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 report presents a usecase for a document scanner following the international standard ISO 216. The explanations contained in this report will be divided into five steps. First a preprocessing step reads the image and performs basic operations on the image. Then 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 report will give a step by step guide on how it is possible to implement a document scanner with Python and OpenCV[**?**].

2 PREPROCESSING

Before we can apply any preprocessing step we have to read the image (Figure **??**). This is done with the imread function[**?**] of OpenCV which outputs the image as numpy array with the shape (Height, Width, Channel).

$$imread(path)$$

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.

output/hoch_3_1_original.jpg

Fig. 1. Original image.

For further steps the photo is needed in a grayscale format (Figure **??**). This convertion is made with the cvtColor[**?**] function of OpenCV. The convertion of colorspace is made with the following formula[**?**]:

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

$$cvtColor(image, COLOR_BGR2GRAY)$$

output/hoch_3_2_grayscaleimg.jpg

Fig. 2. Grayscale image.

output/hoch_3_3_gaussianblur.jpg

Fig. 3. Grayscale image with Gaussian Blur.

The next preprocessing step includes the blurring of the grayscale image. This is done to remove noise from the image[**?**]. In this case the Gaussian Blurring[**?**] is used. There 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 optimal 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[**?**].

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

The gaussian filter coefficients are computated with the following formula[**?**]:

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

The Gaussian Blur[**?**] function is used as follows:

$$GaussianBlur(grayscale_image, ksize = (5,5), sigmaX = 0, sigmaY = 0)$$

3 CORNER DETECTION

After preprocessing steps are taken and a suitable noise reduction algorithm was chosen, the edge detection can be started. For edge detection the Canny edge[**?**] detector is used, 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 already described in the **??** 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:

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

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

To implement this method in the document scanner, the function which implements the full Canny edge detector[**?**] can be called:

$$Canny(blurred\_image, 75, 180,$$
$$L2gradient = True, apertureSize = 3)$$

The functions input is the noise reduced image described in **??**. 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 [**?**], the more precise L2-Norm[**?**] 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 **??**

output/hoch_3_4_canny.jpg

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

4 CAMERA CALIBRATION

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

The OpenCV function[**?**] is used as follows:

$$findContours(edges,$$
$$RETR\_LIST, CHAIN\_APPROX\_SIMPLE)$$

Where the input are the previously generated edges, the retrieval mode[**?**] which is set to retrieve the contours without establishing any hierarchical relationships and the contour approximation mode[**?**] 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 **??**

The assumption is made that the document is the largest polygon in the image. The contours are hierarchically sorted according to their area, with the largest polygon with four vertices being selected[**?**][**?**]. The polygon is found by using following functions[**?**][**?**]:

$$arcLength(contour, true)$$

$$approxPolyDP(contour,$$
$$epsilon, true)$$

Where we first calculate the curve perimeter which will be used as input for the approximation. The approximation accuracy is set to 0.05 times the curve perimeter and the bool is set to true, which indicates that the curve must be closed, thus the first and last vertices are connected.

output/hoch_3_5_contouredimage.jpg

Fig. 5. The corners of the document.

5 UNDISTORTION

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 provided documents are in ISO 216 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 **??**. What is known from the list of the given four corners from the previous section is that the first element is the most upper point in the image. Next, the distance of this point must be calculated with the second and fourth elements 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. Otherwise it was recorded landscape. Relative to the previous result, the list of found points must be put into this form:

$$[\text{upper\_left}, \text{upper\_right}, \text{lower\_right}, \text{lower\_left}]$$

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 ISO 216 format.

$$[[0, 0], [\text{width}, 0], [\text{width}, \text{height}], [0, \text{height}]]$$
$$\text{height} = 3000$$
$$\text{width} = \frac{\text{height}}{\sqrt[2]{2}}$$

output/a.png

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

$$getPerspectiveTransform(src_pts, dst_pts)$$

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 **??**.

output/hoch_3_6_transformation.jpg

Fig. 7. Document after the transformation.

6 SUMMARY

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[**?**] is best suited. Both an adaptive gaussian threshold[**?**] and an adaptive mean threshold show great performance, though an adaptive mean threshold tends to generate less noise in the outputs.

$$adaptiveThreshold(transformation, 255,$$
$$ADAPTIVE\_THRESH\_MEAN\_C,$$
$$THRESH\_BINARY, 7, 6)$$

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 **??**

output/hoch_3_8_adaptive_binary_image_mean.jpg

Fig. 8. The generated binary image.