Project 3

Isabela Telles Furtado Doswaldo - RA: 170012

MC920 - Introduction of Digital Image Processing
University of Campinas

1 Introduction

The goal of the project is to implement an algorithm that is able to align images of documents.

The document image analysis is used to recognize the text and graphics components in image of documents in order to extract the intend information as a human would. One of the most applied product of this analysis is the Optical Character Recognition (OCR) software that is able to recognize characters in a scanned document. It is really useful nowadays, since the document digitization has become more and more common.

Nevertheless, an inherently associated problem with document image analysis is the skew angle detection, which specifies the deviation of the text lines from the horizontal axis. Some degree of skew is unavoidable either a document image is scanned manually or mechanically. Since the skew can influence directly in procedures as the OCR, it is necessary to apply a skew detection and correction technique in order to obtain a skew-free document image.

The program developed in this project implements two algorithms to skew detection and correction, one of them based in horizontal projection profile and the other one base in Hough transform, both using Python, along with the following libraries: OpenCV, Matplotlib, Numpy, Pytesseract and PIL.

The images used for testing are available in http://www.ic.unicamp.br/~helio/imagens_inclinadas_png/.

2 Methodology

The program was designed using two different methods as base, so the user can choose which one is going to be used to align the image. Thus, the program requires the method as an argument, but it also requires the input and output image, as you can see below:

1. Input image: path to the PNG image file before the alignment
2. Mode: technique used to align the image (0: based on Hough transform; 1: based on horizontal projection profile)
3. Output image: path to the PNG image file after the alignment

See an example of usage to execute the program when the input image is `pos_24.png`, the mode is based on Hough transform and the output image is `correct_pos_24.png`

```
$ python3.5 align.py pos_24.png 0 correct_pos_24.png
```

After the program has initiated, the image was loaded from a file into a 3D numpy array using the OpenCV library's function *cv2.imread()* along with the flag `cv2.IMREAD_GRAYSCALE`, which allowed that the image was read in grayscale mode.

In order to verify how the program can be used in document image analysis, the development of OCR before and after the process of skew detection and correction was pointed out. Therefore, after loading the image, a thresholded image was obtained through an algorithm of local thresholding:

the adaptive thresholding, given by the function *cv2.adaptiveThreshold()*, which calculates the threshold for small regions of the image. The method used was the Gaussian method, defined by a flag `cv2.ADAPTIVE_THRESH_GAUSSIAN_C`, that determines the threshold value as the weighted sum of neighbourhood values where weights are a gaussian window. Since the goal was to obtain a clear image in order to obtain better results using OCR, the threshold type used was the binary, defined by the flag `cv2.THRESHOLD_BINARY`.

2.1 Horizontal Projection Profile-Based Technique

The horizontal projection profile is a traditional and simple approach to detect the skew angle of a document image. A series of horizontal projection profile H is a useful region-based signature defined by the sum of the black pixels in each line of the image, as you can see in equation (1).

$$H(y) = \sum_{x=0}^{M-1} f(x, y) \tag{1}$$

First of all, it was necessary to obtain a binary image, where pixels of the text were represented by 1 and pixels that weren't text were represented by 0. So, the max value of the thresholded image was set to 0 and the zero value of the same image was set to 1.

For comparison purposes, the calculation of the horizontal projection of the input image was calculated, using the Numpy function *np.sum()*, that performs a sum of the array elements over the axis given, the horizontal axis in this case. The projection was then exhibited as a histogram, as you can see in Figure 2.

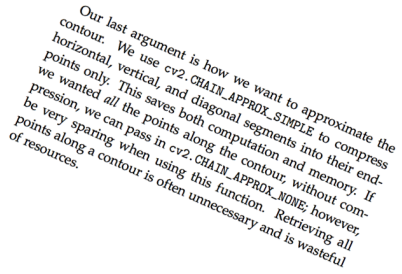

Figure 1: Input Image

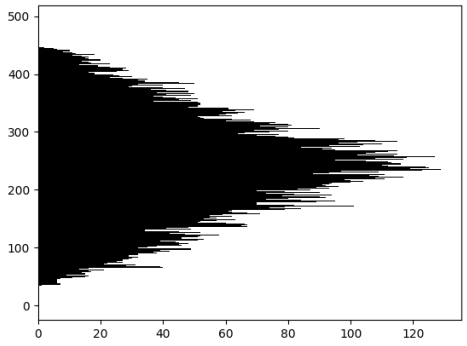

Figure 2: Histogram of Horizontal Projection Profile of Input Image

In order to find out the skew angle the image must be rotated in a range of -90º to 90º, which covers 180 degrees of rotation, since it is assumed that the document image provided won't be upside down.

The function *cv2.getRotationMatrix2D()* of OpenCV finds a transformation matrix M when the center of rotation is the center of the image.

$$M = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

Then, it is easy to see that the sine and cosine will be given by M[0, 1] and M[0, 0]. The new bounding dimensions of the image can be calculated with the sine and cosine found, ensuring the image won't be cut off. With the new height and width of the image, the rotation matrix can finally be adjusted to take into account the translation of the image and the function *cv2.warpAffine()* is able to perform the actual rotation of the image.

For every rotated image, the horizontal projection was calculated. The profile with the maximum variation between peaks and valleys is the one that provides the best alignment of the document image, once the number of collinear text pixels maximizes. In order to find out which one of the profiles is the one there is a optimization function given by (2) that calculates a score S for every profile, assuming x_i is an element of the profile, with $i = 0, 1, ..., N$.

$$S = \sum_{i=1}^{N} (x_i - x_{i-1})^2 \tag{2}$$

The angle of rotation in which the profile has the maximum score is the skew angle, since the maximum score only quantifies the biggest variance between the level of side by side bars. The horizontal projection profile of the skew free image can be seen in Figure 4.

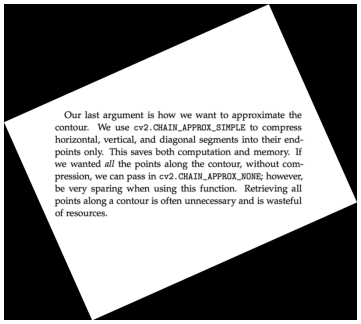

Figure 3: Output Image

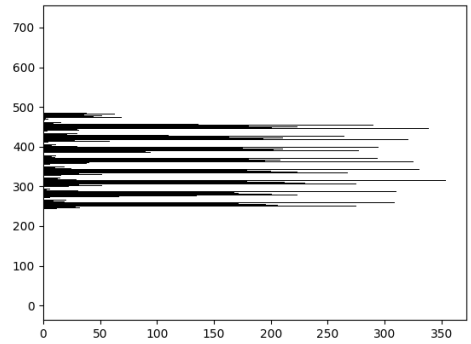

Figure 4: Histogram of Horizontal Projection Profile of Output Image

The OCR can be applied over the output image, using the function *image_to_string()* from Pytesseract, along with the function *Image.fromarray()* from PIL library that creates an image memory from a numpy array. The return for the image used as example in all figures is the following text, which is a much better result than when OCR was applied in input image, since it returned an empty text.

```
 Our last argument is how we want to approximate the
contour.  We use cv2.CHAIN_APPRDX_SIMPLE to compress
horizontal, vertical, and diagonal segments into their end-
points only.  This saves both computation and memory.  If
we wanted all the points along the contour, without com-
pression, we can pass in cv2.CI-IAIN_APPRDX_NUNE; however,
be very sparing when using this function.  Retrieving all
points along a contour is often unnecessary and is wasteful
of resources.
```

2.2 Hough Transform-Based Technique

The Hough transform is a popular approach to detect the skew angle of a document image, because it is a technique used primary to detect lines. A line is represented as $y = mx+b$, or in a parametric form $\rho = x\cos\theta + y\sin\theta$, where ρ is a perpendicular distance from the origin to the line, and θ is the angle formed by this perpendicular line and the horizontal axis measured in counter-clockwise. Then, any line can be represented by the pair (ρ, θ), which has an accumulator that is incremented by one in its corresponding cell for every pixel of the line present in the image.

OpenCV has a function *cv2.HoughLines()* that returns an array of (ρ, θ), so it takes a lot of computation, because the endpoints of the lines are not calculated automatically. That is why the

function used was actually *cv2.HoughLinesP()*, which implements a Probilistic Hough Transform, an optimization of the Hough Transform. The function receives a binary image as argument, so first of all, a canny edge detection is applied with the function *cv2.Canny()*. The result can be seen in Figure 5. With the edges of the image, the minimum length of line defined by 100 and the maximum allowed gap between line segments to consider them a single line defined by 10, the function *cv2.HoughLinesP()* is applied and returns two endpoints of each line.

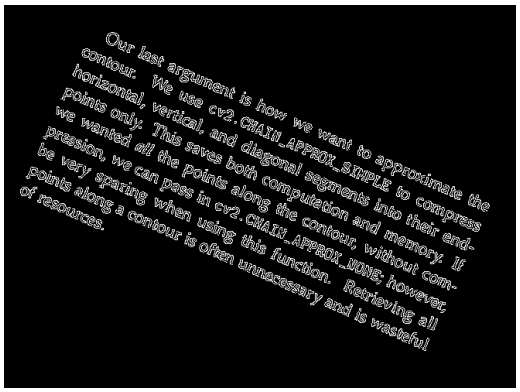

Figure 5: Edges of Image

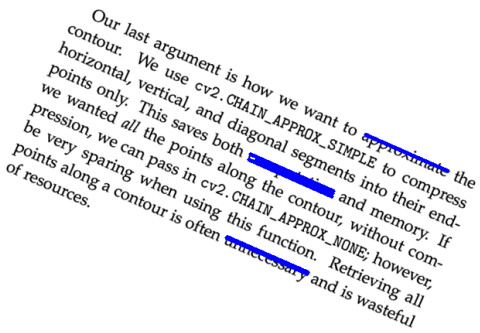

Figure 6: Detected Lines in Image

The angle of each dominant line found by the algorithm is given by (3). The first endpoint of the line is (x_1, y_1) and the second is (x_2, y_2).

$$\theta = \arctan2(y_2 - y_1, x_2 - x_1) \tag{3}$$

The angles were stored in an array of angles, and the skew angle was found by finding the median of these angles with the function of Numpy library *np.median()*.

Again, the OCR was applied over the output image and it returned a much better result than when the OCR was applied in input image, as you can see below.

```
 Our last argument is how we want to approximate the
contour.  We use cv2.CHAIN_APPRDX_SIMPLE to compress
horizontal, vertical, and diagonal segments into their end-
points only.  This saves both computation and memory.  If
we wanted all the points along the contour, without com-
pression, we can pass in cv2.CHAIN_APPRDX_NDNE; however,
be very sparing when using this function.  Retrieving all
19th along a contour is often unnecessary and is wasteful
of resources.
```

3 Conclusion

Both methods have fulfilled their role, the horizontal projection profile-based found the skew angle of 24º for the example presented in sections above, while the Hough transform-based found the skew angle of 23.83º. Since none of the images provided were an upside down image, all of them were correctly aligned. The performance of both algorithms weren't excellent when it was runned with larger images, and the OCR couldn't succeed in recognizing the characters of the larger image of test `sample2.png`, probably because it is an image with a lot of information spread around the document in columns, one centered, one to the left and one to the right. The use of a bounding box technique to segment the image in each column would likely improve the result of an image like that.

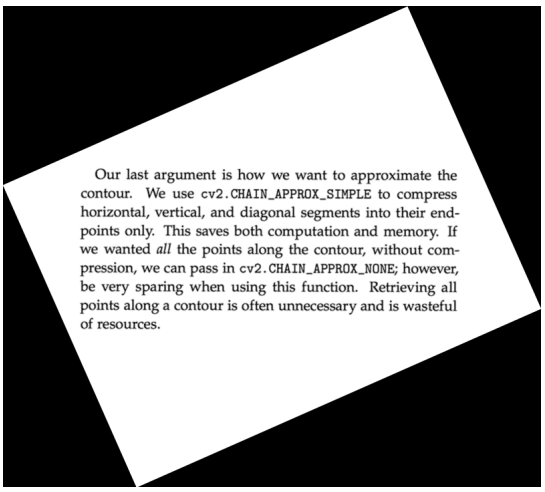

Figure 7: Output Image