Skew Detection for Complex Document Images Using Fuzzy Runlength

Zhixin Shi and Venu Govindaraju
Center of Excellence for Document Analysis and Recognition(CEDAR)
State University of New York at Buffalo, Buffalo, NY 14228, U.S.A.

Abstract

A skew angle estimation approach based on the application of a fuzzy directional runlength is proposed for complex address images. The proposed technique was tested on a variety of USPS parcel images including both machine print and handwritten addresses. The testing results showed a successful rate more than 90% of the test set.

1. Introduction

The skew angle estimation and correction of a document page is an important task for document analysis and optical character recognition (OCR) applications. A wide variety of methods have been proposed in the literature. The Projection Pro£le technique[5, 2] detects the skew angle by creating a histogram at each possible angle. Then a cost function is applied to these histograms. The skew angle is the angle at which this cost function is optimized. The methods using Hough Transform are theoretically identical to the methods using projection pro£les[1]. Hough Transform is usually applied on a set of selected angles along each angle straight lines are determined with a measurement for the £t. The best £t for the lines gives the skew angle. Hough Transform can be applied on all black pixels[6], on reduced data from horizontal and vertical runlength computations[4] or only on the bottom pixels of connected components[3]. Another method uses nearest neighbor clustering of connected components found in [9].

However, most of the proposed approaches are designed mainly for machine printed documents. They are usually able to deal with small skew angles within $\pm 15°$, failing to manage cases of documents that may exceed this limit. Moreover, some of them entail high computational cost, especially in the case where the Hough transform is used. Also, certain approaches are font, column, graphics or border dependent. There are very few methods proposed to handle handwritten or mixed documents[8, 7]. Some of them are designed for speci£c applications. Most of these methods can not handle handwritten documents very well.

This is mainly because of the complexity of handwritten documents including variations in writing styles, sizes of characters and touching or connection of characters, words and text lines.

In this paper, we propose a novel method for complex document skew angle estimation. The method is able to detect the per-dominant direction of the text in the document image. The text can be handwritten, machine printed or mixed. The main advantage of the proposed method is the ability to extract the text line information from document mixed with non-text elements such as graphics, bar-codes and forms. The method £rst transforms the content of a document page into components representing text words, line or graphics areas. Then the extracted components are ranked as being possible types of text or non-text. Finally the per-dominant direction of the text in the image is estimated using minimum squared distance optimization and component grouping method.

In Section 2 we will discuss the complexity of handwritten and mixed documents. The basic principle of our approach will be described. In section 3 the steps in detecting skew angles will be described in detail. Section 4 presents a skew correction scheme for handwritten documents. In Section 5 we will present experiment results and Section 6 gives a £nal conclusion.

2. Background

As indicated in [9], without recognizing character and/or understanding the document, human usually determine document orientation by collecting some regularly aligned symbols, mostly text. A symbol line is de£ned as a group of regularly aligned symbols that are adjacent, relatively close to each other and through which a straight line can be drawn. Connected components are chosen as such symbols in [9]. Foreground pixels can also be chosen as such symbols. For example in methods using projection pro£le or Hough transform, pixels are used to determine the skew direction along which a most favorable pro£le or straight lines can be drawn. But in some complicated document images such as the images in Figure 1, obviously not all of the

Figure 1. Complex images

foreground pixels can contribute in making the symbols to determine the document orientations. The previous methods fail on some of these images. For example images (b), (c) and (d) in Figure 1 can easily mislead the projection pro-£le or Hough transform because of the multiple directions of text lines or the noise, the big black border and graphics. In image (a) in Figure 1, there are both too small and too big connected components from the broken text and connected text crossing lines. They make the methods using connected component approaches also hard to £nd the correct directions.

For the purpose of document recognition/understanding, we are mostly interested in determining the orientation of the document texts regardless of they may be buried in other document object such as graphics. Without recognition, to separate the texts from the other graphic elements is a dif£-cult issue. To correctly identify text lines without recognition, we have to combine grouping of the broken segments and separating of the touching or connected characters between lines together.

From our analysis of the complex document images we found some general properties of text lines and other graphics. We noticed that for both handwritten and printed text, the relative distances between characters in same lines are generally smaller than the distances between text lines. Human identi£cation of the text lines utilizes these differences ef£ciently. We can also tolerate touching or connection between text lines by £nding a background path between each pair of text lines. The touching or connections between text lines are usually made by oversized characters or characters with long descendents running through the neighboring lines. Therefore if we could ignore some bridging strokes across lines, we may see the background paths between lines clearly. As for graphics, they are either in relatively irregular shapes as isolated connected components or a group

of connected components occupying a bigger area than any usual text line. Examples are bar-codes, stamps, pictures and pepper noises.

Based on these observations, we decide to use the background runs to build a type of separators. They should separate the different text lines by running through the connecting strokes between lines. They should also be able to group the neighboring characters in same text lines together.

Background runlength has being used in the literature for page segmentations and skew detections[5]. Background regions(white streams) are built from adjacent background runs and those wide white streams are used in estimation of skew angles. For page block segmentation, the assumption that the text lines and blocks are well separated by white background is required. For tolerating noise and run-away black strokes, runlength smearing[6] is applied as an image processing. The favorite runlength such as foreground runs are created by skipping small runs in background color. The expected result from this process is that the most of the foreground text characters are grouped together. The text lines and text blocks are extracted by using a connected component analysis approach. The method again works well for printed documents with mostly text. It will fail on documents with touching or connection between text lines and text blocks.

We may use the runlength smearing approach for building our background runs. "Smearing" is as same as skipping or ignoring some foreground pixels. Hopefully it will break the touching between text lines. But setting up the threshold for skipping the small foreground runs is dif£cult. If we set the value too small, we may not be able to break the crossing strokes connecting the text lines. If we set the value too big, then it may be bigger than the stroke width of most of the characters and end up erasing the text areas.

To solve the problem we designed a new kind of runlength – fuzzy runlength. we *trace* a background run starting from a background pixel along two directions, to its left and right(this is for horizontal runs, otherwise the up and down directions for vertical runs). On the way of the tracing we skip some foreground pixels. When the accumulated number of skipped pixels exceeds a pre-set threshold, we stop the tracing and do the same along the other direction. The total number of the traced positions is the length of the run associating to the position where we start the tracing. Intuitively, the fuzzy runlength at a pixel is how far we can see from standing at the pixel along horizontal(or vertical) direction. Like a human standing in a forest looking for a path out of the forest, the length that he can see along a direction may not be the distance from where he is standing to the £rst tree in front of him. Rather he may be able to "see through" a few trees to get a longer view. But he may not be able to see through too many trees.

In next section we will present the algorithms of con-

structing the fuzzy runlength and estimation of skew angle using a selected set of above foreground components.

3. Skew detection

3.1. Build fuzzy runlength

We assume that the input document images are binary images with foreground color black and background color white. We generate the horizontal fuzzy runlength by scanning each row of the image twice, from left to right and then from right to left.

1. *In_img* ← the input binary image.

2. *Fuzzy_img* ← out put buffer of same size as the input image, initialized to 0 for holding the fuzzy runlengths.

3. Scan each row of the input image from left to right:

    (a) Initialize a FIFO queue **BlkQ** for holding black pixels which can be seen through from the current pixel to its left.

    (b) For the current position j, if the current pixel is black, put the current position j on top of the queue **blkQ**.

        * If the number of black pixel in **blkQ** is bigger than **MaxBlockCnt**, assign the fuzzy run image at the current position the value of j - **blkQ**[*bottom*]. Then pop out the bottom element from blkQ;

        * Else assign the fuzzy run image at the current position the value of the fuzzy run image at previous position + 1.

4. Similarly we scan the row from right to left.

In the above algorithm **MaxBlockCnt** is a threshold value we have to decide before scanning the image for calculating the fuzzy runs. It can be either set as a £xed value for an application running similar set of images or determined at run time dynamically. Since the value of **MaxBlockCnt** is for determining how many black pixels we want to see through in building the fuzzy background runs, it can actually be set as a large value, as far as it is not too large so that we can see through all the black pixels in a sizable word. For example, in each word if we assume there are at most two descendent characters such as g or y, and the average stroke width is 5 pixels. We want the see-through pass 4 strokes then we can set **MaxBlockCnt** to be a little more than 20, say, 25.

3.2. Detection of possible text areas

After above scan we get a buffer *Fuzzy_img* for holding fuzzy runlengths for each pixel. We take the buffer as an image and binarize it to two values. The threshold value for the binarization can be a pre-set value or can be determined dynamically. The fuzzy runlength at each pixel represents the maximal extend of the background along the horizontal direction through the pixel position. Considering the runs insider a text word being shorter than the runs in between text lines, we can choose the threshold value as bigger than the estimated maximal runlength insider the words. In our experiment we empirically determined the threshold based on the estimations of average character size, average distance between characters and average stroke width. These values can all be calculated dynamically too.
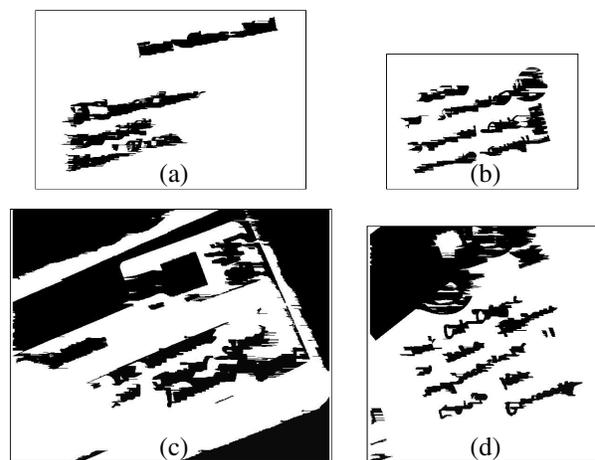

(a)   (b)

(c)   (d)

Figure 2. Binarized fuzzy runlength images

The binarized fuzzy runlength image *Fuzzy_img* in Figure 2 consists of connected components each as a pattern represents either part of a text line(made of one or a few words) or other graphic elements. For a document with text line skew angle between $\pm 45°$ the fuzzy runlengths can push the text line patterns clearly stand out. The skew angle will be calculated from each of these text line patterns.

To identify a connected component as a pattern of a text line, we have to give a clear de£nition of text line pattern. We de£ne a component as a pattern of a text line if it satis-£es the following conditions:

1. The height of the component should be near the estimated text height. This height can not be the height of the bounding box of the component. To calculate the height we evenly divide the component to many small pieces horizontally and calculate the vertical extend from each one of these pieces. The average values of these vertical extends is the estimated height of the component.

2. the length of the component should be long enough, for example it should be at least twice as long as the calculated height of it. For the length we simply use the length of the bounding box.

3. We may also add a condition on its density to £lter out noise. This is optional. The identi£ed text line pattern are shown in Figure 3.
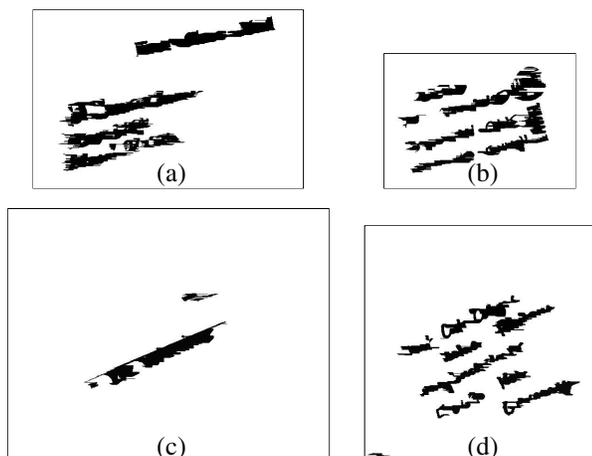

Figure 3. Identi£ed connected components as text line patterns.

3.3. Detection of skew angle of text areas

Each text line pattern as a connected component is a long rather than wide strap of black pixels. It is like a blurred image of a text line or word. Since the descendents and ascendents are striped off, its orientation along the longitude represents the orientation of the underlying text line or word. Therefore we £rst compute the orientation for each of the components.
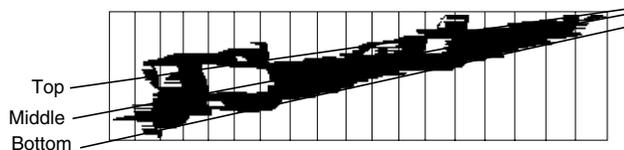

Figure 4. Compute a best £t orientation for a component

For each text line pattern we £rst evenly divide it horizontally into n pieces, see Figure 4. The distance d between the adjacent dividing lines is £xed. When $d = 1$ we simply use all the columns in its bounding box as dividing lines. Choosing a bigger value for d is a matter of ef£ciency. On

each dividing line we take three points, the top most black pixel, the lowest black pixel and the middle point between these two pixels. We name them *top*, *middle* and *bottom* points. We then use minimum squared distance method to get three best £t straight lines for all the top points, all the middle point and all the bottom points. Compare the £ts(the minimal squared distances) of the three lines we choose the best among them. In our example in Figure 4 the bottom line is our best choice. The orientation of the line pattern is de£ned as the orientation of the chosen straight line.

There are several other ideas for computing the orientation. One of them is to get the estimated convex hull of a component £rst, then use the convex hull to compute the orientation.

3.4. Calculation of skew angle

Now that we have the orientation information for each text line patterns in the image. To use them to come up an unique value as orientation of the text lines in the image, we have several choices.

We may simply compute an average value of all the orientations. But we found the orientations of some relatively small pieces are not accurate enough and they are the cause of the errors in the £nal results. Therefore we need to rank all the available information base on not only the orientation of each component, but also other information including the sizes and the shapes of the components. For example the longer components should give more reliable orientation estimations. For our application we £rst compute a weighted average of the orientations of the components. The weights are determined by using the lengths of the components. Then we compute an average again after we weed out the out layers.

One of the side effects we have experienced is that the skew detection approach we propose above can detect some up-side down handwritten documents. From each text line pattern component we actually compute three best £t lines. If majority of the orientations are returned from using the top lines, we can say the document is probably in an upside down orientation. From our experiment this kind of probability is quite reliable for handwritten documents with accurate rate above 70%.

4. Skew correction

We present a very straightforward approach for skew correction. For a detected skew angle we rotate the document image using the following formula

$$
\begin{aligned}
X &= x\cos\theta + y\sin\theta \\
Y &= y\sin\theta - x\cos\theta
\end{aligned}
$$

to transfer a point (x, y) from original image to a new point (X, Y) in the same coordinate system. Here θ is the skew angle between the document orientation and the x-axis along horizontal direction. Of course for transferring the original image to a new de-skewed images, we need to adjust the size of the output buffer and shift the (X, Y) coordinates in a usual way.

The transform is too simple for skew correction of binary images. Due to round-off errors in the computation the output binary images have holes inside the character strokes. These holes make document recognition dif£cult. To solve the problem we designed a look-up table each its entry is a £lling pattern for a 3×3 window. Place the center of the window on a *white* pixel in the transformed image. According to the corresponding pattern we decide to change the white pixel to black color. The £lling patterns are designed following a simple rule: £ll holes without creating new touching or connections.

5. Experiment

To test our method, we used a set of USPS parcel images. We manually selected 813 of 1864 binary images with clear skews. These images are cropped and binarized from the original parcel address images automatically. The original scan resolution was 300dpi. For detecting skew angle, we don't need such high resolution. We down sampled them to 1/4 of the size before running our programs. The skew corrected images after running the implementation of our method are viewed to compare with the original images. The correct rate of skew detection and correction is 92.1%. There is no image failed with wrong rotating skew direction. Apart from some images with less than enough skew angle, the rest of the failure cases are images failed from being using £xed parameters in our simple implementation. For example the threshold value for "running through" black pixels is too small for images with extremely large size characters with thing strokes. The dynamically determined such threshold value based on evaluations of character size and stroke width would help for the problem.

6. Conclusion

In this paper we present a novel method for complex document skew angle detection. The method uses a new concept of fuzzy runlength which imitates an extended running path through a pixel of a document. The fuzzy runlength can be use as an image processing tool to partition a complex document to separate the content of the document to texts in terms of text words or text lines, and to other graphic areas. Classi£cation of these areas can lead us to getting information for document segmentations and skew

Figure 5. De-skewed images

angle detections. Our simple experiment also showed the success of the method. Further research along this direction will be using the method on document segmentation and other content locations.

References

[1] A. Amin, S. Fischer, T. Parkinson, and R. Shiu. Fast algorithm for skew detection. *IS&T/SPIE Symposium on Electronic Imaging, San Jose, USA*, 1996.

[2] G. Ciardiello, G. Scafuro, M. Degrandi, M. Spada, and M. P.Roccotelli. An experimental system for of£ce document handling and text recognition. *Proc 9th Int. Conf. on Pattern Recognition*, pages 739–743, 1988.

[3] G. T. D.S. Le and H. Wechsler. Automated page orientation and skew angle detection for binary document image. *Pattern Recognition*, 27:1325–1344, 1994.

[4] S. Hinds, J. Fisher, and D. D'Amato. Document skew detection method using run-length encoding and the hough transform. *Proceeding of 10th International Conference on Pattern Recognition*, pages 464–468, 1990.

[5] T. Pavlidis and J. Zhou. Page segmentation by white streams. *Proc. 1st Int. Conf. Document Analysis and Recognition (ICDAR),Int. Assoc. Pattern Recognition*, pages 945–953, 1991.

[6] S. Srihari and V.Govindaraju. Analysis of textual images using the hough transform. *Machine Vision and Applications*, 2:141–153, 1989.

[7] A. H. W. Chin and A. Jennings. Skew detection in handwritten scripts. *Proc. IEEE on Speech and Image Technologies for Computing and Telecommunications*, pages 319–322, 1997.

[8] B. Yu and A. Jain. A generic system for form dropout. *IEEE trans. On Pattern Analysis and Machine Intelligence*, 18(11):1127–1134, 1996.

[9] B. Yu and A. Jain. A robust and fast skew detection algorithm for generic documents. *Pattern Recognition*, 9:1599–1629, 1996.