Detecting Contours of Human Organs in CT Images Using the Canny Edge Detector

Ana Poklukar
OBSS 2024/25 , FRI, UL
ap3956@student.uni-lj.si

***Abstract*—In this project, we developed a program for detecting contours of human organs in CT images from the CTMRI database using the Canny edge detector. A detailed discussion on the performance and applicability of the Canny edge detector for CT image analysis, along with the results, was compiled in a report.**

I. Introduction

The analysis of medical imaging is a cornerstone of modern diagnostics, with edge detection playing a critical role in identifying anatomical structures and abnormalities. Among the widely used edge detection algorithms is the Canny edge detector, proposed by John F. Canny in 1986 [1]. This algorithm is renowned for its ability to detect edges with precision, while maintaining robustness against noise, making it particularly suitable for analyzing complex medical images such as computed tomography (CT) scans.

This project focuses on the implementation of the Canny edge detector from scratch, applying it to the task of detecting contours of human organs in CT images from the CTMRI database. The CTMRI database was developed by the Laboratory for Biomedical Computer Systems and Imaging (LBCSI) [2] at the University of Ljubljana. After implementing the algorithm, we identified its limitations when applied to this specific dataset and proposed enhancements to address these weaknesses. To evaluate the effectiveness of our implementation, the results were compared with those obtained using the Canny edge detector provided by the widely used computer vision library, OpenCV (cv2) [3]. Our findings provide insights into the applicability of the Canny edge detector for medical imaging and demonstrate the impact of targeted enhancements on its performance.

II. Methodology

The methodology of this project involved the implementation of a Canny edge detector algorithm based on the approach described by Canny. The algorithm was implemented in Python. In the following, the key steps of the methodology are outlined.

A. Preprocessing and Gaussian Filtering

An input image was first converted to grayscale to simplify processing. Gaussian smoothing was then applied to reduce noise. A Gaussian filter was created, where the sigma value was set to 0.5% of the minimum image dimension, and the kernel size was calculated as the smallest odd integer greater than 6σ. The Gaussian filter was then applied to the grayscale image to produce a smoothed result, removing noise.

B. Gradient Calculation

To compute image gradients, Sobel operators were used. These operators were convolved with the smoothed image to calculate the gradient magnitude and angle. The gradient magnitude was computed as the square root of the squared gradients in the x- and y-directions, while the angle was calculated as the arc tangent of the gradient components.

C. Non-Maximum Suppression

Using the computed gradient magnitude and angle, non-maximum suppression was performed to thin the edges. This step involved suppressing pixels in the gradient magnitude that were not local maxima along the direction of the gradient, ensuring that only the most significant edge pixels were retained.

D. Thresholding with Improvements

To avoid manually setting the thresholds for hysteresis thresholding, we introduced an improvement by computing thresholds dynamically based on the image. The high threshold was set as

$$\text{mean magnitude} + 2 \times \text{standard deviation},$$

and the low threshold was defined as half the high threshold. This method ensured thresholds were adaptive to the input image.

E. Hysteresis Thresholding

Pixels with gradient magnitudes above the high threshold were classified as strong edges and set to 255, while those between the high and low thresholds were classified as weak edges and set to 75. Weak edges were later connected to strong edges to form continuous boundaries.

F. Edge Linking

During edge linking, weak edge pixels were retained only if they were connected to strong edge pixels. This ensured that only meaningful edges were preserved, and noise or

isolated weak edges were removed. The resulting binarized image was saved for analysis.

To compare our implementation with OpenCV's *cv2.Canny*, the input image was pre-blurred using the same Gaussian smoothing parameters applied in our implementation. Since the threshold parameters for OpenCV's Canny detector differ from ours, we tested a range of thresholds. For the low range, thresholds were varied between 10 and 400 in steps of 10, and for the high range, between 40 and 500 in steps of 10. Performance metrics including F1 score, precision, and sensitivity were calculated for each configuration to evaluate the results. The low and high thresholds that yielded the highest F1 score were then saved and used for further comparison with our Canny edge detection implementation.

III. Results

The results of our edge detection implementation are presented in the following figures and table.

Figures 1 and 2 show examples from testing image "/2/0001.png" from patient 240163-01. In Figure 1, you can see the original image, the image with detected edges after non-maximum suppression, and the final binarized image after edge linking. Figure 2 presents a comparison between the results of our Canny edge detector and the OpenCV Canny edge detector.

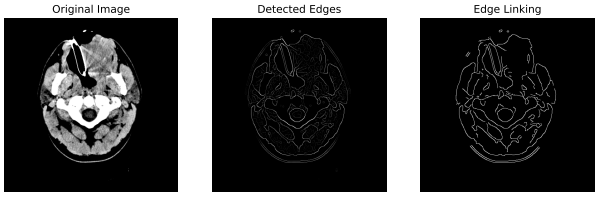

Fig. 1. Example from the CTMRI database showing the original image, the image with detected edges after non-maximum suppression, and the final binarized image after edge linking.

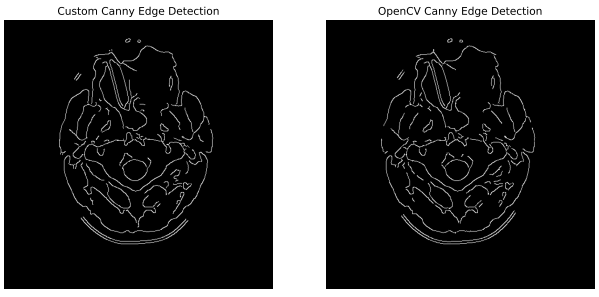

Fig. 2. Comparison of edge detection results for the same example from the CTMRI database as in Fig. 1. The images show the result of our Canny edge detector and the result of the Canny edge detector from OpenCV.

Since testing on the entire database was computationally demanding (the CTMRI database contains around 11,000 images), we decided to use a diverse subset of 100 randomly selected images for performance evaluation. The algorithm was executed multiple times on this varying subset, and the average precision, sensitivity, and F1 score were computed, as shown in Table I.

TABLE I
Average Precision, Sensitivity, and F1 Score for a Subset of the CTMRI Database.

Precision [%]	Sensitivity [%]	F1 Score [%]
83.7	80.6	81.8

IV. Discussion

In this project, we implemented and improved the Canny edge detector for detecting contours in CT images. The algorithm was tested on the CTMRI database, and the results were compared to the Canny edge detector from OpenCV.

As seen from the average precision and sensitivity in Table I, the performance is not as high as expected. Several factors can explain this discrepancy. For instance, the edges detected by our implementation might be slightly offset compared to the OpenCV version, leading to incorrect edge detection. Additionally, the thresholds used in OpenCV's Canny detector did not exactly match those computed in our implementation, as can also be seen in Figure 1.

Despite these challenges, our Canny edge detection implementation works well, especially with the adaptive thresholding improvements. However, it is computationally slower than the OpenCV version, and its performance does not match the efficiency of OpenCV's implementation in terms of speed and accuracy.

References

[1] J. 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] L. for Biomedical Computer Systems and Imaging, "Ctmri database," https://lbcsi.fri.uni-lj.si, university of Ljubljana.

[3] Itseez, "Open source computer vision library," https://github.com/itseez/opencv, 2015.