Using Local Slant Correction to Normalize Handwritten Text Samples

Adrian Kuhn
University of Bern
akuhn@iam.unibe.ch

December 2005, University of Bern

Abstract. In this paper we propose to use variable slant correction instead of an global correction, since the slant of handwritten text is not constant over a line of text. We present an algorithm that computes local slant, based on generalized projection and dynamic programming, and introduce a technique called *slant map propagation.*
We apply it on a case study and report the results: local slant correction improves the word recognition rate from 37.3% to 42.24% and the word level accuracy from -14.99% to -3.18%, compared to global slant correction.

1 Introduction

Preprocessing handwritten text samples with slant correction is crucial to the recognition process. Since the HMM based recognition system [4] divides the text sample into thin slices, it is important that vertical strokes are as perpendicular as possible. Otherwise vertical strokes do not fit into the slices and appear to the recognition system as slanted strokes that harm the modeling of the characters. For example the letter P might once start like a slash, once like a backslash, where indeed it should start with a perpendicular stroke.

Handwritten text is not uniform and slant may change over the course of a line. Therefore a global and thus uniform slant correction can not guarantee that all vertical strokes are as perpendicular as possible. In fact we need a variable slant correction to make sure each vertical stroke is corrected exactly. But such a variable slant correction must not be too sensitive, since there are a great many false positives.

most false positives are produced by short strokes and can be avoided by a simple weighting scheme: short strokes are ignored and a larger weight is assigned to long strokes. But still, there remain two classes of false positives: First long but slanted strokes as found in *e.g.* the letters A, M, N or X, and second the connectors in cursive handwriting.

Thus our requirements are: a local slant correction that smoothly conforms to the variation of slant in handwritten text, corrects each vertical stroke while avoiding false positives. As a further requirement, we demand that the algorithm must not require any parametrization.

In this paper we combine two previously published approaches. Nicchiotti and Scagliola proposed to use *generalized projection* for normalization of handwriting [2], while Uchida *et al.* used *dynamic programming* for the same task [3].

2 Local Slant Correction

In short the algorithm works as follows: Compute the *generaliyed projection* [2] for each column and each possible slant angle, which results in a *slant map*. Then modify the resulting *slant map* to avoid false positives. And finally apply a *dynamic programming* algorithm to get an optimal but smooth solution.

This section explains each step in detail and motivates why it is applied and how it affects the result. While Figure 1 to 2 illustrate the algorithm on as series of images.

Input is a black-white image containing a line of handwritten text. The background is white, the handwritten text is black. In our case study these images have a height of 128 pixel and a length of about 2000 pixels, however the algorithm is independent of these numbers and works with input of any size.

Output is a vector $\mathbf{p} = \{p_1, p_2, .., p_{\text{width}}\}$ with the optimal slant angles for each column of the input image. A slant line $sl(x_0, p_0)$ is a discrete line ranging from position $(x_0 + p, 1)$ to position $(x_0 - p_0, \text{height})$ on the input image, with the angle p_0 given as pixel offset.

The solution has to meet the requirement that subsequent angles may not differ by more than ± 1, thus $|p_j - p_{j+1}| \leq 1$ holds true for all elements of $\mathbf{p}$. This requirement guarantees a smooth solution, but there is another reason it has to be met – since otherwise subsequent slant lines might intersect and cross each other and thus fall behind their predecessors leading to an undesired repetition of input pixels on the output image.

2.1 Setting up the Slant Map

A *slant map* is a matrix $\mathbf{A}$ containing the generalized project values of all possible slant lines. Its row indices $i \in P$ correspond to all possible slant

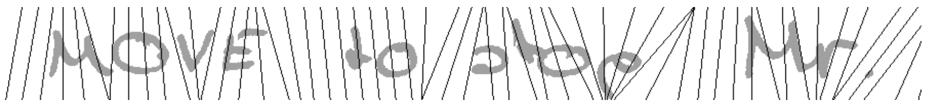

Fig. 1. The input is a line of handwritten text as black-white image. The vertical lines show the output of the algorithm, *i.e.* the local slant detected by the algorithm.

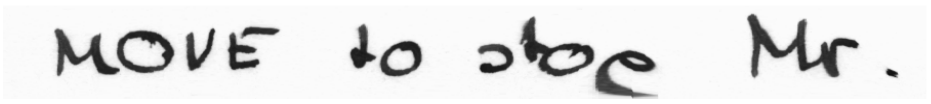

Fig. 2. The same line of handwritten text but with corrected slant, *i.e.* the local slant correction has been applied to the input image.

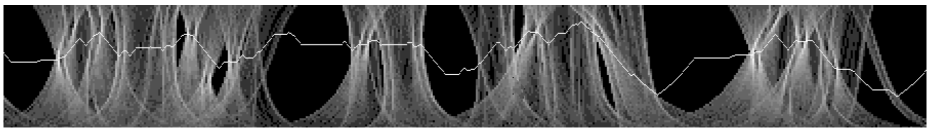

Fig. 3. A raw *slant map* of the text line: the columns correspond to the columns of the input image, the rows ranges from slant angle $-45°$ to $+60°$. The grayscales show the *generalized projection* at the corresponding column and angle – more see Section 2.1.

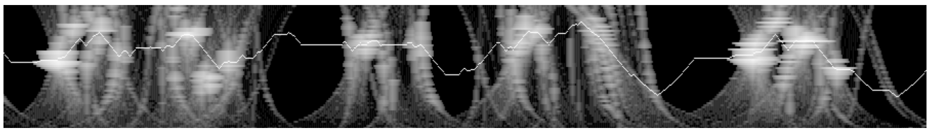

Fig. 4. The modified *slant map*, all hotspots have been propagated along the x-axis as explained in Section 2.2. The white line shows the solution found by the *dynamic programming* algorithm.

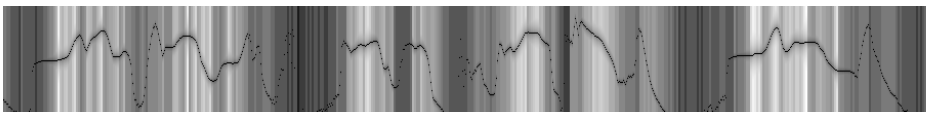

Fig. 5. A visualization of the cost-function of the *dynamic programming* algorithm: the axes are the same as on the *slant map*, the grayscales show the penalty of the cost function – more see Section 2.3

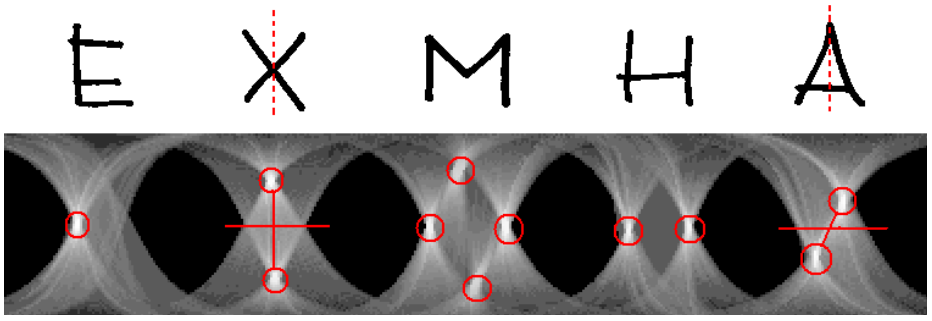

Fig. 6. Illustrates different kind of hotspots: **E** – one hotspot due to the long vertical stroke; **X** – two false hotspots and halfway between these two points the *ad-hoc* hotspot with the correct slant; **M** – in total four hotspots, two of them false positives; **H** - two hotspots; **A** – again as with the X, two false positives and halfway between them the *ad-hoc* hotspot with the correct slant.

angles and the columns indices $j \in X$ to the columns of the input image. Each entry $a_{i,j}$ has the value of the *generalized projection* along the slant line $sl(i,j)$:

$$a_{i,j} = gp(sl(i,j))$$

The *generalized projection* along slant line sl is defined as the sum of the squared lengths of all subsequent runs of black pixels:

$$gp(sl) := \sum_{y/inY} b_y$$

$$b_y := \begin{cases} 0, & \text{if white pixel} \\ 1, & \text{if first black pixel} \\ b_{y-1} + 2, & \text{if subsequent black pixel} \end{cases}$$

Thus subsequent runs of black pixels weight more than the same number of single black pixels. The longer a run the higher its weight. The weight of n subsequent black pixels is n^2. With the *generalized projection* we ignore short stokes and add weight to long strokes.

Note: We use a slightly different *generalized projection* than [2], by our definition the weight of n subsequent black pixels is n^2, instead of $\frac{1}{2}n(n+1)$. Which is basically the same – apart from a factor of $\times 2$ – but allows for a much easier handling of formulas.

2.2 Modifying the Slant Map

Once we obtained the *slant map*, we modify its content to make sure the *dynamic programming* algorithm will find each vertical stroke and leave out the false positives.

A hotspot on the *slant map* is a local maximum in the matrix $\mathbf{A}$, these maxima appear where the *generalized projection* algorithm detects long strokes in the input image. Real hotspots are *e.g.* the vertical strokes of the letters E, H, M or P, while *e.g.* the oblique strokes of the letters A, V, N or X are false positives.

Furthermore what we call *ad-hoc* hotspot marks the bisecting line between a pair of oblique strokes, see Figure 6. On the *slant map* this is the the position halfway between the hotspots of both oblique strokes. Even though there is no hotspot at that position on the *slant map*, the *dynamic programming* algorithm has to cross this position, since the bisecting line between *e.g.* the oblique strokes of A, X or V is a strong indicator of the correct slant.

Propagating Hotspots On a raw *slant map* the hotspots show up as *butterfly* patterns, but this is suboptimal for the *dynamic programming* algorithm, there is too much distracting noise around the hotspot which leads to erratic solutions. We would like it to cross the hotspot as "horizontal" as possible to ensure a smooth solution. Figure 7 shows a hotspot before and after this propagation.

Thus we propagate the hotspots along the x-axis. This is done by "reversing" the *generalized projection* to the left and right:

$$a'_{i,j} := \max_{l \in X} \left\{ a_{i,l} - |j - l|^2 \right\}$$

In that way, the hotspot of a stroke with length n is propagated n pixels to the right and n pixels to the left. For example a stroke of length 4 has a hotspot of value 16 at position j. This value is propagated into both direction such that at position $j \pm 1$ it is $16 - 1^2 = 15$, at $j \pm 2$ it is $16 - 2^2 = 12$, at $j \pm 3$ it is $16 - 3^2 = 7$, and at $j \pm 4$ it is $16 - 4^2 = 0$. If two propagated hotspots overlap, the larger value is used.

A First Estimation In this step we compute a vector $\mathbf{p}'$ with a first estimation of optimal slant angles for each column in the *slant map*. Be aware that these optima are based only on the current column and do not take into account their context, thus they are a sub-optimal solution of

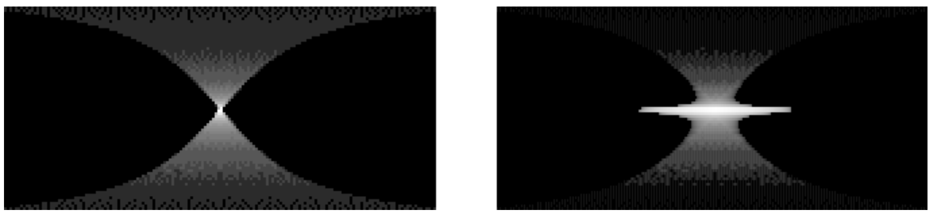

Fig. 7. Illustrates the *slant map propagation* technique: On the left the butterfly shape of a raw hotspot; on the right the propagated hotspot.

problem that do not meet our requirement of smoothness. However this vector is the input of the *dynamic programming* algorithm in the next step.

The sub-optimal slant angle p'_j is defined as the weighted mean of column j, with the row indices i as data set and the entries $a'_{i,j}$ as weights:

$$p'_j := \frac{\sum i \times a'_{i,l}}{\sum a'_{i,l}}$$

With this step, we solve the problem of the *ad-hoc* hotspots. The weighted mean of two false positives is halfway between the two propagated hotspots, and thus exactly at the location of the desired *ad-hoc* hotspot.

2.3 Applying Dynamic Programming

In this step we apply *dynamic programming* to find the optimal solution $\mathbf{p}$ based on the sub-optimal solution $\mathbf{p}'$. The *dynamic programming* algorithm is used to find the cheapest path in the *slant map* fulfilling the condition that subsequent angles may not differ by more than ± 1, thus $|p_j - p_{j+1}| \leq 1$ holds true for all i. The cost function $f(i,j)$ is:

$$f(i,j) := \min \left\{ |p'_j - i|^2,\ \max_{k \in P} a'_{k,j} \right\}$$

The first parameter of the minimum is obvious: the farer away from the estimation p'_j the more expensive the path. The second parameter is the squared length of the longest stroke found in column j of $\mathbf{A}'$, that is if an estimation p'_j is missed by more than the length of its corresponding stroke the cost does not increase any more. Thus missing a short strokes has less impact on the solution than missing a long one, which is exactly the behavior we aim for.

3 Validation

We applied the local slant correction on a random subset of writer independent task [4] from the the IAM database [1] using single gaussian recognition, and compared the results to an ordinary global slant correction. First a brief description of global slant correction, then the results.

3.1 Global Slant Correction

Global slant correction differs from local correction in that it returns only a single over-all slant angle p and not a vector $\mathbf{p}$ with multiple slant angles.

$$p := \frac{\sum i \cdot r_i}{\sum r_i} \qquad r_i := \sum_{j \in J} a_{i,j}$$

The algorithm starts with the same *slant map* as the local correction, but then computes the weighted average of the row sums and returns this as the optimal global slant correction angle.

3.2 Results

Both word recognition rate and word level accuracy improved significantly, as shown in the table below.

	global	local
word recognition rate	37.3%	42.24%
word level accuracy	-14.99%	-3.18%

The results are very promising and show that our local slant correction clearly outperforms global correction.

References

1. U.-V. Marti and H. Bunke. A full english sentence database for off-line handwriting recognition. In *ICDAR '99: Proceedings of the Fifth International Conference on Document Analysis and Recognition*, page 705, Washington, DC, USA, 1999. IEEE Computer Society.
2. Gianluca Nicchiotti and Carlo Scagliola. Generalized projections: A tool for cursive handwriting normalization. In *ICDAR*, pages 729–732, 1999.
3. Seiichi Uchida, Eiji Taira, and Hiroaki Sakoe. Nonuniform slant correction using dynamic programming. In *ICDAR*, pages 434–438. IEEE Computer Society, 2001.
4. M. Zimmermann. *Offline Handwriting Recognition and Grammar based Syntax Analysis.* PhD thesis, University of Bern, Bern, Switzerland, 2003.