Analysis of computed tomography (CT) images

Žiga Trojer

Abstract - for the purpose of the second seminar paper in the subject of OBSS, we implemented the procedure of detecting contours of human organs in CT images using the Canny edge detector. The original Canny edge algorithm provides a threshold to be entered as input to the algorithm. We have improved this process by implementing an automatic threshold adjustment process. The algorithm in this case works better than the algorithm with a fixed threshold. Compared to the edge function already implemented in Matlab, our detector works a bit worse, but we are quite satisfied with the results.

Introduction

The edge detector process serves to simplify the analysis of image by drastically reducing the amount of data to be processed, while at the same time preserving useful structural information about object boundaries. In all edge detectors, it is important that edges that occur in the image should not be missed. It is also important that the edges are well localized - that means that the distance between the detected edge and original edge is minimum. This also means that our edges needs to be thin, as we would not like to have too wide boundaries [1].

Methods

We implemented the algorithm in Matlab. We downloaded images of CT scans from the Computed Tomography-Magnetic Resonance Imaging Database [2], where we tried to find CT images of several different human organs. For the analysis, we used several different images - those images represent CT scans of brains, lungs, heart, etc. There was no need to transform the images prior to running the algorithm. The process of implementing Canny edge detection algorithm [3] can be broken down to 5 steps:

1. Applying Gaussian filter to smooth the image.
2. Finding the gradients of the image.
3. Applying non-maximum suppression.
4. Applying double threshold.
5. Tracking edges.

Gaussian filter.

Gaussian filter is applied to smooth the image in order to remove the noise. It is applied to convolve with the image. The

Figure 1. Input CT image of head. Image is already in grayscale, so no prior transformation was required.

equation for a Gaussian filter kernel of size $(2k+1) \times (2k+1)$ is given by equation 1

$$\mathrm{H}_{ij} = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}\right) \quad (1)$$

where $1 \leq i, j \leq (2k+1)$. The size of the Gaussian kernel is very important, since it affects the performance of the detector. Larger the size, lower the detector's sensitivity to noise. Localization error to detect the edge will increase with the increase of Gaussian filter kernel size. Size of Gaussian filter is determined with the size of an image - $\sigma = min(\text{size of image}) \times 0.005$. Parameter k is now calculated from σ - about 3 times the σ size.

Gradient of the image.

After smoothing the image, we need to find the intensity gradients of the image. We filter the image with a Sobel kernel [4] in both horizontal and vertical directions. Sobel convolution kernels S_x and S_y are defined by

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \qquad S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}.$$

One kernel is the other rotated by 90 °. In that way, we get first derivative in horizontal direction $G_x = \mathrm{S}_x * \mathrm{A}$ and first derivative in vertical direction $G_y = \mathrm{S}_y * \mathrm{A}$. Edge gradient G and direction Φ of each pixel is then calculated by the following formulas:

$$G = \sqrt{G_x^2 + G_y^2}, \qquad \Phi = \tan^{-1}(\frac{G_y}{G_x}).$$

We rounded each direction angle to one of four angles: 0°, 45°, 90°and 135°. Those angles represents vertical, horizontal and two diagonals.

0.1 Non-maximum suppression.

This is an edge thinning technique. It is applied to find the locations with the sharpest change of intensity value. For each pixel in the gradient image, we

- compare the edge strength of the current pixel with the edge strength of the pixel in the positive and negative gradient directions,
- if the edge strength of the current pixel is the largest compared to both, the value is preserved. If not, the value put to zero (suppressed).

Figure 2. CT image after detecting edges.

Applying double threshold.

After the non-maximum suppression, we need to suppress some pixels which remained from the noise. We do this by using the double threshold. If the pixel gradient is higher than the upper threshold, we mark the pixel as a strong edge pixel. If it is below the lower threshold, the pixel is suppressed. Otherwise, the pixel is marked as a weak edge pixel. The original algorithm uses a fixed threshold, which should be determined empirically. It was recommended that the upper:lower ratio is maintained between 2:1 and 3:1 for good results. The choice of thresholds was then improved by automatically adjusting to the image. It is calculated by calculating the average value of the image (pixel intensity between 0 and 255) and then subtracting / adding one standard deviation. We then multiply this value by the selected constant and this is the basis for for the lower and upper threshold.

Tracking edges.

The strong edges are considered as a true edges. The weak edge pixels obtained from the above thresholding are classified as edges or non-edges based on their connectivity. We check if they are connected to a strong pixel - we use 8-connectivity.

Results

We tested the algorithm on as many images as possible, representing different parts of the body and organs [2]. With this approach, we wanted our algorithm to be robust. Our algorithm finds the correct edges, but not all of them. How many edges the algorithm detects depends very much on the choice of threshold. In the first implementation of the algorithm, we used a fixed threshold: the lower threshold was 0.060 and the upper was 0.160, since this gave the best results. Those thresholds were determined empirically. Figure 3 shows the CT image after edge linking. It detected strong edges very good, but some edges were left out. For example, there is no edge of the outer bound. Inner edges were successfully detected, also the edges are thin and partially connected.

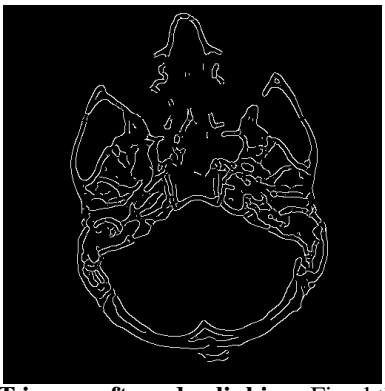
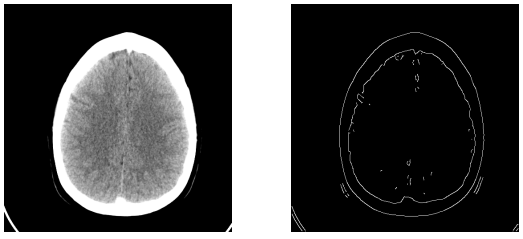


Figure 3. CT image after edge linking. Fixed threshold of 0.060 and 0.160 was used.

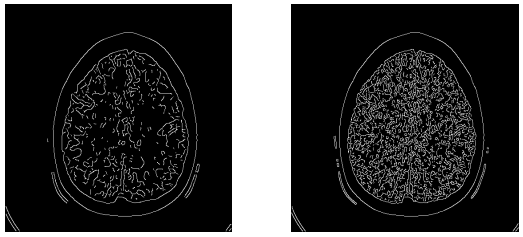
Our algorithm works poorly in the case where there is a large contrast in the original image, but not between the edges and between the others. One such example is the Figure 4.

The problem here is to set a threshold in advance. The optimal threshold, of course, varies from image to image. For this purpose, we improved the algorithm by implementing an adaptive threshold. This is based on the average and standard deviation of the gray intensity in the image. The formula for the threshold is: threshold $= \mathrm{C}(\mathrm{mean(I)} \pm \mathrm{std(I)})$, where C



(a) Original CT **(b)** After edge linking

Figure 4. CT image where the algorithm fails to detect important edges.



(a) Using $k = 4$ **(b)** Using $k = 2$

Figure 5. CT image after linking using improved algorithm and two sizes of Gaussian kernel. First one is using σ to determine size, the other one is determined empirically.

is constant determined empirically. The improved algorithm thus works better, as evidenced by the examples on Figure 5.

It is worth mentionin that edge detection in CT images of the brain works better when we choose a smaller Gaussian filter. We compared two sizes - for $k = 2$ and k, which is calculated from the size of the image. Figure 5 shows the detected edges using different sizes of Gaussian filters. k for the left one is calculated from the size of the image the right one uses $k = 2$. The second Gaussian filter is approximately once smaller than the first one. Second one works better, since it takes smaller neighborhood for each pixel - the result is very similar to a result of Matlab function 'edge'.

Figure 6 shows the CT image of a head after edge linking using adaptive threshold. It is obvious that this algorithm works much better than before. A lot more edges were detected and the edges stayed thin. A lot more information is saved with this algorithm. We also want to compare our algorithm to the algorithm that Matlab uses for detecting edges. This function is 'edge' function with an argument 'Canny' - Figure 7. We see that those two algorithm now work very similar, Matlab function returned a bit better results. Both algorithms now distinguish noise well from data and detects edges well.

Figure 6. CT image after linking using improved algorithm. Adaptive threshold was used.

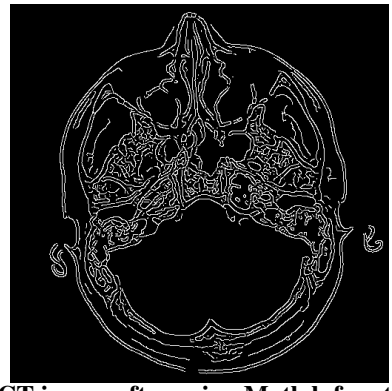


Figure 7. CT image after using Matlab function 'edge'.

Discussion

We were initially satisfied with the implementation of the algorithm, but when we compared it with the Matlab function, we wanted to improve this algorithm. We are very pleased with the improved algorithm, as we did not significantly increase the complexity of the algorithm, and we greatly influenced the quality of the output images. Other improvements would be possible, such as choosing a Gaussian filter, depending on what is in the picture. Thus, a smaller Gaussian filter would be used for CT of the brain and a larger one for other images. This could contribute to the quality of the detected edges.

References

[1] John Canny. A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-8:679 – 698, 12 1986.

[2] Computed tomography-magnetic resonance imaging database. https://lbcsi.fri.uni-lj.si/OBSS/Data/CTMRI/. Accessed: 2020-12-23.

[3] Summer school session 3: Features - corner and edge detection. https://iitmcvg.github.io/summer_school/Session3/. Accessed: 2020-12-23.

[4] Sobel edge detector. https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm. Accessed: 2020-12-23.