Slant Correction using Histograms

Frank de Zeeuw

Bachelor's Thesis in Artificial Intelligence
Supervised by Axel Brink & Tijn van der Zant

July 12, 2006

Abstract

Slant is one of the characteristics that make handwriting harder to process automatically than printed text. For this reason slant correction is a standard step in systems for processing written text. In this article I will present an algorithm for automatically correcting slant. It works by first segmenting the text into words using horizontal and vertical projection histograms. Each word is then sheared by the angle that maximizes the height of peaks in the vertical projection histogram of the word. I will argue that the presented method is simple and fast, yet produces results similar to more advanced approaches. The algorithm has been incorporated into a system that uses an evolutionary algorithm to preprocess images of handwritten text for writer identification.

1 Introduction

Any piece of handwriting is written at a certain *slant*, which is best explained by a picture like Figure 1. To be precise, slant is the angle that near-vertical strokes in writing make with the exact vertical direction, i.e. the perpendicular to the baseline of the word. It is not to be confused with the *slope* of the text, which is the angle that the baseline of a word or sentence makes with the horizontal direction. Slope can be removed by rotating the entire text, while slant is corrected by *shearing* each word. This means shifting every line of pixels sideways by an amount depending on its distance from the

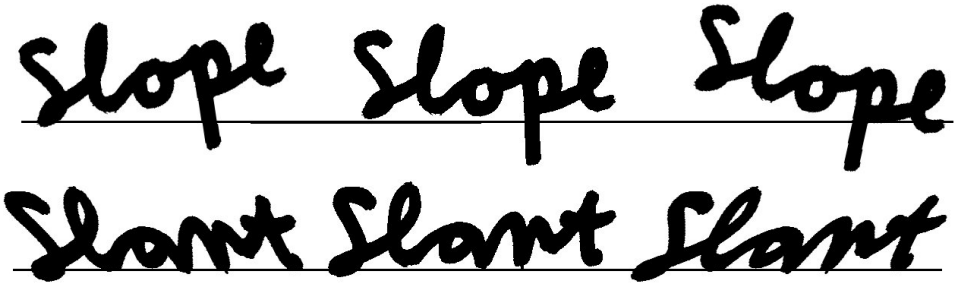

Figure 1: Examples illustrating slope and slant. Above: negative, zero and positive slope. Below: negative, zero and positive slant.

Figure 2: The shear operation.

baseline. Figure 2 illustrates this.
Slant can vary greatly between writers, as well as within a given piece of writing, or even within an individual word. In automatic processing of handwritten text, it is often useful to reduce such variation by correcting slant. For instance, in character recognition, however it is approached, removing slant will reduce variation within classes of characters, making those classes easier to recognize. It will also make character segmentation easier, since characters that are straight up have more distinct space between them than characters that are at a slant. Another application is writer identification, where slant correction can improve performance by reducing within-writer variation. Specifically, it can help to identify a writer that has tried to disguise his writing by writing at an unnatural slant, which occurs in forensic science applications. On the other hand, between-writer variation in slant could be helpful for writer identification, and slant removal might have a negative effect on this. However, a slant correction algorithm can also be used to return the slant profile of the text separately, while at the same time preparing the image for better extraction of other characteristics. For these reasons, slant correction has been a standard step in automatic handwriting processing, usually following foreground/background separation, slope correction and segmentation of the text into words.
The algorithm below and its implementation were developed as part of a preprocessing system for automatic writer identification. The system consists of separate functions for a variety of preprocessing steps, such as background correction, text region detection, slope removal and slant correction. Parameters and the order in which these functions are to be applied is then determined by a genetic algorithm, using feedback from a user. This preprocessing system will be integrated with an existing method for writer identification. Intended applications are automatic searching in a 19th-century Dutch handwriting database and forensic writer identification.
In the next section I will explain the different approaches to slant correction that have been presented in the literature, and which of those the algorithm in this paper belongs to. Section 3 will describe the algorithm. Section 4 will show and discuss the result of the implementation on some examples.

2 Related work

There are several approaches to slant correction in the literature, which can be roughly divided into three groups.
The first, also historically the first, tries to determine the slant angle by detecting near-vertical strokes and taking the average angle of those as the shear angle. This approach was begun by the authors of [1] and several others have followed their lead, using different ways to select the strokes. For instance, in the method of [1] two horizontal lines are drawn above and below the core region and all intersection points of the writing with these lines are determined. Any two points that are close to being vertically aligned are then assumed to be part of the same near-vertical stroke. From each such stroke a local slant estimate is taken and a global estimate is obtained by averaging the local estimates. Compared to the other approaches this method is quite fast, but on

the downside, it relies heavily on heuristics, hence is not very robust.

The second approach evaluates a measure function of the image on a range of shear angles and selects the angle with highest value of the measure. The measure is most often computed from the vertical histogram, based on the idea that deslanted writing will have a more intense histogram, i.e. with higher and more pronounced peaks than slanted writing. In [3], for instance, Kavallieratou et al. treat the histogram as a time signal, calculate its Wigner-Ville distribution (known to have several nice properties and often used in pattern recognition), and take the intensity of that distribution as the measure. Another interesting example is [2], where entropy is used as a measure of slantedness. These approaches use less heuristics and seem to give better and more robust results, although it is hard to evaluate or compare them. They do come with a computational price tag, because the shear transform, the histogram and the measure function have to be computed for a lot of angles.

The third approach distinguishes itself from the first two by correcting the slant *nonuniformly*. The techniques above shear a word (or bigger units) uniformly, i.e. by a single angle, hence can never fully cope with variant-slanted words, which are quite common in ordinary handwriting. In [5] and [4] a method is described to slant words nonuniformly using dynamic programming techniques. To apply different shear angles at different points within a word, one has to split the word up into intervals and shear each of those individually. To determine what intervals to take, and by what angle to shear over each interval, a criterion is optimized that evaluates the sequences of intervals and angles simultaneously. Such a method has a lot of potential, since it can cope with variant-slanted words. The results are indeed promising, although there are more robustness issues, as the algorithm has greater freedom to make errors within a word. Also the theoretical background and mathematical techniques are somewhat more demanding.

I have chosen to use the second, histogram-based, approach for my project, but aiming for a simpler and faster measure function, for the following reasons. First of all, the use of histograms is convenient because they can also be used for line and word segmentation, which are essential for slant correction but were not yet implemented in the system my deslant module was to be part of. They could even have been used for slope removal, but a module for that was already present. Nevertheless it seemed convenient to approach these problems with similar techniques.

The second reason for taking the second approach is that I thought it might be worthwhile trying to improve it by using a simpler measure function. I feel that in [3], Kavallieratou et al. (who were my main source for this approach) do not provide much justification for using the Wigner-Ville distribution, other than that it is popular in other applications. In fact, I'm not sure if using a distribution is necessary at all. Although higher, more distinct peaks, which characterize deslanted words, are more likely in a distribution of high intensity, they are not quite the same thing. It is not obvious that characters without vertical strokes also provide a more intense distribution of the histogram when deslanted.

Because of these doubts I propose to base the measure function solely on the properties of the highest peaks. An added advantage is that working with those peaks is just simple string manipulation, and is therefore likely to be computationally more efficient. We will return to these considerations in the section 4.

3 The Algorithm

We will describe how a given text image is split up into lines, how the lines are divided into words, and how the slant of those words is corrected.

3.1 Line segmentation

As segmentation is not implemented separately in the system this algorithm will be part of, I implemented it as part of the slant correction process. This is especially justified since I chose to do both segmentation and slant correction by using histogram techniques.

For line segmentation, the image is first converted to a grayscale image, which is then inverted, to make dark pixels have high values and light pixels low values. Then the horizontal projection histogram is computed, as in Figure 3, returning a list of numbers. An element of the list is the sum of all pixel values in the corresponding line of pixels in the image. In this list, a line of writing should occur as a peak, while inter-line spacing should give a valley. Then the boundaries between lines can be found by the following simple list manipulations.

First the peaks in the list are determined as intervals with high values, i.e. with values above a certain percentage, say 10%, of the maximum value in the list. This percentage was determined by some testing on examples, but the precise value actually does not matter that much, because of the following refinement procedure. We compute the average of the list of peaks, which provides a temporary estimate of the height of a line. Based on this estimate, peaks that are too narrow to be a line are thrown out, and peaks that are too wide are refined into smaller peaks, using a higher threshold percentage. This process is repeated until no more changes occur. Here the thresholds for 'narrow' and 'wide' are taken to be half resp. twice the average height of a line. Finally, the lines are spread out so that they all meet and cover the entire height of the image, and extra peaks are introduced to cover any open space at the top or bottom of the image.

In this way we create a partition of the image, which is used to split up the original image (from before the grayscale was taken) into lines by cropping the strip corresponding to each peak. The resulting lines are then returned in a list of images, so that each can be manipulated separately (in particular, segmented into words and deslanted). Afterwards the pieces are glued back together by a separate function, into an image of the same size as the original image.

Figure 3 illustrates the procedure. The black lines are just visualization; in fact they are the lines where the cuts between lines are made.

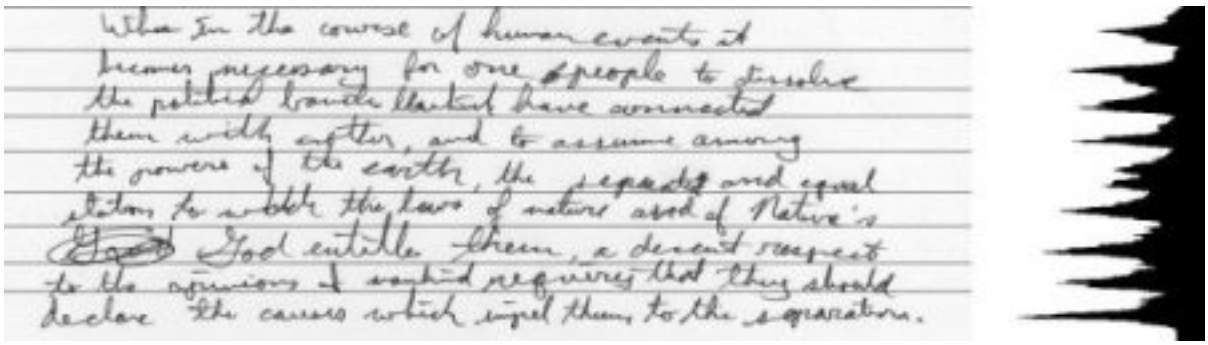

Figure 3: Line segmentation.

3.2 Word segmentation

Next we segmenting each of the lines we obtain into words in a very similar way (although not similar enough to use the same code). The main differences are that the vertical histogram is used, and that while lines are all of similar height, words have strongly varying length. On the other hand, the length of inter-word spaces are more uniform. Therefore, after taking the vertical histogram as in Figure 4, we extract valleys as intervals with relatively small histogram values. We then remove the valleys that are too narrow, say less than a fourth of the height of the line. We do not need to refine wide valleys, since large spaces can occur. Then we take the complements of the valleys, resulting in a list of intervals, each of which should consist of one or more words. These intervals we spread out to cover the whole line again, and a similar cropping and glueing procedure as for line segmentation is used.
Figure 4 visualizes the procedure.

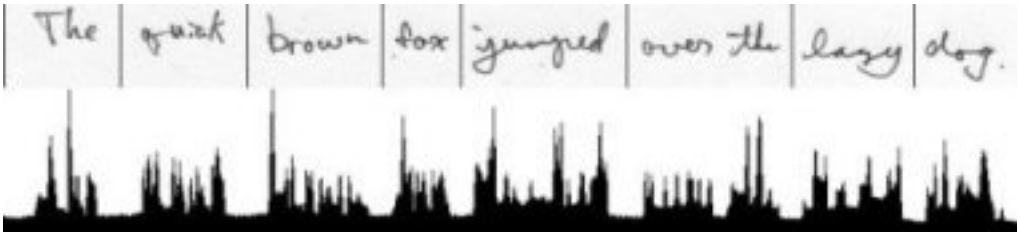

Figure 4: Word segmentation.

Figure 5: A fully segmented text.

3.3 Deslant

Our slant correction method uses the vertical projection histogram. The idea is that the histogram of a word that is written straight up will have larger and more distinct peaks. Therefore, we could look at the histogram of the word at different shear angles and take the one with the highest peaks.
We do this for angles between -45 and 45 degrees, which is the most common range of slant angles in normal writing. For each angle, we compute the vertical histogram, as is depicted for a few angles in Figure 6, and apply a measure function that measures the height of the peaks. The angle with highest measure will win and will be used as shear angle. To save time, we first go through the range with big steps, of say 5 degrees. For each of those angles we determine the ones with highest measure, and we search around each of those with smaller steps. This gives something of a tradeoff between speed and robustness, as bigger steps risk missing the best angle. This did not seem to result in any problems so we did not investigate it any further.
The measure function looks for the peaks in the histogram (with the same function as used in the segmentation), selects the highest n peaks, and returns the average height of those. If less than n peaks occur, it returns the average of those available.
The number n is passed to the algorithm as a parameter. From experimenting a little we found out that for n less than 3, the results varied with n, and were not always good. For higher values the results are better, and usually the same for different n. Therefore we set the default value to 5.

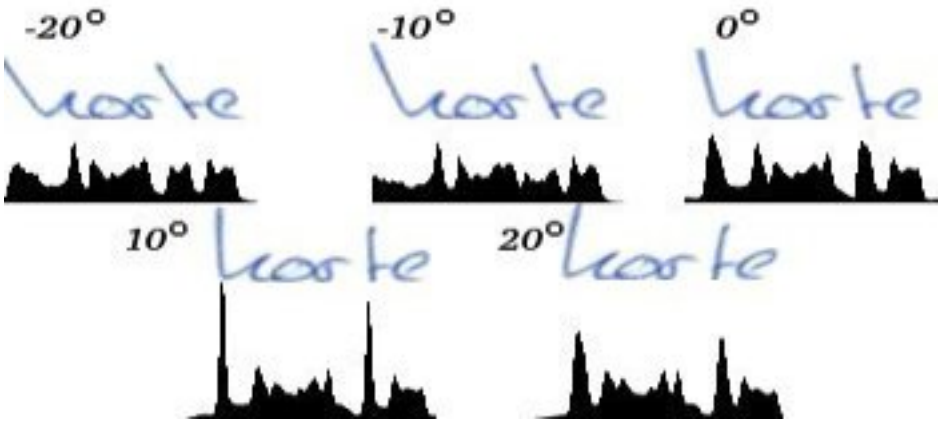

Figure 6: Several steps in the deslant algorithm.

Figure 7: How to deslant a variant-slanted word?

4 Results

First of all, we should emphasize that it is hard to evaluate the results of a slant correction algorithm, because the slantedness of a word is not defined exactly. To see this, consider Figure 7. The problem is that slant can vary *within* a word, so that there is no objective standard of deslantedness (at least for uniform slant correction; for non-uniform approaches, see section 2).

Therefore we cannot evaluate the algorithm by counting how often it is right and how often wrong. One could let a person judge the quality of the deslanted words in a text, but that would be very time-consuming. Besides, the purpose of the algorithm is not to make the writing more readable for humans; we have no problem reading slanted text, and the shear transformation will often make the writing look *less* natural.

Instead, the purpose of slant correction is to make text more suitable for digital processing by a certain system. Therefore, the best way to evaluate an algorithm would be to measure the change in performance of the system it is incorporated into. For instance, [3] report a 9% improvement and a reduction of training time of their character recognition system when their slant correction algorithm was added. Of course, it remains hard to compare different algorithms when they have been designed for and evaluated in different systems.

In our case, the system that the algorithm is part of is still under construction at the time of writing, so that it was not yet possible to evaluate the algorithm in this way. Consequently, in this article we will have to limit ourselves to considering the algorithm's performance on some examples, observing where it works and where it fails, and trying to explain why.

Figure 8 shows the result of this deslant algorithm on a printed text that has been artificially slanted. Clearly, the result is quite satisfying. Figures 9 and 10 show the algorithm working nicely on clear handwritten text. Figure 11 is a bit harder, and illustrates some of the issues involved.

First of all, in the center of Figure 11 we see some isolated artifacts that should not be there. This occurs because the line segmentation cuts off descenders, so that after shearing those will be separated from the characters they belong to. This could be prevented by using a better segmentation algorithm, namely one that can segment the lines of text using more flexible curves than straight lines. That would make it possible to leave descenders intact. In fact, this would improve the performance of this slant correction algorithm, because the descenders will give clear peaks in the histogram. However, we decided to leave this for the time being, as it is not essential to our project.

Also, just below and to the right of the center of Figure 11 we see something going wrong: the word 'dus' is slanted much too far to the right. This could be caused by the stains on the background, or perhaps by varying thickness of the ink. It is not a big problem, since in the preprocessing system that the algorithm is part of, such stains should be removed, and the image should have been thresholded so that all writing is equally thick.

Furthermore, a potential solution to misslanted words like this would be to keep track of the surrounding slant angles, so that if an estimated slant angle varies strongly from the neighboring ones, as happens here with 'dus', the algorithm could adjust it. It could do this by looking for a well-scoring angle that more

resembles the surrounding angles, or perhaps by just leaving it as in the original; at least then it has not been made worse.
In conclusion, our algorithm works reasonably well, although as said we cannot really compare its performance to other approaches. However, it is simple and fast: compared to [3], our measure function is easier to understand and implement, and because it only uses elementary list operations, it performs quite fast.

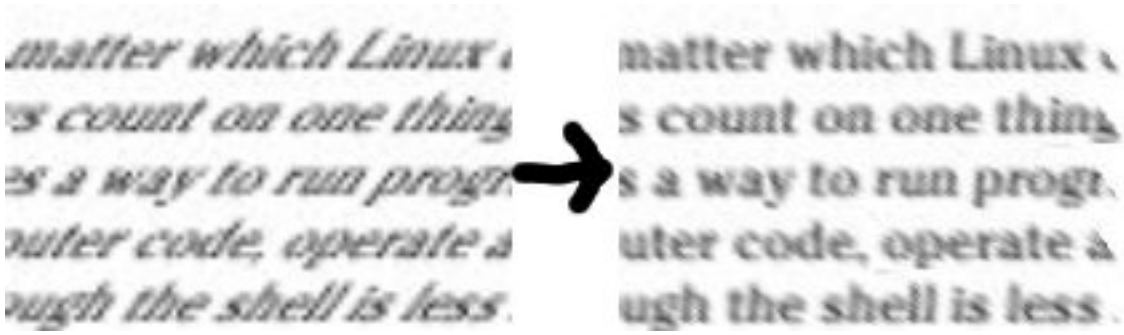

Figure 8: Deslanting an artificially slanted text.

Figure 9: A part of a text before and after deslanting.

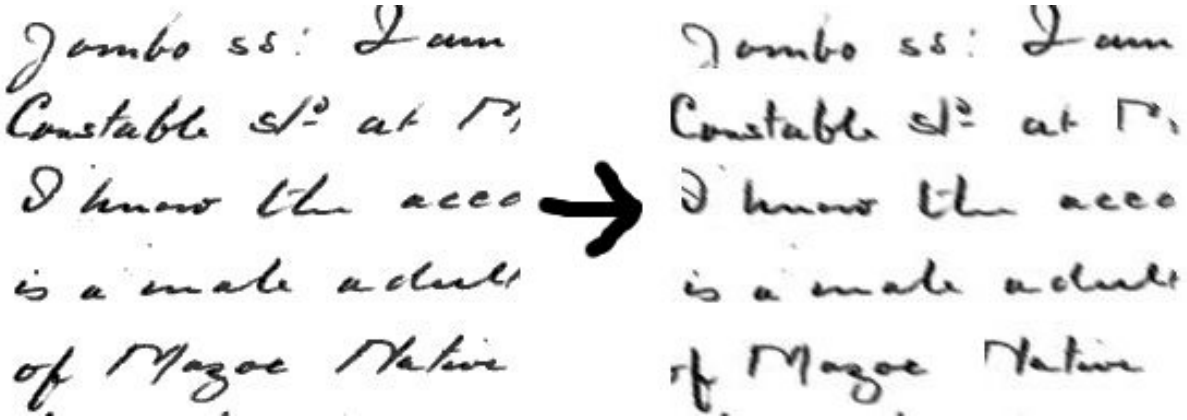

Figure 10: Another text fragment before and after deslanting.

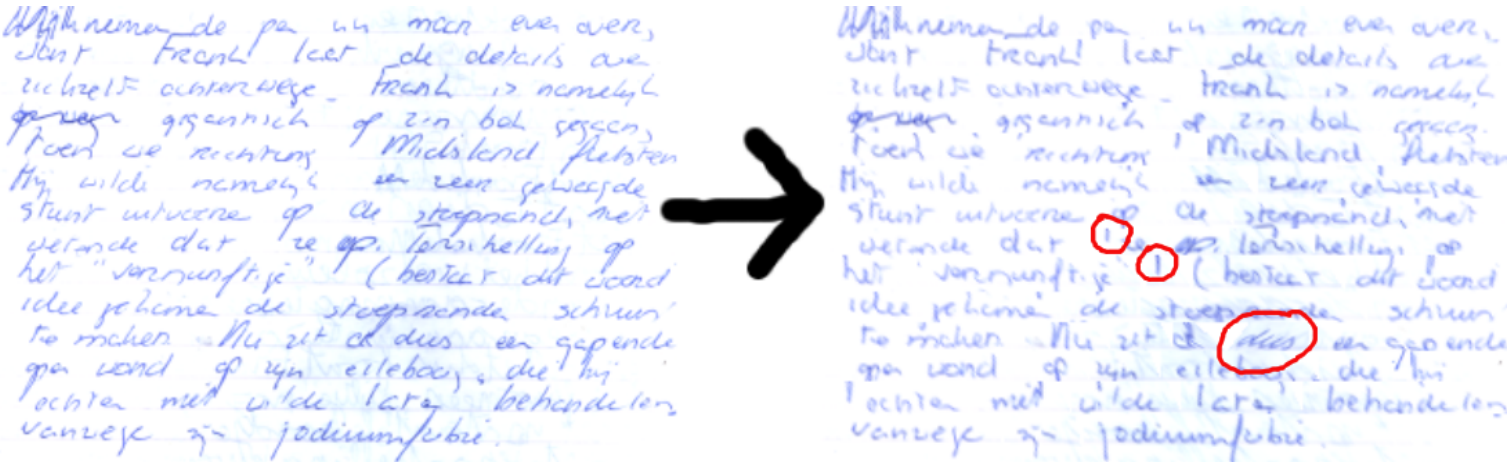

Figure 11: Some issues with deslanting.

References

[1] R.M. Bozinovic and S.N. Srihari, Off-line Cursive Script Word Recognition, *IEEE Trans. on PAMI*, vol. 11, no. 1, pp. 68–83, 1989.

[2] M. Côté, E. Lecolinet, M. Cheriet, and C. Suen, Automatic reading of cursive scripts using a reading model and perceptual concepts, *International Journal on Document Analysis and Recognition*, vol. 1, no. 1, pp. 3-17, 1998.

[3] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis. Slant estimation algorithm for OCR system. *Pattern Recognition*, vol. 34, no. 12, pp. 2515–2522, 2001.

[4] E. Taira, S. Uchida, and Hiroaki Sakoe, Nonuniform slant correction for handwritten word recognition, *IEICE Transactions on Information & Systems*, vol. E87-D, no. 5, pp.1247–1253, 2004.

[5] S. Uchida, E. Taira, and H. Sakoe, Nonuniform slant correction using dynamic programming, *Proceedings of 6th International Conference on Document Analysis and Recognition (ICDAR 2001, Seattle, USA)*, vol. 1, pp. 434–438, 2001.