EE 417 Computer Vision

Berkant Deniz Aktaş

19515

Homework 1- Edge Detection

October 23, 2018

Sabancı University

Faculty of Engineering and Natural Sciences

Computer Science and Engineering

INTRODUCTION

This report is written for EE 417 Computer Vision Homework 1. Aim of the homework is implementing various edge detectors, comparing them and changing the parameters in order to get familiar with MATLAB Image Process toolbox and understand the topic.

Prewitt, Roberts, Canny, Laplacian of Gaussian(LoG) edge detectors will be used for processing the images. There will be general comparision of the filters in the first part then various functions will be used in order to interpret the images.

APPLYING FILTERS

In this part, different kinds of images will be used for edge detection and methods will be compared.

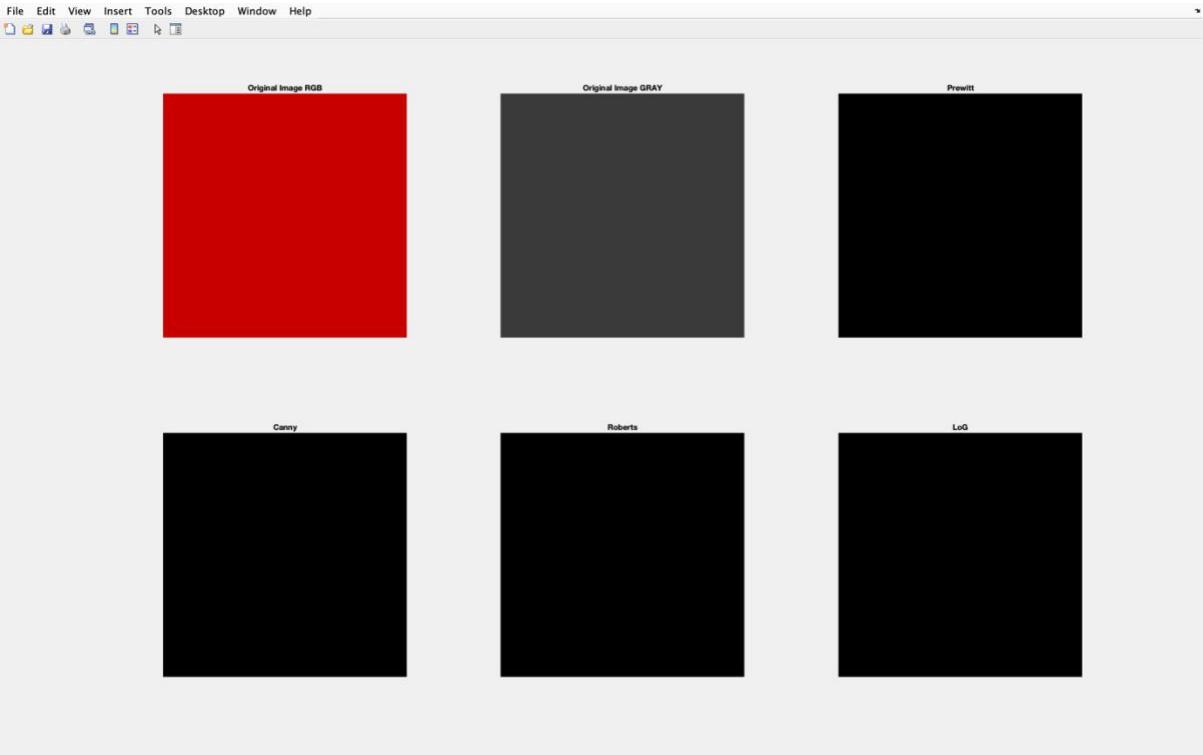

Figure 1. Edge detection on ‘red.png’

As its shown in the Figure 1. all of the edge detectors can not find any edged on a flat red image. It is an obvious and expected results because intensity values of the all pixels are same and detectors can not find any change between pixels. Thus, no edges detected in this case.

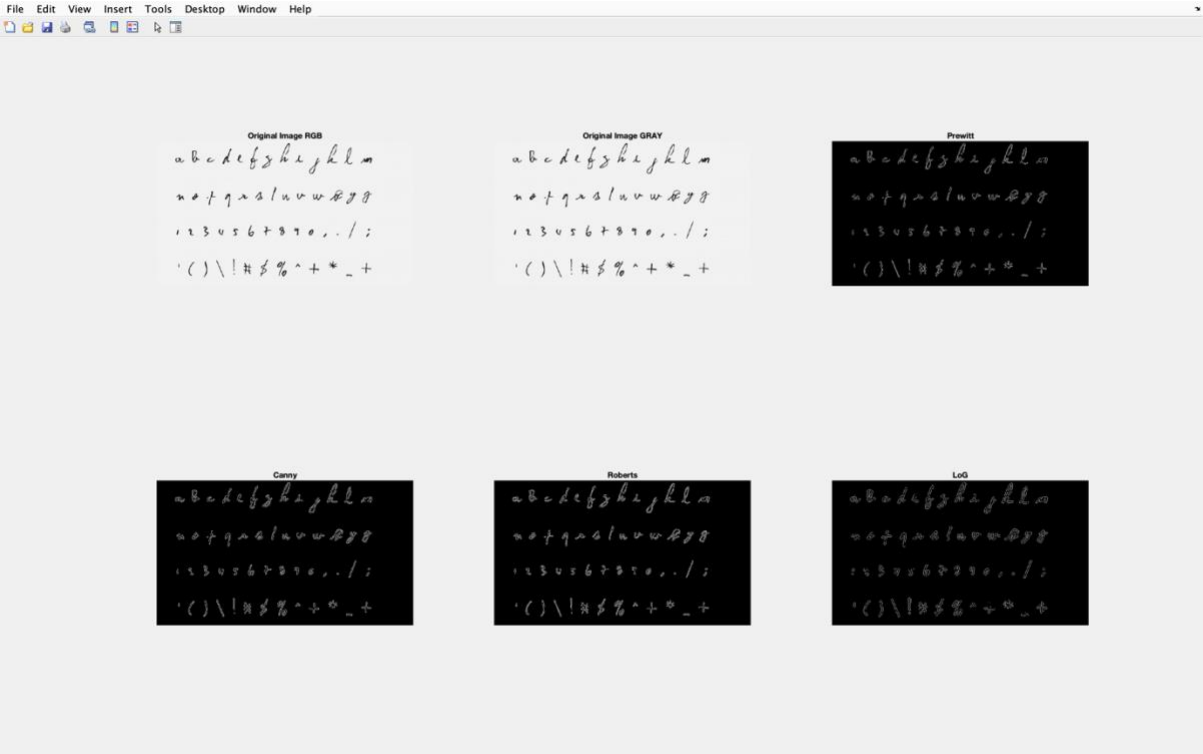

Figure 2. Edge detection on ‘alphab.jpg’

Figure 2. shows the performance of the edge detectors against an image which has white background and black text is written on it. As it is expected all filters performed well and found the edges. One can observe LoG detector and see thickness of the edges are thinner comparing the other images. It may be related with second derivating of the image.

Also, Canny detector has thinner edges because of non-maximum suppression feature of the detector.

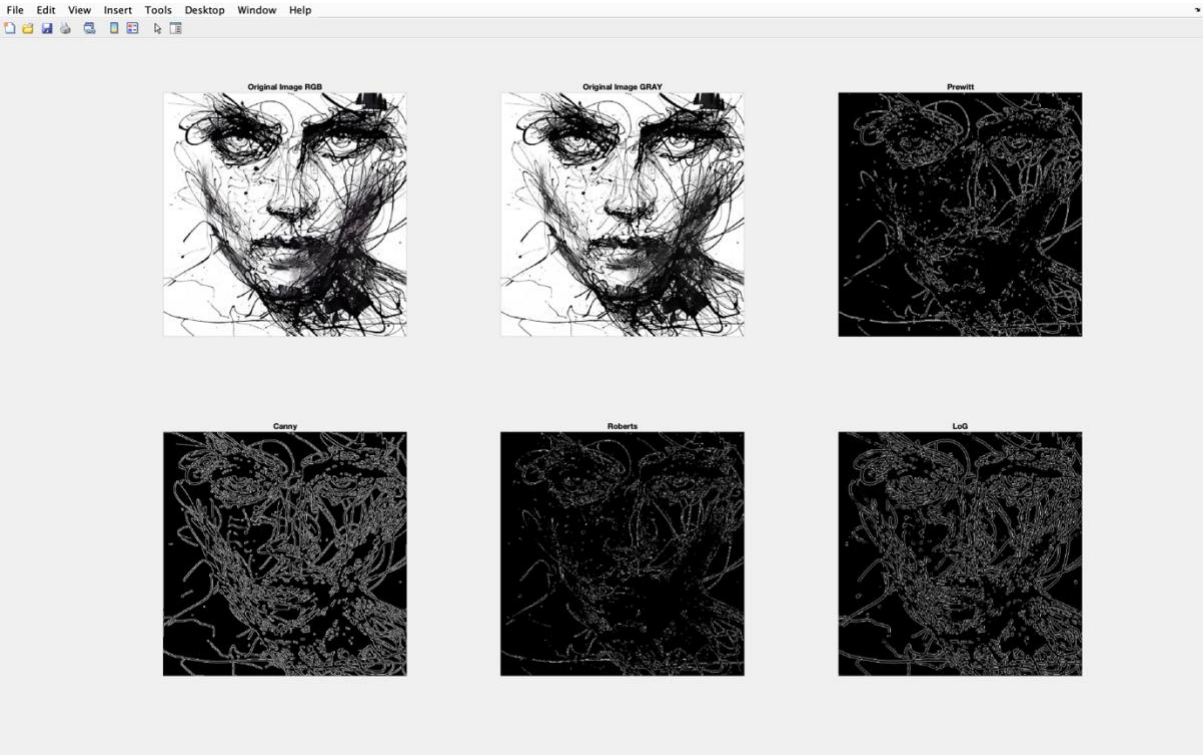

Figure 3. Edge detection on ‘sketch.jpg’

Another kind of image is drawing. As one can guess when you draw something you are drawing the edges if you are not painting. Expected outcome is finding the edges, lines in this case, and not finding the filled parts of the image. As its expected Canny detector finds strong edges and detects them. Roberts and Prewitt detectors are not performing well in this case. The common thing about Robert, Prewitt and also Sobel detectors are using approximation based on their kernel matrix and having local maximum of the gradient.

It may be better to use Canny or LoG in this case because of their computationally advanced methods. Note that, Canny also uses a kernel (Gaussian) in order to detect but due to two threshold values it is more resistable and consistent.

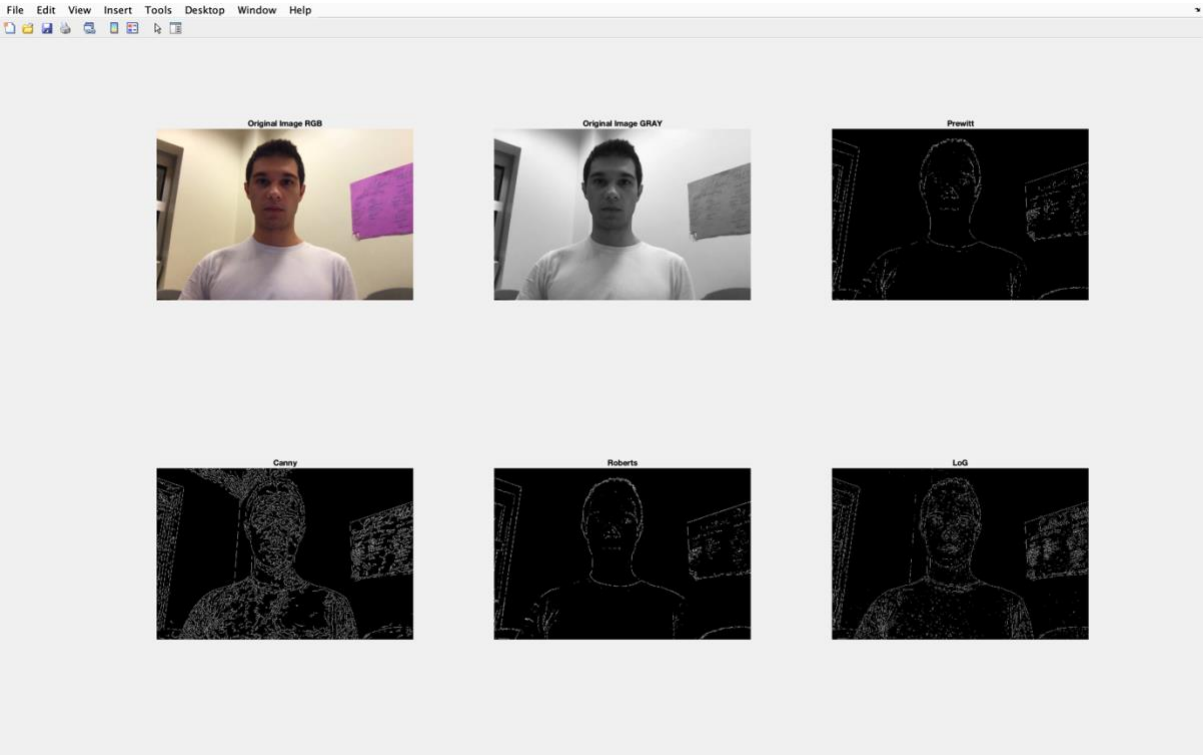

Figure 4. Edge detection on ‘berkant.jpg’

As its shown in the Figure 4. Canny and LoG found more edges comparing with Roberts and Prewitt. Finding more edges may mislead us sometimes. In this case, Prewitt and Robert found strong edges that are desired. Since Canny and LoG found lots of edges due to nature of the image, one can derive an idea to play with the parameters of these filters to understand and improve the results. As its shown in the previous examples, edge detectors behave differently based on the input image and its nature.

PLAYING WITH PARAMETERS

In this part of the report, parameters of the edge detectors will be tested on inputs and they will be compared. Since the base of Prewitt and Robert detectors are similar similar codes will be used to test for different parameters.

PREWITT

The idea of Prewitt detector is having horizontal and vertical operators, which are 3x3 windows. Moving the window and computing the difference between 1st row or column and 3rd row or column depending on the filter. Since middle row or column will be zero, the operator is actually finding the derivative of the window.

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Horizontal and Vertical Operators, y and x, is stated above. MATLAB does all the work for us but since Sobel edge detector is implemented in the lab, I tried to change the mask and apply Prewitt operator by hand for horizontal and vertical edge detections. Normally function returns the combination of horizontal and vertical edges but with a parameter one can achieve both directions seperately. Here is the code I wrote for Prewitt edge detection.

```
function [img2,img3] = prewittT(img)

[row, col, ch] = size(img);
if(ch==3)
   img = rgb2gray(img);
end
img = double(img);
K=1;
x = [ -1 0 1 ; -1 0 1; -1 0 1 ]; % for vertical edges
y = [ -1 -1 -1 ; 0 0 0; 1 1 1 ]; % for horizontal edges
for i=K+1:1:row-K-1
   for j=K+1:1:col-K-1
      subImg = img(i-K:i+K,j-K:j+K);
      valueX = sum(sum(subImg.*x));
      valueY = sum(sum(subImg.*y));
      img2(i,j)= valueX;
      img3(i,j)= valueY;
   end
end
```

```
img = uint8(img);
img2 = uint8(img2);
img3 = uint8(img3);
figure;
subplot(1,3,1);
imshow(img);
title('Original Image')
subplot(1,3,2);
imshow(img3);
title('Prewitt Horizontal Image')
subplot(1,3,3);
imshow(img2);
title('Prewitt Vertical Image')


end
```

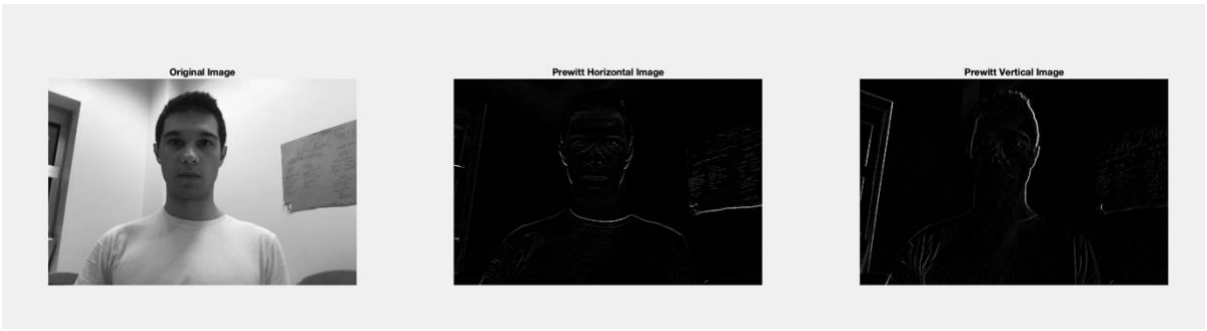

Figure 5. Shows Prewitt with manuel implementation.

Somehow, manuel implementation of the Prewitt found the edges but the difference of the builtin Prewitt detector and my implementation is obvious. This led me to check the functions implemented in Image processin toolbox of the MATLAB from

https://www.mathworks.com/help/images/ref/edge.html

According to function defination there are some parameters that can be applied to Prewitt detector as: Threshold, thinning and directions. The idea of direction is trivial and can be computed with using only one mask, so I did not test this attribute. I tested same image with different threshold values.

```
function [img1,img2,img3,img4] = prewittTest(imgRGB)
```

```
[row, col, ch] = size(imgRGB);
if(ch==3)
    img = rgb2gray(imgRGB);
end
img = double(img);
img1 = edge(img, 'Prewitt', 1);
img2 = edge(img, 'Prewitt', 10);
img3 = edge(img, 'Prewitt', 50);
img4 = edge(img, 'Prewitt', 60);
% OBS HEAD THICK
% 0-1 threshhold ?
img = uint8(img);

figure; % Compare different thresholds
subplot(2,3,1);
imshow(imgRGB);
title('Original Image')
subplot(2,3,2);
imshow(img);
title('Grey Image')
subplot(2,3,3);
imshow(img1);
title('Image with treshhold = 1')
subplot(2,3,4);
imshow(img2);
title('Image with treshhold = 10')
subplot(2,3,5);
imshow(img3);
title('Image with treshhold = 50')
subplot(2,3,6);
imshow(img4);
title('Image with treshhold = 60')

img5 = edge(img, 'Prewitt','nothinning');
img6 = edge(img, 'Prewitt');
figure;
subplot(1,2,1);
imshow(img6);
title('Default prewitt')
subplot(1,2,2);
imshow(img5);
title('Default prewitt with no thinning')

% direction could be defined but manuel implementation.
end
```

The idea is testing the image with different thresholds, while playing with the parameters I realized It should variate between 1-70 in this image. After incrementing threshold slowly, I discovered that after a point edges are gone. It is because of the nature of the image.

In this example, the most important and high valued edges are lost after threshold 70.

This maximum value is strongly depended on the image and having a black and white image will probably increase the value.

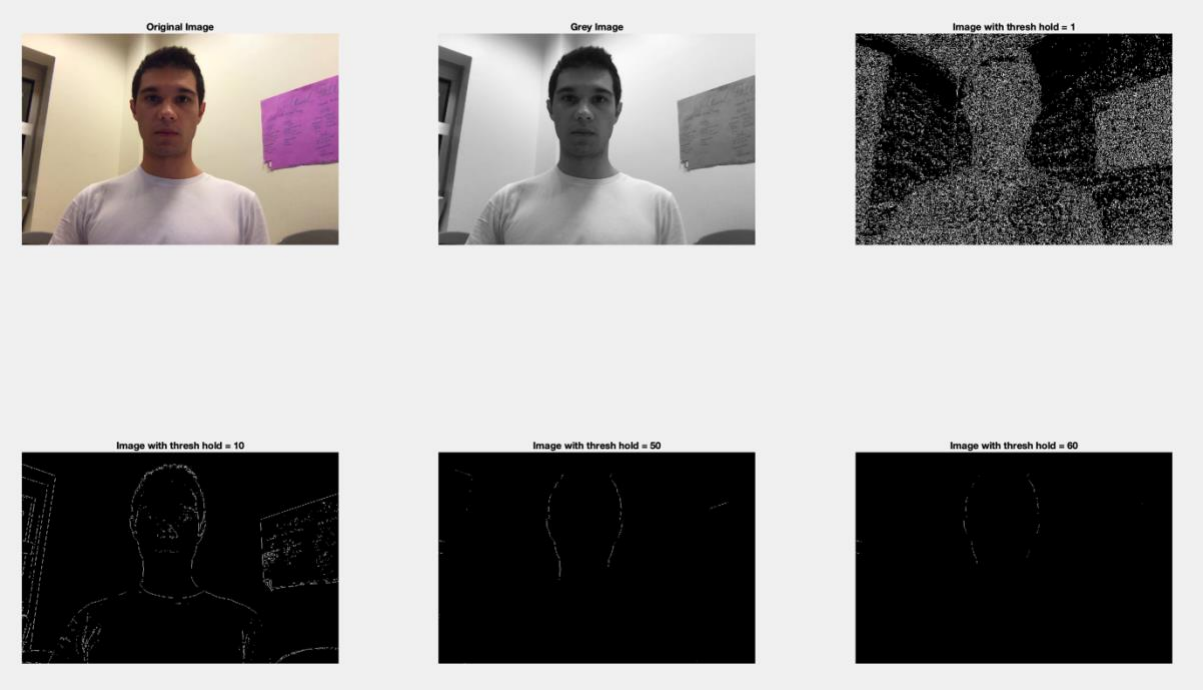

Figure 6. Shows Prewitt detector with different thresholds.

Having threshold value 1 cause the image to detect lots of edges. It is because of different kind of noise in the image such as shadows and illumination based. Another observation is the edges of the my head are still visible. I think this is because of black hair and white background and shadows on my face. Another function I tried is using ‘nothinning’ function in order to see how detector thins the lines.

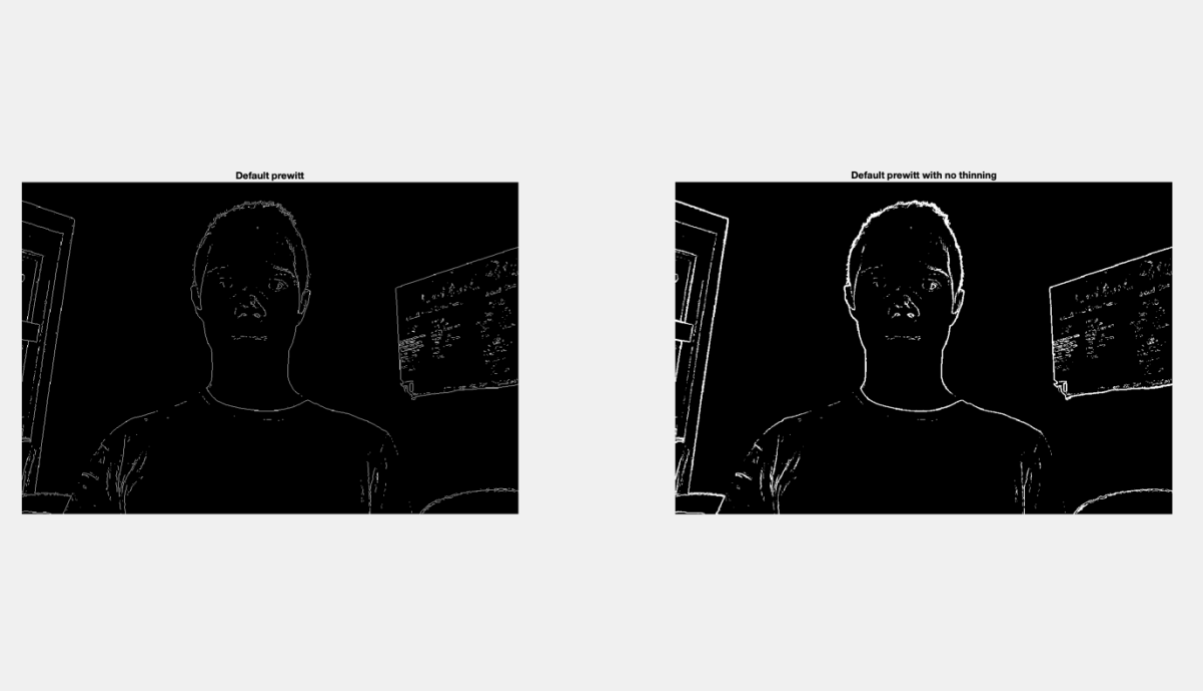

Figure 7. Shows the Prewitt detector without any threshold parameter but ‘nothinning’ parameter.

With comparing Figure 7. and Figure 6. one can guess the default threshold value of the builtin Prewitt detector as 10 or very close number by looking 4th image in the Figure6. Having thick lines as its shown in the Figure 7. may cause some edges to get lost in the image because of others thickness. If we compare the top of hair we can see this result

Left image has more detailed thinner edges comparing with the right image. Having thin lines are beneficial because future algorithms may need thinner edges e.g corner detection.

ROBERTS

Roberts detector is very similar with Prewitt or Sobel, I found two implementations of Roberts, one is having 2x2 windows and other one is having 3x3 window.

I tested Roberts detector with a similar code like Prewitt detector and applied same filters in order to see the difference.

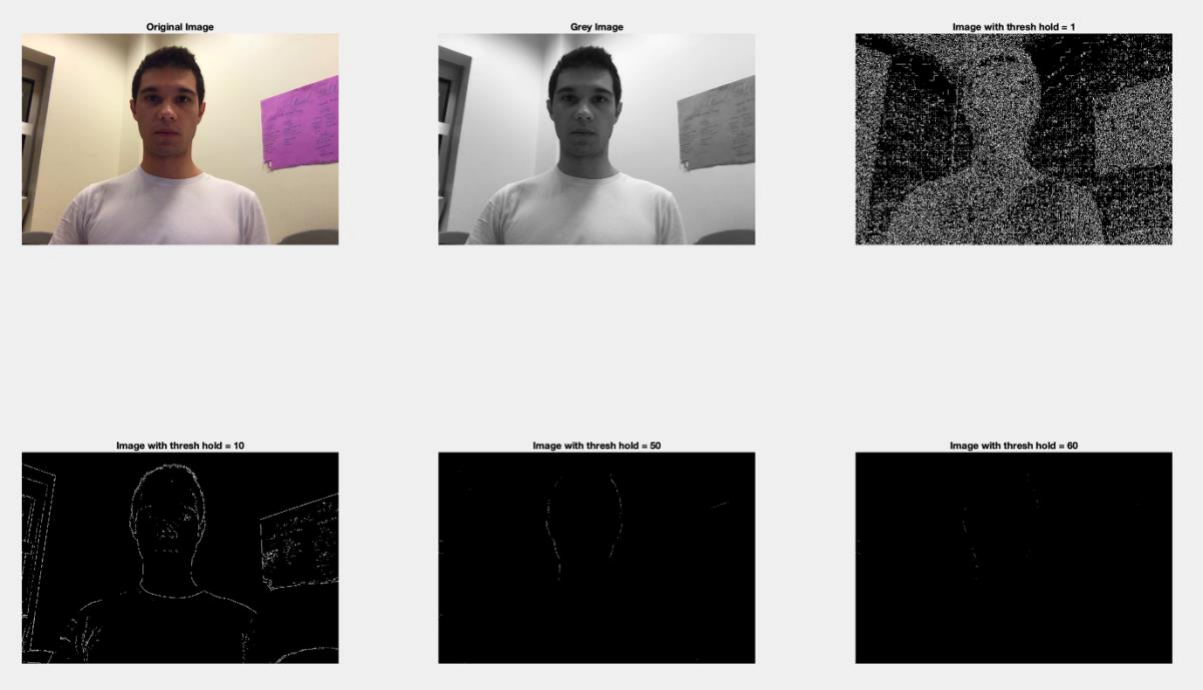

Figure 8. Shows Roberts detector with different thresholds.

As its expected the results are similar with Prewitt detector but threshold value for edges differ. Comparing the results; when having thresholder 60, prewitt is more succesfull detecting the edges. So, We can conclude that Prewitt works better than Roberts in this image. On the other images, Roberts may beat Prewitt.

I also tested thinning parameter on Roberts edge detector. Figure 9. Shows the results of the edge detected image.

Figure 9. Shows the Roberts detector without any threshold parameter but ‘nothinning’ parameter.

CANNY

It can be concluded as a best detector among others because of its complex and computationally good nature. It is also known that It is robust and work pretty well on the noisy images. As it is described in the lecture, the idea is first implementing gaussian smoothing in order to reduce noise of the image. Second step bases on taking the gradient of the image in order to find edge strength. This can be computed with using sobel mask which we implemented in the lab. The sum of the magnitues horizontal and vertical gradients at each points will give the edge strength of the image. Third, detectors finds the gradient direction with using some geometric operations. Fourth part is classifying this edge direction and setting the direction to 45, 90 or 135 degrees. Fifth part is using non-max supression to detect the local maximum and finding the values between two thresholds in order to supress them. Last parts are classifying the edges, If a pixel has higher value than high thresh hold, we can directly call it an edge then any pixels which are connected to our edge pixel will be traced until its value is less than T2. The last part is about having strong edge pixels and candidates which may selected as an edges. With these five operations the edges will be detected and most importantly they will be thin due to non-max supression. As stated before having thin edges are beneficial for future algorithms.

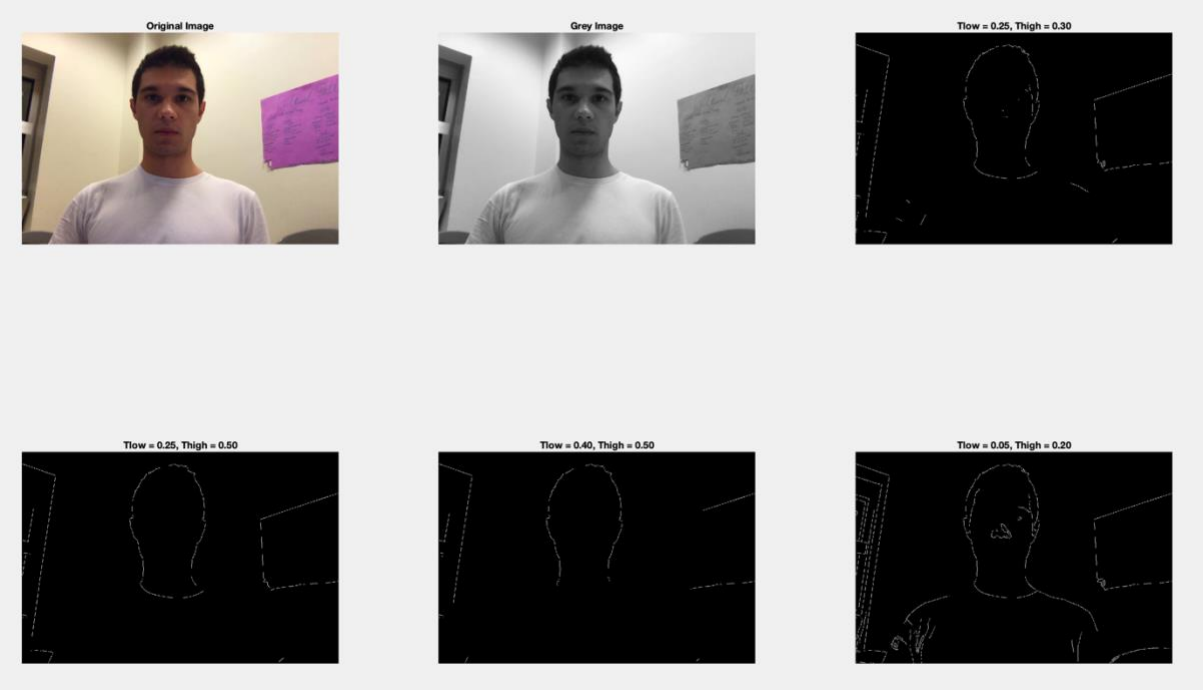

Figure 10. Canny implemented with different thresholds.

As its shown on the Figure 10. I tried different threshold values. First I tried 0.25 as low threshold and 0.30 as high threshold. For testing purposes, I incremented high threshold to 0.50 and did not change low threshold. As I expected the number of edges are lowered because detector did not classify pixels lower than high threshold. The next thing I tried was the opposite of previous experiment. I incremented low threshold to 0.40 and did not change high threshold and hold it at 0.50. Comparing the images I can conclude that the number of edges are again decreased but the reason is different in this case. After finding the edges which are higher than 0.50 threshold, detector tried to trace the pixels which has a value between 0.40 and 0.50. Since lower threshold is 0.40 It could not took the edges below this threshold. Previous value was 0.25, comparing the image 4 and 5 we can conclude that the difference is the edges which are formed from the pixels with values between 0.25 and 0.45. Both experiments with changing thresholds resulted losing edges but their reason and logic is different from each other. After testing different values for thresholds I found that having

0.05 as low, 0.20 as high threshold is suitable for this image.

The second parameter that I played is sigma parameter. According to definition in the MATLAB image processing toolbox functions, sigma is standart deviation of the Gaussian filter. So changing the sigma value will be result us different smoothed image after first part of the algorithm which will led different results while detecting the edges. As its stated in the defination normal Canny detector uses sqrt(2), 1.41, as default sigma value.

Figure 11. Shows different sigma values for image derived from Figure 10.

After finding good threshold values I tried to change the sigma value of the detector.

As I was expecting more edged are detected for smallar sigma values. Sixth image in Figure 10. has default sigma value which is around 1.41. Figure 11. Images has smallar sigma values. One can easily guess that having a smaller sigma value will reduce the smoothing of the Gaussian Smoothing and more edges could be detected after the algorithm terminates.

Figure 11. shows the results for this statement, when sigma is 0.1 more edges are detected and the edge density decreases when sigma is inceremented. The problem is noisy images. Without proper smoothing (sigma lower) with Guassian Smoothing, finding edges of the noisy images will be problematic. Thus, one should find good sigma and threshold values in order to get full performance of the Canny detector. Default parameters are working really nice but may led different results depending on the noise.

LAPLACIAN OF GAUSSIAN

Unlike other detectors LoG works with the second derivative. If we combine our computer vision and calculus experience, we can find that having second derivative will give us the changes on image. Thus, we can find edges who are going inside and outside, inward and outward. Instead of having horizontal and vertical masks, LoG has one positive and one negative mask which 3x3 window and has a -+4 in middle, +-1s at adjacent cells. Idea is applying one filter to image and based on the applied filter we can add or substract the image after applying it. Before doing this operation we need to smooth the image with gaussian in order to prevent problems which may be caused by the noise. So one can combine(convolution) Gaussian Smoothing with Laplacian operator and have a different kernel to use.

```
function [img1,img2,img3,img4] = logTest(imgRGB)
[row, col, ch] = size(imgRGB);
if(ch==3)
   img = rgb2gray(imgRGB);
end
img = double(img);
img1 = edge(img, 'log', 1);
img2 = edge(img, 'log', 2);
img3 = edge(img, 'log', 3);
img4 = edge(img, 'log', 4);
```

```
img = uint8(img);

figure;
subplot(2,3,1);
imshow(imgRGB);
title('Original Image')
subplot(2,3,2);
imshow(img);
title('Grey Image')
subplot(2,3,3);
imshow(img1);
title('sigma=1')
subplot(2,3,4);
imshow(img2);
title('sigma=2')
subplot(2,3,5);
imshow(img3);
title('sigma=3')
subplot(2,3,6);
imshow(img4);
title('sigma=4')

end
```

I tried implementing it different sigma values. According to MATLAB Image Processing tool box, it defines the standart deviation of LoG and by default it is 2 and the window size of the filter is based on sigma which can be computed as n = ceil(sigma*3)*2+1.

One can conclude that as we increase the sigma value the image will be better smoothed and the number of the edges are decreased in this case. Increasing the sigma will be benefical for highly noisy images in order to reduce to noise and find the edges with less error.

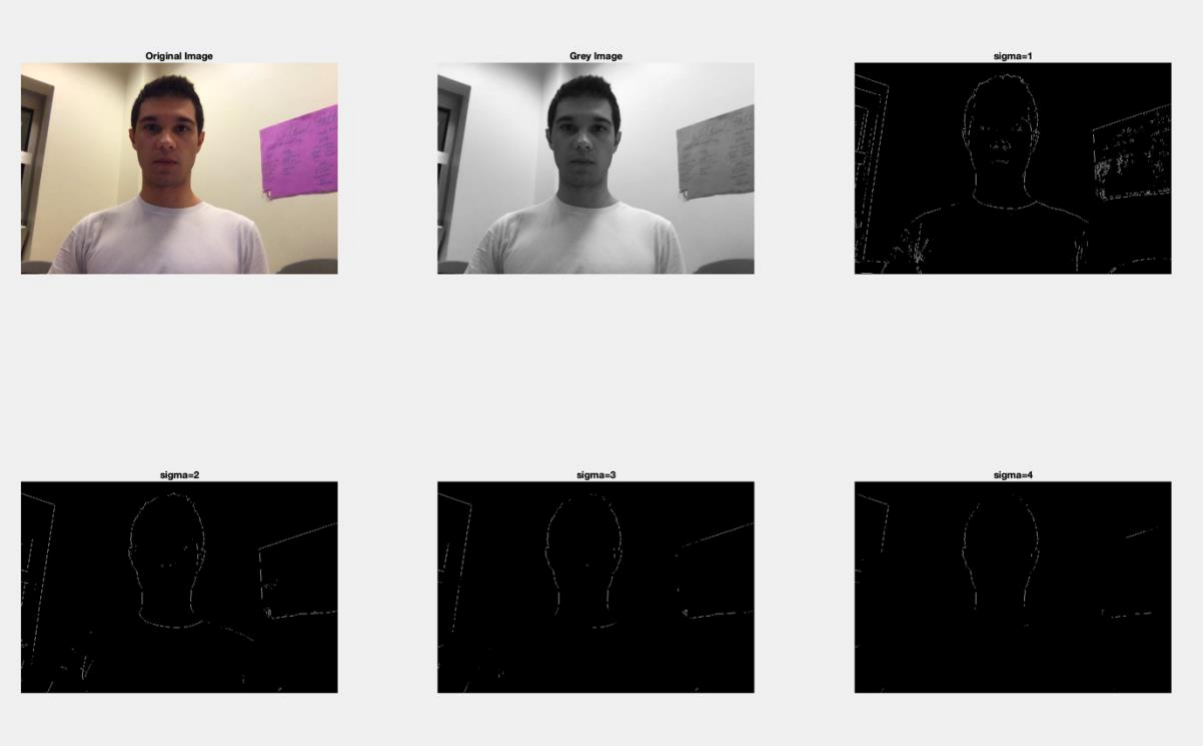

Figure 12. Shows LoG with different sigma values

As I expected, the number of edges decreased with the incrementation sigma value.

But one question that should be discussed is the builting sigma value. In Figure 4. And 4th image in Figure 12. According to MATLAB website default sigma is 2, but one can easily see that the resulted pictures are different from each other. There is a missing part with the defination of treshold of LoG in MATLAB. I tried to check formulas from papers but could not find where and why treshold value is used. First idea is having different Laplacian Approximation filters but I think I am not experienced about image processing to make comments about this topic.