

## A METHOD FOR HANDWRITING INPUT AND CORRECTION ON SMARTPHONES

GARETH LOUDON, OLLE PELLIJEFF, LI ZHONG-WEI

*Cyberlab Singapore, Ericsson Research,*

*#18-00 SLF Building, 510 Thomson Road, Singapore 298135*

[gareth.loudon@ericsson.com](mailto:gareth.loudon@ericsson.com), [olle.pellijeff@ericsson.com](mailto:olle.pellijeff@ericsson.com), [zhong-wei.li@ericsson.com](mailto:zhong-wei.li@ericsson.com)

This paper describes a new approach for entering text on a “Smartphone” via natural handwriting input. The approach focuses on ease of use within the confines of a Smartphone display size and processing limitations. Therefore there are two integrated components to the approach. The first is a new handwriting recognition engine that has been designed to have a very high recognition accuracy (98.3% character accuracy), support sentence-based handwriting input, and have a small memory footprint (84 kb) and fast processing time. The second is a method that allows very simple editing and correction of recognized characters.

### 1 Introduction

Mobile devices that are available today are having more and more functionality built into them. There are new devices called “Smartphones” that can support voice communication, PIM tasks, as well Internet access for e-mail and browsing. These Smartphones support text input via a stylus. One example of such a Smartphone is the Ericsson R380 shown in Figure 1. Although a software keyboard can be used to enter text, a more convenient method is to use handwriting.



Figure 1. The Ericsson R380 Smartphone.

Some requirements on handwriting recognition engines for Smartphones are that they must run on slow processors with a low memory footprint, have very high recognition accuracy, enable fast text entry and allow easy correction. Several handwriting recognition engines have been developed to meet these requirements [1][2][3]. Limitations on some of these handwriting recognizers are that users have to write one character at a time and wait for recognition to take place. This waiting time is sometimes known as a “time-out”. One way around the problem of time-outs is to make the user write every character with one stroke. This is done in the Graffiti recognition engine used in the Palm devices. This enables very fast text entry speed. The system also has high accuracy because of the unique strokes representing each character. The disadvantage is that users have to learn the special way to write the

characters. To overcome the problems that exist with other handwriting recognizers we allowed users to write a phrase at a time. To achieve very high recognition accuracy we placed some restrictions on the handwriting styles. Users are only allowed to write in a hand printed style; they must write in lowercase letters and convert to uppercase during an editing phase; and they must write the characters in a particular way as shown in Figure 2. Accents are supported to allow entry of European specific characters. Other characters supported are common punctuation, symbols and also gestures for editing.

Lowercase chars	a b c d e f g
	h i j k l m n o p q r s
	t u v w x y z
Numerals	0 1 2 3 4 5 6 7 8 9
Accents	^ ˇ \ " ° ~
Misc. chars	œ œ †
Punct.	. , ? ! @ & '
Symbols	\$ £
Gestures	→ Erase ↓ Insert

Figure 2. Writing styles of characters supported.

## 2 Methods

The structure for the handwriting recognition architecture is shown in Figure 3.

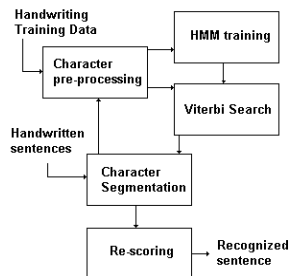


Figure 3. Handwriting recognition architecture.

The technique used in the handwriting recognition engine is based on discrete Hidden Markov Models (HMMs) [4]. HMMs are used because they handle the problems of non-linear shifting and multiple representations of handwritten characters well. There are two major modes to the handwriting recognition system. The first mode is the training of the HMMs, and the second mode is the recognition. Each handwritten character sequence is stored as a sequence of x,y coordinates in time. The next sections describe the training and recognition methods.

## 2.1 Training

Only handwritten examples of isolated characters are used for training. The following sections describe the training process.

### 2.1.1 Character Pre-processing

There are five steps in the pre-processing stage.

*Character Normalization:* Find the mean and variance of a handwritten character, and scale the character to have zero mean and standard variance. Characters are not distorted in shape during the normalization.

*Stroke Joining:* For characters written in two strokes, connect strokes together to form a one-stroke character. Fill in points between the first and second strokes. The number of points to fill in depends on the distance between the last point in the first stroke and the first point in the second stroke.

*Stroke Re-sampling:* Resample the pre-processed characters so that the average speed of writing a character is constant across all characters, while maintaining the unique acceleration and deceleration information within a stroke. This is important because some characters are written slowly and some characters are written quickly, but all examples of the same character tend to slow down at corners of a stroke and speed up for straight lines.

*Feature extraction:* Once the handwriting data has been re-sampled, four features are used to describe the character for each data point  $(x_i, y_i, x_i - x_{i-1}, y_i - y_{i-1})$ . If  $i < 2$   $x_i - x_{i-1}, y_i - y_{i-1}$  are set to zero.

*Vector Quantization:* The final step of the pre-processing is the quantization of the features into a fixed set of quantization codes. This helps to reduce the computation complexity in the HMMs. Vector Quantization (VQ) is a standard procedure that can be found described in several textbooks [4]. In the training stage of the process, two codebooks are formed; one 32 element codebook for the  $x_i, y_i$  features and one 32 codebook for the  $x_i - x_{i-1}, y_i - y_{i-1}$ . All handwriting training examples are then quantized using these two codebooks in preparation for the HMM training.

### 2.1.2 HMM Training

The discrete HMMs are trained using the standard Baum-Welch algorithm [4][5]. Two ten-state HMMs are used for each character. One HMM is based on the features  $x_i, y_i$  and one HMM is based on the features  $x_i - x_{i-1}, y_i - y_{i-1}$ .

## 2.2 Sentence Recognition

During the recognition stage, handwritten phrases are input into the handwriting recognizer. An example of handwritten text is shown in Figure 4 below. The recognition stage is split into four main steps. The first is the segmentation of handwritten sentence into isolated characters, the second is the extraction of features for character recognition, the third is the search algorithm to find the most likely character, and the fourth is the re-scoring of the recognition results.

here's my contact info!  
gareth.loudon@ericsson.com  
tel. +65 356 1661

Figure 4. An example of a handwritten sentence for recognition.

*Character segmentation:* Several parameters are calculated for each stroke during the character segmentation step. The key parameters are shown in figure 5.

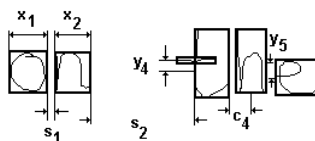


Figure 5. Some parameters for the handwritten phrase “on the”.

Parameter “x” describes the width of a stroke, parameter “s” describes the space between adjacent stroke, parameter “y” describes the difference in height between the centres of adjacent strokes and parameter “c” describes the distance between the centre of a stroke and the right-hand edge of the preceding stroke. If  $(s_i > \max(x_i))$  or  $(-s_i > 2 * \max(x_i) \text{ and } y_i > \max(x_i))$  stroke i is a character at the end of a word, else if  $(c_i > 0)$  stroke i is a character within a word, else stroke i should be merged with the following stroke to form a character. Each character segmented from the input sentence is passed onto the next steps for character recognition.

*Character Pre-processing:* The pre-processing step during recognition is the same as the training stage, with the only difference being that the codebooks are re-used from the training stage during the VQ process.

*Viterbi Search:* The time sequence of VQ codes for a character are matched against all the HMMs using a standard Viterbi search algorithm [4]. The probabilities of match between the input data and all characters are recorded and sorted. The five characters with the highest matching probability are stored.

*Re-scoring*: Once the segmentation and character recognition steps have been completed, a re-scoring step takes place that makes use of simple language knowledge to improve recognition performance. The first part of the re-scoring step uses a simple statistical language model based on character unigram and bigram probability values. The second part of the re-scoring step uses a word dictionary, in this case 16,000 words.

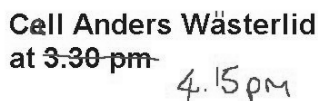
### 2.3 Preliminary Recognition Results

A test set of handwritten English sentences containing 5559 characters (including punctuation and numerals) were used to evaluate the recognition performance. The test set contained handwriting samples collected from seven different writers. All characters were written in a style that conformed with the requirements of the recognizer. The handwriting training set contained 11984 different characters written by thirteen different writers. The average character recognition accuracy was 98.3% for the 5559 characters when not including the use of a dictionary.

## 3 Editing and Correction of Recognized Characters

The most common approach for correction in recognition systems in the past has been to present a list of alternative characters or words for a user to choose from. If the correct character or word is not in the list then the user has to write the character or word again in some writing area that is separate from the recognized text. The process is also complicated if the user wants to edit a passage of text, because again the user has to switch between a text-editing mode and a handwriting-input mode.

The approach we have taken is to integrate the text editing and correction task with the handwriting input task. We created a mode of correction that is similar to the correction mode used when modifying text on paper with a pen. The user just has to write on top of an existing character to change it. The user can also delete words by crossing them out or inserting new letters or words using a combination of gestures and character writing. Figure 6 gives an example of the correction mechanism. The handwriting recognizer keeps track of re-written characters so that the same recognized character does not keep appearing after recognition.



Call Anders Wästerlid  
at ~~3:30 pm~~ 4.15 pm

Figure 6. The correction mechanism.

Accents can be added in the correction/editing stage, as shown above, so that specific European characters can be created. To change the case of a character from lower to uppercase (or vice-versa) simply tap on a button (that acts as a shift key) and then tap on the character you wish to change.

#### **4 Implementation**

The handwriting recognition engine was coded in “C” and compiled on the EPOC OS used by Ericsson’s Smartphones. The memory size of the basic handwriting recognition engine is 84kb (excluding the re-scoring module). If the dictionary is added the overall memory is 216kb.

#### **5 Conclusions**

The approach we have taken for handwriting input has been tailored to meet the needs of Smartphones and other mobile devices of today. The approach not only includes a new handwriting recognition engine but also introduces a new mode of editing and correction that is a very simple and easy to use. It is planned to undertake a subjective evaluation of our handwriting recognition engine as compared with the Graffiti recognition system by involving a large group of users. This is to study the advantages and disadvantages of both methods.

#### **References**

1. CalliGrapher by Paragraph, <http://www.paragraph.com>.
2. smARTwriter by ART, <http://www.artrecognition.com>.
3. JOT by CIC, <http://www.cic.com>.
4. L Rabiner and B,H Juang, *Fundamentals of Speech Recognition*, Prentice Hall, 1993.
5. L,E Baum, “Inequality and Associated Maximization Technique in Statistical Estimation of Probabilistic Functions of Markov processes”, *Inequalities*, vol. 3, 1972, pp.1-8.