Data-Embedding Pen — Augmenting Ink Strokes with Meta-Information

Marcus Liwicki
Kyushu Univ. / DFKI
Fukuoka, Japan / Germany
marcus.liwicki@dfki.de

Seiichi Uchida
Kyushu Univ.
Fukuoka, Japan
uchida@ait.kyushu-u.ac.jp

Masakazu Iwamura
Osaka Prefecture Univ.
Osaka, Japan

Shinichiro Omachi
Tohoku Univ.
Miyagi, Japan

Koichi Kise
Osaka Prefecture Univ.
Osaka, Japan

ABSTRACT

In this paper we present the first operational version of the data-embedding pen. During writing a pattern, this pen produces an additional ink-dot sequence along the ink stroke of the pattern. The ink-dot sequence represents, for example, meta-information (such as the writer's name and the date of writing) and thus drastically increases the value of the handwriting on a physical paper. Since the information is placed on the paper, it can be extracted just by scanning or photographing the paper. There is no need to get access to any memory on the pen to recover the information. This is useful especially in multi-writer or multi-pen scenarios. The experiments using an encoding scheme and a decoding algorithm showed very promising results. For example, it was proved that we can embed 28 or more bits of information on simple handwritten patterns and decode them with a high reliability.

Categories and Subject Descriptors

I.7.5 [Document and Text Processing]: Document Capture; E.4 [Coding and Information Theory]: Error control codes; B.m [Hardware]: Miscellaneous—Design Management

General Terms

Design, Algorithms, Experimentation

1. INTRODUCTION

Handwriting is a popular modality for writing down information, making annotations, or just marking items. Unfortunately, as soon as the ink is on the paper, many important information is already lost. In fact, we cannot access meta-information about the handwritten pattern from itself; for example, 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 seem to be a possible choice to store and retrieve such meta-information — unfortunately, they cannot increase the value of handwriting on paper either. Nowadays, several digital pens to capture handwriting on normal paper have been developed and those pens can store the stroke sequences on a computer along with some meta-information, such as the writer ID. However, the handwriting on the paper is still just an ink pattern without any meta-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 the increased 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 world (paper) and the cyber-space (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.

Another property is the omission of preparing any special paper or sensing device before writing. There is no need to take care where to write down a note or comment. Thus this device does not interrupt the thinking process, especially when something important pops into ones mind. This fact is also supported by the feature that a final data-embedding pen will be immediately ready for use without long startup times. While the current prototype is comprised of several devices for controlling several parameters, those devices can be removed or down-sized for a final version.

2. RELATED WORK

To the authors' best knowledge, this is the first trial on implementing a new pen device which can embed arbitrary

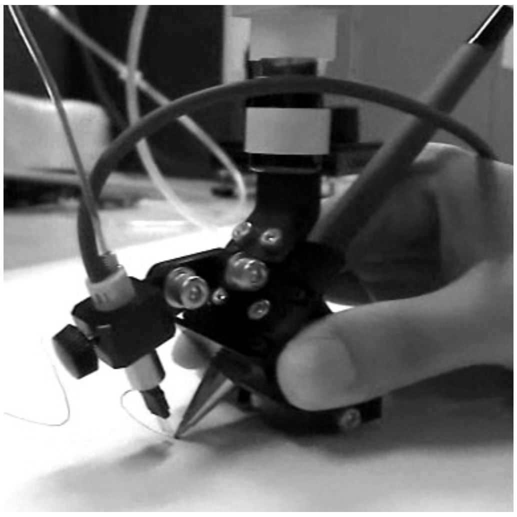
Figure 1: The first prototype of the data-embedding pen.

information dynamically into handwriting on a paper. Generally, data embedding into papers has been done statically by a printer. For example, XEROX DataGlyph [?] is a kind of digital watermarks and information is printed and embedded as a fine texture into font images (or photographs).

Nowadays, the most famous digital pen may be Anoto[1]. 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 patterns. Anoto and the data-embedding pen have very different purposes. The purpose of Anoto is to get the pen motion and to identify the absolute position on the paper. Thus, Anoto is considered as a kind of pen-tablet and the handwritten strokes on the paper have no additional value. In contrast, the purpose of the data-embedding pen is to enhance the value of the strokes on the paper.

The embedded ink-dots will also be helpful for the problem of online information recovery from the handwritten stroke image. There have been many attempts to solve the problem of so-called stroke recovery [?, ?, ?]. Since stroke recovery is a kind of inverse problem, it is an ill-posed problem with intrinsic difficulties. For example, no one can always give the correct online information of "X", which was written as "$\backslash \rightarrow /$" or "$/ \rightarrow \backslash$" or "$> \rightarrow <$." If we embed any temporal information (such as the writing direction) by an ink-dot sequence, it is possible to make the problem well-posed. This implies that we can convert handwritten images into online patterns and thus apply online handwriting recognition [?, ?], which is generally more accurate than offline recognition [?].

3. THE DATA-EMBEDDING PEN

3.1 General Idea

The data-embedding pen is a device which comprises a usual ball-point pen and an ink-jet nozzle element. Figure 1 shows a picture of this device. 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. (Invisible ink has already been tested as a good alternative for the future.) The number of the ink-dots and their timing are used to encode the desired information. An automatically processed version of the ink-dots in Fig. 1 is shown in Fig. 2 (a).

[1]http://www.anoto.com

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

In order to extract the information, the image is first scanned or photographed. Then document analysis algorithms are applied for recovering the online stroke representation (see Section 5.2). Next, the sequence information of the ink-dots is determined by using the online stroke representation. Finally the sequence is decoded (see Section 5.3).

The idea of the data-embedding pen has been presented previously in [?]. In [?], however, no functional device was at hand and thus only simulated data served for an experimental evaluation. In this paper, the first pen prototype is presented and experiments on real data are conducted. Furthermore, an error-redundant and error-correcting coding scheme is proposed and a novel set of document processing algorithms are introduced.

3.2 Hardware

As shown in Fig. 1, the data-embedding pen basically consists of a usual ball-point pen (right) and an inking nozzle element (left). In the current prototype, three devices are used to control the nozzle element and its ink tank. The first device is an amplifier for activating the piezoelectric jet inside the nozzle. It is connected with the nozzle element by a cable. The second is a vacuum pump for controlling the pressure of the ink; it is connected to the ink tank to avoid too high ink pressure (which would result in a blocking of the nozzle) and too low pressure (which would restrain the ink flow). The third is a programmable D/A converter for providing triggers to the amplifier for controlling the timing of ink-dots.

Although those devices make the prototype less practical, they are necessary during feasibility study for fixing several parameters and the coding scheme. In other words, they can be removed or down-sized after the fixation. Several ideas to improve the hardware are mentioned in Section 8.3.

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.

Although our past simulation experiment [?] assumed three colors for ink-dots, we now assume just a single color (yellow, in the following experiment) as noted above, due to the fact that the color that the current prototype has a single inking nozzle. Of course, multiple colors will increase the information embedding density. On the other hand side the use of multiple colors will make the pen much larger and thus be less feasible.

4. INFORMATION EMBEDDING

4.1 Units of Coding

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 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. The leftmost dot in Fig. 2 depicts such a synchronization blob.

Our coding scheme is defined by three units, called frame, block, and bit. (These namings are motivated by the terminology of network protocol design.) 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 (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.

4.2 Parameters

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. 2, $bF = 6$ and $bB = 4$. Since those parameters are important for embedding performance their influence will be discussed in the experiments section.

Another important parameter is the bit time (bT). It controls the time duration assigned for each bit. For example, if $bT = 20$, there would be 19 pulses without any inking after a 1-pulse line and 15 pulses without any inking after a 5-pulse line. Having a constant bit time increases the probability that the pulse lines within a block have equal distance to each other, which is beneficial for the information recovery. After each block we make a pause of length bT and before the start of a synchronization pulse line, we make another pause of length bT. This results in a larger gap at the end of a frame, making it possible to recover the writing direction (see Section 8.1).

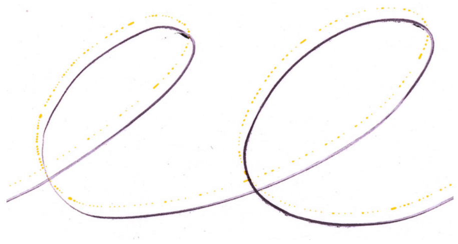

Figure 3: Two loops with ink-dots.

4.3 Error Correction

Error correction must be considered in the coding scheme because the ink-dots are not always extracted correctly. In fact, there are many problems in the extraction. For example, sharply curved strokes, stroke intersections, writing speed fluctuation, and noise on the paper surface often disturb the extraction. Consequently, we must assume missed bits, merged bits, and spurious bits.

Two popular error correction methods are introduced in this paper. The first method is a repetitive code and the second is a parity check. The repetitive code is very simple; the same bit sequence (often comprised of several frames) are embedded repeatedly. We can detect and correct errors by comparing frames, which should contain the same bit sequence. For making this comparison easier, the first two bits of a frame is used for showing an address of the frame. In the frame of Fig. 2, the first two bits (01) represent the address $0 * 2^0 + 1 * 2^1 = 2$.

For the error correction by the parity check, the last bits of each frame are parities. In the frame of Fig. 2, the last 8 bits are considered as the parities for detecting and correcting the errors of the first 16 blocks. The parity check is done by a matrix; the first 4 blocks are aligned as a 4×4 matrix (each row corresponds to a block) and then the 5th block is used as parity bits for each column and the 6th block is used similarly for each row. Note that by introducing the frame address (2 bits) and the parity check (2 blocks $\times 4$ bits), the amount of encodeable data by a frame (24 bits) becomes 14 bits.

5. INFORMATION RECOVERY

5.1 Image Processing

Information recovery begins with image processing which extracts ink-dots and black ink strokes from a scanned image. In the this section, four steps of image processing are explained using Fig. 4 (a), which is an intersection part of Figure 3 ("two-loops").

The first step of ink-dot extraction is a simple thresholding operation to extract the black ink stroke and yellow ink-dots. The second step is noise removal because the black ink stroke image extracted includes many noisy pixels, as shown in Fig. 4 (b). Thus, erosion and dilation are applied. Figure 4 (c) shows the result. Similar operations are also applied to the ink-dot image (Fig. 4 (d)). Note that the parameters for those operations can be optimized on a small training set.

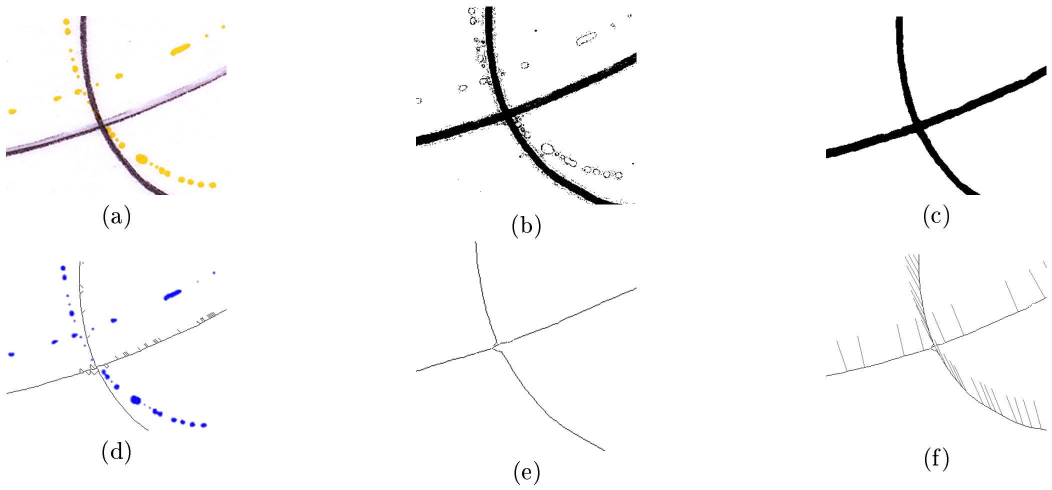

Figure 4: Image processing on an intersection part of Fig. 3. See text for details.

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 dots could be recovered by this approach.

The fourth step is a thinning operation on the black ink stroke. Figure 4 (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. 4 (e).

5.2 Aligning Ink-Dots by Stroke Recovery

In order to decode the ink-dots, they should be aligned according to their original temporal order. Since this order is lost in the scanned image, we must estimate it by using the result of stroke recovery. Specifically speaking, after recovering the writing order of the black ink stroke based on the algorithm presented in [?] and establishing the correspondence between the ink-dots and the stroke, we align the ink-dots.

The basic idea of establishing the correspondence as shown in Fig. 4 (f) 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 directions (with 10° interval). Then, we select the direction $\overline{\theta}$ with minimum variance, i.e., $\overline{\theta} = \mathrm{argmin}_{\theta} \operatorname{Var}\{d_{1,\theta}, \ldots, d_{K,\theta}\}$. This direction is the most stable direction and thus represents the pen tilt. Finally, for each ink-dot k, the corresponding point is determined as the closest point in the direction $\overline{\theta}$.

5.3 Data Decoding

For decoding, the bit information (i.e., 1-pulse and 5-pulse lines and synchronization blob) is first recovered at 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 replacement errors (i.e., $1 \to 0$ or $0 \to 1$), the parity check is performed within each frame. If there is a failure in only one parity bit, it can be ignored as the error of the parity bit. If there are two failures, one in a row and one in a column, the corresponding data bit is negated. In any other case the frame is rejected because of uncertainty. Note that since we employ a repetitive code, the information of a rejected frame can be recovered by a frame with the same address at another repetition.

6. EXPERIMENTS

6.1 Parameter Optimization

6.1.1 Data

Two sets of data-embedded handwriting were collected, using the current pen prototype. The first set (Set1) contains horizontal straight lines. The second set (Set2) contains loop curves. (See Fig. 3 as an example). Each sample was written on an A4 sheet. The widths of lines and loop curves were about 20cm. (The size of each loop was less than 3cm × 3cm.) On each sheet, 14 straight lines or 7 loop curves were written. To investigate the influence of the velocity, the first part of each sheet has been drawn with a speed lower than normal writing speed (about 3 seconds for 20 cm). The succeeding part has been drawn with a normal writing speed (about 1.5 seconds for 20 cm), and the last part has been drawn with a higher writing speed (less than a second for 20 cm). All data has been written by the same writer in order to make the results comparable.

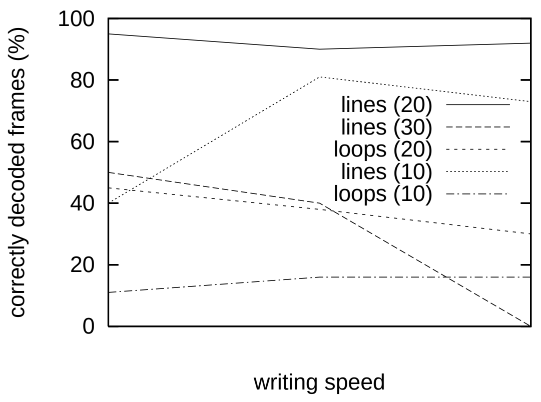

Figure 5: Influence of writing speed and bit time on accuracy at $bB = 4$ and $bF = 6$. (The number in parenthesis denotes the value of bT).

For each data set, 6 paper sheets were prepared. Each sheet contains the data-embedded handwriting produced with one of six different parameter settings: $(bB, bF, bT) = (4, 6, 10)$, $(4, 6, 20)$, $(4, 6, 30)$, $(5, 3, 20)$, $(12, 2, 20)$, $(7, 5, 20)$. Consequently, Set1 contains 6×14 straight lines and Set2 contains 6×7 loop curves.

A sequence of four frames was the unit representing the entire information to be embedded and the unit was repeated along the black ink stroke. In other words, we used the repetitive code for error correction. The dot sequence for each frame was as follows:

- 1st frame: $1, 1, 1, 1, 1, 1, \ldots$
- 2nd frame: $0, 0, 0, 0, 0, 0, \ldots$
- 3rd frame: $1, 0, 1, 0, 1, 0, \ldots$
- 4th frame: $1, 1, 0, 0, 1, 1, \ldots$

Note that the length of the was changed by bB and bF. In all cases two bits for the frame address were added in front and the last bits were reserved for the parity information as stated in Section 4.3. For example, if $bB = 4$ and $bF = 6$, the dot sequence of the first frame being comprised of (i) two bits of the address $(0, 0)$, (ii) 14 bits of 1, and (iii) 8 bits of parity.

All the handwriting was written on an ordinary white paper. The paper was scanned by a flat-bed scanner with the resolution of $1,200$ dpi. The average size of an ink dot (1-pulse line) was 0.1 mm.

Table 1 shows a summary of the coding styles used in our experiments, with the corresponding matrix dimension for the parity check, the bits needed for the parity check, and the number of information bits (i,e., bits for representing the information to be embedded).

6.1.2 *Influence of Writing Speed and Bit Time*

The influence of writing speed and bit time bT was investigated with the parameters fixed at $bB = 4$ and $bF = 6$. We tested the recovery algorithm on both sets mentioned above, using all three writing speeds and three different values of the bit time, i.e., $bT = 10, 20, 30$.

The results appear in Fig. 5, where the percentage of correctly decoded frames is given in the y-axis, corresponding to the writing speed (slow, medium, fast) given in the x-axis. As expected, better results have been achieved on Set1, where only straight lines were written. For the more difficult set, the loop curves, only about half of the frames were correctly decoded.

It is very important to note that all erroneous frames have been rejected successfully by the error detection procedures. In other words, no false decoding result occurred. This makes the system very reliable, because the code is repeated every four frames. We observed that within the span of two loops each code (the unit of four frames) appears at least twice, resulting in a nearly perfect decoding rate for two-loop length.

Setting $bT = 20$ produced the best results for all writing speeds. Therefore we used this bit time in the following experiments. If there were long spaces between the bits ($bT = 30$ combined with a fast writing speed), the accuracy drops to 0, even for straight lines. The main reason for this can be seen in the fact that the written line is not long enough to store the information of a whole frame.

6.1.3 *Influence of Coding Styles*

Table 2 shows the decoding performance for different codes. For Set1, at all code styles, almost perfect decoding was realized at each frame. Considering the fact that the same code was embedded repeatedly, we can expect a perfect decoding in the case of straight lines.

The results on Set2 reveal interesting properties of the different code styles. The highest accuracy of 67% was achieved with the shortest code (7 bit per frame). With an increasing code length, the recognition rate drops dramatically. In fact, with the 34-bit/frame code (not in the table), no frame has correctly been decoded. However, we did not exhaust all the error correction possibilities of our encoding. The application of more sophisticated error correction methods is beyond the scope of this paper and will be subject to future work. (See Section 8.1.)

The last column in Table 2 shows the results when only 2-loop patterns were examined on Set2. The value shows how many information can be decoded correctly if only two succeeding loops are at hand. All encoded information was correctly decoded with the shortest code, which is a very promising result. Consequently we can retrieve $4 \times 7 = 28$ bits of information perfectly from 2-loops. Two loops have a pattern size of about 3cm $\times$ 6cm.

Table 2 shows the rejection rate of frames which either did not pass the plausibility control or were rejected after the parity check. Removing the plausibility control, all those frames would be considered as output, which would result in a large increase of incorrect frames.

In order to investigate the impact of the parity check, we have performed experiments without a parity check. For

Table 1: Variations of coding styles examined in the experiment. Recall that two bits are used for the frame address.

bB	bF	parity matrix	parity bits /frame	info bits /frame	total info bits
5	3	3×3	6	7	28
4	6	4×4	8	14	56
12	2	4×4	8	14	56
7	5	5×5	10	23	92

Table 2: Average decoding rates for different code styles.

				Set1 (straight)			Set2 (loop)			
bB	bF	info bits /frame	total info bits	correct frames (%)	rejected frames (%)	incorrect w/o parity (%)	correct frames (%)	rejected frames (%)	incorrect w/o parity (%)	recovery from 2-loop (%)
5	3	7	28	100	0	0	67	33	7	100
4	6	14	56	96	4	0	45	55	15	94
12	2	14	56	96	4	16	20	80	12	35
7	5	23	92	100	0	0	9	91	9	6

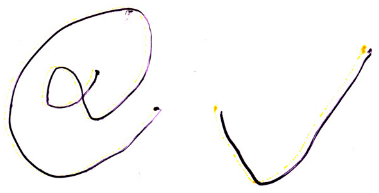
Figure 6: Example of a written "@" and a checkmark.

Set1, without the parity check, only 4% of all frames had errors. This implies that we can embed more information into straight lines by removing parity bits. In contrast, for Set2, we can see a larger impact. The rate of incorrect frames is not negligible. For example, in the case of $bB = 5$ and $bF = 3$, 7% of the correct frames (67%) were helped by the parity check.

6.2 Examples on Real Data

Besides the experiments described above, we also have performed tests on handwritten patterns which might appear in a real world scenario. That is, we have acquired three small sets with real handwritten strokes. The first set comprises 18 "@" symbols, the second set contains 21 checkmarks and the final set consists of 17 simulated signatures. Since the 7-bit code performed best in the previous section, we have used it in these experiments. Examples of the three data sets can be found in Figs. 6 and 7.

In our experiments, at least 21 bit were correctly extracted from each "@" symbol and each checkmark. Considering that these symbols are less than 3×3 cm in size, this is a very promising result. Note that 21 bit is enough to distinguish 2^{21} 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 in a huge company ($>$ 2 million!)

The most challenging task is the decoding of information added to a small handwritten signature "Meyer." The signatures have a size of 1 cm in height an 4 cm in length. In the case of the signature of Fig. 7, even three frames (out of four) were correctly decoded.

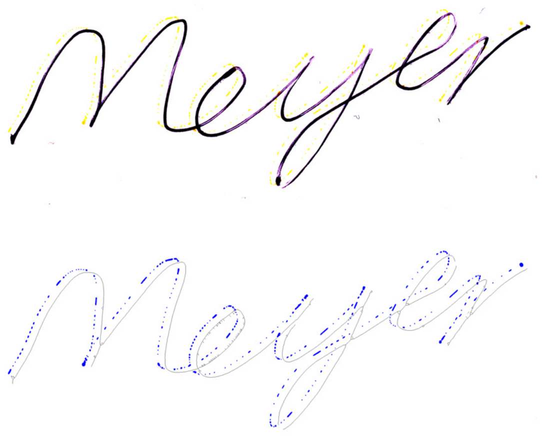
Figure 7: A simulated signature (top) and the extracted ink dots (bottom).

7. APPLICATIONS

What can this data-embedding pen be used for? It is very novel device and thus we have not assumed some specific "killer application". A possible application depends on how many bits can be embedded and decoded successfully – on this point, there is many room for improvement. However, we can consider many possible applications even though we can just embed, say, 28 bit information in a single handwriting pattern.

As indicated before, meta-information about the handwriting is the most straightforward target to be embedded; for

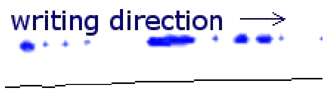

Figure 8: The direction of the closest stroke to the synchronization blob can be interpreted as the writing direction.

example, time/date of writing, geo-location, writer ID, and pen ID. Of course, this meta-information is useful for signature verification and other forensic applications. In addition, if we know the author of a handwritten note, 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 we embed the price of a product onto its package by a handwritten checkmark, the checkmark can be considered as a "handwritten" bar-code. If we embed a URL onto a physical paper by a handwritten word (or phrase), the word becomes a link between physical world (paper) and a cyberspace (the Internet).

Embedding information on the paper opens up new possibilities for diaries and notebooks. The owner can always find out when and where the information has been written down. After recognizing and recovering all information we can easily bridge to the digital world and fill Web 2.0 communication platforms with contents. For creating a blog of a journey, for example, you just need to write down short notes about the places where you are, the information where you have been and when the event took place would be automatically available.

One dreamy application is to embed the recognition result of the current handwritten word onto itself or the succeeding word. For this application, the handwriting pattern should be captured by a pen-tip camera or another sensor and recognized quickly.

8. DISCUSSION

In this Section we will discuss important aspects to be considered in order to make the pen device useful in praxis.

8.1 Coding Scheme

The encoding presented in Section 4 has much more capabilities than those currently exploited by the decoding algorithm. The pause before the synchronization blob, for example, can be used to determine the writing direction during stroke recovery. 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. 8. This information can improve stroke recovery when the writing direction cannot be determined by conventional techniques.

Another feature of the encoding is the possibility to derive the writing speed from the length of the n-pulse lines and the distance between succeeding bits. Longer n-pulse lines would correspond to faster writing (see Fig. 9). Furthermore, the writing angle and the tilt of the pen can be roughly

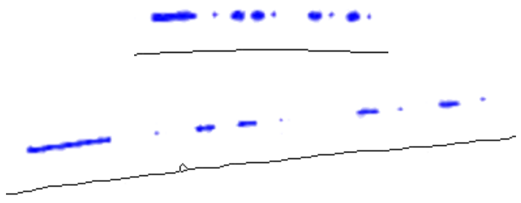

Figure 9: Different length of the n-pulse lines depending on the velocity. (top: slow velocity, bottom: faster velocity)

estimated by using the correspondence information from the ink-dots to the line (see Section 5.2). Note that a shorter distance would indicate a larger tilt angle if the nozzle is mounted on the pen as in Fig. 1.

Another crucial point is that in the current system most of the errors are detected but not corrected yet, since the main purpose of this work was to investigate the feasibility of our approach. Currently, only the parity information is used to correct 1-bit errors. Other errors, like missing ink dots resulting from occlusion are perfectly detected but not corrected yet. Developing algorithms to correct those errors will be subject to future work.

8.2 Writing Speed

As shown in the experiments, the writing speed is crucial for the reliability of the algorithm. Especially at stopping points and turning points there is no chance to recover the correct information because the ink dots occlude each other. The reason of this problem is that the ink dots are produced according to time intervals and not to spacial distances.

To overcome this problem, we plan to take into account the movement information of the pen. Therefore we can use gravity sensors (gyroscopes) or the information of a camera. Only a rough estimation of the movement speed is needed, so we can use fast means with moderate precision.

8.3 Design Issues

We aim to make the pen more handy and independent from external devices. Although the experimental results are quite promising, the first prototype as shown in Fig. 1 is far from being useful in praxis. The device is rather clumsy and there is a need of three external devices, a computer, an amplifier, and a vacuum device (see Section 3.2).

In order to increase the usability of the pen, its size will be reduced and the haptic feeling will be adjusted to make it similar to a normal pen. Some ideas were already realized. We have modified an angled pen device which was originally designed to facilitate writing[2]. The main casing is used as a channel for the nozzle and side-channel is used for a ball-point pen. The result is depicted in Fig. 10. Using this setup makes it very comfortable to use the pen, because it is similar to using a normal pen. Preliminary experiments with this new design have led to improved results.

[2]http://www.yoropen.com/

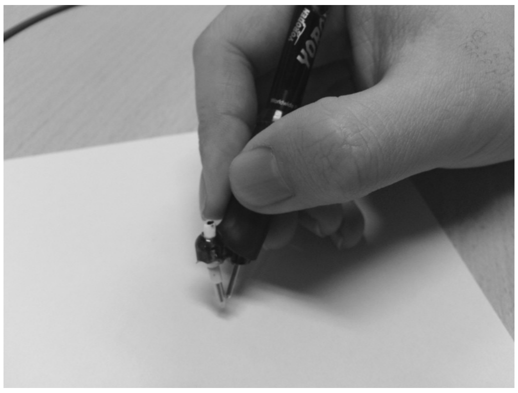

Figure 10: The new design of the data-embedding pen

For making the pen independent from external hardware, one has to choose smaller devices or finally develop an integrated solution. Since the main purpose of our work is a feasibility study and document analysis, this aspect was not regarded yet. However, we could already manage to remove the vacuum device by mounting the ink-container at a predefined height. Our ink-jet producer confirmed that we could use a smaller ink-container filled with a sponge mounted directly on the pen. To get rid of the big amplifier, we could use a tiny one as soon as all parameters regarding the nozzle and the encoding are fixed. Instead of the external computer used in the current setup, a small micro-controller can be used, as done in other electronic pen devices.

9. CONCLUSION

In this paper we have presented the first functional prototype of the data-embedding pen. This pen makes it possible to augment handwritten patterns with meta-infomation 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.

A first evaluation on handwritten strokes (straight lines and loop curves) showed that (i) we could embed and retrieve information ($\sim$ 100 bit) on straight lines almost perfectly, (ii) even on loop curves at least 28 bit were retrieved without error, and (iii) the main parameters on the encoding influenced the performance. In order to assess the practicability of our system, we have performed initial experiments on real handwritten words and characters. The encoded information could be recovered quite accurately from short strokes which appear in many real-world cases. This shows a high potential of the proposed approach.

In our discussions we have tackled the main problems of the current hardware architecture. Ideas for the solution of these problems have been proposed and some of them are already implemented.

The data embedding pen presented in this paper is a successful integration of state-of-the-art image processing methods and novel algorithms. The embedding makes it possible to reliably recover the information from scanned images without any human input. With this approach we increase the value of handwritten patterns on paper, introducing a promising direction for offline handwriting processing.

10. ACKNOWLEDGMENTS

The authors thank Mr. Y. Hori, Microjet Corp., Japan, for his kind support on customizing their inkjet nozzle technologies for our pen device. This research has been supported by the Japanese Society for the Promotion of Science (JSPS).