2011 International Conference on Document Analysis and Recognition

Reliable Online Stroke Recovery from Offline Data with the Data-Embedding Pen

Marcus Liwicki*, Yoshida Akira†, Seiichi Uchida†, Masakazu Iwamura‡, Shinichiro Omachi§ and Koichi Kise‡
*DFKI, Germany, Email: marcus.liwicki@dfki.de
†Kyushu University, Japan, Email: yoshida@human.ait.kyushu-u.ac.jp, uchida@ait.kyushu-u.ac.jp
‡Osaka Prefecture Univ., Japan
§Tohoku Univ., Japan

***Abstract*—In this paper we propose a complete system for online stroke recovery from offline data. The key idea of our approach is to use a novel pen device which is able to embed meta information into the ink during writing the strokes. This pen-device overcomes the need to get access to any memory on the pen when trying to recover the information, which is especially useful in multi-writer or multi-pen scenarios. The actual data-embedding is achieved by an additional ink-dot sequence along a handwritten pattern during writing. We design the ink-dot sequence in such a way that it is possible to retrieve the writing direction from a scanned image. Furthermore, we propose novel processing steps in order to retrieve the original writing direction and finally the embedded data. In our experiments we show that we can reliably recover the writing direction of various patterns. Our system is able to determine the writing direction of straight lines, simple patterns with crossings (e.g., "x" and "ll"), and even more complex patterns like handwritten words and symbols.**

***Keywords*-data-embedding pen; stroke recovery; information encoding**

I. Introduction

It is widely agreed that online handwriting recognition perform better than their offline counterpart [1]. Therefore, one focus of the handwriting recognition community is to recover the online trajectory from offline images in order to improve the performance of the recognizer.

A summary of pen trajectory recovery methods until 2010 has been given in [2]. There different approaches for image processing and local examination have been compared. [2] identified two different types of input material, i.e., skeleton or contour. Furthermore, they presented methods to treat ambiguous zones, double traced handwriting, hidden loops, and end-points. Finally, methods for global reconstruction have been summarized.

In this paper we embed additional information during the writing process which encodes the direction and velocity information. Such information may be beneficial for improved handwriting recognition or writer identification. Furthermore, we enhance the stroke recovery process to make use of this information. This idea is realized by using a pen which is capable of producing small ink-dots alongside the handwritten stroke [3]. If these ink-dots are produced in a time-synchronized manner, we are able to recover the sequence information later. In this paper we propose methods implementing this idea.

The data-embedding pen was originally developed to increase the value of handwritten information on ordinary paper and add meta-information to the offline data. This is an orthogonal approach to other digital pens which store the stroke sequences on a computer along with meta-information. While those pens work on paper, they cannot increase the value of handwriting on paper, because the handwriting left on the paper is still just an ink pattern without any meta-information. The general feasibility of the data-embedding pen has been proven in [3]. The main novel contribution of this paper is that we introduce an ink-dot generation method which can be used for stroke trajectory recovery.

The remainder of this paper is organized as follows. First, Section II describes the novel digital pen device and the data embedding. Second, Section III presents the steps for image processing. Subsequently, Section IV Next, Section V reports on stroke recovery experiments performed with the new method. Finally, Section VI draws some conclusions and gives an outlook to future work.

II. The Data-Embedding Pen

The data-embedding pen is a device which comprises a usual ball-point pen and an ink-jet nozzle element. Figure 1 depicts this device. During the writing, the nozzle produces small ink-dots (called *information ink*) alongside the handwritten stroke. The color of the ink-dots is different from the color of the stroke. In this paper, yellow is used for the ink-dots. (Invisible ink has already been tested as a good alternative.) The number of the ink-dots and their timing are used to encode the desired information.

The nozzle is able to generate up to $2,000$ ink-dots per second. Using this high frequency, we can form a connected line by a sequence of several ink-dots. Hereafter, a line by n sequential ink-dots is called *n-pulse line*. If $n = 1$, the n-pulse line forms a single ink-dot. The line, of course, becomes longer by increasing n.

Our coding scheme is based on the combination of three different n-pulse lines. Specifically, we use $n = 1$ (a dot), 5 (a short line), and 20 (a long line). The ink-dot sequence of Fig. 2 consists of those n-pulse lines. Roughly speaking, the information is converted into a binary (0 and 1) sequence

1520-5363/11 $26.00 © 2011 IEEE
DOI 10.1109/ICDAR.2011.278

Figure 1. The data-embedding pen.

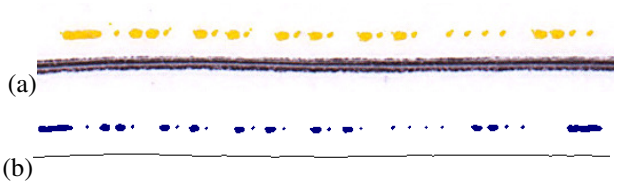


Figure 2. (a) Ink-dots (light) nearby a handwriting stroke (black). (b) After image processing.

and embedded by using the 1-pulse line as 0 and the 5-pulse line as 1. A short pause is prepared between each bit information (1-pulse or 5-pulse line) like in the Morse code. The 20-pulse line, hereafter called *synchronization blob*, is used as an anchor to make sure that a correct position is extracted (see the leftmost dot in Fig. 2).

Our coding scheme is defined by three units, called *frame*, *block*, and *bit*. The bit is the smallest unit and defined by a 1-pulse line or a 5-pulse line. Several consecutive bits comprises a block and several consecutive blocks comprises a frame. Each frame begins (or, equivalently, ends) at a synchronization blob.

Figure 2 is an example of a single frame. From left to right, the ink-dot sequence of the frame is comprised of a synchronization blob, 6 blocks, and another synchronization blob. In each block, 4 bits are encoded and thus in the frame 24 bits $(0110 - 1010 - 1010 - 1010 - 0000 - 1100)$ are embedded. More information of the encoding of information is given in [4].

In the following we present the main idea of how to encode the direction and speed information, since the main purpose of this paper is stroke trajectory recovery. As described above, a sequence of n-pulse lines is produced. This is done with a fixed frequency, i.e., the information is produced with 100 Hz. Consequently, the bits in a block are produced every 10 ms. After each block, a pause of another 10 ms is added. Furthermore, an additional pause of 10 ms is added before and after each frame, resulting in a longer pause before synchronization blobs than after synchronization blobs. In our experiments, the number of bits per block is 4 and the number of blocks per frame is 2. This results in an overall time of 130 ms for one frame

Figure 3. The direction of the closest stroke to the synchronization blob can be interpreted as the writing direction.

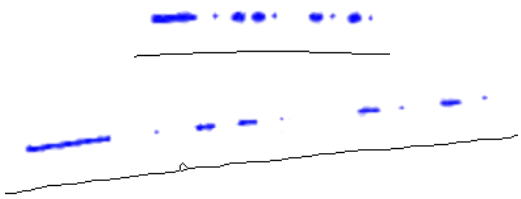


Figure 4. Different length of the n-pulse lines depending on the velocity. (top: slow velocity, bottom: faster velocity)

followed by one synchronization blob.

The direction information is now embedded at the synchronization blobs (appearing at 7.7 Hz). The pause before the synchronization blob is twice as long as the pause after the blob. Hence, the side with the shorter distance to the next ink dot would be the one in writing direction. An illustration of this property is shown in Fig. 3. Ideally, it would be possible to get the correct direction information of non-overlapping lines with a duration of at least 260 ms.

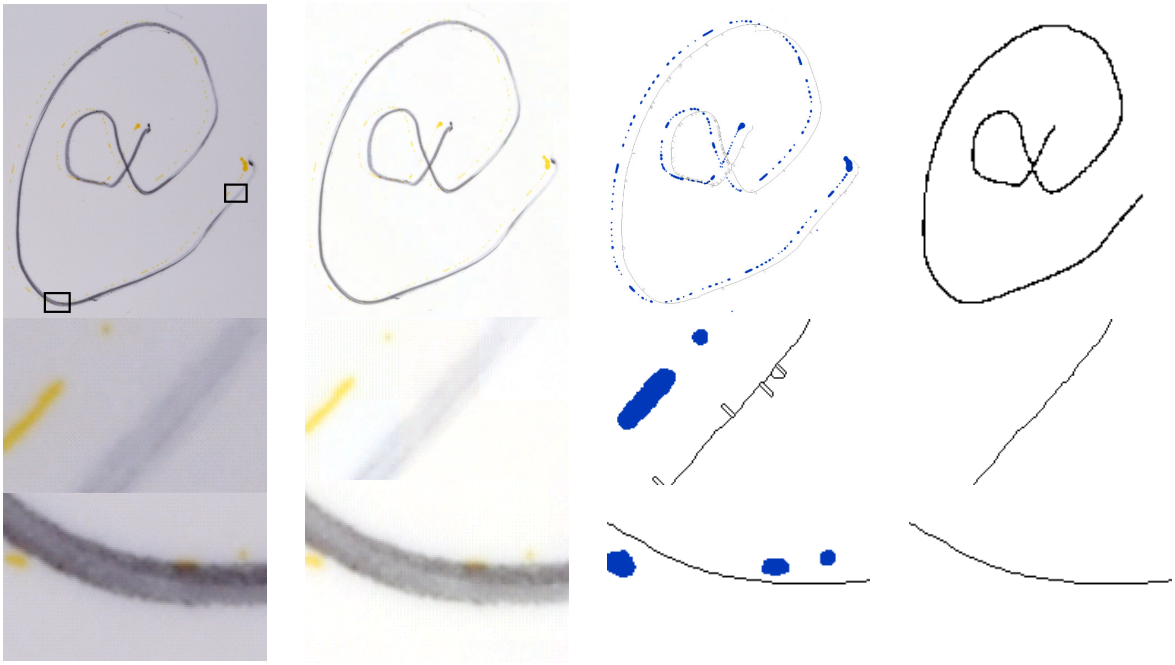
The speed information can be derived from the blocks within the frames. Since a new n-pulse line starts every 10 ms, the actual distances between the n-pulse lines can be used to estimate the speed. Furthermore, longer n-pulse lines would correspond to faster writing (see Fig. 4).

Furthermore, the writing angle and the tilt of the pen can be roughly estimated by using the correspondence information from the ink-dots to the line (see Section IV). Note that a shorter distance would indicate a larger tilt angle if the nozzle is mounted on the pen as in Fig. 1.

III. Image Processing

Our system works on scanned and photographed images. Especially the latter type introduces many difficulties which do not occur on scanned images: different illuminations (depending on the flashlight, varying sizes for the ink dots, and different thickness for the strokes (depending on the distance between the camera and the paper. In the this section, the main stages for assessing these problems, as well as the further preprocessing steps will be explained.

The results of the individual steps are depicted in Fig. 5. There, an example of a photographed "@" symbol is given on the top. Below, two example regions which are part of the "@" symbol are illustrated to visualize the behavior of



(a) (b) (c) (d)

Figure 5. Example of a photographed "@" symbol (a) before image processing, (b) after white normalization, (c) after color extraction and thinning, (d) after final skeletonization.

the algorithm.[1]

As stated above, we can not assume the same lighting for all the photographs. The situation is even worse, i.e., we face the problem of inhomogeneous lighting (usually, the center is more exposed by the flashlight). Simply using a global threshold for color extraction would fail under these circumstances. Therefore, we use a low-resolution grid (motivated from work in related areas [5], [6]) and determine the brightest point in each sub-region. This is then used as a reference for white in this sub-region and the color values are normalized according to this value (*white normalization*). The size of the sub-region was set to 50×50 pixels to make sure that at least one target white pixel occurring in this region. Note that this method also works on homogeneous background colors (we have successfully performed small tests with *blue* and *yellow!* background). The result of white normalization in Fig. 5 (a) can be seen in Fig. 5 (b).

The next three steps basically follow the approach of [7]. The second step is noise removal, i.e., erosion and dilation are applied. Figure 5 (c) shows the result. Note that the parameters for those operations were optimized on a small training set. The histogram of the n-pulse lines sizes is then further investigated. The sizes are clustered in three regions, small, medium and large regions. These clusters represent the three different ns of the n-pulse lines, i.e., the binary information or a synchronization blob.

The third step is a special treatment of ink-dots occluded by the black ink stroke. During writing complex patterns, it often appears that yellow ink-dots overlay with the stroke. Fortunately, those yellow ink-dots are still visible on the stroke (they just appear to be a bit darker). To overcome this problem another thresholding operation is performed on already extracted pixels with a lower thresholds to recover dark yellow ink-dots. As can be seen in Fig. 5 (c), the yellow ink dot becomes visible, even if it was totally occluded by the black line.

The fourth step is a thinning operation on the black ink stroke. Figure 5 (c) shows the result of an orthodox thinning method. Then, after removing many small loops and short spurious edges by unifying neighboring branches as introduced in [7], the final thinning result is obtained as shown in Fig. 5 (d).

The final result of image processing is a graph (obtained by using the graph-representation of the skeleton, c.f., Fig. 5 (d)) of ink traces where nodes represent end-points in the traces or crossings, whereas the edges represent the paths between the crossings or end-points. The graph in Fig. 5 has three nodes (the end-points and the crossing) and three paths (the two paths at the end points and the loop).

[1]The authors of this paper are aware that the yellow ink-dots in (a) and (b) are hardly visible on a grayscale-printout. We have done this intentionally for several reasons. First, we did not want to alter the original image in order to show the difficulties for the algorithm. Second, the extracted ink-dots will be marked in blue for all processed images (e.g., (c)). Finally, in the electronic version of the publication one can see the yellow ink.

IV. Stroke Recovery

As mentioned in Section II the information ink dots were produced in such a way that the direction, velocity, and angle information is embedded into the ink-dot sequence to some degree. In the following we will describe the trajectory recovery, which is the main focus of this paper.

First, the correspondence between the information ink dots and the stroke[2] is established. The basic idea of establishing the correspondence is to find the closest point on the stroke for each ink-dot. A simple nearest neighbor, however, cannot always provide a correct result because a dot and its corresponding point might be a bit distant due to the pen tilt. Thus, at each ink-dot k, we first calculate the minimum distance $d_{k,\theta}$ to the stroke for each θ of 36 angles (with 10° interval). Then, we select the angle $\overline{\theta}$ with minimum variance, i.e., $\overline{\theta} = \operatorname{argmin}_\theta \operatorname{Var}\{d_{1,\theta}, \ldots, d_{K,\theta}\}$. This angle is the most stable angle and thus represents a projection of the actual pen angle and tilt during writing. Finally, for each ink-dot k, the corresponding point is determined as the closest point when using angle $\overline{\theta}$. If many ink-dots were not assigned to a stroke, this process is repeated, because it might be that the tilt has been changed during writing.

Second, an initial estimation of the path directions is performed based on the information which is available from the ink dots. This particular step makes use of the information which has been embedded into the code during writing, i.e., the pauses between the ink-dots and the synchronization blobs differ (see above).

Hence the synchronization blobs are taken into account for direction estimation. For each synchronization blob, the path of the corresponding stroke is traced until the next information ink dot appears. If there is an information ink dot in both directions, the one which is closer is assumed to lie in the direction of writing and thus a counter of this direction is increased for that path. Then the guesses are summed up and the direction with a higher counter is taken. This leads to a good guess, especially for longer strokes.

Third, the nodes of the graph are taken into account. Each node of the graph has a number of inbound, outbound and unassigned paths (edges). As the direction of some paths has been determined already in the second step, we now propagate the direction information by using the following procedure for all unassigned paths p_i. The steps are applied repeatedly (Note that each path has two corresponding nodes):

1) If a path p_i exists where one of the nodes (e.g., Node n_1) has a missing outbound edge (i.e., more paths are ending at this node than beginning at this node) and the other node n_2 has a missing inbound edge (i.e., more paths are beginning at this node than ending at this node), the direction of the edge goes from n_1 to n_2.
2) Repeat 1 until no such path exist anymore.
3) If a path p_i exists where one of the nodes has a missing inbound or outbound edge, the path direction of p_i is set according to the missing path.
4) Repeat 3 until no such path exist anymore.
5) Up to now, small loops were not affected by the previous steps. For those remaining loops we apply the methods initially proposed by [7].

Finally, the remaining paths are considered to be double traces. This is usually the case for crossings with three edges, where a path continues and a short edge goes into one direction. Due to overlay of the ink-traces in those paths as well as the ink-blobs it is impossible to get the exact trajectory information if there is a small difference of the up- and down-stroke.

As a result of these processes we obtain the direction information of all paths. Now we start to trace the lines starting at the top-left path with an end-node of degree 1. When arriving at a node of degree > 3 during tracing, we go into the direction of a loop and also consider the heuristics introduced in [7] in tied situations. After reaching an end, the next untraced path with an end-node of degree 1 is taken into account, etc.

[2]Note that a stroke is seen as the sequence of points between two consecutive pen-down and pen-up movements.

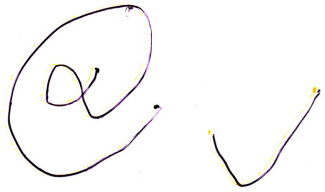


Figure 6. Example images

V. Experiments

In our experiments we evaluated the performance of the stroke trajectory recovery method on patterns with different difficulties. We collected a set of $80 - -120$ samples for each of the following patterns (Some examples are given in Fig. 6):

- closed circles (diameter of 5 cm and 3 cm)
- straight lines (5cm) in all four directions (right, left, up, down)
- drawn "x" (size of 3 cm and 5 cm)
- "@" symbols (size of 3 cm and 5 cm)
- "ll" (size of 3 cm and 5 cm)
- hooks (size of 3 cm and 5 cm)
- the word "Clever" (height 4 cm)

All examples have been written by a person who was not aware of the processing methods. Note that most of these patterns contain samples where it is not possible to perfectly

Table I
DIRECTION DETECTION ACCURACY IN %

System	circles		lines				"x"		"ll"		hooks		clever
	3 cm	5 cm	down	right	up	left	3 cm	5 cm	3 cm	5 cm	3 cm	5 cm	
[7]	—	—	100	100	0	0	50	50	100	100	100	100	80
Proposed	100	100	100	100	100	100	79	98	45	100	100	100	95
With post-processing	100	100	100	100	100	100	79	98	100	100	100	100	96

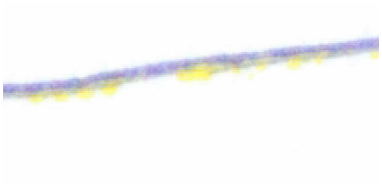


Figure 7. A problematic case where the information ink is smudged by the ballpoint pen

recover the trajectory information if no information ink is available.

In our experiments on this data set we investigated the performance of the direction detection algorithm, because this is the main advantage of our method (Note that the image processing is the state-of-the-art skeletonization [2]). A method without using the information ink trace, i.e., the approach of [7], has been used as a reference system. The accuracy is defined as the number of path with a correctly identified direction divided through the number of all paths.

The results of the trajectory recovery appear in Table I. As can be seen, the algorithm of [7] performs already good on many patterns. However, it has some complications with closed circles, lines which go against the natural direction, and two-stroke patterns.

Using the embedded information significantly increases the performance. Our method works perfect on most patterns. Only small patterns introduce some complications. A simple idea for post-processing is to apply the method of [7] if only a single path is available and no direction could be determined. The last row shows the performance if this strategy is applied. The final method performs with 100 % on 10 out of 13 patterns.

An analysis of the failures shows that during acquisition the ink-dots were often overlapping the pen-stroke. Often, this causes no problem, however, when the tip of the ballpoint pen touches already existing information ink-dots. In this case the ink is smudged by the pen and therefore our algorithm is not able to recover the correct information (see Fig. 7. In future we will try to tackle this problem by improving the image processing technologies.

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an orthogonal approach for online stroke trajectory recovery. We propose a special pen device which is able to produce a sequence of information ink-dots next to the handwritten stroke. We introduce a special procedure for generating these ink-dots making it possible to recover the direction information, as well as the speed and up to some degree the angle and tilt of the pen during writing. Furthermore, we added novel steps to the recovery process which take the ink-dot information into account.

Our experiments emphasize the strengths of our approach. For most patterns we could perfectly recover the correct writing directions. By the help of the information ink-dots we are able to outperform standard recovery approaches.

In future we plan to improve the image processing of our method. Furthermore, we have received an ink-device which is able to produce even smaller dots. This will be used in a large-scaled evaluation experiment in order to assess the overall recovery and embedding power of our algorithms.

ACKNOWLEDGMENT

This work has been financially supported by the ADIWA project.

REFERENCES

[1] R. Plamondon and S. N. Srihari, "On-line and off-line handwriting recognition: a comprehensive survey," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 22, no. 1, pp. 63–84, 2000.

[2] V. Nguyen and M. Blumenstein, "Techniques for static handwriting trajectory recovery: a survey," in *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, 2010, pp. 463–470.

[3] M. Liwicki, S. Uchida, M. Iwamura, S. Omachi, and K. Kise, "Data-embedding pen — augmenting ink strokes with meta-information," in *9th Int. Workshop on Document Analysis Systems*, 2010.

[4] ——, "Embedding meta-information in handwriting — Reed-Solomon for reliable error correction," in *12th International Conference on Frontiers in Handwriting Recognition*, 2010, pp. 51–56.

[5] A. Jain, *Fundamentals of Digital Image Processing*. Prentice-Hall, 1989.

[6] M. Simon, S. Behnke, and R. Rojas, "Robust real time color tracking," in *RoboCup-2000*, 2001, pp. 239–248.

[7] Y. Kato and M. Yasuhara, "Recovery of drawing order from single-stroke handwriting images," *IEEE Trans. Pat. Anal. Mach. Intell.*, vol. 22, no. 9, pp. 938–949, 2000.