Lab #3 Post-Lab Report

pciftcioglu@sabanciuniv.edu

November 2020

1 Introduction

In this lab, we tried to do several operations with related to edge detection such as Sobel filtering, Prewitt filtering and Laplacian of Gaussian filter. All of the operations have been performed on gray-scale images and has been checked whether if they are gray-scale before each process and if not, they have been transformed. To avoid any confusion, during the rest of the paper, the images that we will use our operations on will be referred as "Original Image", and images we will be creating after processes will be referred as "Resulting Image". Initially, the algorithms for each operation that we have done in the lab will be explained with the methods used and accordingly, further trials with different parameters (e.g. different window size) or different images will be discussed with my own results.

2 In Lab Implementations

2.1 Sobel Filtering

As we have done in the previous lab, we know that Sobel filtering is a discrete 2D derivative local operation that returns the first derivatives of the image along x and y directions with using the following kernels:

$$xFilter = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, yFilter = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

However, different from the previous lab, we also binarized the results of calculated gradients (first derivative of the pixel values for x and y direction) of the image by a threshold value to make the edges of the objects in the image appear clearer than last time. To do so, steps were implemented as following:

First, we used the same function that we have used from the previous lab "lab2sobelfilt" which takes an image and returns the x and y gradients of the image's pixel values (with window size k = 1, 3x3). Right after we converted the return values back to the double form since it was giving the 'uint8' image format. After that, we calculated the gradient magnitude for every index in the matrix that "lab2sobelfilt" returned using the formula below:

$$G(p) = \sqrt{G_x(p)^2 + G_y(p)^2}$$

After that, for binarization, we created an empty matrix of zeros as the same size as the magnitude of gradient matrix and checked whether the value at every index is bigger then the threshold value which was T = 100 in our case. If so, we flipped the value of the index to 255 from 0 (to make it appear as a white pixel).

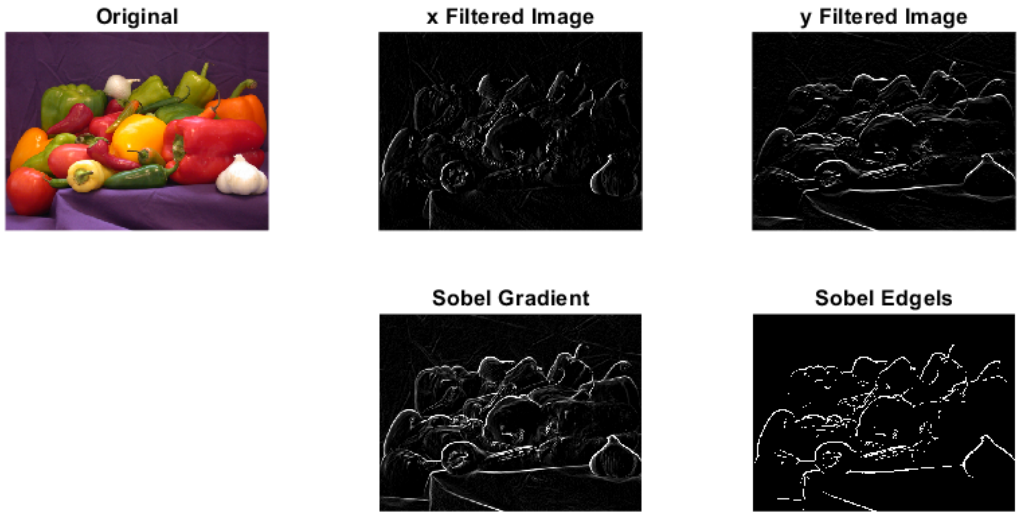


Figure 1: Sobel Filters, T = 100

As it can be seen from the 1 that, even though the edges of the objects in the x Filtered and y Filtered image are visible, the edges overall are not as clear. And when we look at the Sobel Gradient Image we can see that the details are way more visible as nearly seems like a combination of both x Filtered and y Filtered. However, since we only want to detect the edges of the objects within, as an end result, we achieved that purpose and Sobel Edgels image in 1 is that the edges are visible enough that we can see where the distinction between different objects. We will see if we can do better than this in the following experiments.

2.2 Prewitt Filters

Prewitt filtering is another discrete 2D derivative local operator which is also used for edge detection but uses different kernels as following:

$$xFilter = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, yFilter = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

For this function, first we initialized two matrices with all zeros as the same size of the original image and selected window size as 3X3 (k=1) again. As we did in the previous lab with sobel filtering, for each window we again calculated the multiplication of x Filter (for x values) and y Filter (for y values) with the pixel values in the corresponding window and inserted the values to the newly created matrices.

Then, we again did the same procedure like the previous operation that we calculated gradient magnitude (G_{mag}) and initialized an empty matrix as size G_{mag} and changed the pixel values to 255 for every value int the matrix that are greater then the threshold, again T = 100 for our case.

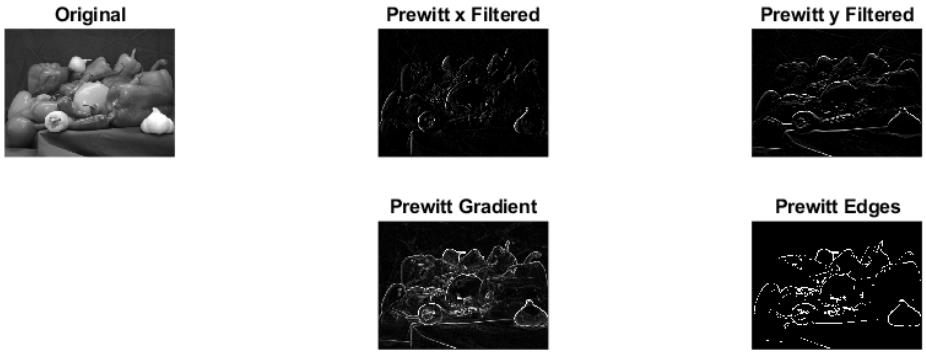


Figure 2: Prewitt Filters, T = 100

On the first look of the results of the Prewitt filters in Figure 2, it is nearly the same results as in the Figure 1 and it can be said that it again was mostly successful for detecting the edges of the objects but a bit hard to see the difference. However, when taken a close look, it can be seen that in Prewitt Gradient, the resulting lines are more detailed and clear compared to previous one and Prewitt filter was more accurate in detecting the edges of the certain objects as it can be seen from Figure 3.

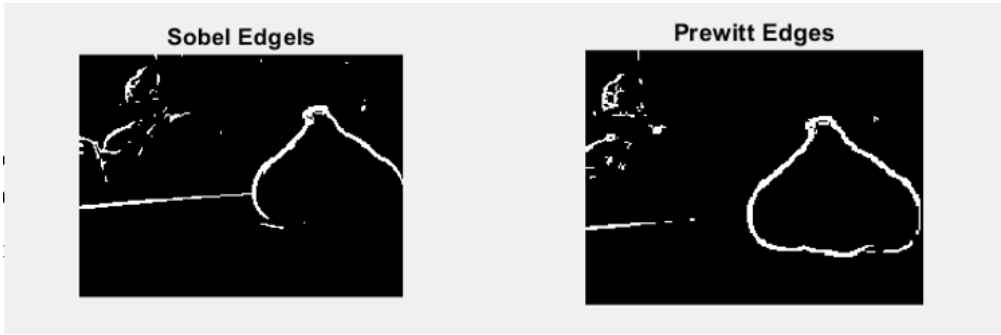


Figure 3: Sobel Edges and Prewitt Edges, T = 100

2.3 Laplacian of Gaussian

As it is described in the lecture Lapplace operator is also used for detecting the edges of the objects and also the noise in the image but, since it is desirable to do so, we also applied a Gaussian filter to first smoothen the image. And required steps has been implemented as following:

We used 2D Gaussian Filter with a built-in Gaussian filter function with $\sigma = 1.1$. After returning a (relatively) blurred image, we performed the Laplacian operation with convolution on the image using the kernel below:

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Then, we initialized a new matrix with all zeros as the same size as the original image, and k = 1 for the window size 3x3. After that, as described in the pseudo code in the lab document we checked the following conditions and implemented the related operation for every (sliding) window in the image. As an important note, to check whether the sign change between the pixel and the related neighbor in the operations we multiplied their values and checked whether if its greater than 0 or not. Since if they are both in the same sign the result of the multiplication would be positive and if they are different the result would be negative. After checking the related conditions we assign the pixel's values that are suitable to 255 (white) on the final image matrix. In addition threshold values are used as T1 = 1 for the first condition and T2 = 2 for the second condition.

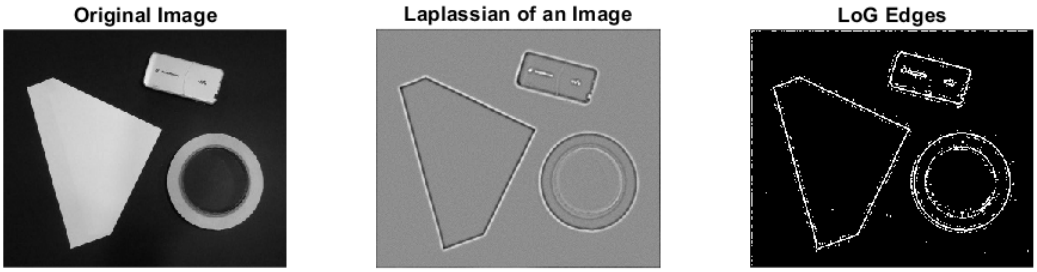


Figure 4: Laplacian of Gaussian, T1 = 1 and T2 = 7

As it can be observed from the Figure 4 that Laplacian of an image is successful for detecting the edges of the objects within the image an as well as the noise such that the image looks like a smooth version of the image with added some noise in it (especially in the background). It is also clear to see that Laplacian of Gaussian filter made all pixel values in the edges clear enough and even the details but further trials will be also taken into account to see the effects of the threshold values and the success of the filter in different images.

3 Post-Lab Experiments

3.1 Sobel Filtering

As we implemented Sobel edgels with the threshold value T = 100, I wanted to see how the output will change with smaller and greater threshold values on the same image.

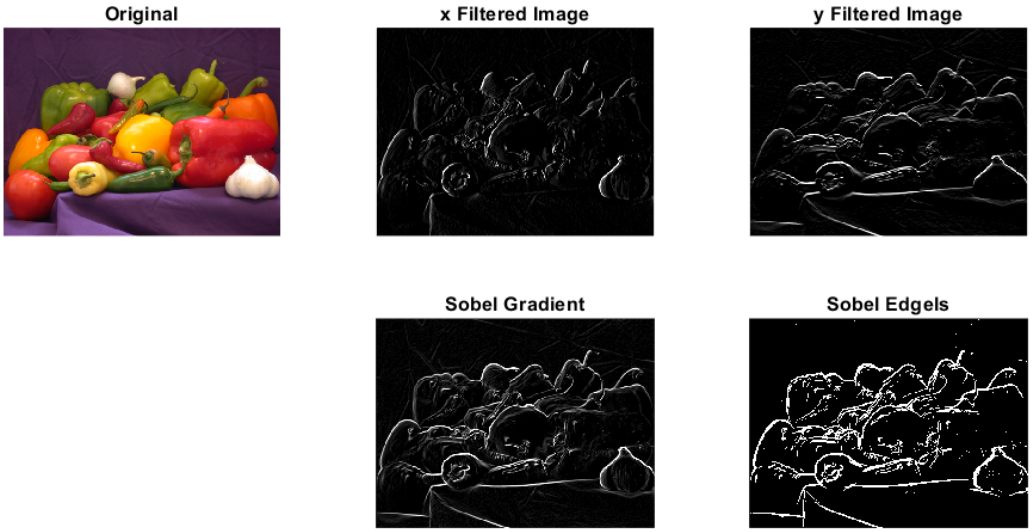


Figure 5: Sobel Filters, T = 50

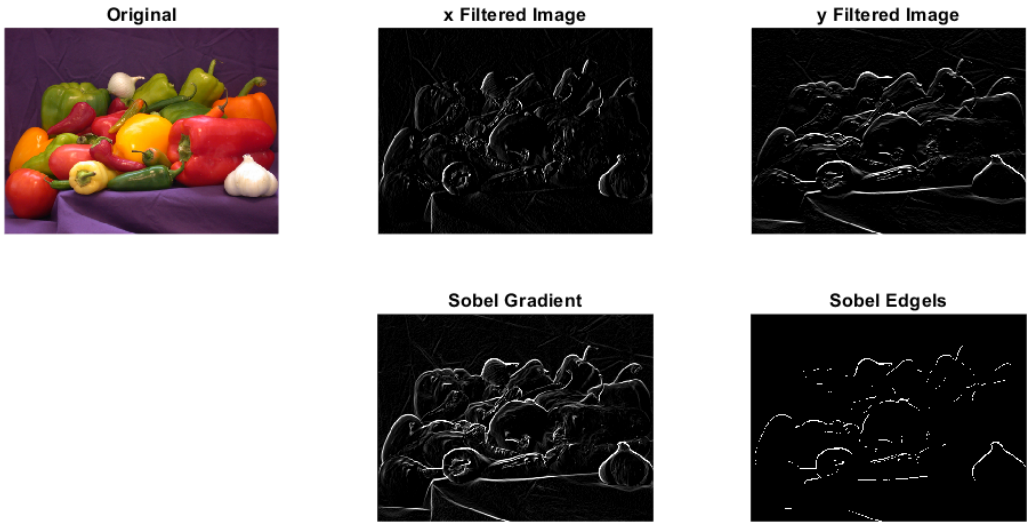
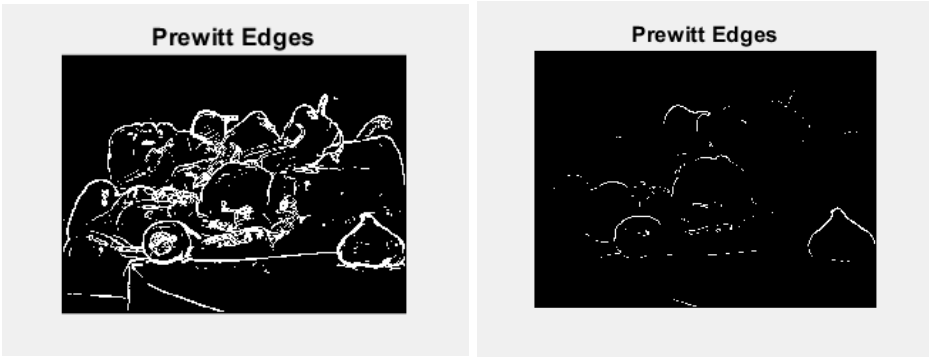


Figure 6: Sobel Filters, T = 200

Since we keep using the same image and the same kernels as filters, the filtered images and sobel gradients doesn't get affected from the change in threshold value. The important thing we need to observe in here is that how the white pixel density is being affected. As can be seen from the Figure 5 and 6 that as the threshold value increased, the details (as in white edges) are decreased. For instance, even though it is a bit unnecessary to go into the details of the objects when we are trying to only identify the borders of the objects to identify different objects, as we increased the the threshold towards 200 as in the 6, it started becoming hard to understand which objects are present. The reason behind that is quite clear that as the T increased, less pixel values were able to be greater than the magnitude of gradient so it resulted in less white (value = 255) appearing in the resulting image.

3.2 Prewitt Filters

Since we observed how the affects of different threshold values in the previous example, I wanted to know if we use different kernels (i.e. kernels used for Prewitt filter) whether the effect will be that obvious to observe again.



(a) T = 50

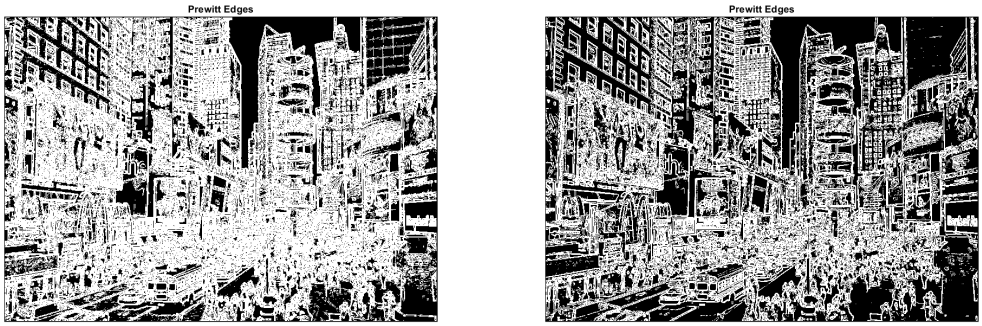
(b) T = 200

Figure 7: Prewitt Filters

As probably guessed before, Prewitt filtering resulted in the same effect with smaller and greater threshold values as can be seen from Figure 7. So it is possible to conclude that T = 100 seems like the optimal value for identifying different objects with edge detection for both of the filters (Sobel and Prewitt) since it becomes way too detailed with smaller values and hard to see the boundaries with greater values.

Figure 8: Crowded City Image

In addition to the images on the previous experiments, I wanted to try another image for edge detection to see whether the optimum threshold value will change or not. To do so, I used a different type of an image which is more detailed and has more objects within as well as a lot of background details as can be seen in Figure 8. Another aim of selecting this image is that I want to see whether the filters will be successful in identifying the edges in this kind of an image. And it might be also useful to mention that this one is higher in image quality (800*563 pixels).



(a) T = 50 (b) T = 100

Figure 9: Prewitt Filters on Crowded City Image

As can be seen from the Figure 9 that T = 50 was still mostly white pixel overall in the image and hard to see where the edges are but this time T = 100 was not that successful either since it is still hard to determine where the boundaries between objects are. Thus, I tried 2 more higher threshold values to see if they could perform better.

(a) T = 200 (b) T = 300

Figure 10: Prewitt Filters on Crowded City Image

When compared with Figure 9b it can be stated that T = 200 in 10a performed better since it is easier to see what the image represents such as the boundaries of the buildings and where the different objects such as cars, windows, billboards are. However, when looked at the 10b, some boundaries starting being less visible such as boundaries of the some buildings appeared as disconnected. Thus, it is possible to conclude that the optimal threshold value varies in different types of images and it should be derived by experimenting with different values but still successful to detect the edges somehow even though not as clear as it can be in every image.

3.3 Laplacian of Gaussian

Since we tried this function with another image and saw it actually performed pretty good, I wanted to try whether Laplacian will work better on the city image from the previous experiment which was shown in the Figure 8.

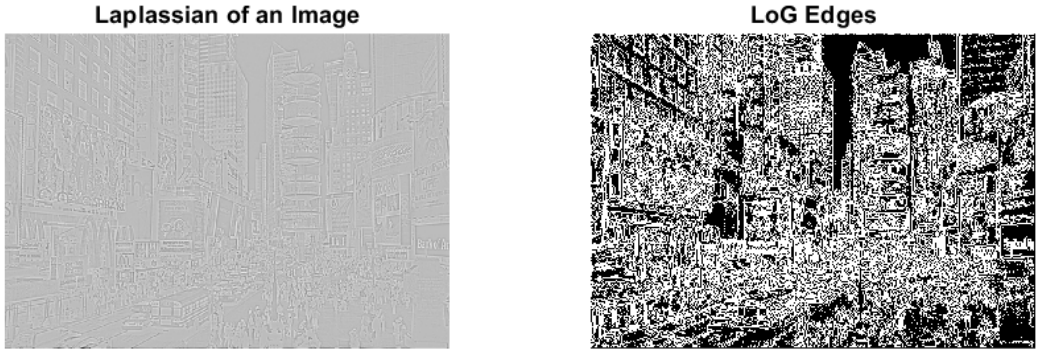
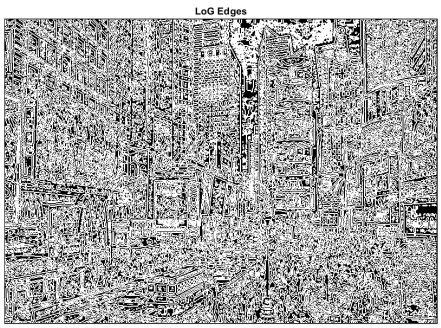


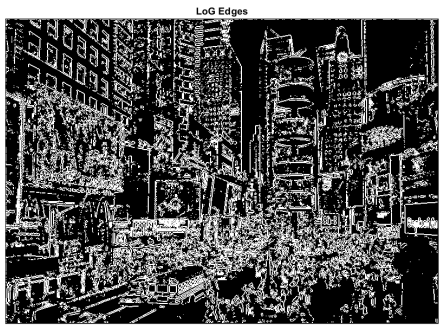
Figure 11: Laplacian of Gaussian with T1 = 1, T = 7

As it can be seen from the Figure 11 that it is really hard to identify which objects there are in the image on the first look to the Laplacian of the image however it is still performing a (not that succesful) edge detection when looked at the final result Laplacian of Gaussian when implemented the threshold method. The threshold values used are still the same as we did in the lab experiment which are T1 = 1 for detecting a sign change and T2 = 7 to identify the zero crossings due to noise, so I experimented further to find better fitting threshold values for a complex image like that one.

First, I kept the first threshold very small T1 = 1 and tried one small (T1 = 1) and one greater (T2 = 20) value for the second threshold.



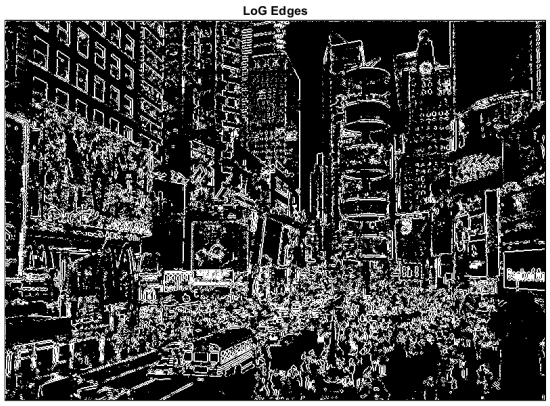
(a) T1 = 1 and T2 = 1



(b) T1 = 1 and T2 = 20

Figure 12: Laplacian of Gaussian on Crowded City Image

After, I did the same for the first threshold (T1 = 10 and T1 = 20) while keeping the second one same. The reason behind I kept the value as T2 = 20 was that, it can be seen from the Figure 12a that the edges of the objects within the image were more differentiable compared to 12b.



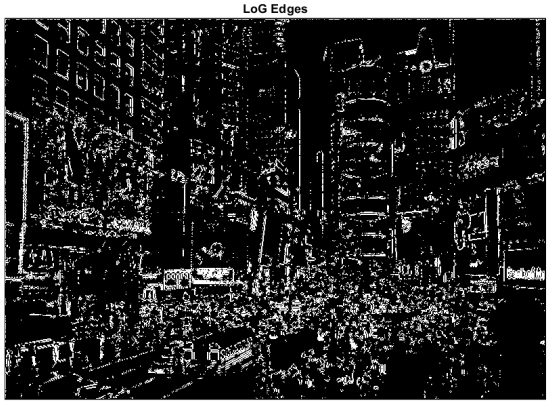
(a) T1 = 10 and T2 = 20

(b) T1 = 20 and T2 = 20

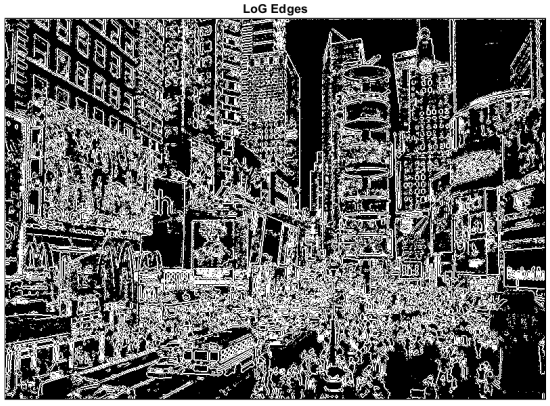
Figure 13: Laplacian of Gaussian on Crowded City Image

At the end, after I played with the first threshold value as can be seen in the Figure 13, they both returned an image that has been contoured in terms of the edges are visible. And there is a slight difference between the images that smaller T1 resulted in a more detailed image at the end.

Finally, I also wondered if I can make the image less detailed but more only the edges of the objects so I played around with the values for more. And I realized that when I manipulate T1, compared to T2 the pixels that are able to appear as white are more and tried until I find the better edge detection. But not increased T2 as much so the edges become disjoint again such as in the Figure 14a. Thus, I decided on an optimal value for both of the values as T1 = 10 and T2 = 10 and final image with edge detection can be seen in the Figure 14b.



(a) T1 = 10 and T2 = 30



(b) T1 = 20 and T2 = 10

Figure 14: Laplacian of Gaussian on Crowded City Image

Appendix

3.4 main.m

```
%% Lab3 Edge Detection
clear all; close all; clc;

%% Problem 1: Sobel Filtering

I = imread('peppers.png');
Thresh1 = 100;
Im_sobel = lab3sobel(I, Thresh1);

%% Problem 2: Prewitt

Thresh2 = 100;
Im_prewitt = lab3prewitt(I, Thresh2);

%% Problem 3: Laplacian of Gaussian LoG
I2 = imread('Object_contours.jpg');
G = lab3log(I2, 1, 7);

```

3.5 lab2sobelfilt.m

```
function [x_filtered, y_filtered] = lab2sobelfilt(img)

    [row, col, ch] = size(img);
    if (ch == 3)
      img = rgb2gray(img);
    end

    img = double(img);
    x_Filter = [-1 0 1; -2 0 2; -1 0 1];
    y_Filter = [-1 -2 -1; 0 0 0; 1 2 1];
    k = 1;

    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            window = img(i-k:i+k,j-k:j+k);
            Xvalue = sum(sum(window.*x_Filter));
            Yvalue = sum(sum(window.*y_Filter));
            x_filtered(i,j) = Xvalue;
            y_filtered(i,j) = Yvalue;
        end
    end

    img = uint8(img);
    x_filtered = uint8(x_filtered);
    y_filtered = uint8(y_filtered);

%        figure;
%     subplot(2,2,[1 2])
%         imshow(img)
%         title('Original Image');
%     subplot(2,2,3)
%         imshow(x_filtered)
%         title('Sobel x Filtered Image: Vertical Edgels');
%     subplot(2,2,4)
%         imshow(y_filtered)
%         title('Sobel y Filtered Image: Horizontal Edgels');

end

```

3.6 lab3sobel.m

```
function [I_edge] = lab3sobel(I, T)

[row, col, ch] = size(I);
if ch == 3
   Im = rgb2gray(I);
end
% Calculating Gradient Magnitude
[Gx, Gy] = lab2sobelfilt(Im);
Gx = double(Gx);
Gy = double(Gy);

% Find the magnitude of gradient
G_mag = sqrt(Gx.^2 + Gy.^2);

% Thresholding
I_edge = zeros(size(G_mag));
I_edge(find(G_mag > T)) = 255;



Gx = uint8(Gx);
Gy = uint8(Gy);
G_mag = uint8(G_mag);
I_edge = uint8(I_edge);


figure;
subplot(2,3,1)
imshow(I)
title("Original");

subplot(2,3,2)
imshow(Gx)
title("x Filtered Image");

subplot(2,3,3)
imshow(Gy)
title("y Filtered Image");

subplot(2,3,5)
imshow(G_mag)
title("Sobel Gradient");
subplot(2,3,6)
imshow(I_edge)
title("Sobel Edgels");

end

```

3.7 lab3prewitt.m

```
function [I_edge] = lab3prewitt(I, T)

[row, col, ch] = size(I);
if ch == 3
   I = rgb2gray(I);
end

I = double(I);
Sx = [-1 0 1;-1 0 1;-1 0 1];
Sy = [-1 -1 -1; 0 0 0; 1 1 1];

k = 1;

```

```
    Gx = zeros(size(I));
    Gy = zeros(size(I));

    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            window = I(i-k:i+k,j-k:j+k);
            Xvalue = sum(sum(window.*Sx));
            Yvalue = sum(sum(window.*Sy));
            Gx(i,j) = Xvalue;
            Gy(i,j) = Yvalue;
        end
    end

    G_mag = sqrt(Gx.^2 + Gy.^2);

    I_edge = zeros(size(G_mag));

    I_edge(find(G_mag > T)) = 255;

    Gx = uint8(Gx);
    Gy = uint8(Gy);
    G_mag = uint8(G_mag);
    I_edge = uint8(I_edge);
    I = uint8(I);


    figure;
    subplot(2,3,1)
    imshow(I)
    title("Original");

    subplot(2,3,2)
    imshow(Gx)
    title("Prewitt x Filtered");

    subplot(2,3,3)
    imshow(Gy)
    title("Prewitt y Filtered");

    subplot(2,3,5)
    imshow(G_mag)
    title("Prewitt Gradient");
    subplot(2,3,6)
    imshow(I_edge)
    title("Prewitt Edges");

    end

```

3.8 lab3log.m

```
    function [E] = lab3log(I, T1, T2)
    %% Step 1 : Input iamge and convert to grayscale
    [r, c, ch] = size(I);
        if (ch == 3)
          I = rgb2gray(I);
        end

    %% Step 2 : Blur the image, using Gaussian Filtering

    J = imgaussfilt(I, 1.1);

    %% Step 3 : Perform the Laplassian on the Blurred Image

    W = [0 1 0; 1 -4 1; 0 1 0];
    G = conv2(J,W,'same');

```

```
    %% Step 4 : Find the zero crossing of the Laplassian and compare the local change at this
    point to a threshold value

    E = zeros(r, c); %initialize all zero edge image

    k = 1; % 3*3 sliding window

    for i = k+1:1:r-k-1
            for j = k+1:1:c-k-1
               if abs(G(i, j)) >= T1
                   if (G(i-1, j)*G(i, j)<0 || G(i+1, j)*G(i, j)<0 || G(i, j-1)*G(i, j)<0 || G(i, j
    +1)*G(i, j)<0)
                       if abs(G(i-1, j) - G(i, j)) >= T2 || abs(G(i+1, j) - G(i, j)) >= T2 || abs(
    G(i, j-1) - G(i, j)) >= T2 || abs(G(i, j+1) - G(i, j)) >= T2
                           E(i, j) = 255;
                       end
                   end
               else
                   if (G(i-1, j)*G(i+1, j))<0 || (G(i, j-1)*G(i, j+1))<0
                       if abs(G(i, j+1) - G(i, j-1))/2 >= T2 || abs(G(i-1, j)- G(i+1, j))/2 >= T2
                           E(i, j) = 255;
                       end
                   end

               end
            end
        end

    %% Step 5: Visualization
    figure
    subplot(1,3,1)
    imshow(uint8(I))
    title("Original Image")

    subplot(1,3,2)
    imshow(G,[])
    title("Laplassian of an Image")

    subplot(1,3,3)
    % E = medfilt2(E); Shows the image better
    imshow(E)
    title("LoG Edges")

    end
```