

On-Line Handwritten Text Line Detection Using Dynamic Programming

Marcus Liwicki, Emanuel Indermühle, Horst Bunke
Institute of Computer Science and Applied Mathematics, University of Bern
Neubrückstrasse 10, CH-3012 Bern, Switzerland
{liwicki,bunke}@iam.unibe.ch

Abstract

In this paper we propose a novel approach to the detection of on-line handwritten text lines based on dynamic programming. We try to find the paths with the minimum cost between two consecutive text lines. Most steps of the proposed algorithm are based on off-line information. Hence the method can also be applied to off-line documents after a few minor changes. In our experiments we show that this dynamic programming based approach is better than a common on-line segmentation procedure.

1. Introduction

Although the problem of handwriting recognition has been considered for more than 30 years [3, 10, 11], there are still many open issues, especially in unconstrained handwritten sentences recognition. Text line detection is an essential preprocessing step in nearly all recognition systems for general handwritten text. In on-line systems, text line detection is usually handled by simple heuristics, i.e. if the pen-movement to the left and to the bottom exceeds a certain threshold it is assumed that a new text line starts [7, 10]. However, these simple methods fail if the user writes sequentially on different lines. This happens when a missing letter has been inserted later or when notes have been taken during a meeting. Therefore a superior method that can handle those cases has to be applied.

For the off-line case rather simple methods can be applied if the gap between consecutive neighboring text lines is large enough [9]. Other approaches for off-line text line detection are based on connected components [8] or on projection [12]. In [4] an approach based on finding a segmentation path has been introduced, but this method has many problems if the text lines are close to each other. Recently a new general algorithm for detecting text lines has been introduced in [5].

As on-line data can be easily converted into the off-line format, any off-line method can be applied to on-line data.

However, in order to segment on-line data, one can – and should – take additional benefit from the on-line information. The approach proposed in this paper is based on finding an optimal path between two consecutive text lines. The optimal path between two lines is found using dynamic programming. The input of our approach consists of on-line data, but it additionally takes off-line information into account. The motivation that most of the steps are performed in off-line data is that the spatial information is not dependent on the order of the strokes, and that humans also use the off-line information for this task.

In our experiments on 100 documents of the IAM-OnDB [6]¹ we could outperform an approach proposed previously. Without on-line postprocessing we achieved a stroke classification rate of 99.79 %, which could be increased to 99.94 % by using the on-line information for re-assigning small strokes to other text lines. Only on two documents small mistakes occurred.

The rest of the paper is organized as follows. Section 2 gives an overview of the text line detection system. The dynamic programming procedure is introduced in Section 3 in greater detail and the cost functions to find an optimal path are described in Section 4. Section 5 outlines some postprocessing steps. Experiments and results are presented in Section 6, and Section 7 draws some conclusions and gives an outlook to future work.

2. System Overview

Figure 1 gives an overview of the text line detection system. The input is a handwritten text in on-line format. First some preprocessing steps are applied. The purpose is to filter out some noise. This filtering is needed because the recorded on-line data usually contain noisy points and gaps within strokes, which are caused by loss of data.

Another preprocessing step is the initial estimation of the starting point of each text line. Between two consecutive starting points the dynamic programming paths will start.

¹<http://www.iam.unibe.ch/fki/iamondb/>

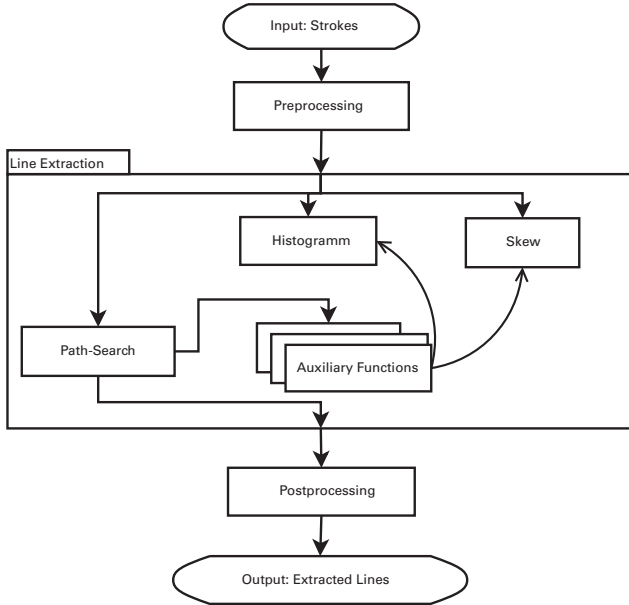


Figure 1. System Overview

The starting points of the text lines are estimated by calculating the vertical histogram of the foreground pixels in the left part of the document. To overcome the skipping of lines, we also set a new starting point if the gap size between two consecutive starting points exceeds the median gap size. It is usually no problem if too many starting points have been found, because this often results in empty spaces between two paths, which can be easily detected and deleted.

As the last preprocessing step, the skew of the document is estimated. We experimented with two methods for skew estimation, i.e. linear regression and a new approach. In the new approach two histograms of the foreground pixels are calculated. One histogram is in the middle of the left-hand side of the text area, i.e. it covers the area from 12.5 % to 37,5 % of the text width; the other histogram is situated in the middle of the right-hand side, i.e. it covers the area from 62.5 % to 87,5 % of the text width. The maxima of these two histograms are then matched using a greedy search algorithm, which iteratively matches the maxima with the lowest y -difference and the largest number of strokes intersecting the connecting line. The approach is illustrated in Fig. 2, where the areas considered for the histograms are highlighted with a gray background. In our experiments described in Section 6 the second approach based on histograms performed better than linear regression.

After preprocessing, a dynamic programming procedure is applied to detect the text lines. It searches an optimal path guided by a cost function that consists of a combination of several auxiliary functions. The dynamic programming procedure is outlined in Section 3, while the cost functions

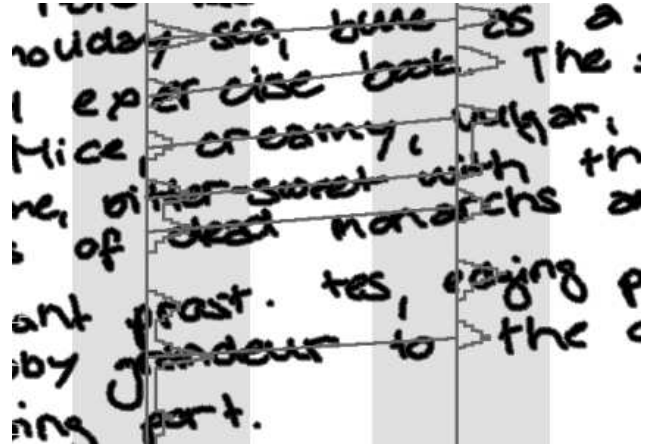


Figure 2. Skew estimation by using two histograms

including the auxiliary functions are described in detail in Section 4. The dynamic programming procedure already outputs lines of good quality. At the end, however, some postprocessing steps are applied to further improve the results.

3. Dynamic programming

Dynamic programming is a well established technique for solving optimization problems [2]. It has been successfully applied in various subfields of image analysis, particularly in edge detection [1]. Dynamic programming is applicable wherever the given optimization problem consists of similar subproblems and the optimal solution can be determined using the optimal solutions of the subproblems. In general the optimal solution for a given function $h(x_1, \dots, x_N)$ can only be found using an exhaustive search through all parameters x_i , resulting in an exponential calculation time. If it is known that the problem can be divided into subproblems h_1, \dots, h_{N-1} and the optimal values of $h_{n-1}(x_{n-1}, x_n)$ are independent of all other parameters the problem can be solved in $O(N * k^2)$ time, where k is the number of discrete values each x_i can take on.

Dynamic programming works according to the following formula:

$$\min_{x_i, 1 \leq i \leq N} h(x_1, \dots, x_N) = \min_{x_N} f_{N-1}(x_N) \quad (1)$$

where $f_0(x_1) = 0$ and

$$\min_{x_n} f_{n-1}(x_n) = \min_{x_{n-1}} (f_{n-2}(x_{n-1}) + h_{n-1}(x_{n-1}, x_n)) \quad (2)$$

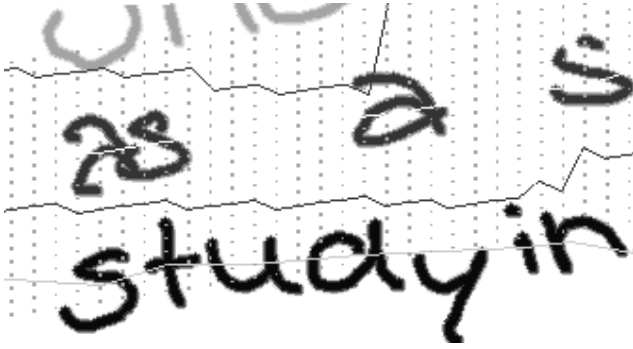


Figure 3. Example grid for a path

In the problem considered in this paper, the N parameters are the y -positions of the path at each horizontal position. This results in a grid of possible positions for the best path. Figure 3 illustrates a grid for an example text². Note that the grid has been adjusted to the skew.

4. Cost Function

The function $h_{n-1}(x_{n-1}, x_n)$ for the subproblems is calculated using three auxiliary functions. These functions are described in this section. Since the optimization problem is to find a path with minimal cost, each function can be seen as a penalty for any deviation from the optimal path. Eventually, the path with the smallest overall penalty, or cost, is the desired solution.

To avoid very high differences in y -direction between consecutive points on the path, we use the vertical distance between two adjacent positions for the calculation of the first auxiliary function. After subtracting the skew the penalty is calculated according to the following formula:

$$h_{y\text{-difference}}(x_{n-1}, x_n) = d(y_{n-1}, y_n)^3 \quad (3)$$

In the letters of the Roman alphabet, delayed strokes, such as i-dots and quotation marks, usually belong to the text below. Only some punctuation marks, like dots and commas, have to be assigned to the text above, but they usually are already at the correct y -position. One can benefit from this property if the optimal path of the dynamic programming procedure is closer to the text line above than to the text line below. This idea is captured in the *evade*-function, which adds more penalty if the path gets closer to strokes below. To avoid jumping to another line, we also add a penalty to the grid points to the right and to the left. However this penalty in horizontal direction decreases

²Note that in this and the following figures, different shades of gray have been used to represent the individual lines detected by our algorithm.

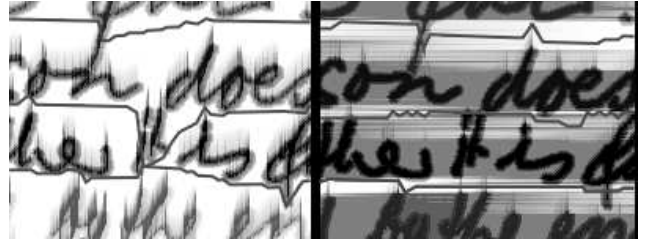


Figure 4. Effects of the evade-function; note that there is the same text on the left and right - only the gray background that represents the penalty function differs

with distance. More formally, the penalty added to a given point x_n is defined as follows:

$$h_{evade}(x_{n-1}, x_n) =$$

$$\max_{p \in P} ((C - d(x_n, p_x)) + (C - 2 * d(y_n, p_y))) \quad (4)$$

where P is the set of foreground pixels in the area below x_n where both summands are greater than or equal to 0, and C is a constant. Thus the penalty decreases if the distance becomes larger. Figure 4 illustrates the effect of adding this penalty. In the left-hand side of this figure only the distance in vertical direction is taken into account, while on the right-hand side both distances are considered. In this figure the penalty for all grid points is displayed in light gray. We observe that there is a segmentation error in the left part of Fig. 4. However, thanks to the influence or the penalty in horizontal direction, this error is avoided in the right-hand side. In the area of small strokes we apply this penalty in the opposite direction, because most likely these strokes are points corresponding to the line below. Figure 5, on the left-hand side, illustrates what happens if we do not apply this special handling for small strokes, i.e. an i-dot is assigned incorrectly. The correct segmentation obtained under the extended procedure is shown on the right-hand side.

The third auxiliary function is derived from the assumption that the words have roughly been written next to each other. To prevent the path from jumping into the text line above or below, we add a fixed penalty if the path crosses space of a supposed text line. We assume that large strokes that have their center of gravity in vertical direction near the maximum of the vertical histogram are all in the same text line. This initial guess is similar to the connected component based approaches [8]. We define:

$$h_{cross}(x_{n-1}, x_n) = 1, \text{ if there is a crossing} \quad (5)$$

The effect of this auxiliary function is illustrated in Fig. 6.

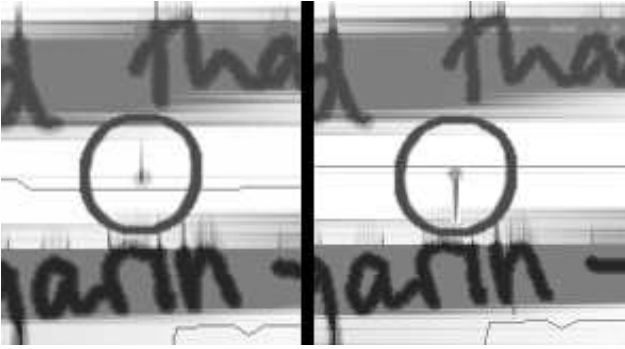


Figure 5. The i-dot is correctly assigned with the special handling for small strokes

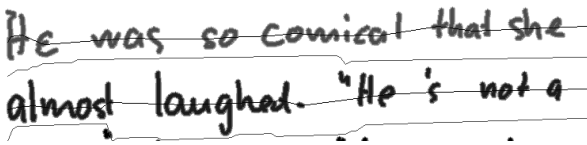


Figure 6. Cost function for preventing the crossing of lines

Finally, the cost function $h_{n-1}(x_{n-1}, x_n)$ used in the dynamic programming procedure consists of a linear combination of the three auxiliary functions introduced above and is defined as follows:

$$h_{n-1}(x_{n-1}, x_n) = h_{y-difference}(x_{n-1}, x_n) + h_{evade}(x_{n-1}, x_n) + c * h_{cross}(x_{n-1}, x_n) \quad (6)$$

where c is a parameter that is optimized on an independent validation set.

5. Postprocessing

After the dynamic programming approach has detected the text lines as described above, we perform postprocessing operations to eliminate a few common mistakes which may have occurred during preprocessing and the search for the optimal path. The first correction is the removal of empty text lines, which appear if too many starting points have been estimated.

Another effect of too many starting points is the existence of pseudo text lines which consist of quotation marks and i-dots only. These lines usually contain only a small number of strokes, and the strokes have a few points only. To recover from errors of this kind we merge these lines with the corresponding text lines below.

The last postprocessing step is the only one which directly uses on-line information. Since some i-dots and punctuation marks may have been assigned to a wrong text line, we look at the strokes in the on-line vicinity of each stroke s_i . If these strokes are also nearby in space, but in another estimated line, we reassign s_i to the other text line. In the next section the effect of this on-line postprocessing step is demonstrated.

6. Experiments and Results

Our experiments have been conducted on the IAM-OnDB, a large on-line handwriting database acquired from a whiteboard [6]. We used a set of 100 documents for testing. Each document has about six to ten text lines with an average number of 25 strokes each. A different set has been used for validating the parameters of the dynamic programming approach.

In order to compare the performance of the proposed system, we implemented an on-line reference system. This reference system is based on simple heuristics, i.e. if the pen-movement to the left is larger than N pixels and there is a pen-movement to the bottom it starts a new text line. This is a common procedure found in many on-line systems [7, 10]. The reference system usually fails if a punctuation mark has been written later than the words next to it, or if a letter or a word has been forgotten and inserted later.

The parameters of the proposed method and the reference system have been optimized manually on the validation set. The weights of first two auxiliary functions of Section 4 have been set to 1, while the last one has a factor of 1000. We tested the proposed system two times. First we applied the dynamic programming and the off-line postprocessing; in the second test we also applied the on-line postprocessing.

System	Stroke	Document
On-line reference	93.40 %	62 %
Proposed system 1	99.79 %	90 %
With on-line postpr.	99.94 %	98 %

Table 1. Results on 100 documents: stroke and document classification rate

Table 1 shows the classification rates of the proposed system and the reference system on the stroke level and on the document level. The strokes classification rate is the number of correctly assigned strokes divided by the total number of strokes. Similarly, the document classification rate is the number of correctly processed documents divided by the total number of documents. First we notice a very remarkable improvement from 93.40% (62 %) to 99.79 % (90 %) achieved with the proposed system without

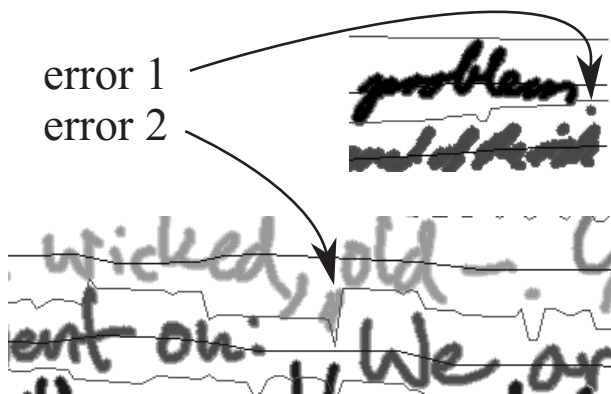


Figure 7. Examples of misclassified strokes

on-line postprocessing. Adding the on-line postprocessing step, we can further improve the performance. Note that the improvement from 99.79 % to 99.94 % is statistically significant ($\alpha=0.05$). In the system with on-line postprocessing only two documents contained small mistakes.

7. Conclusions and Outlook

In this paper we presented a new approach for detecting text lines in on-line handwritten documents. This approach is based on dynamic programming and uses both the off-line and on-line information of the strokes in a handwritten document. Our system tries to find a path between two consecutive text lines based on a heuristic cost function that takes different criteria into account. In addition to two other criteria, which enforce the smoothness of the optimal path and prevent it from crossing a text line, we propose the novel *evade*-function which adds more penalty if the path is closer to strokes below. Using this function the path is forced to run closer to the text line above and many punctuation marks and i-dots are correctly assigned.

In our experiments on 100 documents we could outperform a commonly used on-line approach based on simple heuristics. Without on-line postprocessing we achieved a correct assignment rate of strokes of 99.79 %, which could be increased to 99.94 % by using on-line information for reassigning small strokes to other text lines. Only two documents out of 100 contain small mistakes. Figure 7 shows some examples of the errors. Even for a human it is not easy to assign the misclassified strokes to the correct text line without reading the text. The first error is a period which has been treated as an i-dot and assigned to the text line below. The second error are misclassified quotation marks from the line below which have been treated as commas.

While the proposed system already works very well, it is a hard problem to detect text lines perfectly. It can be argued

that significant improvements in text line segmentation can be achieved only if text line detection and text recognition are integrated with each other. Such an integration is a challenging direction for future research.

References

- [1] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice-Hall, Inc., 1982.
- [2] R. Bellman and S. Dreyfus. *Applied dynamic programming*. Princeton University Press, 1962.
- [3] H. Bunke. Recognition of cursive roman handwriting - past present and future. In *Proc. 7th Int. Conf. on Document Analysis and Recognition*, volume 1, pages 448–459, 2003.
- [4] E. Kavallieratou, N. Fakotakis, and G. Kokkinakis. An unconstrained handwriting recognition system. *International Journal on Document Analysis and Recognition*, 4(4):226–242, 2002.
- [5] Y. Li, Y. Zheng, D. Doermann, and S. Jaeger. A new algorithm for detecting text line in handwritten documents. In *Proc. 10th Int. Workshop on Frontiers in Handwriting Recognition*, pages 35–40, 2006.
- [6] M. Liwicki and H. Bunke. IAM-OnDB - an on-line English sentence database acquired from handwritten text on a whiteboard. In *Proc. 8th Int. Conf. on Document Analysis and Recognition*, volume 2, pages 956–961, 2005.
- [7] M. Liwicki and H. Bunke. HMM-based on-line recognition of handwritten whiteboard notes. In *Proc. 10th Int. Workshop on Frontiers in Handwriting Recognition*, pages 595–599, 2006.
- [8] R. Manmatha and J. Rothfeder. A scale space approach for automatically segmenting words from historical handwritten documents. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27, 2005.
- [9] U.-V. Marti and H. Bunke. Text line segmentation and word recognition in a system for general writer independent handwriting recognition. In *Proc. 6th Int. Conference on Document Analysis and Recognition*, pages 159–163, 2001.
- [10] R. Plamondon and S. N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(1):63–84, 2000.
- [11] A. Vinciarelli. A survey on off-line cursive script recognition. *Pattern Recognition*, 35(7):1433–1446, 2002.
- [12] B. Yu and A. K. Jain. A robust and fast skew detection of algorithm for generic documents. *Pattern recognition*, 29(10), 1996.