Report Blatt 3

Alexander Attinger, Yannic Kilcher

November 16, 2012

1 Exercise 1 Edge Detection

As descriped in sheet one for the Laplacian, edges are usually denoted by strong change of intensity over short distances. Thus they can be detected by calculating the gradient of the intensity and make use of the fact that the gradient will usually have a maximum at potential edges. Both the sober edge detector and the canny edge detector try to make use of this fact.

1.1 Sobel Edge Detection

The sobel operator is a specific filter convolved with the Image P. It is used to approximate the calculation of the gradient in x and y direction. To compute the gradient G_x in x direction:

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} * P$$

is used and for y direction G_y:

$$G_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} * P$$

is used. OpenCV implements the Sobel function, i.e. it allows to calculate $G_x and G_y$. The gradient magnitude G can be calculated in different ways, we used:

$$G = .5 * |G_x| + .5 * |G_x| \tag{1}$$

We then implemented a simple thresholding on G to label pixels as Edge/background. As can be seen in the pictures, the detection works ok, at least for the human eye. Some draw backs:

- Only local information is used. The sober edge detector does not try to extend edges or try to close gaps.
- Threshold value had to be adjusted for each picture individually

- In images with a clear foreground/background edge detection on the foreground object works fairly well (e.g. Butterfly)
- In images with no clear foreground/background distinction such as the stairs image or the outdoors image, the problem gets really hard.

1.2 Canny Edge Detection

The canny edge detector tries to make up of some of the shortcomings of the sobel method. At the heart of the canny detector still lies a sobel kernel. But it includes some extra mechanisms (as presented in the lecture):

- Non-maximum supression
- Hysteresis

The OpenCV implementation lets you set a range of parameters, most importantly the upper and lower hysteresis thresholds. Before passing the images to the Canny Detector, they also need to be smoothed, this was done with a simple gaussian filter. In order to test a wide range of values quickly, I set the upper threshold to be three times the lower threshold and implemented slider buttons for both the kernel size of the gaussian and the lower threshold. They both have a similar effect, generally, increasing the values will decrease the number of edges detected, which is what we would expect. Again, different values have to be used for the different pictures. Also the result is somewhat arbitrary, since in some pictures, even for us it was sometimes hard to see all edges. In general noise is a big problem.

1.3 Comparison

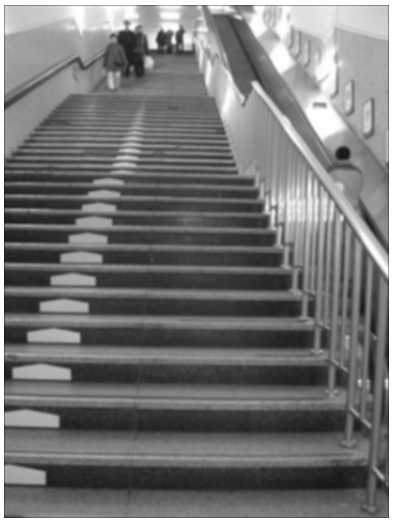
Canny Edge detector lets you set more parameters, this can be useful, but it usually also takes longer to find an optimal combination of the different parameters. Canny edge detector seems more robust than the sobel one, additionally edges are only 1px thick, allowing clear localization of the edge. I would always prefer the canny edge detector over the sobel edge detector, since at the heart of the canny edge detector lies a sobel operation. However, with the hysteresis thresholding, the canny edge detector tries to make edges more continuous. Although finding an optimal set of parameters can be tedious, I guess once it is found for a particular picture, it can be reapplied to similar pictures (i.e. pictures taken under the same conditions). In general, edge detection is a very complex problem, especially in complex scenes. Even for us it is sometimes hard to judge in pictures such as cells.jpg where the true edges are. Additionally sometimes we don't want to detect all the edges, but only a subset of them. Also, for us humans equipped with stereo vision, we know at leas to basic classes of edges which are fundamentally

(a) Original Image 1

(b) Sobel

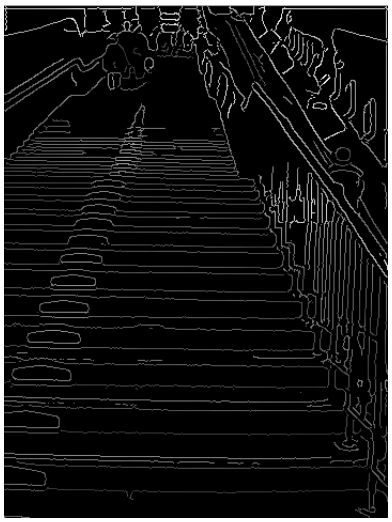
(c) Canny

Figure 1: Butterfly



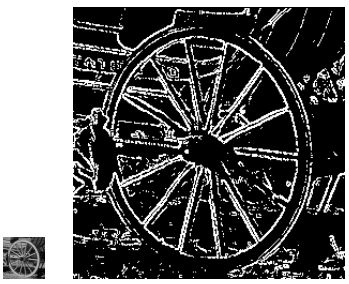
(a) Original Image

(b) Sobel



(c) Canny. In the original sized image, edges are visible more clearly.

Figure 2: Stairs



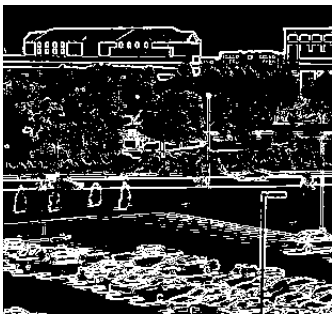
(a) Original Image

(b) Sobel

(c) Canny.

Figure 3: Wheel

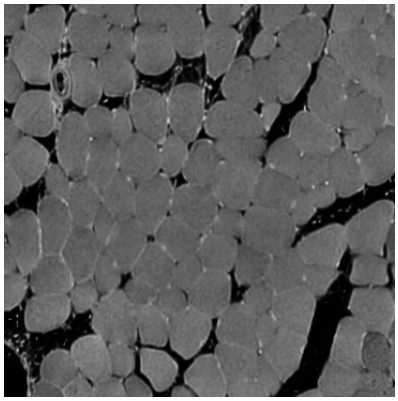
(a) Original Image



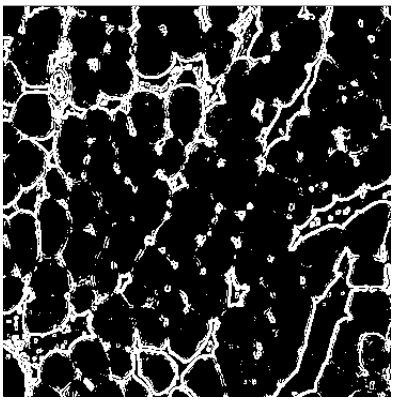
(b) Sobel

(c) Canny.

Figure 4: Outdoor



(a) Original Image



(b) Sobel

(c) Canny.

Figure 5: Cells

different: texture edges (such as black paint on white paper) and spatial edges (edge of table,...). In a 2D image, this distinction cannot be made.

2 Hough transformation

2.1 Step size

We have compared the results of the Hough transformation while varying different parameters. For the parameter rho (Fig. 6), we have discovered that making it too large will result in some lines being "overlooked", as the big step sizes simply step over the direction of a line. For theta (Fig. 7), a larger value will discover more lines. This is, because when more finely grained, votes for the same line in the image will be distributed among neighboring accumulator cells, where as when coarsly grained, a single accumulator cell might indicate multiple similar lines. For the number of votes needed to be classified as a line (Fig. 8), we obtained the intuitive result of getting more lines, the less votes are needed.

2.2 Line detection

Here, we have summarized our results in Fig. 9 with tuned parameters.

2.3 Complexity

Since the algorithm has to test for every combination of rho and theta, the complexity is proportional to the number of rhos times the number of thetas.

2.4 Multiple lines detected

Our approach to solving the propblem of multiple similar lines being detected for the same line in the image is a rather simple one. Since these similar lines lie closely together in the hough accumulator (mostly neighboring cells), we need a way to combine clusters of high valued cells. We would do this by blurring the image, for example by convolving with a gauss kernel and then either apply a threshold function to the accumulator or subsample the accumulator to halve it in size. Repetition of this should eliminate the double-detection of the same lines.

(a) Rho = 0.5° (b) Rho = 1.0° (c) Rho = 2.0°

Figure 6: Hough transform for detecting lines with varying rho.

(a) Theta = 0.5 (b) Theta = 1.0 (c) Theta = 2.0

Figure 7: Hough transform for detecting lines with varying theta.

(a) N = 80 (b) N = 100 (c) N = 120

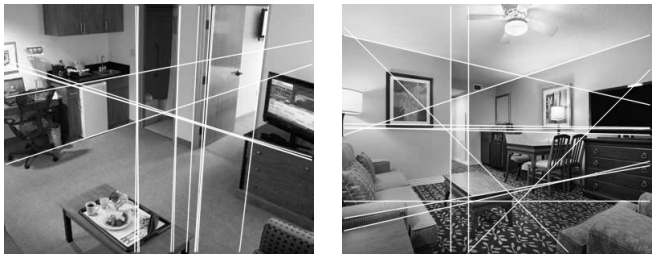
Figure 8: Hough transform for detecting lines with varying number of votes N needed to be classified as line.

2.5 HoughLinesP

We have tried out the HoughLinesP function with the same parameters as above, except that we adjusted the number of votes needed to 1/3rd of the above numbers. We have discovered that this function is better at finding also short lines. The results can be seen in Fig. 10

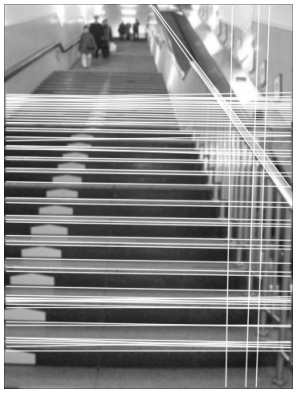
(a) theta=1,rho=1°,N=100

(b) theta=3/4,rho=3/4°,N=140



(c) theta=3/4,rho=3/4°,N=77

(d) theta=3/4,rho=3/4°,N=105



(e) theta=1,rho=1°,N=130

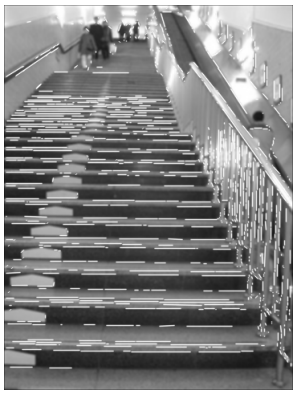
Figure 9: Line detection with Hough, preprocessed with Canny

(a) theta=1,rho=1°,N=33

(b) theta=3/4,rho=3/4°,N=46

(c) theta=3/4,rho=3/4°,N=25

(d) theta=3/4,rho=3/4°,N=35



(e) theta=1,rho=1°,N=43

Figure 10: Line detection with HoughLinesP, preprocessed with Canny