Computer Vision I - Sheet 4
Group 2

Jonas Otto
jonas@jonasotto.com

Dominik Authaler
dominik.authaler@uni-ulm.de

July 10, 2019

1 Moravec Operator

The source code for this task can be found in the files `sh04ex01.m` and `moravec.m`. Because there was no reference grid for the four directions given, we've chosen the mathematical model of defining the $0°$ direction along the positive x-axis, with growing angles when turning towards the positive y-axis. For the y-axis we assumed the model that the point $(0, 0)$ is located in the upper left corner of the image and the y-axis is pointing downwards. To avoid the need of flipping all the matrices to be able to apply them as convolution (where they get flipped back), we apply the non-flipped matrices as correlation. In addition to that we also apply the box filter as correlation, because it's symmetric that doesn't make any changes.

Comparing the two resulting images shown in the figures 1 and 2, one can easily stand, that the detector procudes a lot more false positive detections when the image is noisy. This can be explained quite well, as the high-frequency noise increases the difference between two neighbored pixels. To reduce this effect, one could apply a low-pass filter to the noisy image before trying to detect the corners.

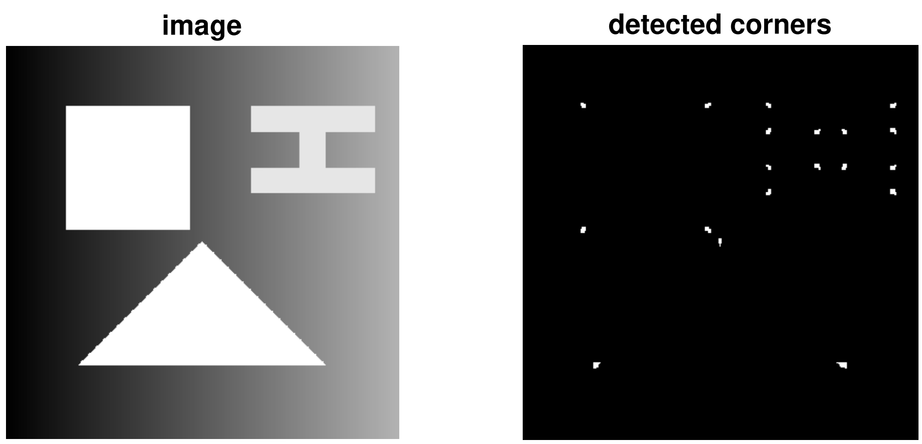

Figure 1: Corner detection applied to figures1.png

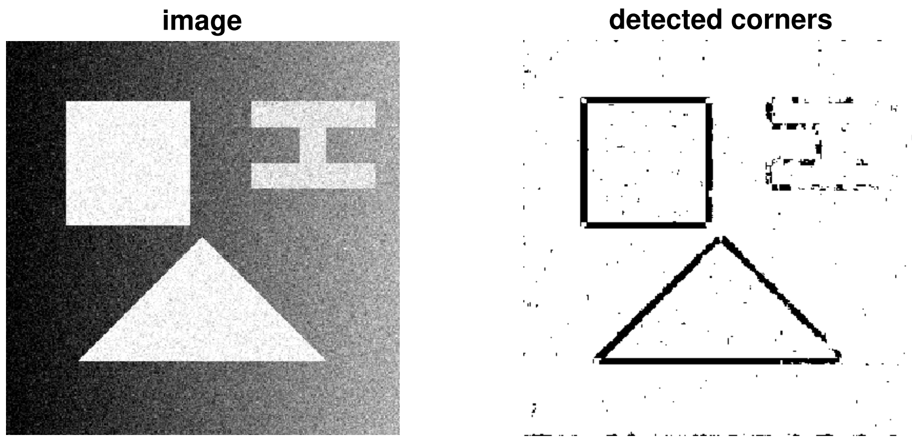

Figure 2: Corner detection applied to figures1_noisy.png

```
function corners = moravec(img)

[height, width] = size(img)

%Matrices used to calculate the differences between two neighboored pixels
%in the given direction
D0 = [0, 0, 0; 0, 1, -1; 0, 0, 0];
D45 = [0, 0, 0; 0, 1, 0; 0, 0, -1];
D90 = [0, 0, 0; 0, 1, 0; 0, -1, 0];
D135 = [0, 0, 0; 0, 1, 0; -1, 0, 0];

%Box filter for the 5x5 neighborhood
B = ones(5, 5) * 1/25;

%Apply both filters
imgD0 = imfilter(imfilter(img, D0, 'replicate').^2, B, 'replicate');
imgD45 = imfilter(imfilter(img, D45, 'replicate').^2, B, 'replicate');
imgD90 = imfilter(imfilter(img, D90, 'replicate').^2, B, 'replicate');
imgD135 = imfilter(imfilter(img, D135, 'replicate').^2, B, 'replicate');

%Normalize results
```

```
sum = imgD0 + imgD45 + imgD90 + imgD135;
imgD0 = imgD0 ./ sum;
imgD45 = imgD45 ./ sum;
imgD90 = imgD90 ./ sum;
imgD135 = imgD135 ./ sum;

min12 = min(imgD0, imgD45);
min34 = min(imgD90, imgD135);
min_ges = min(min12, min34);

%correct nan-values occuring inside the figures because the summed value
%(divisor) is zero
min_ges(isnan(min_ges)) = 0;

% Find corners:
% Corner is indicated by minimum > threshold
threshold = 0.1;
corners = zeros(height, width);
corners(min_ges > threshold) = 1;
```

```
%% Group 2: Dominik Authaler, Jonas Otto
close all;
clc;
clear;

%% Reading the images, applying Moravec Operator
img = im2double(imread('../images/figures1.png'));
noisyImg = im2double(imread('../images/figures1_noisy.png'));

corners = moravec(img);
cornerNoisy = moravec(noisyImg);

%% Visualization
rows = 1;
cols = 2;

figure('name','Image');
subplot(rows, cols, 1);
imshow(img);
title("image");

subplot(rows, cols, 2);
imshow(corners);
```

```
title("detected corners");
saveas(gcf,'../images/ex01.eps','epsc')

figure('name','Noisy Image');
subplot(rows, cols, 1);
imshow(noisyImg);
title("image");

subplot(rows, cols, 2);
imshow(cornerNoisy);
title("detected corners");
saveas(gcf,'../images/ex01_noisy.eps','epsc')
```

2 Structure Tensor

The source code used to create the images shown in the figure 3 can be found in the file sh04ex02.m. The figure only shows the results for the original image, because the results for the noisy version are as bad as the ones of the Moravec operator. This is mainly, because the structure tensor is based on the results of the Sobel operators, which approximate derivatives and are therefore quite sensitive to high frequency noise. In the comined image for all detected structures in the top right corner of the figure 3 the homogenous regions are blue, the edges green and the corners are painted red. The middle row of the figure shows the occurences of the different types as binary images. Moreover the image in the bottom left corner of the figure shows a zoomed in version of the original image, in which the eigenvector corresponding to the greatest eigenvalue is shown. For better visualization the vectors were thresholded by their corresponding eigenvalue. Therefore only significant vectors are shown in the image. The other two images in the last row of the figure visualize the both eigen-values. Comparing these to the images in the second row, it's easy to guess how the pixels were classified. For corners both eigenvalues need to be above some threshold. In comparison to that it's sufficient for edges if one of the eigenvalues is above the threshold. Last but not least the homogenous regions are characterized through two eigenvalues below the threshold.

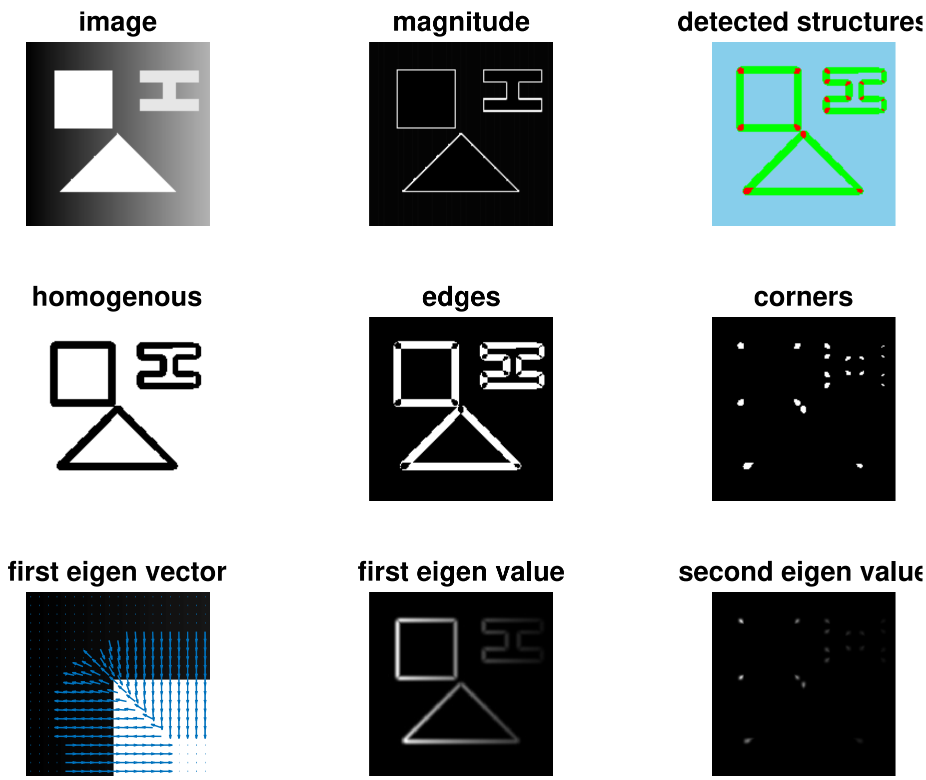

Figure 3: Corner detection applied to figures1.png

```
%% Group 2: Dominik Authaler, Jonas Otto
close all;
clc;
clear;

%% Calculation
% Reading the image in
img = im2double(imread('../images/figures1.png'));
%img = im2double(imread('../images/figures1_noisy.png'));

[height, width] = size(img);

%Sobel operators
sobelX = [1, 0, -1; 2, 0, -2; 1, 0, -1];
sobelY = [1, 2, 1; 0, 0, 0; -1, -2, -1];
```

```
I_x = imfilter(img, sobelX, 'replicate', 'conv');
I_y = imfilter(img, sobelY, 'replicate', 'conv');

magnitude = sqrt(I_x .^2 + I_y .^2);

%Gaussian filter
sigma = 3;
h = 2 * ceil(2*sigma) + 1;

G = fspecial('gaussian', h, sigma);
S11 = imfilter(I_x .* I_x, G, 'replicate', 'conv');
S12 = imfilter(I_x .* I_y, G, 'replicate', 'conv');
S22 = imfilter(I_y .* I_y, G, 'replicate', 'conv');

% images used to present the results
binHom = zeros(height, width, 1);
binEdge = zeros(height, width, 1);
binCorner = zeros(height, width, 1);
output = zeros(height, width, 3);
colorHom = double([135, 206, 235])./255;
colorEdge = double([0, 255, 0])./255;
colorCorner = double([255, 0, 0])./255;

eigThreshold = 0.1;

sEigenValues = zeros(height, width, 2); % 2 eigenvalues at every location
sFirstEigenVectors = zeros(height, width, 2); % one eigenvector with 2 components at every location

%TODO find a solution without these for-loops
for y = 1:height
    for x = 1:width
        S = [S11(y, x), S12(y, x); S12(y, x), S22(y, x)];
        [V, D] = eig(S);

        [eig_values, perm] = sort(diag(D), 'descend'); % Sorted eigenvalues, permutation vector of indices
        eig_vectors = V(:, perm);                       % Sorted eigenvectors

        sEigenValues(y, x, :) = eig_values;
        %Classify pixel
        if eig_values(1) < eigThreshold && eig_values(2)
```

```
                < eigThreshold
                binHom(y, x) = 1;
                output(y, x,:) = colorHom;
            else
                sFirstEigenVectors(y, x, :) = eig_vectors(:,
                    1);

                if eig_values(2) > eigThreshold
                    binCorner(y, x) = 1;
                    output(y, x,:) = colorCorner;
                else
                    binEdge(y, x) = 1;
                    output(y,x,:) = colorEdge;
                end
            end
        end
end

%% Visualization
rows = 3;
cols = 3;

subplot(rows, cols, 1);
imshow(img);
title("image");

subplot(rows, cols, 2);
imshow(magnitude);
title("magnitude");

subplot(rows, cols, 3);
imshow(output);
title("detected structures");

subplot(rows, cols, 4);
imshow(binHom);
title("homogenous");

subplot(rows, cols, 5);
imshow(binEdge);
title("edges");

subplot(rows, cols, 6);
imshow(binCorner);
title("corners");

```

```
roiX = 190:210;
roiY = 40:60;

subplot(rows, cols, 7);
imshow(img(roiY, roiX),[]);
hold on;
quiver(sFirstEigenVectors(roiY, roiX, 1),
    sFirstEigenVectors(roiY, roiX, 2));
title("first eigen vector");

subplot(rows, cols, 8);
imshow(sEigenValues(:, :, 1), []);
title("first eigen value");
subplot(rows, cols, 9);
imshow(sEigenValues(:, :, 2), []);
title("second eigen value");
saveas(gcf,'../images/ex02.eps','epsc')
```