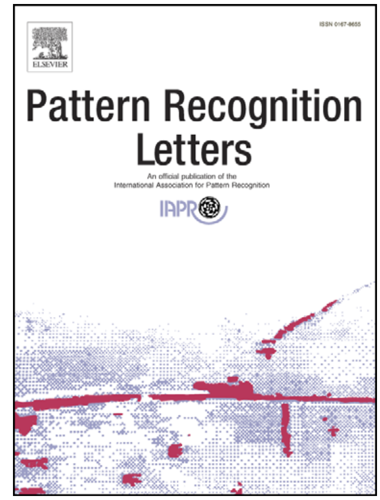


More than Ink — Realization of a Data-Embedding Pen

Marcus Liwicki, Seiichi Uchida, Akira Yoshida, Masakazu Iwamura, Shinichiro Omachi, Koichi Kise

PII: S0167-8655(12)00286-3
DOI: http://dx.doi.org/10.1016/j.patrec.2012.09.001
Reference: PATREC 5501

To appear in: *Pattern Recognition Letters*

More than Ink — Realization of a Data-Embedding Pen

Marcus Liwicki[a], Seiichi Uchida[b], Akira Yoshida[b], Masakazu Iwamura[c], Shinichiro Omachi[d], Koichi Kise[c]

[a] *DFKI, Germany, Email: marcus.liwicki@dfki.de*
[b] *Kyushu Univ., Japan, Email: uchida@ait.kyushu-u.ac.jp*
[c] *Osaka Prefecture Univ., Japan*
[d] *Tohoku Univ., Japan*

Abstract

In this paper we present a novel digital pen device, called data-embedding pen, for enhancing the value of handwriting on physical paper. This pen produces an additional ink-dot sequence along a written stroke during writing. This ink-dot sequence represents arbitrary information, such as writer's name and writing date. Since the information is placed on the paper as an ink-dot sequence, it can be retrieved just by scanning or photographing the paper. In addition to the hardware of the data-embedding pen, this paper also proposes a coding scheme for reliable data-embedding and retrieval. In fact, the physical data-embedding on a paper will undergo various severe errors and therefore a robust coding scheme is necessary. Through experiments on data written by two writers, we show that we can embed 32 bits on short and simple or even on more complex patterns and finally retrieve them with a high reliability.

Keywords: data-embedding pen, stroke recovery, handwriting

1. Introduction

Handwriting is an important modality for writing down information, making annotations, or just marking items. Unfortunately, as soon as the ink is on the paper, many information known during writing is already lost. We cannot access meta-information about the handwritten pattern from itself; it is impossible to retrieve who wrote this pattern or when it was written. In other words, a handwritten pattern on a physical paper is just an ink pattern and thus cannot provide any information but its shape.

Digital pens have emerged as a choice to store and retrieve such additional information. In fact, several digital pens capturing handwriting on normal paper have been developed. Those pens can store the stroke sequences on a computer with some additional information. Unfortunately, the digital pens cannot increase the value of handwriting on paper; even with the digital pens, the handwriting left on the paper is still just an ink pattern without any information.

In this paper, we propose a novel pen device to enrich the handwriting on the physical paper. The proposed pen device, called *data-embedding pen*, can embed arbitrary information, such as meta-information, by an additional ink-dot sequence along the ink stroke of the handwriting. Each ink-dot represents an information bit and thus an ink-dot sequence represents a bit-stream of the information to be embedded. The information can be retrieved by scanning or photographing the paper and decoding the ink-dot sequence.

The most important property of the data-embedding pen is to increase the value of handwriting on the physical paper. If we embed the writer ID, the handwriting on the physical paper itself stores this meta-information and identifies the writer without using an electronic memory. If we embed an URL into the handwriting, the handwriting becomes a link between the physical paper world and the cyber-space, i.e., the Internet. Furthermore, if we embed any temporal information or hints into the pattern, it is possible to convert the strokes into the online representation which is helpful to attain a better handwriting recognition accuracy.

The contributions by this paper are summarized as follows.

- The idea of embedding information into handwriting is very novel, as reviewed in Section 2.
- For this idea, a prototype of the data-embedding pen is implemented using a special ink-jet nozzle element. Such an implementation has never been developed before.
- For reliable data-embedding and data-retrieval, image processing techniques and a coding scheme are proposed, both of which are specialized for the data-embedding pen.
- Experimental results with the prototype proved that arbitrary 32-bit information can be embedded into, for example, a 5cm-length handwriting pattern and retrieved perfectly from the pattern.

Note that this paper is not only the first compilation of the authors' past trials (Uchida et al., 2006; Liwicki et al., 2010a,b, 2011) but also a new report of experimental results on a broader set of handwritten patterns.

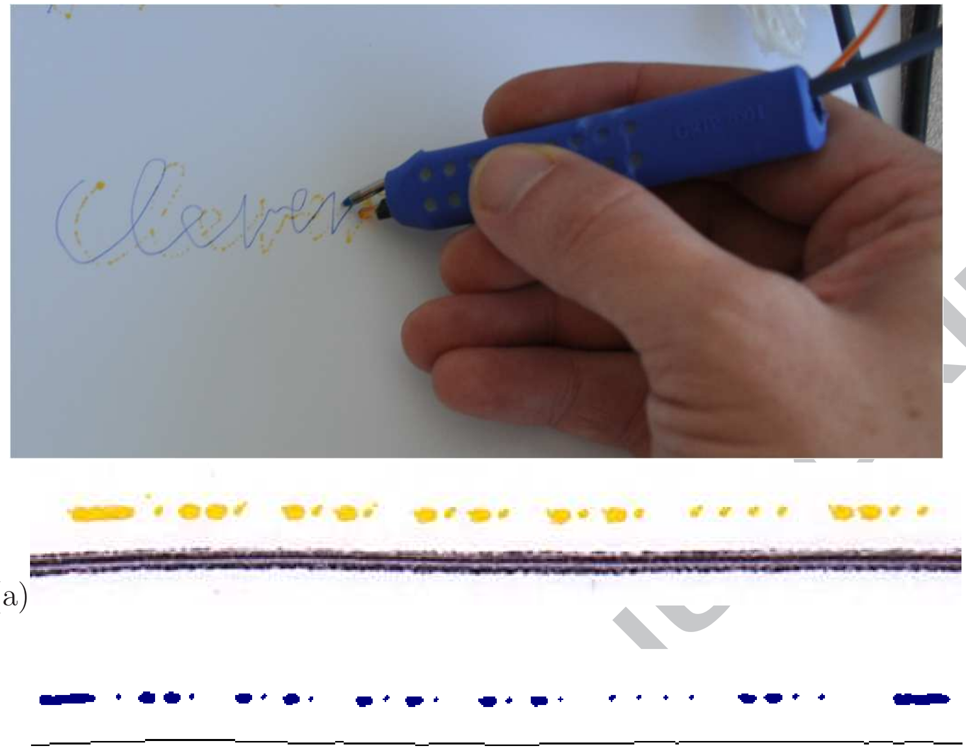
2. Related Work

To the authors' best knowledge, the data-embedding pen is the first trial on implementing a new pen device which can embed arbitrary information dynamically into handwriting on a paper. Generally, embedding data into paper has been done statically by a printer. For example, XEROX DataGlyph (Hecht, 1994) is a kind of digital watermarks and information is printed and embedded as a fine texture into font images or photographs.

Digital pens are popular devices to enhance the usability of handwriting. Various types of digital pen devices are already available. Tablets are the most widely available digital pen-input modality. Wacom tablets[1], for example, capture the pen-tip trajectory by a note-size flat pad with sensor array. Other digital pens use ultrasonic to capture the pen-tip location. All those digital pens are just useful to transfer handwritten patterns to the computer. In other words, they do not enhance the "function" of the handwriting on paper itself.

Nowadays, the most famous digital pen seems to be the Anoto pen[2]. Anoto reads the dot pattern printed on the paper surface from its pen-tip camera and detects its absolute position on the paper by interpreting the pattern. By continuously detecting the position during the pen movement, Anoto can acquire the online trajectory. Like other digital pens, the Anoto pen also has a very different purpose from the data-embedding pen, i.e., just transferring the pattern to the computer. In fact, the handwritten pattern on the paper is just a pattern showing a stroke shape and thus no additional value. In contrast, the purpose of the data-embedding pen is to enhance the function of the handwritten pattern on paper.

[1] http://www.wacom.com — last visited: November 2011
[2] http://www.anoto.com — last visited: November 2011



(a)

(b)

Figure 1: A prototype of the data-embedding pen (top). (a) Ink-dots (light) nearby a handwriting stroke (black). (b) After image processing.

3. The Data-Embedding Pen

3.1. Hardware Prototype

The data-embedding pen is a device which comprises a usual ballpoint pen and an ink-jet nozzle element. Figure 1 (top) depicts a prototype of this device with the ballpoint pen at the top and the nozzle at the bottom. Figure 1 (a) is a handwritten pattern generated by the prototype. During the writing, the nozzle produces small ink-dots 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 for better visibility of the results. Invisible ink is a good alternative for hiding the ink-dot sequence. In the past we have successfully performed experiments using invisible ink in combination with an ultraviolet camera (see Liwicki et al. (2010a)).

It is possible to encode arbitrary information as an ink-dot sequence by changing the number, the timing, and the shape of the ink-dots, as shown in Fig. 1 (a). Very roughly speaking, this coding scheme is similar to Morse code, where short and long segments and a pause are used and arbitrary information is represented as their sequence. Our coding scheme is designed to be more robust and error-tolerant, as described in the later sections.

The ink-dot shape can be controlled by using the high-frequency injection mode of the nozzle. The nozzle is able to generate up to $2,000$ ink-dots per second. Under this high-frequency mode, the ink-dots on the paper are connected and form a line segment. Hereafter, a line produced by n sequential ink-dots is called *n-pulse line*. If $n = 1$, the n-pulse line forms a single ink-dot. The line becomes longer by increasing n.

Note that due to hardware-specific issues the hardware of the actual data-embedding pen differs from the original setup proposed in Uchida et al. (2006). One crucial aspect is that just one nozzle element is used, since it was not possible for us to integrate more than one nozzle element into a practicable device. However, it would be possible to do so by designing specific nozzle elements in cooperation with printer companies. Thus the contribution of this paper can be seen as proving that the data-embedding pen can be realized and therefore motivating printer companies to develop a hardware which would be able to produce smaller dots and including more than one nozzle element and thus enabling the pen to be used with smaller handwriting and to embed even more information.

3.2. Applications

Various kinds of information can be embedded into handwriting by the data-embedding pen. This means that we can consider various applications of the data-embedding pen. In this section, several possible applications will be shown. All applications make use of the fact that any data encoded in a binary sequence can be added alongside with the handwritten pattern. Depending on the length of the pattern, the amount of information varies. As shown later by the experimental results, the current prototype can embed arbitrary 32-bit information into a 5cm-length handwritten pattern, for example.

Embedding information relating to the handwritten pattern itself is the simplest application. For example, writer's ID, writing date, and writing Geo-location, are possible candidates. This "meta"-information of the handwritten pattern can be useful for enhancing signature verification and for usage in other forensic applications. In addition, if we know the writer's ID, the recognition of the handwriting will become easier because we can apply some character recognition model tuned to the writer. Discrimination of multiple writers on a single document is also possible, if the pen embeds the corresponding writer ID. More details of this application idea as well as the idea of embedding a “handwritten” bar-code linking link between physical paper world and the cyber-space are presented in Uchida et al. (2006).

Embedding information on paper opens up new possibilities for diaries and notebooks. The owner of the book has only the paper documents at hand. However, still he or she can always find out when and where the information has been written down. Similarly, one can use the pen for writing an account of one's journey or a diary. If the pen is equipped with a GPS-receiver, the time and place will be automatically attached to the handwritten sentences. After scanning the handwritten pages, the information can be uploaded as a blog or as contributions to a Web 2.0 platform.

If we embed any temporal information by an ink-dot sequence, it is possible to relax the difficulty of the stroke recovery problem (Doermann and Rosenfeld, 1995; Kato and Yasuhara, 2000; Nel et al., 2005), which is an inverse problem to estimate the original writing order of a handwritten stroke pattern. This implies that we can convert handwritten images into online patterns and thus apply online handwriting recognition (Plamondon and Srihari, 2000; Vinciarelli, 2002), which is generally more accurate than offline recognition. Note that in this paper the writing direction, i.e., a kind of temporal information, is already embedded into the handwritten pattern for reliable data-retrieval.

Note that for enabling a pen for such applications it needs to be equipped with the application-specific hardware. While the realization of such hardware is out of the scope of this paper, we will present some ideas for realization here. As mentioned above, the pen could be equipped with a small GPS-receiver and an internal clock to deliver time and place information for embedding. For writer authentification a small fingerprint sensor can be attached to the thumb-side of the pen. Furthermore, small motion sensors and a camera can be equipped for online recognition, document retrieval, and temporal information embedding. The feasability of equipping a pen with such sensors has been proven by the development of specific Anoto pens, such as the livescribe[3]. This pen also includes a small display as a user interface where in the case of the data-embedding pen the user could easily select the kind of data to be embedded.

4. Data-Embedding

4.1. Basic Coding Scheme

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). These numbers have been fixed after a set of initial experiments with various n-pulse lines. The ink-dot sequence of Fig. 1 consists of those n-pulse lines.

The information is converted into a binary (0 and 1) sequence 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). The 20-pulse line, hereafter called *synchronization blob*, is used as an anchor to make sure that a correct position is extracted. The leftmost ink-dot in Fig. 1 is a synchronization blob.

Note that converted binary sequence is not directly embedded into the handwriting. We also apply an error-tolerant coding scheme for the original information after deriving the binary sequence. For example, the data "10" is not just converted into "1010"; after this conversion, the data is further converted into a redundant codeword to be more robust to error. More details will be given in Section 6.

Using the n-pulse lines, three units, called *frame*, *block*, and *bit* are formed. 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. A pause which is longer than the pause between bits is inserted between two consecutive blocks. Each frame begins and ends at a synchronization blob.

Figure 1 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.

The main parameters of the coding scheme are the number of bits per block (bB) and the number of blocks per frame (bF). Accordingly, the number of bits per frame becomes $bF \times bB$. In the example of Fig. 1, $bF = 6$ and

[3]http://www.livescribe.com

$bB = 4$. Another important parameter is the method for correcting errors which eventually occur when ink-dots overlap. More details about this issue follow in Section 6.

4.2. Embedding Dynamic Information

One of the difficulties in realizing the data-embedding pen is the stroke recovery problem. Specifically, since ink-dots are produced and embedded along the black-ink stroke, the writing order of the stroke has to be estimated for retrieving the embedded data. This is the so-called stroke recovery problem (Doermann and Rosenfeld, 1995; Kato and Yasuhara, 2000; Nel et al., 2005) and a well-known difficult inverse problem. For example, no one can always give the correct writing order of a horizontal line "—"; it may be left-to-right, but also may be right-to-left. For handwritten patterns with crossing parts, their writing order becomes more difficult to be estimated.

Fortunately, the ink-dot sequence can be used for relaxing the difficulty of the stroke recovery problem. The idea is to embed the writing direction by controlling the pause (gap) between consecutive n-pulse lines. This embedding can simply be realized by an additional pause added at the end of each frame, or equivalently, before each synchronization blob. Figure 4 in Section 5.4 shows this idea. Finally, the gap size shows the writing direction, i.e., the smaller gap has been produced after the synchronization blob.

5. Data-Retrieval

In this section the main steps for retrieving the information will be summarized. After capturing an image of the data-embedded pattern on paper, we first apply image processing techniques for extracting the ink-dots as well as the black-ink stroke. Then the stroke order recovery is performed to re-align the ink-dots in their original order Kato and Yasuhara (2000). Finally, the embedded data is retrieved through Reed-Solomon based error correction scheme.

We can either use a scanner or a digital camera for image capturing. Camera-based capturing will introduce 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 Section 5.1, the image processing techniques for assessing these problems will be explained.

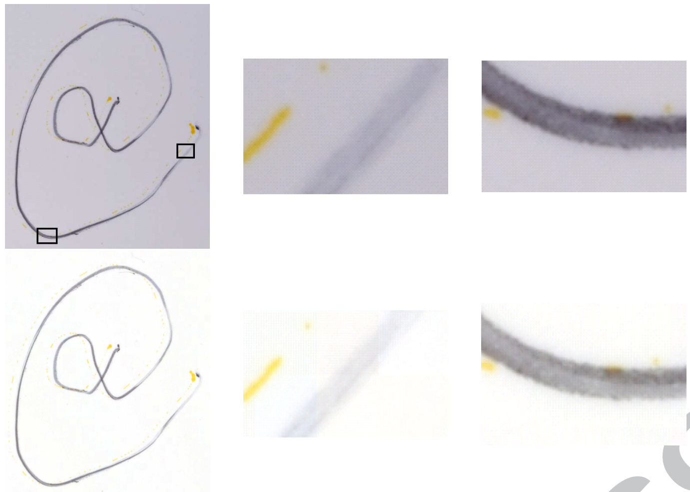


Figure 2: Example of a photographed "@" symbol before image processing (top) and after white normalization (bottom).

Note that the same procedures can also be applied on scanned documents. However, there they would have only minor effects which are negligible.

5.1. Image Processing

The results of the individual image processing steps for camera captured images are depicted in Fig. 2. There, an example of a photographed "@" symbol is given on the top. Next to it, two example regions which are part of the "@" symbol are illustrated to visualize the behavior of the algorithm.[4]

As stated above, we can not assume the same illumination for all the photographs. The situation is even worse when we have inhomogeneous illumination, because usually, the center is more exposed by the flashlight. Simply using a global threshold for color extraction would fail under these circumstances. Therefore, as a first processing step, we use a low-resolution grid motivated from work in related areas (Jain, 1989; Simon et al., 2000) and determine the brightest point in each sub-region. This is then used as

[4]The authors of this paper are aware that the yellow ink-dots in 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., Fig. 3 (d). Finally, in the electronic version of the publication one can see the yellow ink.

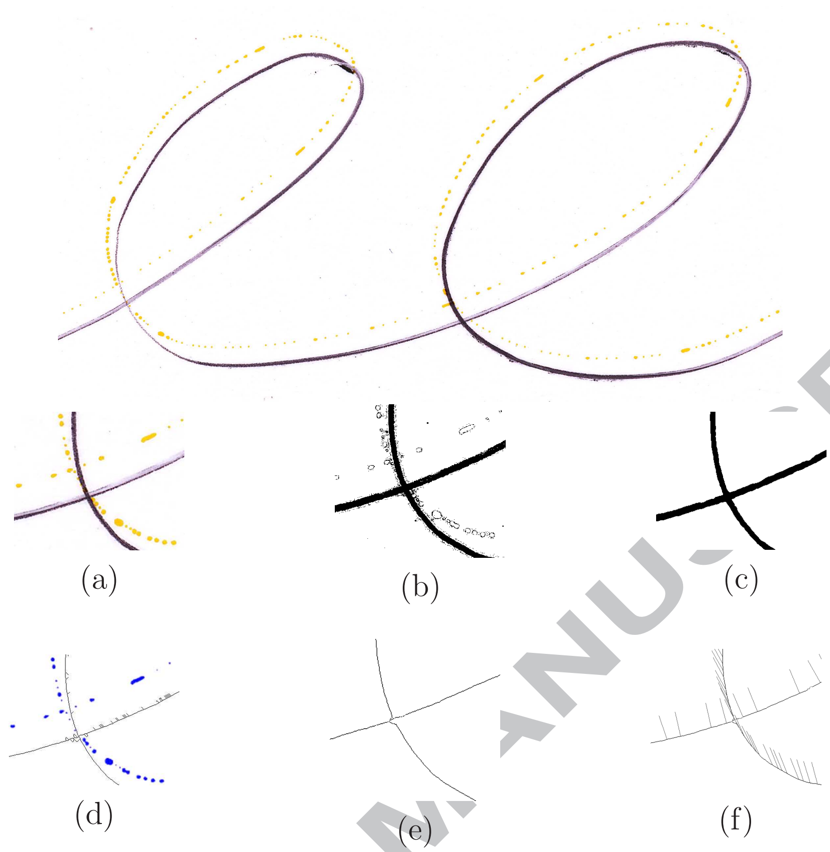


Figure 3: Two loops with ink-dots and vizualization of the image processing steps on an intersection part of the loops. See text for details.

a reference for white in this sub-region and the color values are normalized according to this value. This process is called *white normalization*. The result of white normalization in Fig. 2 (top) can be seen in Fig. 2 (bottom). The size of the sub-region was set to 50×50 pixels to make sure that at least one target white pixel is occurring in this region. Note that this method also works on homogeneous background colors. We have performed small tests which proved that it also works on *blue* and even *yellow* background. Experiments on non homogeneous background might fail, however, if invisible ink was used, the method would be successful again.

5.2. Ink Extraction

The second image processing step is the ink-dot extraction by a simple thresholding operation to identify the black ink stroke and yellow ink-dots. Subsequently, noise removal is performed, because the extracted black ink stroke image includes many noisy pixels, as shown in Fig. 3 (b). Thus, erosion and dilation are applied. Figure 3 (c) shows the result. Similar operations are also applied to the ink-dot image (Fig. 3 (d)). Note that the parameters for those operations can be optimized on a small training set.

The third step is a special treatment of ink-dots occluded by the black ink stroke. Fortunately, those yellow ink-dots are still visible on the stroke, they just appear to be a bit darker. Thus, after extracting the pixels of the black ink stroke, another thresholding operation is performed on those pixels with a lower threshold to recover dark yellow ink-dots. In the following experiments it turned out that about 50% of those ink-dots could be recovered by this approach.

The fourth step is a thinning operation on the black ink stroke. Figure 3 (d) shows the result of an orthodox thinning method. Then, after removing many small loops and short spurious edges by unifying neighboring branches, the final thinning result is obtained as shown in Fig. 3 (e). Up to here, the image processing methods are state-of-the-art methods (Nguyen and Blumenstein, 2010).

5.3. Information Ink Assignment

In order to make use of the embedded information, the n-pulse lines are assigned to the corresponding strokes[5]. The algorithm for finding these correspondence is described in this section.

As shown in Fig. 3 (f), 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 an ink-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

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

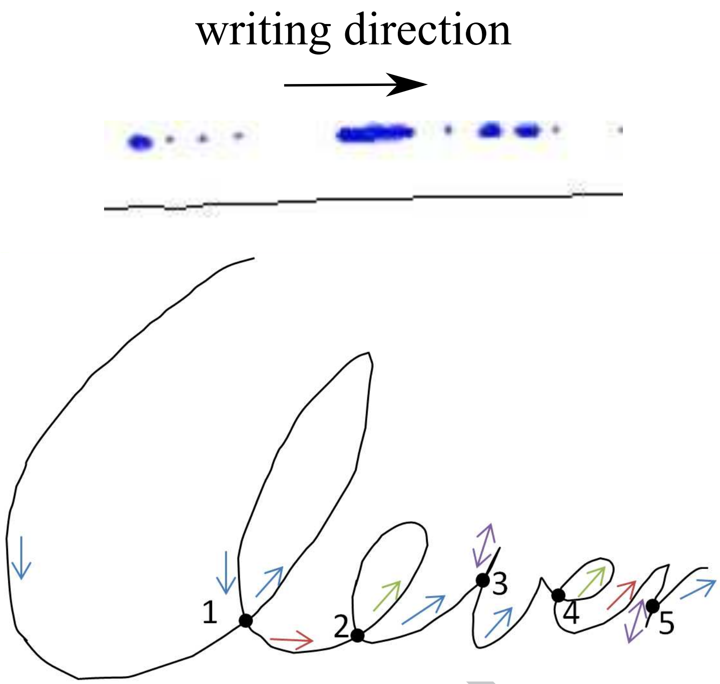


Figure 4: An additional pause is used to determine the initial writing direction (top), this can be observed by the larger gap on the left side of the big synchronization blob. Illustration of the stroke direction recovery process (bottom).

and tilt during writing. Finally, for each ink-dot k, the corresponding point is determined as the closest point when using angle $\bar{\theta}$. 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.

5.4. Stroke Recovery

Having the correspondence between the information ink and the stroke at hand, an estimation of the writing directions of the strokes is performed. In the following, an entire algorithm of the stroke recovery is explained using Fig. 4 as an example. This handwriting pattern can be seen as a graph where each crossing point on the stroke is a node. The algorithm is comprised of three major steps.

First, the direction is estimated by using the gaps around the synchronization blobs. As noted in Section 4.2, the stroke direction is embedded as the difference in the widths of the gaps. Thus for the edges including one or more synchronization blobs it is possible to estimate the corresponding writing direction. The blue arrows in Fig. 4 indicate the initial estimation of the directions. Some short lines may have no synchronization blob and thus no guess is assigned on these edges.

Second, we estimate the direction of the remaining edges. As an example, we consider the edge between Node 1 and Node 2, where no estimation result is given by the first step. The direction of this edge is determined as the outbound direction of Node 1. This is because among the other three edges of Node 1, two have an inbound direction and one has an outbound direction. Since the number of the inbound edges should be equal to the number of the outbound edges, the direction of the edge between Nodes 1 and 2 has to be outbound as well. This is furthermore confirmed by the situation at Node 2. By propagating the detected directions, many directions of unassigned edges can be determined using this strategy. The red arrows in Fig. 4 show the recovered directions after applying the second direction estimation step.

Third, since the direction of edges of looping strokes and double traced edges still cannot be determined by using the first two steps, some special operations are performed on the remaining edges. Double traced edges are detected by counting the degree of the two nodes of the edge. If one or both nodes have an odd-number degree (Node 5 in Fig. 4) and the direction is unknown, the edge is treated as a double traced edges. All the edges identified as double traced are then duplicated. By this duplication, all nodes (except for the ending nodes) have an even-number degree. The duplicated lines are indicated by purple double-arrows in Fig. 4.

The direction of loop edges are determined by a simple strategy called *basic trace algorithm*(BTA) (Kato and Yasuhara, 2000). According to BTA, when we arrive at an intersection node (i.e., a node with degree 4), we will take the center path. For example, when coming from Node 1 to Node 2 in Fig. 4 we take the center path and go to the top-right direction. The green arrows in Fig. 4 represent the directions finally detected.

As a result of these steps we obtain the direction information of all edges. Now we start to trace the lines at the top-left edge with an end-node of degree 1 and an outbound direction. When arriving at a node of degree > 3 during tracing, we go into the direction of a loop and also consider the BTA in tied situations. After reaching an end, the next untraced edge with an end-node of degree 1 is taken into account, etc.

5.5. Data Decoding

Since we have a sequence of the embedded ink-dots by the above processes, we now decode the sequence to retrieve the embedded information. The decoding process starts with the recovery of the bit information, i.e., 1-pulse and 5-pulse lines and synchronization blobs, of every ink-dot, just by checking its size. The sequence is separated into frames using the synchronization blobs. Larger gaps are detected within each frame and assumed as the gaps between block.

Next, a plausibility control is performed on the extracted data. For each block, the number of bits (bB) is confirmed. Sometimes a block has spurious bits, resulting from a wrong mapping or just from noise. In this case, those adjacent bits whose distance deviates too much from the mean distance are deleted. If the number of bits and blocks do not correspond to the values bB and bF, the frame is rejected.

For detecting and correcting errors, the Reed-Solomon error correction is chosen. Details follow in the next section.

6. Error Correction

The process of embedding the ink next to the handwritten stroke is always accompanied with several errors. First, the black ink sometimes overlaps with the information ink (Fig 5 (b)). Second, several ink-dots might overlap at turning points or stopping points (Fig 5 (c)). Finally, the consecutive ink-dots on double traced strokes (Fig 5 (d)), are missed because we discarded them.

In order to recover from the errors, some redundant information has to be added. Simple and intuitive ideas would be to apply repetition and parity check (Liwicki et al., 2010a). However, these encodings show some limitations, especially when it comes to more complicated handwritten patterns like signatures or handwritten words with many crossings and double strokes.

In this paper we use Reed-Solomon error correction (MacWilliams and Sloane, 1977; Reed and Solomon, 1960) for reliably recovering from the occurring errors. The idea is to oversample a polynomial $f(x) = a_1 + a_2 \cdot x^1 + \ldots + a_k \cdot x^{(k-1)}$ from the data with more *points* a_j than needed. This makes the polynomial overdetermined. Therefore it is not needed to recover all points correctly as long as enough points are present. The only drawback of this encoding is that the position j of each point a_j needs to be known for a reliable decoding. In this paper we design each frame to be comprised of two

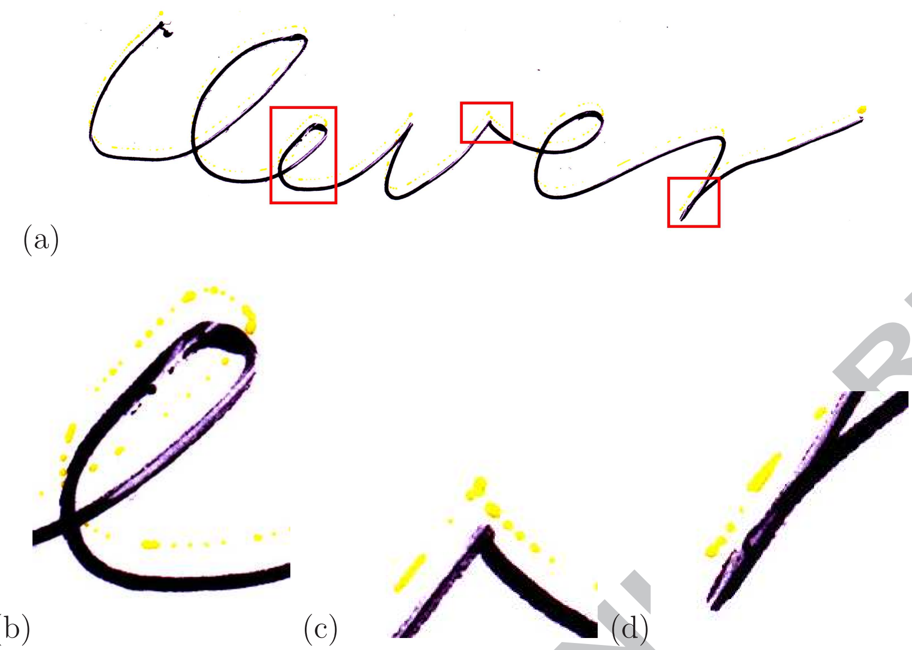


Figure 5: (a) Example of a difficult pattern. (b)–(d) Specific regions of the pattern.

blocks, the first block for the position of the point and the second block for its value. While the details of Reed-Solomon codes can be found in Reed and Solomon (1960), in this paper only the important parameters and properties of this encoding scheme are given.

The first parameter of Reed-Solomon encoding is the base m bits for the points. In this paper we have set $m = 4$. This choice of this value is based on the observation that previous experiments have shown that shorter frames have a higher probability of being correctly decoded. Each frame consists of 8 bit; 4 bit for the position and 4 bit for the value.

The next parameter is the length n of the code (including data and error correction bits). Typically this value is set to its maximum value $n = 2^m - 1$ (the values have to be non-zero). This code is divided into k data points (the data to be encoded) and $n - k$ points for error correction. Given the k data points $a_1, \dots, a_k$ (a message to be encoded), the other values $a_{(k+1)}, \dots, a_n$ of the polynomial are determined and all n points are encoded (sent).[6] If the

[6] The determination of the values is based on the primitive element of the finite field

handwritten patterns are long enough, the code is repeatedly embedded.

In the decoding phase, not all n points need to be correctly recovered. Assuming that c points were correctly recovered, s points are missing (erasures) and e points are erroneous, the code can still be correctly decoded if the following equation holds:

$$2e + s \leq n - k \tag{1}$$

This important property makes the Reed-Solomon codes very useful for applications where burst errors occur. In our case usually the a whole block can be either recognized or not, i.e., it rarely occurs that just one bit is missing (even if only one bit is missing, we do not know the position of the bit).

Since we encode the positions of the points in the frame, the positions of the missing points are known before decoding. In the extreme case, up to $n - k$ points can be missing and still it would be possible to decode the information correctly. In the other extreme case, i.e., if there is no missing point, up to $(n - k)/2$ errors are allowed to occur, which means for each erroneous point, one more correct point should be at hand.

7. Experiments and Results

The aim of our experiments is to prove that the concept of the data-embedding pen can be applied to real data and works on patterns with various complexity. Therefore, after optimizing the system on a few patterns written by one writer and testing them on an independent set of patterns contributed by the same writer, we asked second, independent writer to write down several patterns. This makes sure that there is no bias between the system and the handwriting of one writer.

7.1. Data

Three sets of data-embedded handwriting were collected using the current pen prototype. The first set (Set1) contains 50 horizontal straight lines with a length of 20 cm. All lines have been drawn with approximately the same velocity, i.e., the usual writing speed. Experiments with varying velocity appear in Liwicki et al. (2010a). The second set (Set2) contains patterns

---

α and finding a function $f(x)$ for which holds $f(\alpha^{(i-1)}) = a_i$, for $i = 1, \ldots, k$ and then applying $f(x)$ to the remaining $\alpha^i, i = k, \ldots, n-1$.

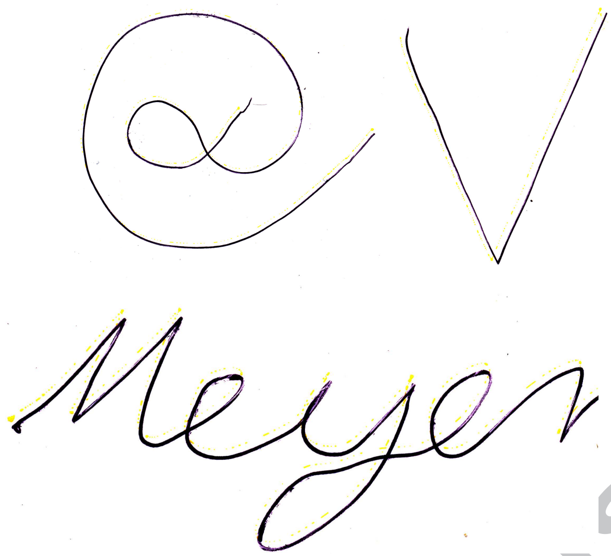


Figure 6: Example images of Set2.

which might appear in a real world scenario, i.e., 12 "@" symbols, 12 checkmarks, 12 simulated signatures, and 12 instances of the handwritten word "Clever". The former two symbols have sizes of 3×3 cm at maximum, the latter symbols have a size of 4×3 cm. Examples for these patterns are shown in Figs. 5 and 6.

The third set contains significantly more data contributed by the second writer. This writer contributed with 80–120 samples for each of the following patterns (the sizes mentiond in brackets correspond the height of the respective pattern):

- straight lines (5cm) in all four directions (right, left, up, down)
- "@" symbols (sizes: 3 cm and 5 cm)
- checkmarks (sizes: 3 cm and 5 cm)
- the word "Clever" (height 4 cm)
- closed circles (diameters: 5 cm and 3 cm)
- "X" (sizes: 3 cm and 5 cm)
- Cursive "ll" (sizes: 3 cm and 5 cm)

All examples have been written by the second person who was not aware of the processing methods. Note, however, that the writer was asked to write the patterns quite fast (using usual writing speed), because there has to be some distance between consecutive ink-dots in order to make them distinguishable.

We intentionally asked the writer to write lines in both orientations in horizontal and vertical direction (resulting in four categories), as well as circles and "X"-shapes, because it is not possible to perfectly recover the trajectory information for these patterns if no information ink is available. Note, that for these patterns the original stroke-recovery method of Kato and Yasuhara (2000) would fail.

7.2. Reed-Solomon Encoding

For the Reed-Solomon encoding, the Shifra Open Source error correcting code library was used[7]. We used a Galois field polynomial of the order 4. The code length was fixed to 15 points ($2^4 - 1$), each point being a hexadecimal number (4 bit).

One aspect of our experiments is to estimate a useful value for the parameter k, i.e., the number of data points, as introduced in Section 6. We have varied k from 1 to 15. In order to overcome needless calculations, we applied the following strategy. First, we set $k = 1$ with $a_1 = 1$ and computed the other values a_i for this setting. The resulting code word is $1, 9, 13, 15, 14, 7, 10, 5, 11, 12, 6, 3, 8, 4, 2$. If we now set $k = 2$ and $a_2 = 9$, the same values for $a_j | j > 2$ would be estimated, and so on. This means that only the encoding of this code is needed and during decoding we can choose the actual value for k. This makes the full use of all collected data, i.e., it is not needed to write down new patterns for each value of k. Note that using this strategy also eliminates side-effects like more noise in some patterns, because always the same patterns are used for the evaluation.

In the experiments on Set1 we wanted to find out how much information can be embedded in a straight line. In this experiments only very few decoding errors occur on the frame level since there are no crossings. Figure 7 provides an example for the extraction result of a 5 cm long part of a straight line where no errors occurred. The only problem were some overlapping ink-dots due to a slow pen-movement. This happened in about 10 % of the

[7] Available at `http://www.schifra.com/` — last visited: November 2011

Table 1: Correctly recovered patterns for Set1 (varying line lengths) and Set2 in %

# data points (k)	# bits	5 cm	10 cm	hook	@	Meyer	Clever
1	4	100	100	100	100	100	100
8	32	100	100	100	100	100	100
9	36	94	100	83	100	75	100
10	40	72	100	75	92	50	100
11	44	56	100	58	83	33	100
12	48	20	100	50	50	0	83
13	52	0	100	17	8	0	42
14	56	0	100	17	0	0	0
15	60	0	92	0	0	0	0

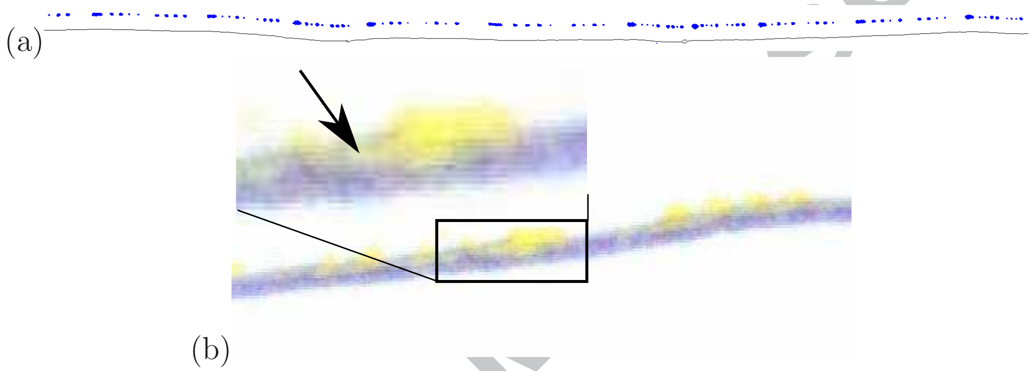


Figure 7: (a)Extraction result (after thinning) of a 5 cm long line of Set1 (enlarged). (b) A problematic case where the information ink is smudged by the ballpoint pen.

frames. Note that these frames were rejected during the frame decoding step presented in Section 5.5, resulting in missing points for the Reed-Solomon error correction.

7.3. Results for Set1

As stated above, the straight lines had a length of 20 cm. Since the code was repeated, no errors occurred on these long lines. We decided to measure the results on shorter lines. Therefore we cut the line first into 10 cm parts and finally into 5 cm parts.

The results of the experiments on Set1 appear in Table 1. This table shows the percentage of samples where the information could be correctly recovered by using the Reed-Solomon error correction. Up to a number of $k = 8$ the codeword was always correctly recovered even for straight lines as short as 5 cm. For larger k value, the performance decreases, because only a limited number of frames appear in a 5 cm line. (In Fig. 7 (a), for example, 10 points (frames) appear.) For the length 10 cm there were only problems if no error correction point appears, i.e., it occured 8 times that there was a missing point which could not be recovered.

In order to investigate the influence of using Reed-Solomon error correction we have performed experiments without using this correction scheme. Alternatively, parity bits were used for each frame (Liwicki et al., 2010a). The advantage of using parity bits is that more data could be embedded into each frame; the disadvantage, however, is that it is harder to recover from burst errors. For both line lengths the recovery rate of the parity-bit method was only half of the Reed-Solomon based error correction (Liwicki et al., 2010b). Therefore we conclude that using Reed-Solomon encoding is the superior strategy.

7.4. Results for Set2

Table 1 presents the results of the experiments on Set2 on the right columns. On all patterns, codes of length 32 could be correctly recovered. It is a very interesting result that even on the more complicated patterns the correct information could be decoded. The main reasons for unsuccessful decoding are either missing points (for short sequences like the hook) or some errors, e.g., a 1-bit was interpreted as a 0-bit if it was partially occluded by black ink (first frame of Fig 5 (b)).

7.5. Results for Set3

In our experiments on Set3 we first evaluated the performance of the stroke recovery methods. A method without taking advantage of the stroke direction embedded into the ink-dot sequence, i.e., the approach of Kato and Yasuhara (2000), has been used as a reference system. The accuracy is defined as the number of edges with a correctly identified direction divided through the number of all edges.

The results of the trajectory recovery appear in Table 2. As can be seen, the algorithm of Kato and Yasuhara (2000), denoted as "System 1", performs already good on many patterns. However, it has some complications with closed circles, lines which go against the more common direction, and two-stroke patterns.

Using the embedded information significantly increases the performance. Our method, denoted as "System 2", works perfect on most patterns. Only

Table 2: Performance comparison on Set3

System	circles		lines			
	3 cm	5 cm	down	right	up	left
Direction detection accuracy in %						
1. Kato and Yasuhara (2000)	—	—	100	100	0	0
2. Proposed	100	100	100	100	100	100
3. With post-processing	100	100	100	100	100	100
Number of embedded bits if 100% information recovery rate is desired						
3.	40	40	32	32	32	32

System	"x"		"ll"		hooks		clever
	3 cm	5 cm	3 cm	5 cm	3 cm	5 cm	
Direction detection accuracy in %							
1.	50	50	100	100	100	100	80
2.	79	98	45	100	100	100	95
3.	79	98	100	100	100	100	96
Number of embedded bits if 100% information recovery rate is desired							
3.	16	36	20	44	12	32	40

small patterns introduce some complications. An idea for post-processing is to apply the method of Kato and Yasuhara (2000) if only a single edge is available and no direction could be determined. The row indicated with "System 1" shows the performance if this strategy is applied. The final method performs with 100 % on 10 out of 13 patterns.

The retrieval results for Set3 are shown in the last row of Table 2. This row shows the maximum amount of encoded bits if a perfect retrieval of all embedded information is desired. Note that the value for large patterns (size 5cm) is at least 32. This result is similar to the results obtained on Set2.

7.6. Failure Analysis

An analysis of the failures shows that during acquisition some of the ink-dots were overlapping the pen-stroke. Specifically, when the tip of the ball-point pen touches already existing information ink-dots, the ink is smudged by the pen. Our algorithm is not able to recover the correct information (Fig. 7 (b)). In future we will try to tackle this problem by improving the image processing technologies.

8. Conclusions and Ideas for Further Research

In this paper we have presented a successful realization of the data-embedding pen. This pen makes it possible to augment handwritten patterns with additional information like the time of writing, the writer ID, and other application-dependent data. The main idea is to encode the desired information in an ink-dot sequence plotted nearby the writing strokes. The hardware design as well as the methods for embedding and recovering information have been also described.

We proposed the use of the Reed-Solomon error correction scheme for successfully encoding and recovering the meta-information. The Reed-Solomon correction scheme uses an overdetermined polynom for encoding the data. During decoding only as many correct points are needed as the number of data points, disregarding their position. Other missing points do not damage the result. For each erroneous point one more correct points is needed to recover from the error.

In our experiments we have shown that the Reed-Solomon error correction scheme is very useful if applied as proposed in this paper. In a first set of experiments, using a stroke length of just 5cm, 32 bits of information could be successfully embedded and recovered from straight lines.

In the second set of experiments we have used more complex patterns, ranging from symbols to handwritten words. Even in this setup we could always recover 32-bit of information. Note that 32 bit is enough to distinguish 2^{32} people. This implies that if a company uses this tiny marks for showing that a certain employee has checked a document, it is possible to identify which employee has checked the document. Also note that small read/write RFID-cards usually allow to store the same amount of information (32 bit).

These results have been confirmed on patterns contributed by another writer in a third data set. On this set, we have furthermore shown, that the direction recovery was improved by using the properties of the information ink-dots. In most cases, the directions of all edges have been correctly identified. Only short edges in complicated patterns were sometimes not correctly recovered. However, this did not harm the information retrieval process as Reed-Solomon error correction has been applied.

An interesting property of the data-embedding is that the shape of the n-pulse lines could be used to retrieve even more information about the dynamics. The speed information, for example, can be derived from the blocks within the frames. Since a new n-pulse line starts every 10 ms, the

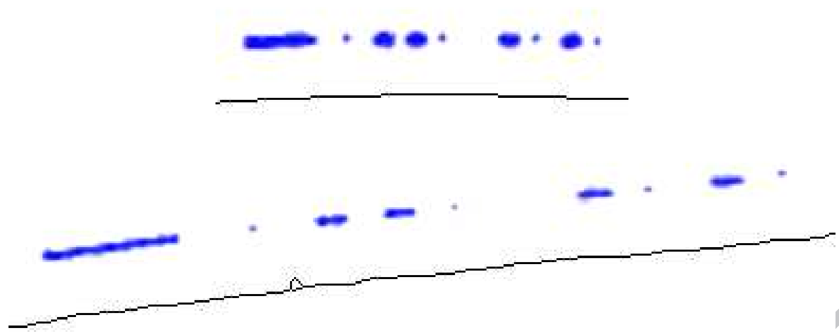


Figure 8: Different length of the n-pulse lines depending on the velocity. (top: low velocity, bottom: high velocity)

actual distances between the n-pulse lines encode the speed. Furthermore, longer n-pulse lines would correspond to faster writing (Fig. 8). Another idea is to estimate the tilt of the pen by using the correspondence information from the ink-dots to the line (see Section 5). Note that a shorter distance would indicate a smaller tilt angle if the nozzle is mounted on the pen as in Fig. 1 (top). Experimenting the accuracy of these algorithms is left to future work.

References

Doermann, D. S., Rosenfeld, A., 1995. Recovery of temporal information from static images of handwriting. International Journal of Computer Vision 15 (1-2), 143–164.

Hecht, D. L., 1994. Embedded data glyph technology for hardcopy digital documents. In: Color Imaging: Devide-Independent Color, Color Hardcopy and Graphic Arts III. Vol. 2171. pp. 341–352.

Jain, A., 1989. Fundamentals of Digital Image Processing. Prentice-Hall.

Kato, Y., Yasuhara, M., 2000. Recovery of drawing order from single-stroke handwriting images. IEEE Trans. Pat. Anal. Mach. Intell. 22 (9), 938–949.

Liwicki, M., Akira, Y., Uchida, S., Iwamura, M., Omachi, S., Kise, K., Sep. 2011. Reliable online stroke recovery from offline data with the data-embedding pen. Proc. 11th International Conference on Document Analysis and Recognition (ICDAR 2011), 1384–1388.

Liwicki, M., Uchida, S., Iwamura, M., Omachi, S., Kise, K., 2010a. Data-embedding pen — augmenting ink strokes with meta-information. In: 9th Int. Workshop on Document Analysis Systems.

Liwicki, M., Uchida, S., Iwamura, M., Omachi, S., Kise, K., 2010b. Embedding meta-information in handwriting — Reed-Solomon for reliable error correction. In: 12th International Conference on Frontiers in Handwriting Recognition. pp. 51–56.

MacWilliams, F. J., Sloane, N. J. A., 1977. The Theory of Error-Correcting Code. New York: North-Holland Publishing Company.

Nel, E.-M., du Preez, J. A., Herbst, B. M., 2005. Estimating the pen trajectories of static signatures using hidden markov models. IEEE Trans. Pat. Anal. Mach. Intell. 27 (11), 1733–1746.

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

Plamondon, R., Srihari, S. N., 2000. On-line and off-line handwriting recognition: a comprehensive survey. IEEE Trans. Pat. Anal. Mach. Intell. 22 (1), 63–84.

Reed, I. S., Solomon, G., 1960. Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics 8 (2), 300–304.

Simon, M., Behnke, S., Rojas, R., 2000. Robust real time color tracking. In: RoboCup-2000. pp. 239–248.

Uchida, S., Tanaka, K., Iwamura, M., Omachi, S., Kise, K., 2006. A Data-Embedding Pen. In: Tenth International Workshop on Frontiers in Handwriting Recognition.
URL http://hal.inria.fr/inria-00103878/en/

Vinciarelli, A., 2002. A survey on off-line cursive script recognition. Pattern Recognition 35 (7), 1433–1446.

Highlights

> A pen device for embedding meta-information in offline handwriting
> Reliable Stroke Recovery using SotA-methods and additional ink
> Error-tolerant data-decoding by using Reed-Solomon Codes
> 32 bits embedding is possible allowing for various applications