86735: Computer Vision
Edge Detection

Due date: Wed 14.15, 21.10.15

Prof. F. Odone, Prof. F. Solari, M. Chessa, N. Noceti

Ernest Skrzypczyk, BSc

Problem 1

The goal of the practical laboratory session was to ultimately perform edge detection on the test images using the Laplacian of Gaussian operator, zero crossing location and thresholding. Most edge detection algorithms use a three part approach: filtering (smoothing), differentiation and detection. There are several aspects that have to be considered.

One of the problems of edge detection is the noise found in the image, which can be incorrectly interpreted by the applied detection algorithm as an edge. A solution could consist out of a process, where there is smoothing of the image first, then differentiating, which could be also accomplished by convolving the image with the derivative of the smoothing kernel.

In the proposed script for the MATLAB environment the Laplacian of Gaussian operator is introduced. The Gaussian filter, which is used for smoothing the image, is described by the following equation (1):

$$G(x,y) = e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{1}$$

The Laplace of Gaussian is defined by equation (2), so it is the convolution of the smoothing Gaussian filter and the Laplacian (second degree derivative) of the image in both dimensions.

$$\Delta^2 G_\sigma = \frac{1}{2\pi\sigma}\left(\frac{x^2+y^2-2\sigma^2}{\sigma^4}\right)e^{\frac{-(x^2+y^2)}{2\sigma^2}} \tag{2}$$

The next step is detecting the zero crossings for both dimensions. Those indicate the location of edges. However for the edges to be thick and easier recognizable, it is necessary to use thresholding or binarizing. All of those steps have been implemented in the following script.

Script 1: Edge detecting script written for Matlab.

```
%% Ernest Skrzypczyk - 4268738
%% 21.10.2015
%% Computer Vision - L02 - Edge detection
%% Matlab 8.6.0.267246 (R2015b)

clear all; close all;


%Reading images and converting the colorspace to grayscale
Image1 = double(rgb2gray(imread('boccadasse.jpg', 'jpg')));
Image2 = double(rgb2gray(imread('cameraman.tif', 'tif')));
Image3 = double(rgb2gray(imread('car.bmp', 'bmp')));
figure; imagesc(Image1); colormap gray; title('Input image "Image1" converted to gray scale');
figure; imagesc(Image2); colormap gray; title('Input image "Image2" converted to gray scale');
figure; imagesc(Image3); colormap gray; title('Input image "Image3" converted to gray scale');

%Processing input images
for l = 1:3

  %Setting processing parameters
  Sigma = 3 / l; SpatialResolution = 3 * l^2 * Sigma; Threshold = 25 * l / 2;

  edgedetection(Image1, Sigma, SpatialResolution, Threshold)
  edgedetection(Image2, Sigma, SpatialResolution, Threshold)
  edgedetection(Image3, Sigma, SpatialResolution, Threshold)

end

function DetectedEdges = edgedetection(InputImage, Sigma = 1, SpatialResolution = 3 * Sigma, Threshold = 25)

%Parameters
Sigma = 1; SpatialResolution = 3 * Sigma; Threshold = 25;
%Laplacian of Gaussian on the Image1
ImageLoG = LoG(InputImage, Sigma, SpatialResolution);
%Determening zerocrossings of Image1
[ZeroCrossingRows, ZeroCrossingColumns] = zerocrossing(InputImage);

figure;
subplot(211); imagesc(ZeroCrossingColumns);
title('Zero crossing - Columns'); colormap gray;
subplot(212); imagesc(ZeroCrossingRows);
title('Zero crossing - Rows'); colormap gray;

%Thresholding
[ZeroCrossings] = thresholding(ZeroCrossingRows, ZeroCrossingColumns, Threshold)

figure; plot(1); imagesc(ZeroCrossings);
```

```
title('Zero crossings in the image'); colormap gray;

end



function ImageOutput = LoG(InputImage, Sigma, SpatialResolution)

%Spatial resolution
[X, Y] = meshgrid(-SpatialResolution:SpatialResolution, -SpatialResolution: SpatialResolution);

%Gaussian
Gaussian = (1 /(2 * pi * Sigma^2)) * ((X.^2 + Y.^2 - 2 * Sigma^2) / Sigma^4).* exp(-(X.^2 + Y.^2) / (2 * Sigma^2));

figure; plot(21);
imagesc(Gaussian); title('Gaussian'); colormap gray;

OuputImage = conv2(InputImage, Gaussian, 'same');

figure; plot(22);
imagesc(ImageOutput); title('Laplacian of Gaussian'); colormap gray;

end



function [ZeroCrossingRows, ZeroCrossingColumns] = zerocrossing(InputImageLoG) ;
[Rows, Columns] = size(InputImageLoG);

%Initalization of zero crossing matrices
ZeroCrossings = zeros(Rows, Columns);
ZeroCrossingRows = zeros(Rows, Columns);
ZeroCrossingColumns = zeros(Rows, Columns);

%Main loop for zero crossing detection runs only between pixels so the
%last and first elements can be skipped
for j = 2:Columns-2
    for i = 2:Rows-2
    %Main loop -- start
        %Slope positive -> negative
        if (InputImageLoG(i, j) > 0 && InputImageLoG(i, j+1) < 0 )
            ZeroCrossingRows(i, j) = ceil(InputImageLoG(i, j) - InputImageLoG( i, j+1) );
        %Slope positive -> zero -> negative
        else if (InputImageLoG(i, j) == 0 && InputImageLoG(i, j+1) < 0 &&  InputImageLoG(i, j-1) > 0 )
            ZeroCrossingRows(i, j) = ceil(InputImageLoG(i, j-1) -  InputImageLoG(i, j+1) );
            %Slope negative -> positive
            else if (InputImageLoG(i, j) < 0 && InputImageLoG(i, j+1) > 0 )
```

```
                    ZeroCrossingRows(i, j) = ceil(InputImageLoG(i, j+1) - InputImageLoG(i, j) );
                    %Slope negative -> zero -> positive
                    else if (InputImageLoG(i, j) == 0 && InputImageLoG(i, j+1) > 0 && InputImageLoG(i, j-1) < 0 )
                    ZeroCrossingRows(i, j) = ceil(InputImageLoG(i, j+1) - InputImageLoG(i, j-1) );
                        end
                    end
                end
            end
            %Slope positive -> negative
            if (InputImageLoG(i, j) > 0 && InputImageLoG(i+1, j) < 0 )
                ZeroCrossingColumns(i, j) = ceil(InputImageLoG(i, j) - InputImageLoG(i+1, j) );
            %Slope positive -> zero -> negative
            else if (InputImageLoG(i, j) == 0 && InputImageLoG(i+1, j) < 0 && InputImageLoG(i-1, j) > 0 )
                ZeroCrossingColumns(i, j) = ceil(InputImageLoG(i-1, j) - InputImageLoG(i+1, j) );
                %Slope negative -> positive
                else if (InputImageLoG(i, j) < 0 && InputImageLoG(i+1, j) > 0 )
                    ZeroCrossingColumns(i, j) = ceil(InputImageLoG(i+1, j) - InputImageLoG(i, j) );
                    %Slope negative -> zero -> positive
                    else if (InputImageLoG(i, j) == 0 && InputImageLoG(i+1, j) > 0 && InputImageLoG(i-1, j) < 0 )
                    ZeroCrossingColumns(i, j) = ceil(InputImageLoG(i+1, j) - InputImageLoG(i-1, j) );
                        end
                    end
                end
            end
            %Main loop -- end
    end
end

function ZeroCrossings = thresholding(ZeroCrossingRows, ZeroCrossingColumns, Threshold)
%Setting dimensions
Rows = size(ZeroCrossingRows, 1);
Columns = size(ZeroCrossingColumns, 1);

for j = 1:Columns
    for i = 1:Rows
    %Muxing of detected zero crossings rows and columns
        if (ZeroCrossingRows(i, j) | ZeroCrossingColumns(i, j)) > Threshold);
      ZeroCrossings(i, j) = 255;
        else
            ZeroCrossings(i, j) = 0;
        end
    end
end
```

```
end
```

The above script is commented in a self explanatory way and according to requirements. Processed images according to the algorithm presented in the script are presented below:

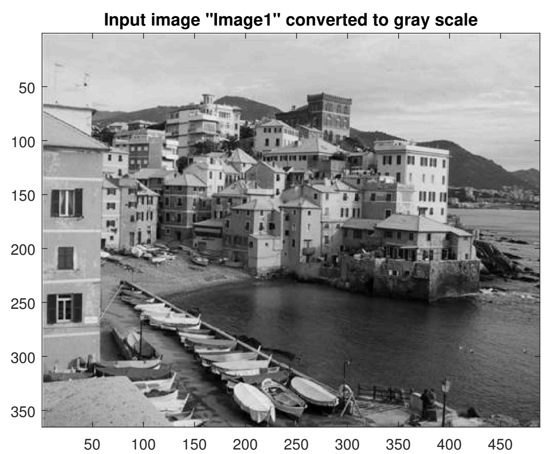

Figure 1: Original image 1 without modifications.

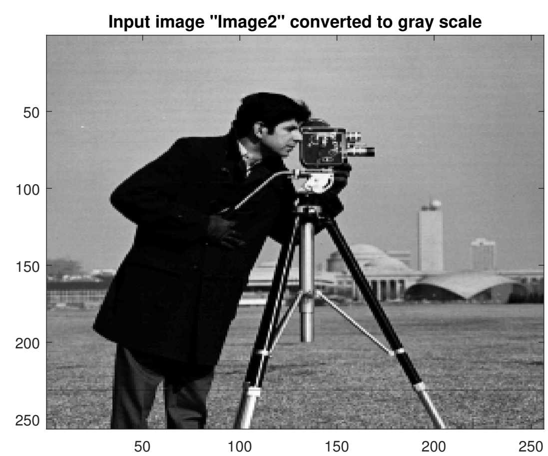

Figure 2: Original image 2 without modifications.

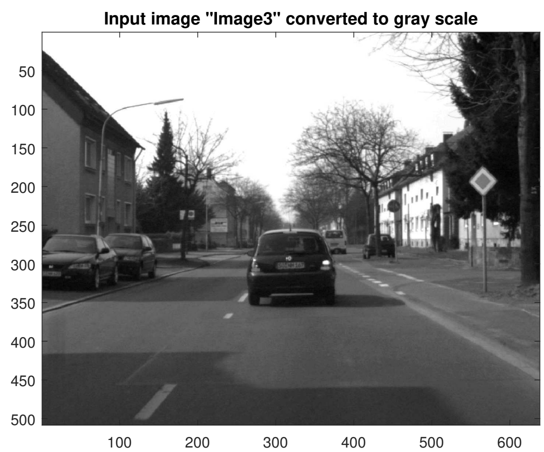

Figure 3: Original image 3 without modifications.

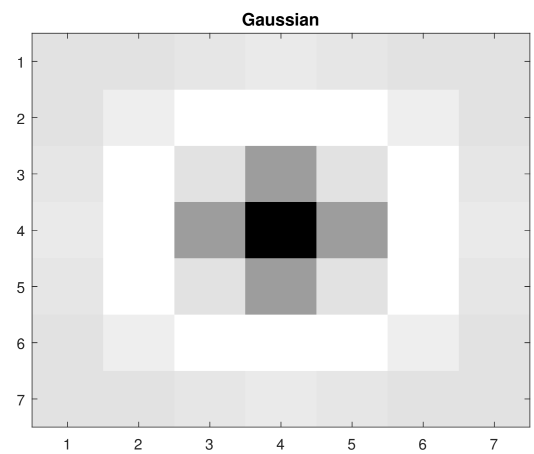

Figure 4: Gaussian filter.

Figure 5: Laplacian of Gaussian.

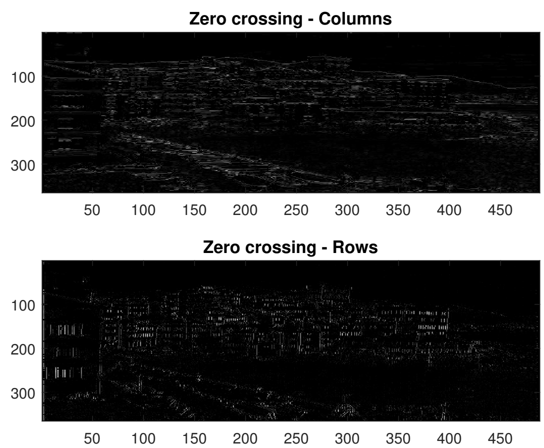

Figure 6: Zero crossings.

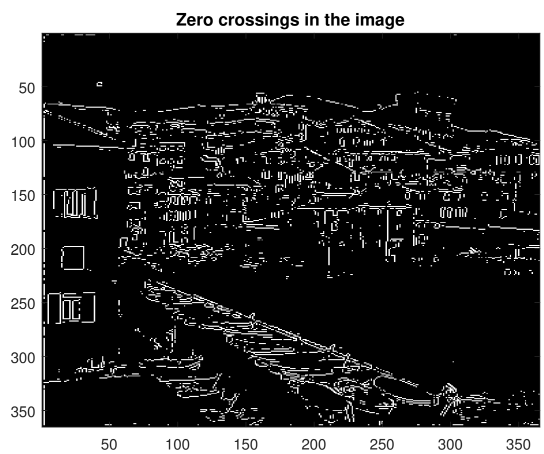

Figure 7: Zero crossings.

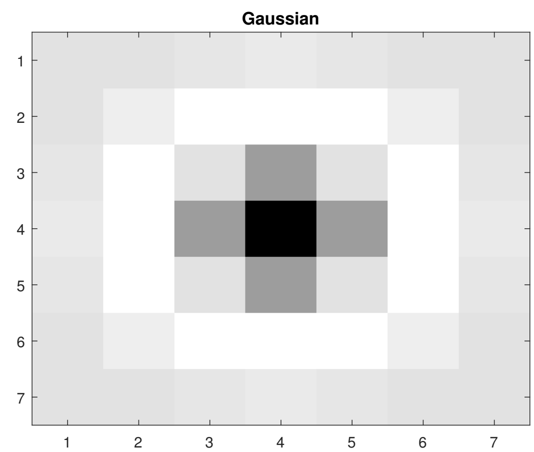

Figure 8: Zero crossings.

Conclusions

The algorithm presented in the script provides a simple implementation of edge detection. As shown in the provided images, the parameters are crucial. However the choice of the appropriate values is highly dependent on the application.