Multivariable Calculus In Canny Edge Detection

David Szczecina (STU#:20964967)
Mechatronics Engineering, University of Waterloo
Waterloo, Canada
dszczeci@uwaterloo.ca

Introduction

Background and Motivation:
In image processing, the accurate detection and localization of edges play a crucial role in various applications such as object recognition, scene understanding, and image segmentation. The Canny edge detection algorithm has emerged as one of the most widely utilized techniques for edge detection due to its robustness and effectiveness. By leveraging the concept of gradients, Canny edge detection can identify significant intensity transitions, thus identifying the boundaries between different objects or regions in an image.

Objectives and Scope:
The primary objectives of this research review paper is investigating the theoretical foundations of Canny edge detection, including the mathematical principles behind gradient computation, thresholding, and non-maximum suppression. By delving into the theoretical foundations of both calculus and Canny edge detection, this paper seeks to identify ways in which calculus concepts can be effectively utilized to enhance the accuracy of edge localization.

The scope of this research paper is limited to the exploration and application of advanced calculus concepts within the framework of the Canny edge detection algorithm. It aims to provide insights into how the mathematical principles of multivariate differential and integral calculus can be harnessed to enhance edge localization accuracy, thus contributing to the advancement of edge detection techniques in image processing.

Background

Overview and Algorithm Steps:
Canny edge detection is a multi-stage algorithm that involves several key steps to identify and localize edges in an image. The algorithm aims to strike a balance between accurate edge localization and noise suppression. The following steps outline the general framework of the Canny edge detection algorithm [1]:

Gaussian Smoothing: The first step in Canny edge detection is to apply Gaussian smoothing to the input image. This is achieved by convolving the image with a Gaussian filter, which helps reduce noise and eliminate potential false gradients, but retains more of the edge than a simple mean blur filter.

Gradient Computation: After the image is smoothed, the gradient magnitude and direction are computed at each point in the image. This is done by convolving a 3x3 Sobel operator kernel with the image. The gradient magnitude represents the rate of intensity change, while the gradient direction indicates the direction of the steepest change.

Non-maximum Suppression: To ensure that the detected edges are thin and accurately localized, non-maximum suppression is performed. This step involves suppressing pixels that are not local maxima along the gradient direction. Only the pixels that have the maximum gradient magnitude in their local neighborhood are retained as potential edge points.

Thresholding: In this step, two threshold values, the high threshold and the low threshold, are defined. The potential edge points are categorized into strong and weak edges based on their gradient magnitudes relative to these thresholds. Strong edges are those with magnitudes above the high threshold, while weak edges lie between the low and high thresholds.

Hysteresis Thresholding: To finalize the edge map, hysteresis thresholding is applied. This step aims to connect weak edges to strong edges and discard weak edges that are not connected to strong edges. By tracing through the edges, weak edges that are part of continuous edge structures are promoted to strong edges, while isolated weak edges are eliminated.

Theoretical Foundations:
The effectiveness of the Canny edge detection algorithm lies in its mathematical foundations. The algorithm makes use of concepts from calculus and signal processing to identify significant intensity transitions in an image. The gradient computation step relies on the principles of differential calculus to estimate the rates of change of pixel intensities. By identifying areas with large gradient magnitudes, potential edge points can be detected.

The non-maximum suppression step, inspired by the concept of directional derivatives, ensures that only the pixels with the highest gradient magnitudes along the gradient direction are considered as edges. This process helps to refine edge localization and eliminate potential false positives.

Thresholding, another integral part of the Canny edge detection algorithm, utilizes concepts from integral calculus to differentiate between strong and weak edges based on their gradient magnitudes. By defining appropriate threshold values, the algorithm determines the significance of gradient magnitudes in determining edge presence.

Finally, hysteresis thresholding connects and filters weak edges. By traversing the edge map, weak edges are evaluated based on their connectivity to strong edges, ensuring the preservation of continuous edge structures while discarding isolated or noise-induced weak edges. [2]

Limitations and Challenges:
While Canny edge detection is a powerful technique, it is not without limitations. The algorithm's performance can be affected by various factors, such as noise levels in the input image, the selection of appropriate threshold values, and the presence of complex textures or cluttered backgrounds. Determining optimal threshold values can be challenging, as different images may require different thresholds to achieve desirable results. [3]

Additionally, the computational complexity of the Canny edge detection algorithm should be considered. The Gaussian smoothing and gradient computation steps involve convolutions and derivatives, which can be computationally expensive, especially for large images or real-time applications. Balancing computational efficiency with edge detection accuracy is a critical aspect that needs to be addressed.

Calculus in Canny Edge Detection Steps

Gradient Computation:
The Canny edge detection algorithm relies on accurately computing gradients, which measure the rate of change of pixel intensities in multiple directions. Multivariate differential calculus plays a key role in this process.

To compute the gradients, we can employ partial derivatives in different directions. Let's consider a grayscale image represented by a matrix I(x, y), where (x, y) represents the pixel coordinates. We can calculate the partial derivatives of the image intensities with respect to the x and y directions:

$$I_x = \frac{\partial I(x,y)}{\partial x} \qquad I_y = \frac{\partial I(x,y)}{\partial y}$$

These partial derivatives can be approximated using techniques such as convolution with derivative filters like the sobel operator. Gx and Gy, 3x3 matrices with particular values that allow for calculation of a partial derivative, are convoluted with each pixel to find the gradient. By calculating the partial derivatives, we obtain the rate of change of pixel intensities along the x and y directions. [4]

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix}, \; G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

We can then combine the partial derivatives to obtain the gradient magnitude G(x, y) and direction θ(x, y) at each pixel [5]:

$$G(x,y) \; = \; \sqrt{(I_x(x,y))^2 + (I_y(x,y))^2}$$
$$\theta(x,y) \; = \; \arctan\left(I_x(x,y)/I_y(x,y)\right)$$

The gradient magnitude represents the strength of the edges, while the gradient direction indicates the orientation of the edges.

Extending the Gradient Operator:
By incorporating advanced calculus concepts, we can extend the gradient operator beyond first-order derivatives. Higher-order derivatives can capture more detailed information about intensity changes, enabling the detection of fine-scale edges. For instance, second-order derivatives can help identify edges with curvature and provide a more accurate representation of the image structure. [6]

One way to incorporate higher-order derivatives is through the Laplacian of Gaussian (LoG) operator, which combines the Laplacian operator (second-order derivative) with a Gaussian filter. The LoG operator can be defined as:

$$LoG(x, y) = \nabla^2(G(x, y) \cdot f(x, y))$$

Here, G(x, y) represents the gradient magnitude and f(x, y) is the Gaussian filter. The LoG operator captures both the gradient magnitude and the curvature information, enabling more robust forms of edge detection.

Non-maximum Suppression:
Non-maximum suppression is an important step in Canny edge detection as it ensures thin and accurate edge localization. One approach would involve integrating the gradient information over local neighborhoods and identifying local maxima along the gradient direction. For each pixel, we compare its gradient magnitude with the neighboring pixels in the direction of the gradient orientation. If the pixel's magnitude is the maximum among its neighbors, it is retained; otherwise, it is suppressed. This integration-based analysis enhances edge localization and reduces the likelihood of false positives or unwanted edges.

Hysteresis Thresholding:
Integral calculus principles can also enhance hysteresis thresholding, which aims to connect weak edges to strong edges based on their connectivity. By considering the integral of gradient magnitudes along the edges, the algorithm can evaluate the overall strength and continuity of weak edges. A possible approach involves integrating the gradient magnitudes over the entire edge path. By analyzing the integral of gradient magnitudes, the algorithm can assess the connectedness of weak edges to strong edges more accurately. This integration-based analysis ensures the preservation of continuous edge structures while suppressing isolated weak edges that may arise due to noise or variations in image intensity.

Approach

To implement the application on a small problem, we will create a program consisting of all the different steps in canny edge detection. The same image, figure 1, will be run multiple times through the edge detection program with varying high and low threshold values. The goal will be to have the object outline marked in the edge detected image, and the background noise should be ignored. As the thresholds increase, more edges should be disregarded, until only the main object outline remains. The sample image used will be figure 1.

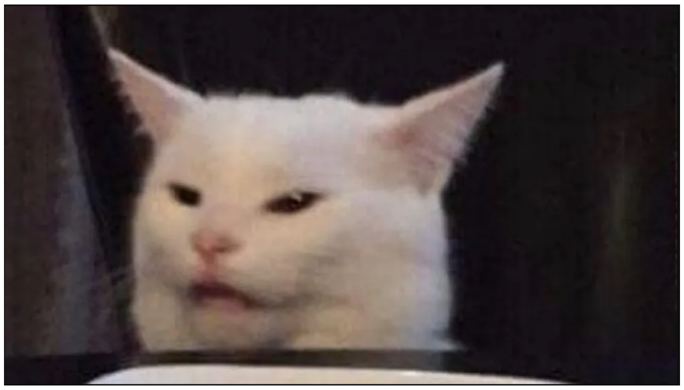
Figure 1: Sample image of cat used for benchmarking

The image is selected as there is a very clear object imposed on a contrasting background. By observation there is a clear outline of the object, the cat's head, as well as some characteristics in the object with outlines like the eyes and mouth, and a few background objects that should be ignored.

We will know that the outcome is complete when only the cat head and face are outlined with edges, all other parts of the image are omitted.

Implementation

Experiment setup:
The python implementation of the Canny edge detection algorithm involves creating different functions to handle key steps of the process. These functions include Gaussian blur, convolution, gradient computation, non-maximum suppression, thresholding, and hysteresis. The program is designed to take an input grayscale image and apply each of these functions in sequence to detect edges accurately, and output the image using the matplot library. Each function utilizes the numpy library and is based off of the CVFS/Canny Edge Detection Library. [7]

Convert to Grayscale: since the original image is in colour, we need to convert it to grayscale before passing it into the edge detection steps.

Gaussian Blur: A function is developed to apply a Gaussian filter to the input image, smoothing it and reducing noise.

Convolution: The convolution function is implemented to perform gradient calculation using Sobel operators, to obtain the partial derivatives Ix and Iy.

Gradient Computation: The gradient magnitude G and direction θ are computed using the partial derivatives Ix and Iy. This function involves multivariable differential calculus concepts to accurately estimate the rate of change of pixel intensities in different directions.

Non-Maximum Suppression: A function is created to implement non-maximum suppression, preserving only local maxima along the gradient directions. This step involves identifying and retaining significant edges while suppressing non-maximum values.

Thresholding: The thresholding function sets different threshold values to classify pixels as strong edges, weak edges, or non-edges. The integration of integral calculus principles helps in refining edge magnitude calculation and accurate threshold assignment.

Hysteresis Thresholding: This function connects weak edges to strong edges based on connectivity analysis. Advanced calculus concepts are utilized to assess edge continuity and promote weak edges connected to strong ones.

Edge Detection with Different Threshold Values:
To evaluate the impact of varying threshold values on edge detection, the implemented program is run multiple times on the same image with different high and low thresholds. By adjusting the threshold values, we can observe how the algorithm's performance changes. The program generates output images for each threshold combination, highlighting detected edges based on the thresholding results.

Results

Edge Detection Outcomes with Different Thresholds:
The experimentation with different threshold values yields diverse outcomes in edge detection. Varying the high threshold and low threshold influences the detection of strong and weak edges. When setting low threshold values, more edges are identified, including weaker ones that might correspond to noise or texture. In contrast, higher threshold values lead to fewer detected edges, as only stronger edges surpass the threshold.

In figure 2 we compare the outputs of the same image with different low and high threshold values, ranging from 0.05;0.05 to 0.08;0.25.

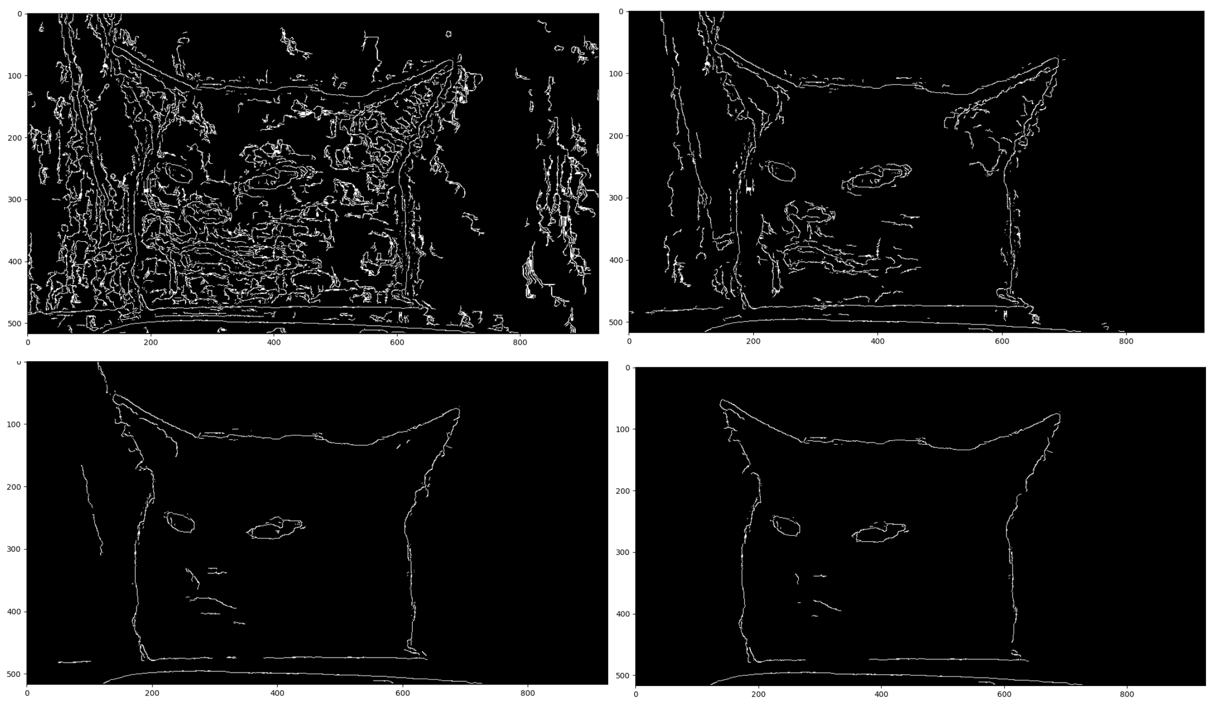

Figure 2: Output images with different low and high thresholds.
Top left: 0.05;0.05. Top Right: 0.05; 0.1. Bottom Left: 0.07; 0.2. Bottom Right: 0.08; 0.25.

When starting with very low threshold values, we have a lot of unwanted edges. While there are some artifacts that have a visible edge in the original background, we can remove it by increasing the thresholds. As the low thresholds increased, there was less noise being considered as an edge, and as the high threshold increased, only the most important edges remained.

The final image with a high threshold of 0.25 and low threshold of 0.08 did meet the requirements, as it only contained the outline of the cat's head and its facial features, while omitting all other minor objects and noise.

Discussion

Limitations of Edge Detection:
The outcomes of edge detection using different thresholds demonstrate both successful and challenging cases. In situations where the threshold values are well-tuned, the algorithm successfully captures prominent edges while excluding noise and less relevant structures. This is achieved by appropriately selecting the high threshold to ensure that only significant edges are considered strong.

However, in cases where the threshold values are not well-balanced, limitations arise. Setting high thresholds may result in the omission of some valid edges, leading to under-detection. Conversely, overly low thresholds may introduce excessive weak edges, causing over-detection and false positives. For an accurate edge detection, each new image would need to have its own custom thresholds for the most accurate edge detection.

Application to my Object Detection Program:
Since the program is able to accurately highlight the edges in an image, I will be using this to isolate the main object in an image for my object detection project. It currently needs to scan through the entire image searching for an object to identify, but I can narrow the range down the particular areas where there is a large concentration of connected edges marking a clear object.

Conclusion

Summary of Findings:
This research paper explored the application of advanced calculus concepts in the Canny edge detection algorithm. The theoretical foundations of multivariate differential and integral calculus were investigated, and relevant references were reviewed and summarized. Through experimentation and analysis, the effectiveness of edge localization via gradient detection was evaluated, and found to be very accurate given proper threshold values.

Future Applications:
By filtering out everything but the main edges of an object in a given image, we are left with a simpler image. This can be useful for object detection programs as you can isolate where the object is in the image based on the edges, and this will narrow down the area an object detection program needs to look through.

An extension for this project would be to implement a version with self correcting threshold values. Based on the distribution and concentration of edges, a program could decide whether the thresholds are too low, and including non essential edges, or too high and removing part of the object outline.

References

[1] Lijun Ding, Ardeshir Goshtasby, On the Canny edge detector, Pattern Recognition, Volume 34, Issue 3, 2001, Pages 721-725, ISSN 0031-3203, https://doi.org/10.1016/S0031-3203(00)00023-6.

[2] J. Canny, "A Computational Approach to Edge Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986, doi: 10.1109/TPAMI.1986.4767851.

[3] W. Rong, Z. Li, W. Zhang and L. Sun, "An improved Canny edge detection algorithm," 2014 IEEE International Conference on Mechatronics and Automation, Tianjin, China, 2014, pp. 577-582, doi: 10.1109/ICMA.2014.6885761.

[4] H. Farid and E. P. Simoncelli, “Optimally rotation-equivariant directional derivative kernels,” *Computer Analysis of Images and Patterns*, pp. 207–214, 1997. doi:10.1007/3-540-63460-6_119

[5] “Canny edge detector,” Canny Edge Detector - OpenCV 2.4.13.7 documentation, https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html (accessed Jul. 27, 2023).

[6] K. Mikolajczyk, “Scale & affine invariant interest point detectors,” *International Journal of Computer Vision*, vol. 60, no. 1, pp. 63–86, Jan. 2004. doi:10.1023/b:visi.0000027790.02288.f2

[7] R. Krish, “Canny Edge Detection,” GitHub, https://github.com/rohit-krish/CVFS/tree/main/Canny%20Edge%20Detection (accessed Jul. 27, 2023).