EE417

POST-LAB #3 REPORT

1. Sobel Operator

To detect edges in a grayscale image, the gradient image obtained by the horizontal and the vertical Sobel operators is binarized by a threshold. The gradient image is calculated as

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

And sobel filtering can be applied as the following kernels

-1	0	1
-2	0	2
-1	0	1

X Filter

-1	-2	-1
0	0	0
1	2	1

Y Filter

As we can see on the images above, sobel x filter gives vertical edges, sobel y gives the horizontal edges of the image. On the other hand, sobel gradient is the norm of the gradient values that emphasizes edges of the image. After sobel gradient, we determined a threshold (as 100) to filter weak edges out and binarizing the image.

```
function [XK,YK,GK,EK] = lab3sobel(img)

    [row,col,ch] = size(img);
    X = zeros(size(img));
    Y = zeros(size(img));
    G = zeros(size(img));
    E = zeros(size(img));
    k = 1;

    if(ch==3)
        img = rgb2gray(img);
    end

    Dimg = double(img);

    xkernel = [-1 0 1;
               -2 0 2;
               -1 0 1];

    ykernel = [-1 -2 -1;
               0 0 0;
               1 2 1];

    threshold = 100;

    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            subIm = Dimg(i-k:i+k,j-k:j+k);
            value1 = sum(sum(subIm.*xkernel));
            value2 = sum(sum(subIm.*ykernel));
            value3 = sqrt(value1^2+value2^2);

            X(i,j) = value1;
            Y(i,j) = value2;
            G(i,j) = value3;
            if(value3>threshold)
                E(i,j) = value3;
            end

        end
    end

    XK = uint8(X);
    YK = uint8(Y);
    GK = uint8(G);
    EK = uint8(E);

end
```

2. Prewitt Operator

To detect edges in a grayscale image, the gradient image obtained by the horizontal and the vertical Prewitt operators is binarized by a threshold. The gradient image is calculated as

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

And sobel filtering can be applied as the following kernels

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

X Filter

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

Y Filter

As we can see on the images above, prewitt x filter gives vertical edges, prewitt y gives the horizontal edges of the image. On the other hand, prewitt gradient is the norm of the gradient values that emphasizes edges of the image. After prewitt gradient, we determined a threshold (as 100) to filter weak edges out and binarizing the image as we did in sobel operator.

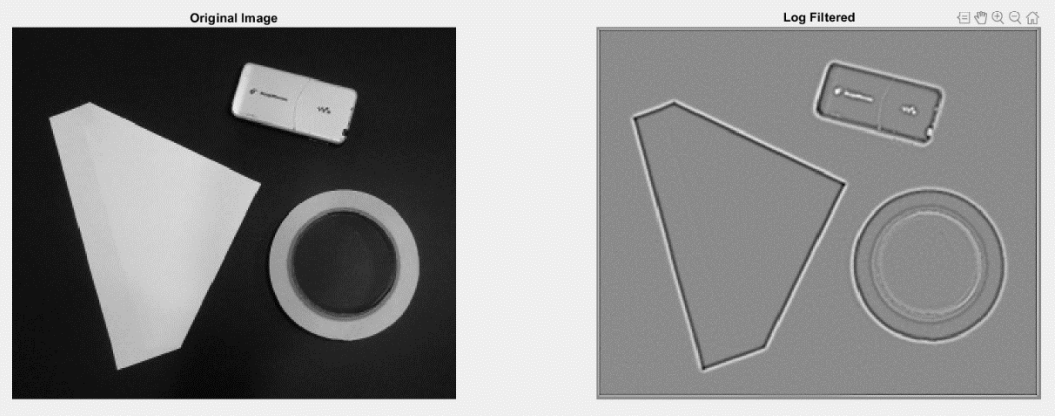
```
function [XK,YK,GK,EK] = lab3prewitt(img)

    [row,col,ch] = size(img);
    X = zeros(size(img));
    Y = zeros(size(img));
    G = zeros(size(img));
    E = zeros(size(img));
    k = 1;
    if(ch==3)
        img = rgb2gray(img);
    end

    Dimg = double(img);

    xkernel = [-1 0 1;
               -1 0 1;
               -1 0 1];

    ykernel = [-1 -1 -1;
               0 0 0;
               1 1 1];

    threshold = 100;

    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            subIm = Dimg(i-k:i+k,j-k:j+k);
            value1 = sum(sum(subIm.*xkernel));
            value2 = sum(sum(subIm.*ykernel));
            value3 = sqrt(value1^2+value2^2);

            X(i,j) = value1;
            Y(i,j) = value2;
            G(i,j) = value3;

            if(value3>threshold)
                E(i,j) = value3;
            end

        end
    end

    XK = uint8(X);
    YK = uint8(Y);
    GK = uint8(G);
    EK = uint8(E);

end
```

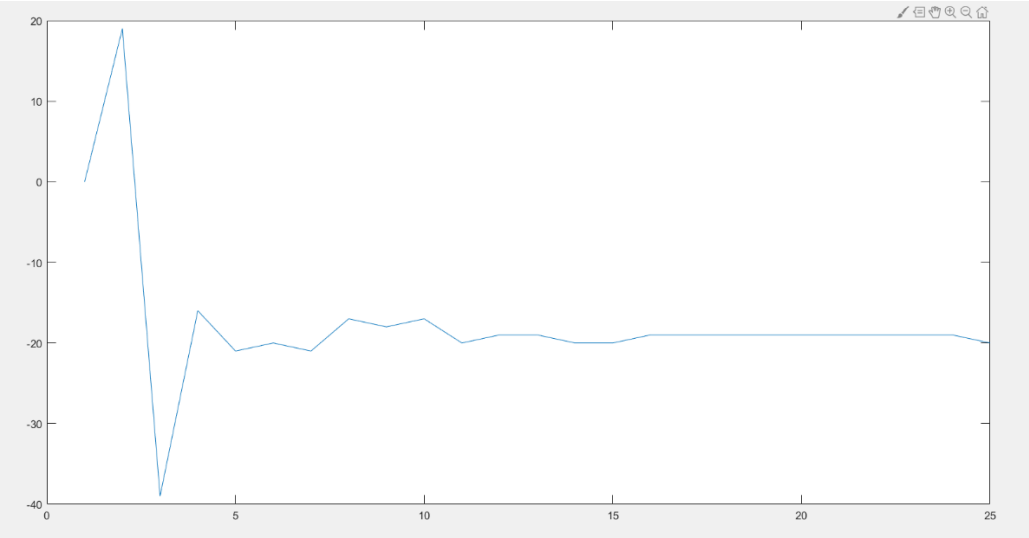
3. Laplacian of Gaussian Filter

In this filter we firstly smooth (Gaussian filter) the image and then find zero-crossings (Laplacian filter)

$$O(x,y) = \nabla^2(I(x,y) * G(x,y) \text{ - > By using linearity}$$

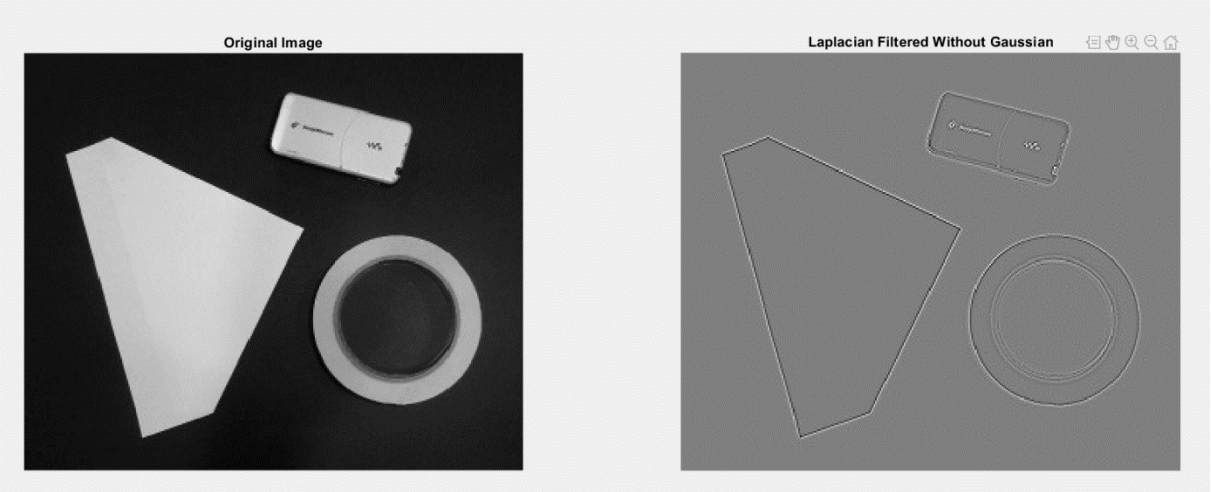
$$O(x,y) = \nabla^2 G(x,y) * I(x,y)$$





We can see that by use of log we can detect edges of the image by use of finding zero-crossing approach which is second derivative and on the graph there is a huge variation of the intensity value (between 0-4) that corresponds to an edge.

Also let us examine if we did not smooth our image before applying Laplacian operator:



In comparison with the log filtered image we can see that edges are seems week and sharp in without Gaussian smoothing.

```
function XK= lab3log(img)

    [row,col,ch] = size(img);
    X = zeros(size(img));
    k = 1;

    % Gaussian Filter Applied
    img = lab2gaussfilt(img);

    if(ch==3)
        img = rgb2gray(img);
    end

    Dimg = double(img);

    xkernel = [0 1 0;
               1 -4 1;
               0 1 0];

    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            subIm = Dimg(i-k:i+k,j-k:j+k);
            value1 = sum(sum(subIm.*xkernel));
            X(i,j) = value1;
        end
    end
    XK = double(X);

End
```

DISCUSSION OF EDGE DETECTORS:

The Sobel operator is more sensitive to the diagonal edge is than to the horizontal and vertical edges. On the contrary, Prewitt operator is more sensitive to horizontal and vertical edges.

In comparison with first derivative edge detectors, higher orders are making the image more and sharp (log filter).

4. Corner Detection

In this section, we will try to find the cornermap of an image by the Kanade-Tomasi algorithm.

Firstly we will compute gradients Gx and Gy of the image

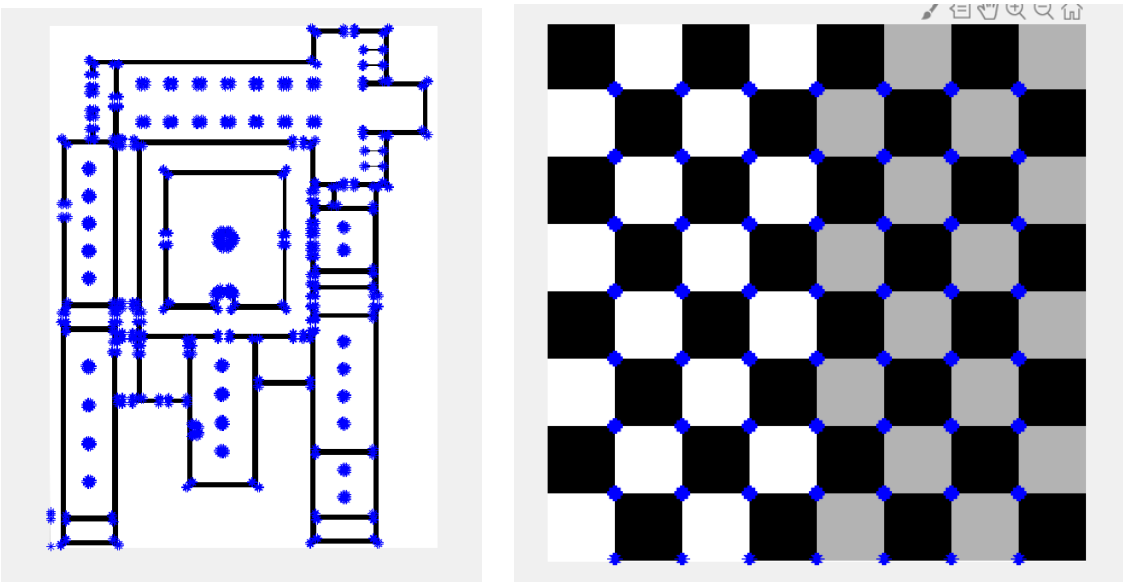
Then, we will create matrix of each pixel in a window as follows:

$$H = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$$

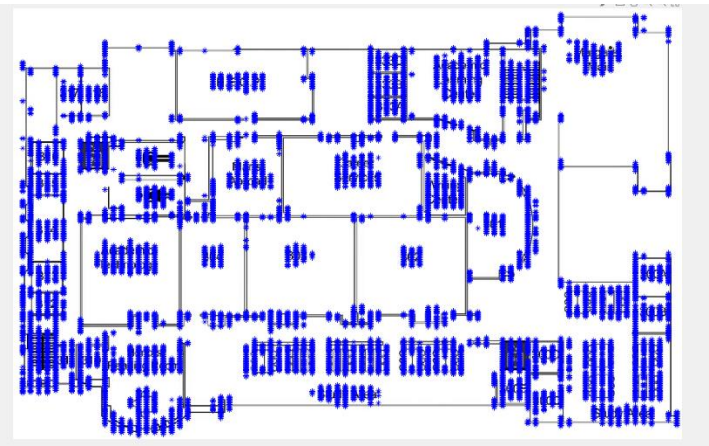
where Ix and Iy are the image gradients of a window along x and y directions, respectively.

Compute the eigenvalues λ1 and λ2 of H.

If min(λ1, λ2) >Threshold, add the pixel coordinates to corner list.



On the image corners are shown as blue points. Actually, that function is creating a corner map and then places them on the image. A corner is a point that shows a sudden intensity change on multiple direction.



Here we can see a floor map that shows the corners in the image.

We can say that Kanadi-Tamasi corner detector gives a better result if the intensity changes are really sharp as in checkboard.jpg above. On the other hand, on Monastry and floor map images these corners are thinner than checkboard.jpg, because of that reason corners that are shown as blue making some variations. Also, in floor.jpg we increased loop increment due to time issues. Otherwise, it did not perform the operations in a reasonable time, because gradient and eigenvalue operations are expensive operations.

```
function corners = lab3ktcorners(img)
    [row,col,ch] = size(img);
    k = 1;
    corners = [];
    if(ch==3)
        img = rgb2gray(img);
    end
    Dimg = double(img);
    threshold = 100;
    [Gx, Gy] = imgradientxy(Dimg);
    for i = k+1:1:row-k-1
        for j = k+1:1:col-k-1
            subImx = Gx(i-k:i+k,j-k:j+k);
            subImy = Gy(i-k:i+k,j-k:j+k);
            H = [sum(sum(subImx.*subImx))
sum(sum(subImx.*subImy)); sum(sum(subImx.*subImy))
sum(sum(subImy.*subImy))];
            eigs = eig(H);
            if(min(eigs)>threshold)
                corners = [corners; i j];
            end
        end
    end
    imshow(img);
    hold on
    plot(corners(:,2),corners(:,1),'b*');
end
```