Detecting contours of human organs in CT images using the Canny edge detector

Biomedical signal and image processing 2020/21, Faculty of Computer and Information Science Ljubljana

Maj Šavli 63150278

***Abstract*—In this seminar we present the Canny edge detection algorithm. We present the algorithm step by step. We show results of applying the implemented algorithm to real images from Tomography-Magnetic Resonance Imaging Database. In the end we discuss the results and some drawbacks of the algorithm.**

I. Introduction

The edge detection is the process of identifying points in a digital image at which the image brightness changes sharply or has discontinuities. It can serve to simplify the analysis of images by drastically reducing the amount of data to be processed, while at the same time preserving useful structural information about object boundaries. The edge detection method we will be addressing in this seminar is called the Canny edge detector [1]. It is one of the most strictly defined methods that provides good and reliable detection. In the next section we will present all steps of the algorithm. In the third section we present the results of applying the implemented detector on images from Computed Tomography-Magnetic Resonance Imaging Database (CT-MRI DB), available at [2]. In the last section we evaluate the results and discuss possible improvements of the detector.

II. Methods

The goal of the Canny edge detector is to minimize the distance between detected edge pixels and real edge pixels, to minimize error (detected edges should be the real edges) and it should not detect false edges, which are not existing. The Canny edge detector can be broken down to a couple of steps, explained below.

A. *Gaussian filtering*

Edge detection results are highly sensitive to image noise. That is why we have to filter the noise out. We can do that by smoothing the image with a Gaussian filter or kernel. Let $f(x, y)$ denote the input image and $G(x, y)$ denote the Gaussian function

$$G(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

We form a smoothed image f_s by convolving G and f

$$f_s(x, y) = G(x, y) \star f(x, y)$$

Filter or kernel can be of different sizes and of different values. The larger the size, the lower the detector's sensitivity to noise. A 5×5 size is good size for most cases. The effects of the Gaussian filtering can be seen in figure 1.

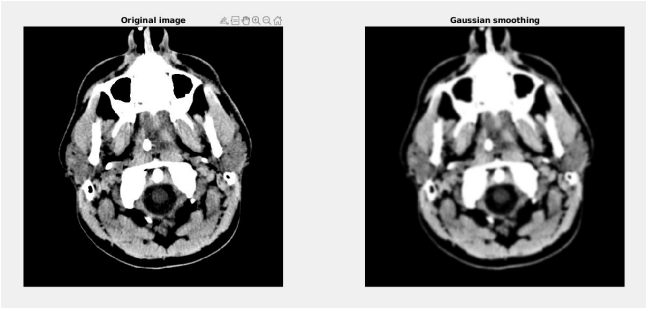

Figure 1: Original image and image after gaussian filtering

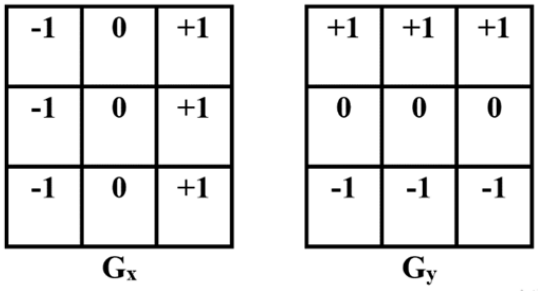

Figure 2: Prewitt mask

B. *Computing the gradient magnitude and direction*

After smoothing the image, we must compute the directions (or angles) and magnitudes of edges in the image. We can do that by first computing the gradient for every pixel in image. We do that by using kernels or masks. There exists many different masks, in this seminar we used the Prewitt mask, which can be seen in figure 2. We use that mask to compute first derivative in horizontal (G_x) and vertical (G_y) direction for every location in image. After acquiring the horizontal and vertical derivative, we can compute the magnitude, given by the following equation

$$M(x, y) = mag(\Delta f) = \sqrt{g_x^2 + g_y^2}$$

The directions or angles can be than computed as

$$\Theta = atan2(G_y, G_x)$$

where $atan2$ is arctangent function with two arguments. The edge direction angle is rounded to one of four angles representing vertical, diagonal and horizontal direction: 0°, 45°, 90°and 135°.

C. *Non-maximum suppression*

Non-maximum suppression is used for edge thinning. With this step, we fulfill the second criteria for the detector, that the edge point detected from the operator should accurately localize on the center of the edge. Non maximum suppression works by finding the pixel with the maximum value in an edge. If a pixel has an intensity that is larger than intensity of both pixels in the gradient direction of q, we keep the pixel, otherwise we set the pixel to zero (make it a black pixel). That means that if we find a pixel on the same edge with higher intensity, the current pixel is probably not the center of the edge. In the Figure below 3, we can see the edge thinning effect.

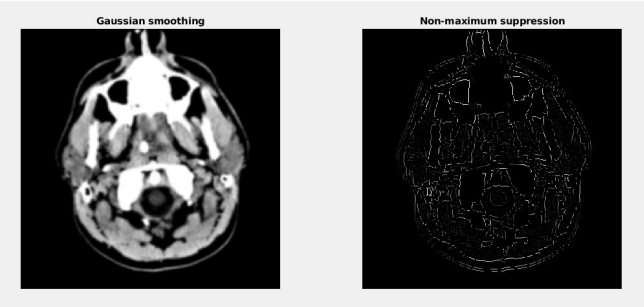

Figure 3: Image after non-maximmum suppression

D. *Double threshold*

After non-maximum suppression, some pixels still remain, being the cause of noise. Using high and low threshold values, we can filter out the pixels with high and weak magnitudes. If an edge pixel magnitude is greater than high threshold value, we mask it as strong edge pixel and if edge pixel magnitude is lower than low threshold value, we suppress it. If an edge pixel magnitude is between high and low threshold value, we mark it as weak edge pixel. The threshold values can be a big factor in the detection quality. If low threshold value is set too low, too much noise will be marked as a weak pixel and possible as an edge pixel in the next step. If high threshold value is set too high, a lot of pixels, which are the actual edge, will be marked as a weak pixel and there will be too little strong pixels for the hysteresis thresholding (next step) to work efficiently. Threshold values we used were $T_{low} = 0.1$ and $T_{high} = 0.4$ and were acquired empirically.

E. *Hysteresis thresholding*

Hysteresis thresholding or edge linking is the step, where we iterate through the weak edge pixels, detected in the previous step. Since the weak pixels are borderline edge pixels, we must make sure they are actually part of some actual edge. We can do that by looking at the weak pixel and its connected neighbourhood pixels. If any of the eight neighbour pixels is a strong pixel, then the weak edge pixel is identified as strong pixel. After iterating through the whole image, we suppress the remaining of the weak pixels, since they were not connected to some strong pixel meaning they are not part of the actual edge. Some of the noise should then be removed and the main edges of the image should be now seen. In the figure below 4, we can see the effect of hysteresis thresholding.

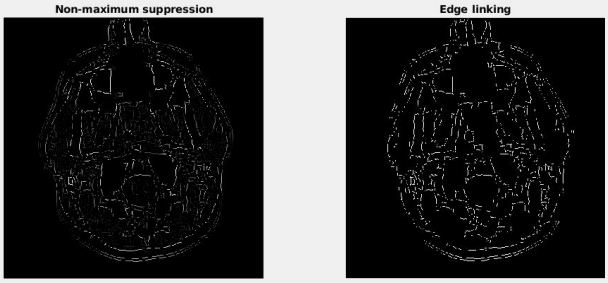

Figure 4: Image after the final step, hysteresis thresholding

III. **Results**

Figure 5: Example images and detected edges

We implemented the algorithm in MATLAB, using inbuilt functions and functions from the image processing toolbox. For computing image derivatives we used Prewitt mask. The low and high threshold for edge linking were acquired empirically and are not optimal for all cases. In the figure

5 above, we can see two example images of human skull and some internal organs alongside with image of its edges, detected with our implementation of the Canny edge detector. We can see how the detector detected the main edges and filtered out gray areas and noise. Some edges may not make a closed circle, but that is also due to empirically set threshold values. Source code available on Github [3].

IV. **Discussion**

The Canny edge algorithm, as we saw, is pretty reliable edge detection algorithm considering its quickness and simplicity. However, it can't handle the challenging edge detection task so well. Moreover, there is a couple of things, that can be improved. For example, the gaussian filter in the first step on the other hand filters out the noise, but it can also smooth out the edge, which increase the possibility of missing weak edges annd appearance of isolated edges in the result. Also, as we said in the previous section, the low and high threshold values for double thresholding are acquired empirically and are global. More complex images may need different threshold values in specific local areas in that image. Overall, the Canny edge detection algorithm is quite adaptable and is still popular and widely used.

References

[1] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[2] "Computed Tomography-Magnetic Resonance Imaging Database," Available at: http://lbcsi.fri.uni-lj.si/OBSS/Data/CTMRI/, [Last visited: 17. 1. 2020].

[3] "Github source code," Available at: https://github.com/saulasciante/obss-izpitni-seminar, [Last visited: 18. 1. 2020].