

Computer Vision SS22 Assignment 1: Document Scanner

Felix Hamburger

Student ID: 35925

Computer Vision SS22

Computer Science Master

Ravensburg Weingarten University

Email: felix.hamburger@rwu.de

Mario Amann

Student ID: 35926

Computer Vision SS22

Computer Science Master

Ravensburg Weingarten University

Email: mario.amann@rwu.de

This paper presents a usecase for a DIN A4 document scanner. The explanations contained in this paper will be divided into three steps. First the corner points of the document will be detected utilizing the canny edge corner detection. By performing a perspective transformation the document will then be provided in a top-view. With the help of a filter the document will then be changed into a binary format.

1 INTRODUCTION

In everyday life, there are often situations in which it is advantageous to scan a document that has been received and keep it as a copy. Unfortunately not everyone has a ready-to-use scanner with them to scan the document. Though most of the people have a device capable of doing just this - a cell phone with a camera. Using various methods, it is possible to scan documents with the help of the cell phone camera, similar to a scanner, and save them in a suitable format. This paper will give your step by step guide on how it is possible to implement these methodologies.

2 PREPROCESSING

The first step is to read the image (Figure 1). This is done with the imread function of openCV which outputs the image as numpy array with the shape (Height, Width, Channel). By default the decoded image is stored in the BGR-format. Which means, the first channel contains the blue color channel, the second one contains the green color channel and the last the red one.

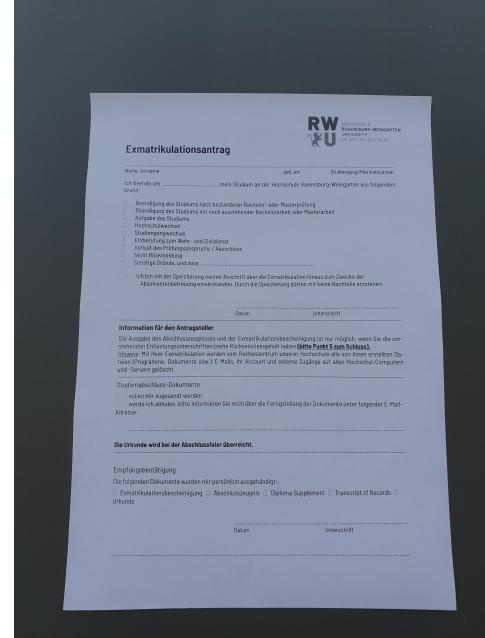


Fig. 1. Original image.

The original image is separately stored, because for further steps the photo is needed in a grayscale format (Figure 3). This conversion is made with the cvtColor[Doce] function of openCV. The conversion of colorspace is made with the following formula[Docc]:

$$Y = B * 0.114 + G * 0.587 + R * 0.299$$

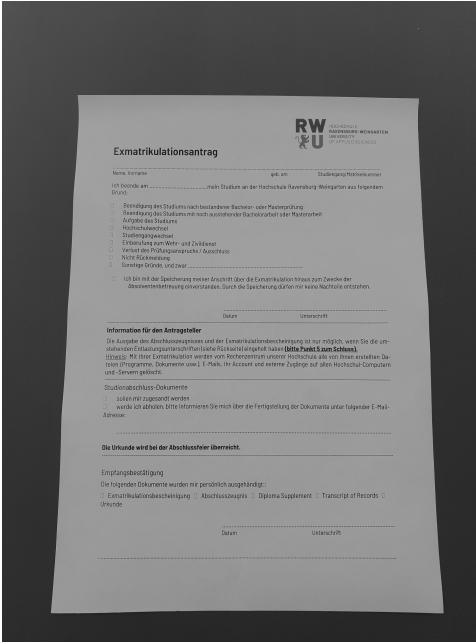


Fig. 2. Grayscale image.

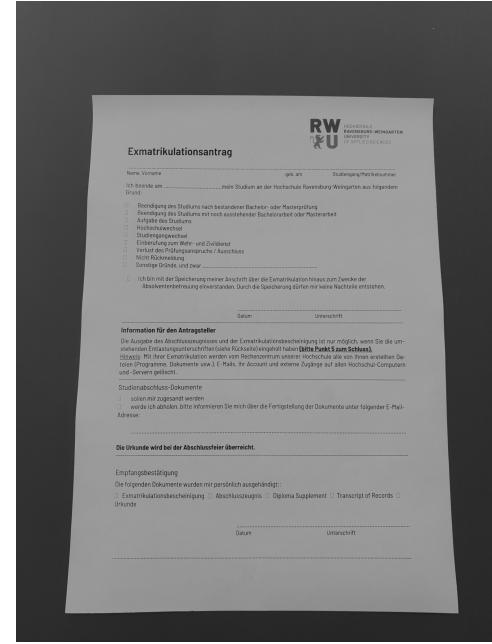


Fig. 3. Grayscale image with Gaussian Blur.

The next preprocessing step includes the smoothing/blurring of the grayscale image. This is done to remove noise from the image[[Docj](#)]. In our case the Gaussian Blurring[[Docg](#)] is used. The image is convolved with the Gaussian Kernel. As size of the kernel the size (5,5) is used. Because a document could be landscape or portrait format, it makes sense to use same values in x and y direction in the kernel size, as well as for the parameters sigmaX and sigmaY. The optimum values of these parameters were determined from the results of several scans of sample documents. The parameters sigmaX and sigmaY are used to calculate the gaussian filter coefficients. In the function call sigmaX and sigmaY are set to zero, which means the Gaussian Kernel computes these variables from the given ksize[[Doch](#)].

$$\sigma = 0.3 * (0.5 * (ksize - 1) - 1) + 0.8$$

The gaussian filter coefficients are computated with the following formula[[Doch](#)]:

$$G_i = \alpha * e^{-\frac{(i - \frac{(ksize-1)}{2})^2}{2*\sigma^2}}$$

3 EDGE DETECTION

After we took our preprocessing step and chose a suitable noise reduction algorithm we start with the edge detection. For edge detection we use the Canny edge[[Can86](#)] detector which is a multi-staged algorithm:

1. Noise reduction.
2. Calculating the intensity gradient of the image.
3. Non-maximum supression.
4. Hysteresis thresholding.

The noise reduction step was allready described in the [Preprocessing](#) section of this paper.

The intensity gradient of the image is calculated using the **Sobel Filter**:

$$S(I(x, y)) := \sqrt{(S_x * I(x, y))^2 + (S_y * I(x, y))^2} \quad (1)$$

Generally we can define the gradient of the Image I as:

$$G(I(x, y)) := \sqrt{I_x^2 + I_y^2} \quad (2)$$

In addition to the gradient we calculate the the orientation given by:

$$\phi(x, y) = \arctan\left(\frac{g_y}{g_x}\right) \quad (3)$$

After getting the gradient magnitude and orientation of each pixel, the edges have to be reduced to a thickness of one pixel which will be realized with **non-maximum suppression**.

Lastly it is decided whether a pixel will be mapped to 1 or 0 with application of **hysteresis**.

To implement this methodology in the document scanner, the function which implements the full Canny edge detector[[Docb](#)] can be called.

```
Canny(blurred_image, 75, 180,
L2gradient = True, apertureSize = 3)
```

The functions input is the noise reduced image described in [Preprocessing](#). For Hysteresis Threshold 75 for the lower-bound and 180 for the upper bound are chosen. In several trials this settings could sufficiently provide the overall best results, when setting the aperture size to 3. Instead of the predefined L1-Norm [[Weia](#)], the more precise L2-Norm [[Weib](#)] is used.

The canny edge detection filter returns an binary output image with edges being set to 1 and other points being set to 0.

This can be seen in Figure 4

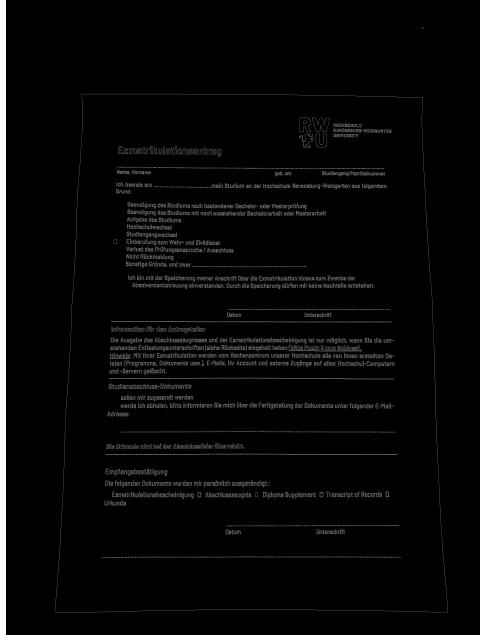


Fig. 4. The output image of the canny edge detection.

4 CORNER DETECTION

To find the corner of the document the previously generated edges in [Edge Detection](#) have to be evaluated. For this purpose, contours are formed which are represented by a curve that connects all continuous points with each other.[[Sb85](#)]

The OpenCV function[[Docf](#)] is used as follows:

```
findContours(edges,
RETR_LIST, CHAIN_APPROX_SIMPLE)
```

Where the input are the previously generated edges, the retrieval mode[[Doci](#)] which is set to retrieve the contours without establishing any hierarchical relationships and the contour approximation mode[[Docd](#)] which is set to output only their endpoints. (E.g: A rectangular contour is defined by the four corner points) This can be seen in Figure 5

The assumption is made that the document is the largest rectangle in the image. The contours are hierarchically sorted according to their area, with the largest polygon with four vertices being selected.

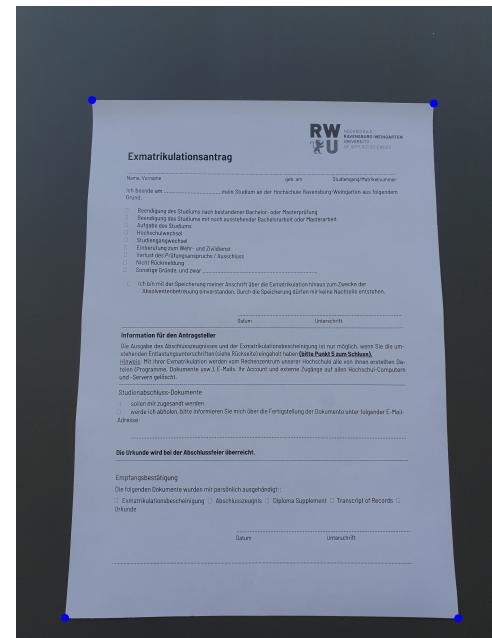


Fig. 5. The corners of the document.

[[DP73](#)] [[Ram72](#)]

5 TRANSFORMATION

Before the transformation can be performed the orientation of the document in the image must be found out.

Without knowing the content of the document, it is not possible to know the real orientation. Therefore a few assumptions are made:

1. All documents are in DIN A4 format
2. All provided documents are portrait format
3. All provided documents are only rotated 90 degrees right or left, otherwise the document is scanned upside down.

These assumptions define the output of the scanner: A document in portrait format. The first task is to find out if the document is landscape or portrait in the image. Four examples of how the document could have been recorded can be seen in Figure 6. What is known from the list of the given four corners from the previous section is that the first element (blue dot) is the most upper point in the image. Next, the distance of this point must be calculated with the second and fourth elements (red dots) in the list and these two distances compared. For this purpose, the Euclidean distance calculation is used. If the distance to the second element is higher than to the fourth element, the document is recorded portrait. Else it was recorded landscape. Relative to the previous result, the list of found points must be put into this form:

[upper_left, upper_right, lower_right, lower_left]

This is used to calculate the perspective transformation from the given points to the destination points with the `getPerspectiveTransform` function from OpenCV[?]. The target points are the set points of the document in DIN A4 format.

```
[[0,0], [width,0], [width,height], [0,height]]
height = 3000
width = height / sqrt(2)
```

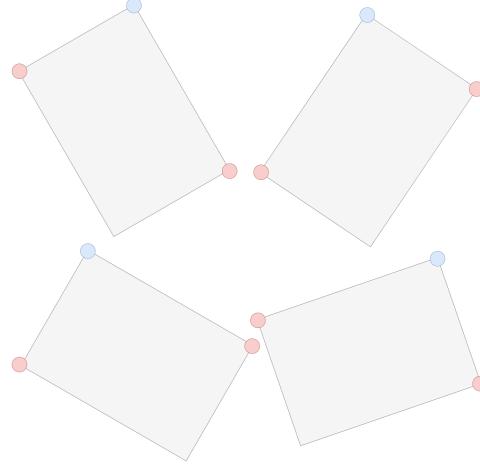


Fig. 6. Four examples of the orientation of a scanned document

The last step of this section is to apply the perspective transformation to the grayscale image. For this purpose the `warpPerspective` function from OpenCV[?] is used:

`warpPerspective(grayscale_image, M, (width, height))`

The function uses the grayscale image, the calculated perspective Transformation(M), and the size of the output image to perform the transformation. The optional parameters are set to default. The output of the transformation can be seen in Figure 7.

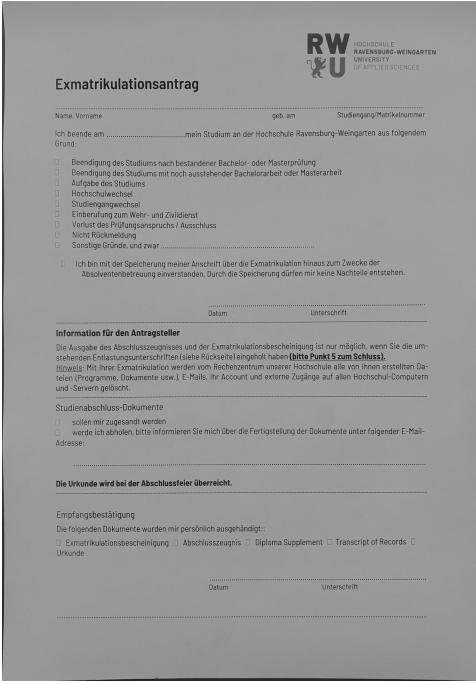


Fig. 7. Document after the transformation.

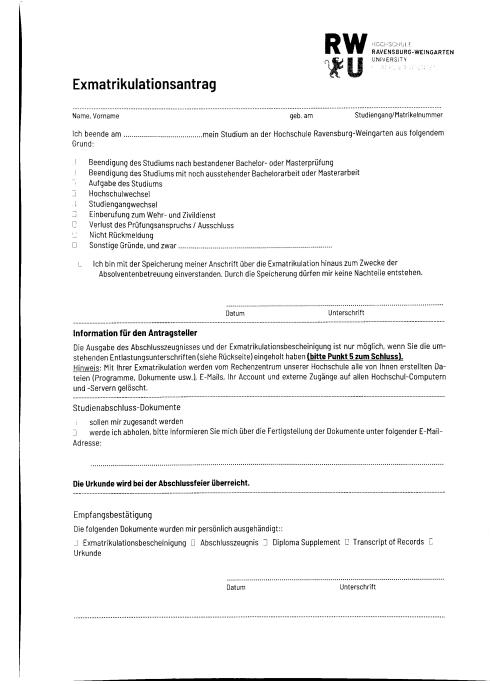


Fig. 8. The generated binary image.

6 BINARY IMAGE

In the last step of the process, the transformed image is converted into a binary image. In various tests, it has been found that an adaptive threshold[[Doca](#)] is best suited here. Both an adaptive gaussian threshold[[?](#)] and a adaptive mean threshold show great performance, though an adaptive gaussian threshold tends to generate noisier outputs.

```
adaptive_binary_image_mean =
adaptiveThreshold(transformation, 255,
ADAPTIVE_THRESH_MEAN_C,
THRESH_BINARY, 7, 6)
```

The chosen adaptive mean threshold value is a mean of the blocksize 7×7 minus the constant $c = 6$. The image generated after this process can be seen in figure 8

7 SUMMARY

The example within the report and the two other examples attached below (Figures [9,10,11,12](#)) show that the scanner does work even if the document is recorded in portrait or landscape format. The pictures of the documents show a small problem, that the recorded documents were wavy. Therefore the output documents show a wavy edge, but the content is still perfectly readable. The image of document of Figure [9](#) is even worse readable than the transformed document seen in Figure [10](#).

For further optimization it would be possible to add a character recognition to find out the real orientation of the document. This would fix the cases when the document has landscape format or is upside down and the assumptions 2 and 3 in the [Transformation](#) section didn't have to be made.



Fig. 9. Document recorded in landscape.

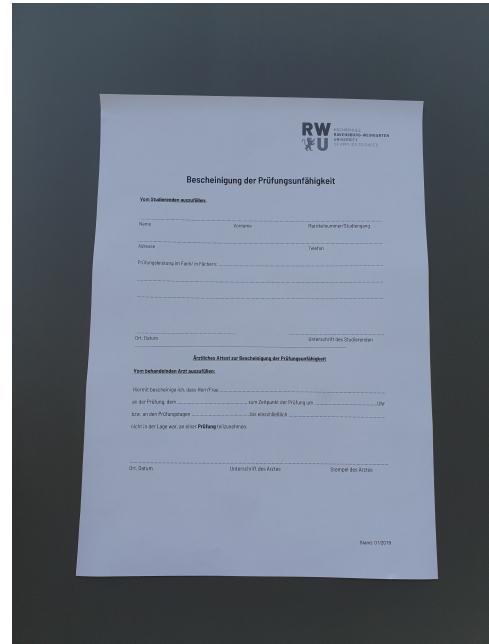


Fig. 11. Document recorded in landscape.

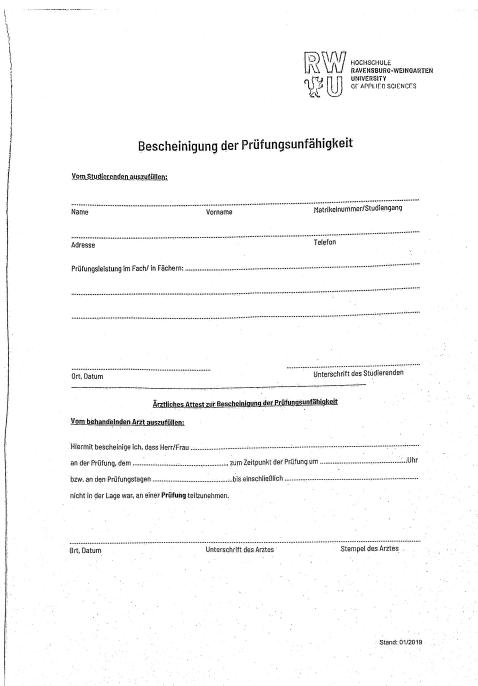


Fig. 10. Document recorded in landscape transformed into binary document

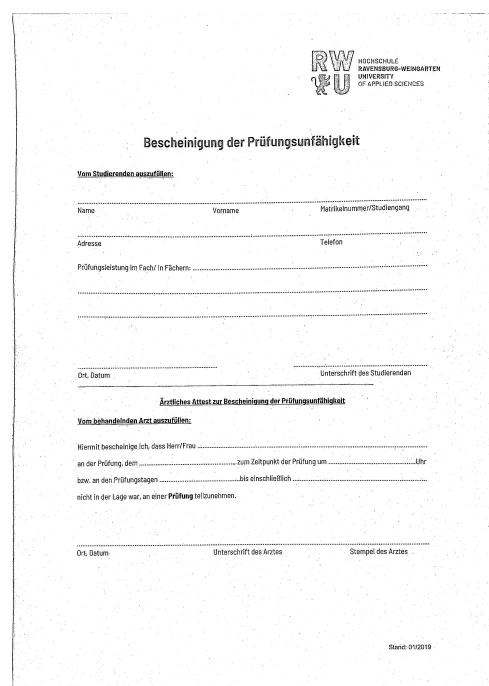


Fig. 12. Another document example transformed into binary document

REFERENCES

- [Can86] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.
- [Doca] OpenCV Documentary. adaptivethreshold. https://docs.opencv.org/4.x/d7/d1b/group__imgproc__misc.html#ga72b913f352e4a1b1b397736707afcde3. Accessed: 2022-05-12.
- [Docb] OpenCV Documentary. Canny. https://docs.opencv.org/4.x/dd/d1a/group__imgproc__feature.html#ga04723e007ed888ddf11d9ba04e2232de. Accessed: 2022-05-09.
- [Docc] OpenCV Documentary. Color conversion rgb to gray. https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html#color_convert_rgb_gray. Accessed: 2022-05-09.
- [Docd] OpenCV Documentary. Contourapproximationmodes. https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html#ga4303f45752694956374734a03c54d5ff. Accessed: 2022-05-12.
- [Doce] OpenCV Documentary. cvtcolor. https://docs.opencv.org/3.4/d8/d01/group__imgproc_color_conversions.html#ga397ae87e1288a81d2363b61574eb8cab. Accessed: 2022-05-09.
- [Docf] OpenCV Documentary. Findcontours. https://docs.opencv.org/4.x/d3/dc0/group__imgproc_shape.html#gadfad6a0b82947fa1fe3c3d497f260e0. Accessed: 2022-05-12.
- [Docg] OpenCV Documentary. Gaussianblur. https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gaabe8c836e97159a9193fb0b11ac52cf1. Accessed: 2022-05-09.
- [Doch] OpenCV Documentary. getgaussiankernel. https://docs.opencv.org/4.x/d4/d86/group__imgproc__filter.html#gac05a120c1ae92a6060dd0db190a61afa. Accessed: 2022-05-09.
- [Doci] OpenCV Documentary. Retrievalmode. https://docs.opencv.org/4.x/d3/dc0/group__imgproc__shape.html# ga819779b9857cc2f8601e6526a3a5bc71. Accessed: 2022-05-12.
- [Docj] OpenCV Documentary. Smoothing images. https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html. Accessed: 2022-05-09.
- [DP73] DAVID H DOUGLAS and THOMAS K PEUCKER. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.
- [Ram72] Urs Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, 1972.
- [Sb85] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.
- [Weia] Eric W. Weisstein. L1-norm. <https://mathworld.wolfram.com/L1-Norm.html>. Accessed: 2022-05-09.
- [Weib] Eric W. Weisstein. L2-norm. <https://mathworld.wolfram.com/L2-Norm.html>. Accessed: 2022-05-09.