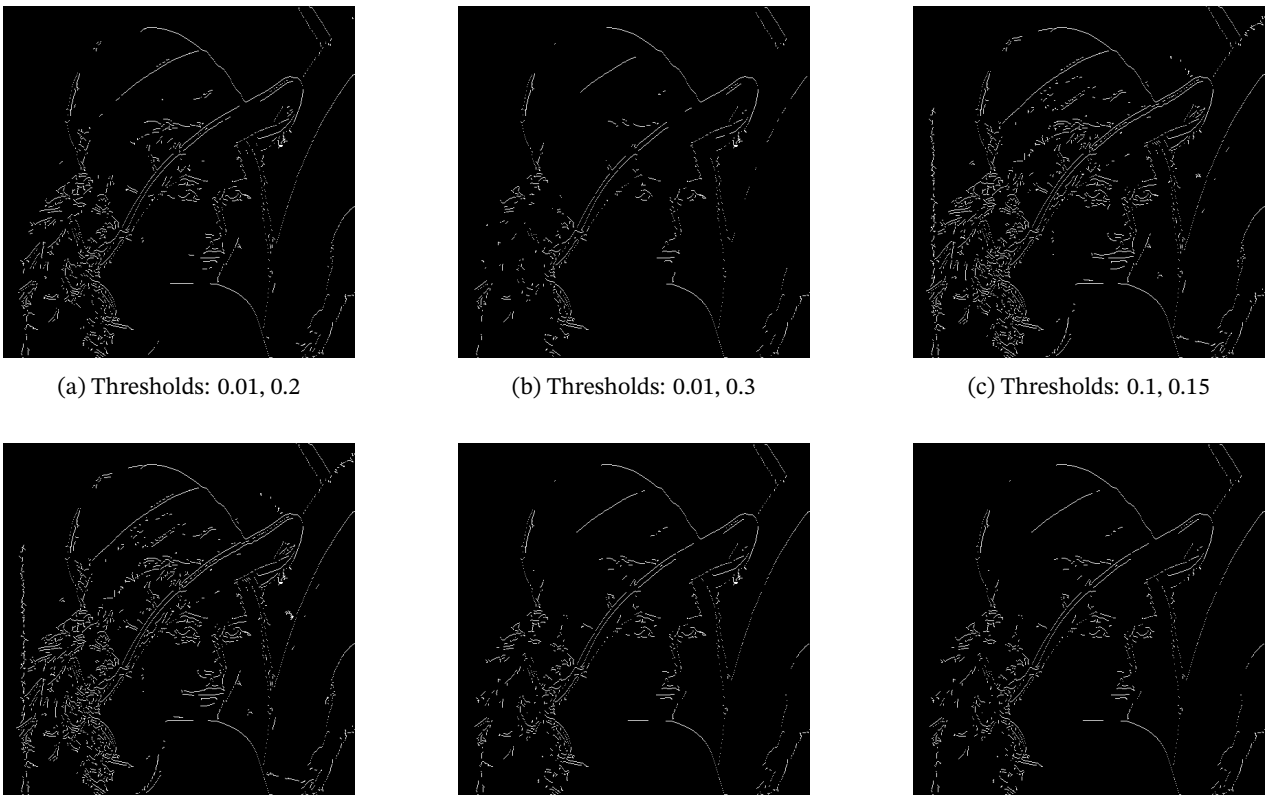


(j) Car Edge Detected

Figure 10: Canny Edge Detection Results for Different Images (Part 5)

8.2 Comparison of Different Thresholds

The Canny Edge Detector was applied to the same image with different thresholds to observe the effect on edge detection. The results are shown in **??**.



(a) Thresholds: 0.01, 0.2 (b) Thresholds: 0.01, 0.3 (c) Thresholds: 0.1, 0.15

(d) Thresholds: 0.01, 0.15 (e) Thresholds: 0.01, 0.25 (f) Thresholds: 0.15, 0.25

Figure 11: Canny Edge Detection Results with Different Thresholds

9 Conclusion

In this assignment, the Canny Edge Detection algorithm was implemented to identify edges in digital images effectively. The process involved applying Gaussian smoothing to reduce noise, calculating intensity gradients, and using non-maximum suppression to refine edge detection. Double thresholding and edge tracking by hysteresis were employed to produce the final edge map.

The results demonstrated the algorithm's ability to detect edges with high accuracy while minimizing noise and false detections. The findings highlight the importance of parameter tuning, such as the thresholds and Gaussian kernel size, in achieving optimal performance. Overall, the implementation validated the robustness and efficiency of the Canny Edge Detector in edge detection tasks.

References

[1] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. 3rd ed. Upper Saddle River, NJ: Pearson Education, 2008.

[2] Sofiane Sahir. *Canny Edge Detection: Step by Step in Python Computer Vision*. https://medium.com/data-science/canny-edge-detection-step-by-step-in-python-computer-vision-b49c3a2d8123. Medium; Accessed on 2023-10-15.

[3] Abhishek Sriram. *Canny Edge Detection Explained and Compared with OpenCV in Python*. https://medium.com/@abhisheksriram845/canny-edge-detection-explained-and-compared-with-opencv-in-python-57a161b4bd19. Medium; Accessed on 2023-10-15.

[4] Wikipedia contributors. *Canny edge detector*. https://en.wikipedia.org/wiki/Canny_edge_detector. Accessed on 2023-10-15.