Online Handwriting Recognition with Direction-Based Character Signatures

Kyle Murray

7.26.06

Abstract

Current online character recognition accuracy is not perfect, and there have been many attempts to perfect the technique over the years to ensure that Tablet PCs, PDAs, and other devices that utilize text recognition are usable and ideally transparent to the user. Many systems rely on context-based recognition, as letter identification without context clues has not been proven to be as accurate. A system was devised using direction-based stroke analysis to attempt to improve letter recognition rates. This system was able to correctly recognize the input character for 62.12% of the samples without the use of context clues. The success rate indicates that the system is not yet accurate enough to stand alone, though it could potentially be improved upon and implemented alongside context-based methods.

Introduction

Digitized pen input can be used to control computer interfaces. There are many ways in which a pen can interact with a computer its user interface (IBM, 2001. Van West, 2003). The pen can be used to control applications, make gestures that perform specific functions (Van West, 2003), or write text to be recognized by the computer and turned into electronic text input. (Chee, 2004. Downton and Impedovo, 1997. IBM, 2003. Liu *et al*, 2003. Heaton, 2005. Van West, 2003) Handwriting recognition is especially useful for computer-based input of non-Latin characters. The amount of characters in some languages is so great that traditional input methods such as keyboards are unable to fully represent all of the characters in an accessible manner, so interfacing with the computer is difficult. (Downton and Impedovo, 1997. Liu *et al*, 2003. Van West, 2003) PDAs have used this technology to assist with character input for years. (Chee, 2004. Heaton, 2005)

The widespread use and general knowledge of written languages on normal media such as paper by ordinary citizens generated the need for handwriting recognition with computers.

(Downton and Impedovo, 1997. Heaton, 2005. Liu *et al*, 2003) One of the primary original uses of handwriting recognition was to recognize the text and addresses written on letters that were going through the United States Postal System. The zip codes were read through early recognition systems inside of the letter sorters which would then transport the letters to the desired location. This system used an optical sensor to differentiate between white blank space and dark or black written text. (Downton and Impedovo, 2003. Heaton, 2005)

No known method of recognition is one hundred percent accurate, even trained humans cannot always read handwriting samples. (Downton and Impedovo, 1997. Liu *et al*, 2003) Publicly available methods of computer-based English *numeral* recognition have reached accuracies of up to 97.10%. There are many high-accuracy methods for recognizing common Latin characters, the highest of which have percent accuracies in the high nineties. (Liu *et al*, 2003)

Recognition based off of pre-recorded, physical handwriting samples is called “off-line” recognition, while recognition based off of live input from a digitizer tablet is called “on-line” (online) recognition. (Downton and Impedovo, 1997. Liu *et al*, 2003) Off-line methods are sometimes referred to as Optical Character Recognition (OCR) systems. (Downton and Impedovo, 1997. Heaton, 2005) Online handwriting recognition is used primarily as a user input system for computers. Written text is transformed into “electronic text”, the text commonly used by computers. (Downton and Impedovo, 1997. Liu *et al*, 2003) Conventional methods of deciphering characters from written text include neural networks, fuzzy logic, (Downton and Impedovo, 1997. Liu *et al*, 2003. Heaton, 2005) expert systems, statistics, and

large “lexicons” of words used to limit possible matches. (Downton and Impedovo, 1997. Liu *et al*, 2003)

Figure 1. A Wacom Graphire 3 digitizer tablet.

Photo by Kyle Murray.

Digitizing tablets, which can also be called graphics tablets or just tablets, have a wide variety of applications. There is not an industry standard application programming interface (API) for collecting pen input data off of tablets on multiple computing platforms. (Wacom Technology Corporation, 2005). Therefore, most online handwriting recognition systems can only be run on one platform without needing the data input program to be rewritten. (Downton and Impedovo, 1997. Wacom Technology Corporation, 2005) The Windows industry standard Wintab API is used to retrieve data from tablets connected to a computer. Most tablets can also function as the default pointing device on a computer. (Bastéa-Forte, 2005. Wacom Technology Corporation, 2005) Adobe Flash 9 applications developed using

ActionScript 3.0 can interface with tablets that present themselves as pointing devices using the DisplayObject mouseX and mouseY properties. (Adobe Systems Incorporated, 2006)

There are two main types of tablets. The first type has a screen directly under the writing surface. This type of tablet is commonly seen on "Tablet PCs" (Van West, 2003), though it is also used on higher-end graphics workstations. The second type, as seen in Figure 1, does not require a screen, the writing surface is blank and the data from the input pen can be shown on a separate, attached screen. These are used by signature verifications systems and by some graphic designers. (Downton and Impedovo, 1997. Liu *et* al, 2005)

Question

Does processing text information from digitized pen input by segmenting the characters into direction-based shapes and strokes increase the accuracy of character recognition versus conventional methods using image data?

Hypothesis

Online handwriting recognition using direction-based strokes to identify letters will recognize letters with higher accuracy than traditional image comparison-based methods.

Procedure

A class for testing handwriting recognition was started in the Adobe® Flex™ Builder™ 2 (Flex) integrated development environment, which was used for compiling ActionScript 3.0 code into Adobe® Flash® Player 9 (Flash) documents. Initially, the class was used to display lines in a trail behind the mouse when the primary mouse button was depressed.

Two utility classes were created, forming the net.reclipse.handwriting package. InkPoint, the first of the two new classes, was used to store coordinate, timestamp, and pressure data from the mouse or tablet. The pressure data point was only used to store one of two values; one representing no pressure and the other representing the presence of pressure. The second resource class, InkTimeline, was to be used for the storage of a collection of InkPoints so that data could be accessed and analyzed later. At first, the InkTimeline class extended the Array class, which is class that stores information in a way not unlike a traditional table, though the Array class is not limited to two dimensions. The InkTimeline class was later un-extended from the Array class, as it was discovered that the data type of the return values of class methods cannot be changed when extending a class and overriding its methods, so a public instance of the Array class was included automatically in each InkTimeline instance to maintain the desired functionality of InkPoint storage.

In the main class, methods were constructed to return the angle relative to the X and Y axis between two points and the angle change from the previous point as the input device moved across the viewable area so that directional input could be analyzed. Each stroke was then recorded and divided up into segments based on the angle difference between InkPoints. The tolerance level was set at 50 degrees, so a new segment was created for every change at or above that limit. The segments were colored for debugging and visualization purposes.

These segments were broken into more segments based on trends in the direction that the line continued in. The most common direction out of every four points (a variable amount) was added to an Array in the InkTimeline which defined the shape of the stroke in a simplified

manner compared to the InkTimeline Array of InkPoints. This array was then normalized to eliminate duplicate, side-by-side numbers which would interfere with recognition by forcing a certain scale to be present for recognition to occur. The normalized collection of segments became the 'signature' property of an InkTimeline instance. If two strokes intersected or one stroke was positioned and shaped like the dot on an 'i' or 'j', the last stroke was concatenated onto the previous stroke making one stroke that defined the character.

A third utility class named CharacterDefinition was created to handle the storage of known characters. The character definitions had three main parts, the simplest of those being the name of the letter it defined. The definition also carried the width:height ratio of the input stroke and all of the known signatures of the letter. This definition could be represented in String form (name:signature; signature:ratio), where any number of signatures could be defined. This String could have been loaded from an external source and parsed by the class using the addStringSample method, or generated from the CharacterDefinition object by the toString method.

In order to recognize an input character from the Object containing the references to character definitions, the input character was analyzed to produce a signature and ratio. The signature was tested against the known signatures in the defined letters. If a character definition had more than one signature definition that exactly matched the input signature, it was given top priority. If only one signature matched, it was given second priority, and if a defined signature contained, but was not exactly equal to the input signature, it was given third priority. The character with the highest priority won, and ties were settled by comparing the

ratios and determining the character with the least different ratio when compared to the input ratio.

To determine the accuracy of the recognition system, 10 letter definitions were supplied for each of the 26 lowercase letters in the Latin alphabet. These letter definitions were documented in String format for later use if necessary. For each letter, 20 attempts at recognition were made, making a total of 540 letter samples. The success was recorded as either a 0 or a 1, failure or success. The log of the console window that all trace operations were sent to was also recorded.

The source ActionScript 3.0 files can be found at: http://handwriting.reclipse.net/source/

ActionScript 3.0 .as files can be compiled for Adobe® Flash® Player 9 by downloading the SDK at: http://www.adobe.com/products/flex/sdk/

Discussion

The online recognition rates shown in Figure 2 are considerably lower than others that have achieved rates up to 97.10%. One of the most likely reasons for this is that these other recognition systems use context-based recognition, where whole words and sentences are fed into the system and grammar and spelling rules are applied to help recognize the letters. (Liu *et al*, 2003) This system only used information from a single character to determine which letter was being drawn, so letters that could not easily be identified without context clues are much less likely to be accurately recognized.

The decision not to use neural networks like similar systems use stems mostly from the fact that this system was designed to transform character information into a form that was not limited to a fixed size and not hindered by small variations in appearance. As a result, the data that is used for recognition is easily fed into a neural network. Neural networks require floating point data input (Downton and Impedovo, 1997. Van West, 2003), and if the sequences of numbers from the character signatures were converted directly into floating point forms, their relationship to the characters would be lost.

Some resolution was lost as a result of the decision to use ActionScript and *not* import data directly from the tablet. The frequency of data collection and the resolution of that data can be greatly increased when obtaining data directly from the tablet as opposed to treating the tablet as a mouse-like pointing device. (Wacom Technology Corporation, 2005) Using the JTablet API (Bastéa-Forte, 2005) and an XML Socket Server (Adobe Systems Incorporated, 2006), it would be possible to send that data to Flash. After adjusting some of the variables

used to control the simplification process, the recognition system should be able to recognize characters using data from the tablet. With the higher polling rate of the tablet, the natural motion of the input pen should be captured more accurately. The current system showed signs of a slow polling rate, as lines that should have been curved were straightened. This is seen in Figure 3.

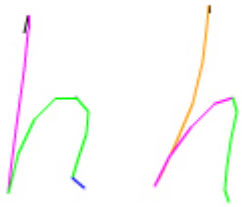

Figure 3: The Recognizer class running after being compiled. The 'h' on the left has a tail caused by the pen lifting off of the surface. The 'h' on the right is broken into three segments as a result of the pen moving faster than data was being provided.

Screenshot by Kyle Murray

A small issue that may have caused problems is also seen in Figure 3. The small tail on the end of the 'h' to the left probably added an extra direction to the signature that was not needed. In an update to the system, small tails like this should be removed before classification occurs. Another small issue is the rotation of the axis. Since the X and Y coordinate axis are the basis for many things, it is not surprising that many letters also contain vertical and horizontal lines. This causes a classification problem when a character with a vertical segment is tilted slightly to the left or right, as the character definition may only account for the version where the vertical segment tilted in one of the directions. By rotating the axis 45 degrees, fewer letters might be falsely identified if they contained vertical or horizontal lines.

One aspect of the trial process that may have falsely inflated the percent accuracy of the system is that the author of the recognition system is the same as the conductor of the tests. Prior knowledge of the way that the system works may have influenced the conductor in such a way that the accuracy was pushed higher than it would have been with a 3rd-party tester who was not familiar with the system.

Further testing could be done on the current version of the system, but due to the relatively poor success rate, better results might be achieved by fixing errors and attempting to improve the polling rate with the aforementioned XMLSocket and Java intermediary step. Combination with a context-based system could result in very high accuracies.

Literature Cited

Adobe Systems Incorporated. 2006. "Adobe® Flex™ 2 Language Reference". http://livedocs.macromedia.com/flex/2/langref/index.html

Bastéa-Forte, Marcello. 2005. "JTablet SDK v0.9.5 Documentation (API) ". http://cellosoft.com/sketchstudio/jtablet-docs/docs/

Chee, Yi-Min. 28 September 2004. "Ink Markup Language". http://www.w3.org/TR/InkML/

Downton, A. C. Impedovo, S. 1997. Progress in Handwriting Recognition. World Scientific Publishing Co. Pte. Ltd.

Heaton, Jeff. 2005. Artificial Intelligence: Programming Neural Networks in Java. http://www.jeffheaton.com/ai/

IBM. 2001. "Pen Technologies". http://www.research.ibm.com/electricInk/

Liu, Z. Cai, J. Buse, R. 2003. Handwriting Recognition: Soft Computing and Probabilistic Approaches. Springer-Verlag Berlin Heidelberg.

Van West, Jeff. 2003. "Using Tablet PC: Handwriting Recognition 101". http://www.microsoft.com/windowsxp/using/tabletpc/getstarted/vanwest_03may28hanrec.mspx

Wacom Technology Corporation. 2005. "Wacom Windows Developer FAQ". http://www.wacomeng.com/devsupport/ibmpc/wacomwindevfaq.html