Tutorial Note on Canny Edge Detector (Canny, 1986)

Student Name: Shajeed Hossain
Student Id: 200041142

Introduction

Edge detection is a fundamental tool in image processing and computer vision, crucial for identifying object boundaries within images. Among various edge detection algorithms, the Canny Edge Detector [1], developed by John F. Canny in 1986, stands out for its accuracy and reliability. This tutorial provides an in-depth exploration of the Canny Edge Detector, emphasizing the motivation behind generating single-pixel thick edges and detailing the algorithm's step-by-step process [2].

Motivation for Single-Pixel Thick Edges

In image analysis, accurately delineating object boundaries is essential. Early edge detection methods often resulted in thick or multiple responses to a single edge, complicating the interpretation of edge information. The Canny Edge Detector addresses this by refining edge detection to produce a precise, single-pixel wide representation of edges, enhancing the clarity and utility of edge information in subsequent image processing tasks [2] [3].

The Canny Edge Detection Algorithm

The Canny Edge Detector is a multi-stage algorithm designed to detect a wide range of edges in images. Its primary objective is to satisfy three main criteria [4]:

1. **Low Error Rate:** Ensuring that edges occurring in images should not be missed and that there are no responses to non-existent edges
2. **Edge Localization:** The distance between the edge pixels detected and the actual edge should be minimized.
3. **Minimal Response:** Only one response to a single edge to avoid multiple responses.

The algorithm [5] comprises the following steps:

1. **Grayscale Conversion**
   The image is to be converted to grayscale, that is the intensity values of the pixels are 8 bit and range from 0 to 255. This helps in simplifying the edge detection process.

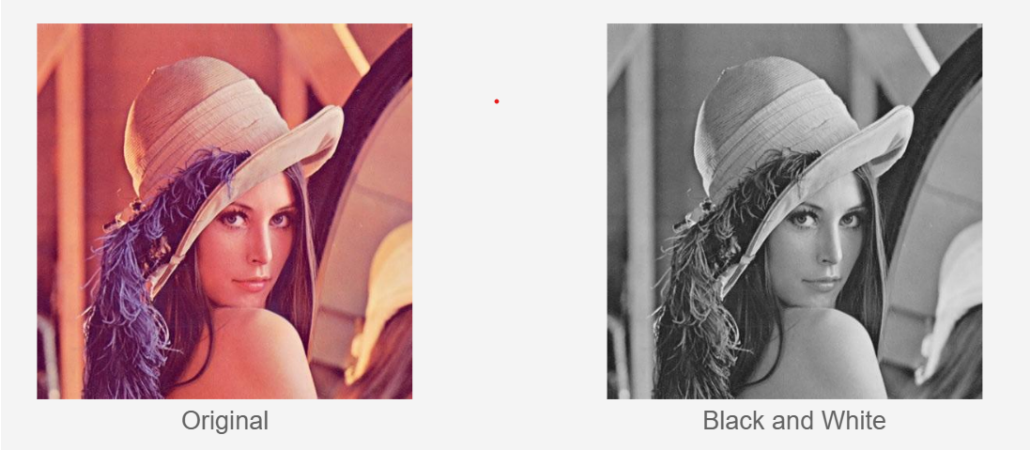

Figure 1: Grayscale Conversion

2. **Noise Reduction**
   Edge detection is susceptible to noise present in raw images. To mitigate this, the initial step involves smoothing the image using a Gaussian filter. This process reduces unwanted high-frequency noise, making edge detection more reliable.

Figure 2: Gaussian Blurring

3. **Gradient Calculation**
   After smoothing, the next step is to compute the gradient intensity of the image. This is typically achieved using Sobel filters in both horizontal (Gx) and vertical (Gy) directions.

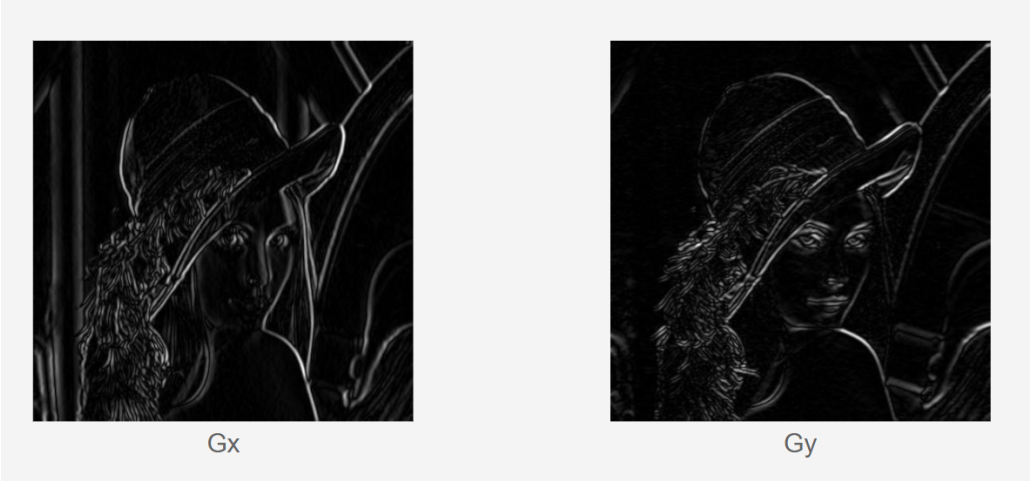

Figure 3: Sobel filtering the image

The gradient magnitude (G) and direction () are then calculated as follows:

- Gradient Magnitude: $G = \sqrt{G_x^2 + G_y^2}$
- Gradient Direction: $\theta = \arctan\left(\frac{G_y}{G_x}\right)$

These calculations highlight regions with high spatial derivatives, indicating potential edges.

Figure 4: Gradient magnitude

4. **Non-Maximum Suppression**
   The gradient magnitude results in thick edges. To achieve a single-pixel edge representation, non-maximum suppression is applied [3]. This technique involves:

   - Comparing each pixel's gradient magnitude to its neighbors in the direction of the gradient.
   - Retaining the pixel if it has the highest magnitude among its neighbors; otherwise, setting it to zero.

   This step effectively thins the edges to a single-pixel width.

Figure 5: Non-maximum suppression with interpolation

5. **Double Thresholding**
   To differentiate between strong, weak, and non-relevant pixels, double thresholding [4]:

   - **Strong Pixels**: Pixels with gradient magnitudes above the high threshold are considered strong edges.
   - **Weak Pixels:** Pixels with gradient magnitudes between the low and high thresholds are considered potential edges.
   - **Non-Relevant Pixels**: Pixels with gradient magnitudes below the low threshold are suppressed.

   This classification helps in identifying definitive edges and potential edges that require further analysis.

Figure 6: Double thresholding

6. **Edge Tracking by Hysteresis**
   The final step involves edge tracking by hysteresis to finalize the edge detection [4]:

   - Strong pixels are immediately considered part of the final edge image.
   - Weak pixels are included if they are connected to strong pixels; otherwise, they are suppressed.

   This process ensures that only valid edges are retained, reducing the likelihood of false edge detection.

Figure 7: Edge tracking

7. **Cleaning up**
   At last, we will iterate through the remaining weak edges and set them to zero, resulting in the final processed image.

Figure 8: Final Output

Intermediate Results

Throughout the execution of the Canny Edge Detector, intermediate results can be visualized to understand the transformation at each stage:

1. **Original Image:** The initial input image.
2. **After Gaussian Smoothing**: The image appears blurred, with reduced noise.
3. **Gradient Magnitude and Direction:** Visual representations of intensity changes and their orientations.
4. **Non-Maximum Suppression Output:** Thinned edges highlighting potential boundaries.
5. **Double Thresholding Result:** Categorization of pixels into strong, weak, and non-relevant.
6. **Final Edge Detection:** The resultant image with well-defined, single-pixel wide edges.

Analysis of Results

The Canny Edge Detector is renowned for its precision and reliability in identifying edges within images. By systematically applying Gaussian smoothing, gradient calculation, non-maximum suppression, double thresholding, and edge tracking by hysteresis, the algorithm effectively isolates true edges while minimizing noise and false detections.

One of the algorithm's strengths lies in its adaptability through parameter tuning. Adjusting the Gaussian filter's standard deviation affects the level of smoothing, allowing for control over noise reduction versus edge preservation. Similarly, setting appropriate high and low thresholds during the double thresholding stage enables differentiation between strong and weak edges, influencing the sensitivity of edge detection.

However, the performance of the Canny Edge Detector can be influenced by the choice of parameters and the characteristics of the input image. For instance, in images with significant noise or texture, the algorithm may produce fragmented edges or miss subtle boundaries. Therefore, careful parameter selection and preprocessing are crucial to achieve optimal results.

Implementation

A simplified implementation and visualization of the Canny Edge Detection algorithm using **cv2** libray can be found here.

A point to note is that, the function **cv2.Canny** (used in the code) internally computes gradients, applies **non-maximum suppression**, **double thresholding**, and **edge tracking by hysteresis** to produce a refined, single-pixel

edge map. Thus, these steps are not shown in code implementation. However, an implementation from scratch can also be found there.

References

[1] J. F. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[2] Wikipedia, "Canny edge detection," n.d. [Online]. Available: https://en.wikipedia.org/wiki/Canny_edge_detector

[3] IIT Delhi, "Canny edge detection," March 23, 2009. [Online]. Available: https://www.cse.iitd.ac.in/~pkalra/col783-2017/canny.pdf

[4] OpenCV Documentation, "Canny edge detection," n.d. [Online]. Available: https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html

[5] Justin Liang, "Canny edge detection," n.d. [Online]. Available: https://justin-liang.com/tutorials/canny/