Aynur Süne - 21041

EE417 – Post-Lab Report 3

Sobel Operator

Sobel Operation detects edges by computing gradient of an image and binarize it with a threshold. Two kernels are used to find vertical and horizontal derivative approximations. This part is computed by using Sobel Filter function from previous lab. Later, from these two horizontal and vertical output images, gradient value for image is computed ('grad' variable in the code below). With a threshold value, gradient values above this threshold are highlighted and others are suppressed.

```
function [newimg] = lab3sobel(img,t)

[r,c,color] = size(img);
if(color==3)
    img = rgb2gray(img);
end

newimg = zeros(size(img));
newimg = double(newimg);

img=double(img);

[newimgx,newimgy] = lab2sobelfilt(img);

newimgx = double(newimgx);
newimgy = double(newimgy);

grad = sqrt(((newimgx).^2 + (newimgy).^2));

k = find(grad>t);
newimg(k)=255;

a = find(grad<=t);
newimg(a)=0;

newimg = uint8(newimg);

end
```

Sobel Filter, t=250

Sobel Filter, t=50

Sobel Filter, t=10

As outputs of the function indicates, if threshold value is too high, number of detected edges is very low. General outlier of the image cannot be observed. If threshold is too low; everything is counted as an edge, even parts of the image with no edge appears as an edge. Sobel operator cannot operate accurately on an image with high noisy pixels since these pixels appear as an edge too.

Prewitt Operator

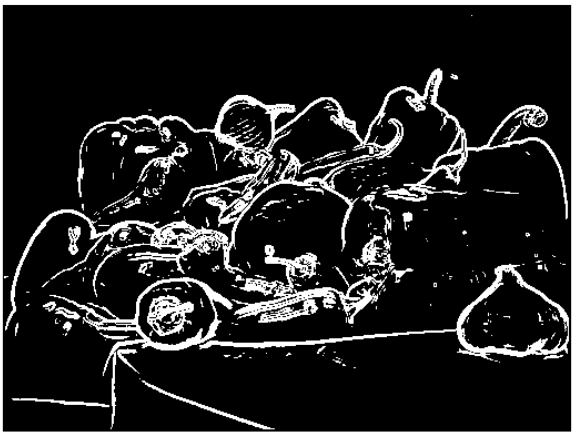
Prewitt Operator works in the same way as Sobel Operator. Implementation is the same except the used kernel filters.

```
function [newimg] = lab3prewitt(img,t)

[r,c,color] = size(img);
if(color==3)
    img = rgb2gray(img);
end

newimg = zeros(size(img));
newimg = double(newimg);

newimgx = zeros(size(img));
newimgy = zeros(size(img));

img=double(img);

filterX = [-1 0 1;-1 0 1;-1 0 1];
filterY = [-1 -1 -1;0 0 0;1 1 1];
k=1;

for i=k+1:1:r-k-1
    for j= k+1:1:c-k-1
        subimg = img(i-k:i+k,j-k:j+k);
        newimgx(i,j) = sum(sum(subimg.*filterX));
        newimgy(i,j) = sum(sum(subimg.*filterY));
    end
end
```

```
grad = sqrt(((newimgx).^2 + (newimgy).^2));

k = find(grad>t);
newimg(k)=255;

a = find(grad<=t);
newimg(a)=0;

newimg = uint8(newimg);

end
```

Prewitt Operator, t=250

Prewitt Operator, t=50



Prewitt Operator, t=10

Outputs of Sobel Operator and Prewitt Operator when threshold is 250; Sobel Operator seems better at finding edges, more edges are detected with Sobel. When threshold is 10; Sobel operator again gave better result than Prewitt Operator. It seems that Prewitt

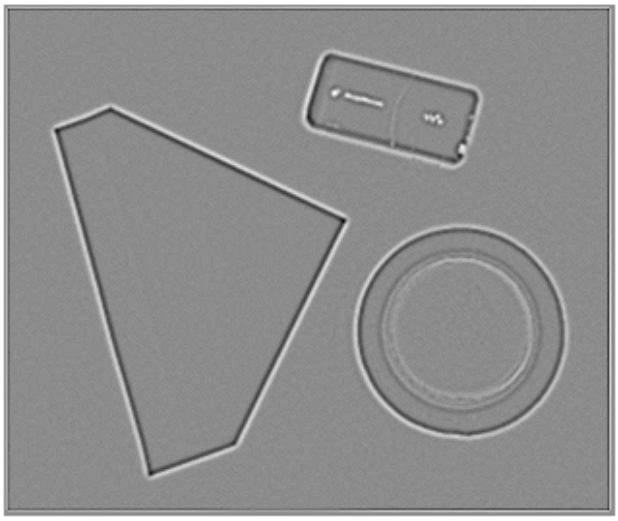
Operator is more sensitive to noise than Sobel Operator. Prewitt Output is noisier and there are a lot of pixels highlighted that counted as edge, which are actually not.

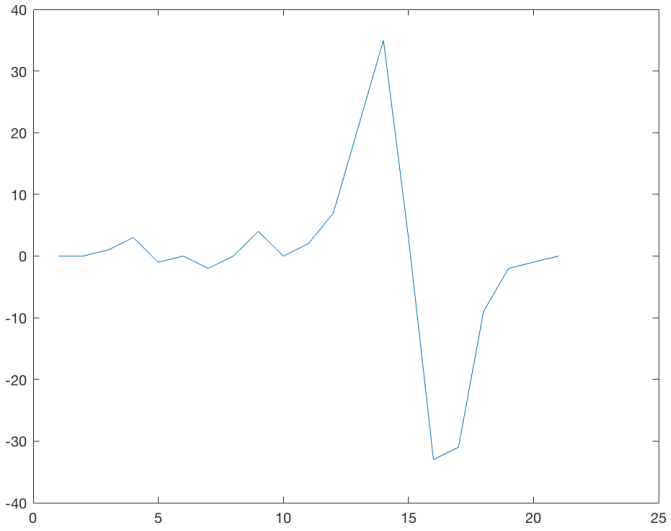
Laplacian of Gaussian

Laplace Operation is second derivative operation to find zero crossings of first derivative. Since derivative operations are sensitive to noise, Gaussian smoothing operation is used on image first. This smoothing operation is done by Gaussian Filter function from previous lab. For Laplace operation, convolution with a given kernel is used.

```
function [newimg] = lab3log(img)
[r,c,color] = size(img);
if(color==3)
    img = rgb2gray(img);
end

newimg = zeros(size(img));
img=double(img);

subimg = lab2gaussfilt(img);
subimg = double(subimg);

filter = [0 1 0;1 -4 1;0 1 0];
k=1;

for i=k+1:1:r-k-1
    for j= k+1:1:c-k-1
        subnewimg = subimg(i-k:i+k,j-k:j+k);
        newimg(i,j) = sum(sum(subnewimg.*filter));
    end
end


end
```

LOG Filtered Image






```
>> img = lab3log(obj);
>> grad = img(70:90,120);
>> plot(grad);
>>
```

Zero crossing can be observed from Gradation function output on the right, as expected. This region indicates an edge.

Corner Detection (Kanade- Tomasi Corner Detection Algorithm)

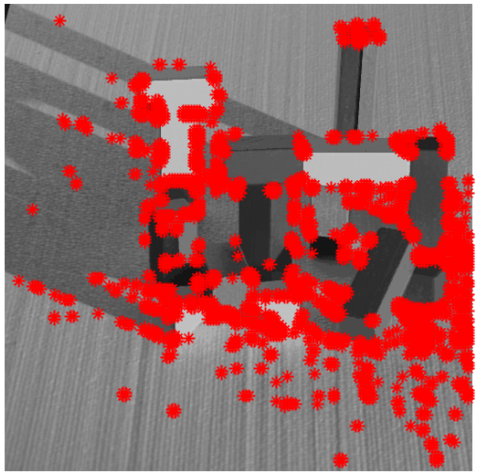
```
function [corners] = lab3ktcorners(img,t,k)
[r,c,color] = size(img);
if(color==3)
    img = rgb2gray(img);
end

newimg = zeros(size(img));
img=double(img);

[Gx,Gy] = imgradientxy(img);

corners = [];

for i=k+1:k:r-k-1
    for j= k+1:k:c-k-1
         subimg_x = Gx(i-k:i+k,j-k:j+k);
         subimg_y = Gy(i-k:i+k,j-k:j+k);

         h1 = sum(sum(subimg_x.^2));
         h2 = sum(sum(subimg_y.*subimg_x));
         h3 = sum(sum(subimg_y.^2));

         H = [h1 h2;h2 h3];
        eigen = eigs(H);

        if(min(eigen)>t)
            corners = [corners; i j];
        end
    end
end

img = uint8(img);

imshow(img);
hold on;
plot(corners(:,2),corners(:,1),'r*');
end
```

First, horizontal and vertical image gradients are computed. Then, for every 2k+1 x 2k+1 window of these gradient values, H matrix is created and its values are computed for every pixel in window where H matrix is as follows:

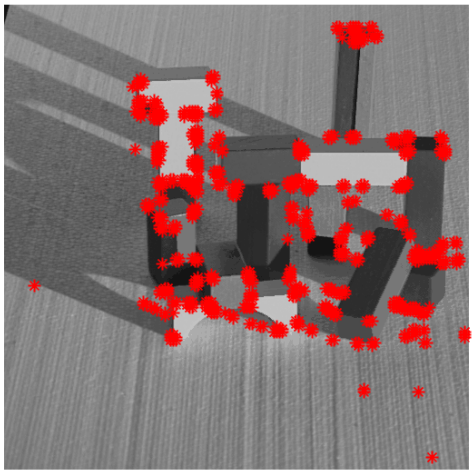
$$E(u,v) = \sum_{(x,y)\in W} [u\ v] \underbrace{\begin{bmatrix} I_x^2 & I_x I_y \\ I_y I_x & I_y^2 \end{bmatrix}}_{H} \begin{bmatrix} u \\ v \end{bmatrix}$$

Later on, Eigenvalues of H matrix are computed. If minimum Eigenvalue is above the threshold, it implies a corner therefore added to a corner list.

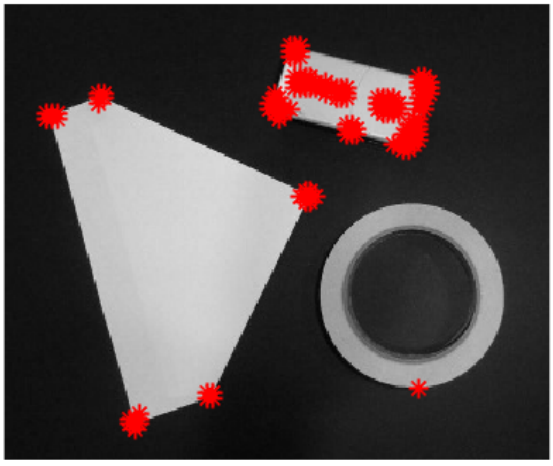
Corner Detection, t=50000, k=3



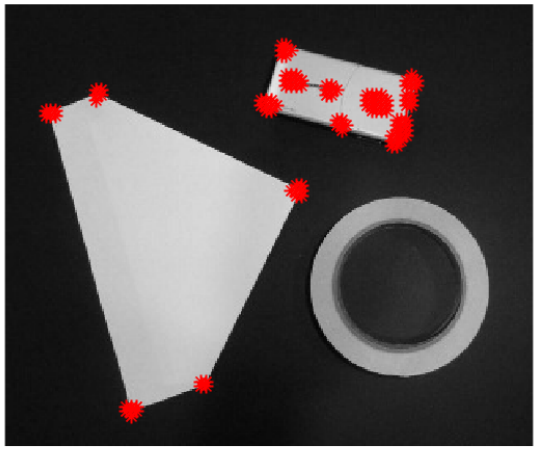
Corner Detection, t=100000, k=3



Corner Detection, t=50000, k=3



Corner Detection, t=100000, k=3



As threshold value increases, accuracy of corner detection also increases because noise in an image may be perceived as a corner as well, and elimination of non-corner pixels is difficult if threshold is low. Outputs are more precise when threshold is high.