

CellWriter: Grid-Entry Handwriting Recognition

Michael Levin
levi0190@umn.edu

December 6, 2007

Abstract

CellWriter is an open source handwriting input panel for the X Window System. Rather than specializing in recognizing a single language or group of languages, CellWriter relies on entry via a grid interface and robust character recognition to provide naturally multilingual handwriting recognition capacity. Elastic matching augmented with a greedy mapping algorithm to account for variability provide the underlying recognition mechanism. This paper describes the design decisions, algorithms developed for, and issues surrounding the program.

1 Introduction

Online handwriting recognition systems are necessary to enable intuitive handwriting-based text input for end users of Tablet PCs and PDAs that do not have convenient access to a keyboard. Special tablet screen and pen hardware in these devices can digitize pen strokes directly into point data, with sub-pixel precision for many devices. The problem of online handwriting recognition is to process digitized handwriting, recognize the text within, and output a string representation.

Specifically open source online handwriting recognition software is important for operating systems that abide by restrictive guidelines forbidding the inclusion of closed source and commercial software in the distribution. This category includes the popular Debian and Ubuntu GNU/Linux distributions. Such distributions today provide a free and competitive alternative to closed source offerings on Tablet PCs and PDAs but have until now

lacked a key software component to enable purely pen-based computing. Despite this need, there have been few efforts to develop a competitive open source product.

The goal of the CellWriter project is to exceed the limitations of existing open source handwriting recognition software and bring reliable natural handwriting recognition to the worldwide open source community. Meeting the need for a multilingual handwriting recognition by specifically tuning a system to every language the program could potentially be used with would be difficult and would ultimately provide better support for some languages over others. Instead, CellWriter takes a general approach and relies solely on the user to train the system with character samples characteristic of their individual handwriting style and language. With full Unicode support these samples can then be associated with the characters of any language, providing generic multilingual capacity.

2 Related Work

Methods for segmenting and recognizing characters and words within the input vary but the field as a whole is well developed and commercial software has been available for some time [1]. Modern commercial systems can achieve recognition rates in the mid-nineties with novice users and without any training data [2]. To achieve this, these systems are specifically tuned for recognizing one language at a time. The recognition system featured in the widely available Microsoft Vista family of operating systems robustly recognizes printed and cursive natural handwriting by using a time-delayed neural network and lexicon, but can only recognize a handful of languages because it requires the development of a separate neural network and lexicon for each language [3].

Existing open source handwriting input systems provide limited recognition. `xstroke` [4] and `wayV` [5] encode gestures as chain code paths through a nine square grid. Matching of chain codes, while sufficient for the rough gestures used in Internet browsers and as application shortcuts [6], provides only a rough recognition capacity insufficient for natural handwriting. Two other projects, `xmerlin` [7] and its descendant `rosetta` [8], use more so-



Figure 1: Main window showing multilingual input in English and Japanese kana.

phisticated elastic-matching with hybrid Euclidean distance and angle difference measures to compare strokes, but are still limited to single-stroke characters drawn in the same order and direction as training input.

3 Design Decisions

The design of the program interface is in some ways more important to the end user than the underlying recognition algorithms. The reliability of the program as a whole depends greatly on the capacity of the interface to gather accurate training and input samples in a convenient manner. CellWriter’s design strives to facilitate quick and intuitive function.

3.1 Grid Layout

Segmenting handwriting is a difficult problem interdependent with later stages of the recognition process even in the reading of machine printed text [9]. Rather than deal with this problem in a multitude of languages, CellWriter utilizes a character grid layout. The user draws one character at a time into the cell grid. When the user has finished the glyph and moved the pointer out of the cell, the glyph is recognized and the result immediately replaces the ink for instant feedback.

Using a grid also has advantages for the interface. Because recognition functions at the character level there is no need to display the ink after writing. Rather than using screen space to display both ink and text, the user’s ink can be transformed immediately into

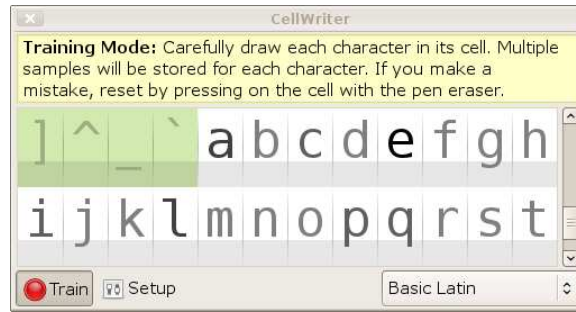


Figure 2: Main window in training mode. Cells that have been trained at least once appear with white background with darker text signifying more samples and those yet to receive any training samples use the disabled cell color, green in this image.

text, character by character. This saves screen space and gives the user the impression of handwriting printed text directly.

Additionally, using a grid layout allows for easy post-entry editing. Cells may be erased with the pen eraser or with a special cross out gesture and inserted by clicking on the divider between the cells. To correct a cell the user only needs to redraw the glyph in the same cell after recognition.

3.2 Mandatory Training

As a writer-dependent system, CellWriter requires that the user train the system before use. To facilitate rapid training, the grid layout can be placed in training mode (see Figure 2). During this mode, the user draws a sample of each character into its corresponding cell. After each sample glyph is drawn, the user moves the pointer outside of the cell to save the sample. Though many samples can be trained for every character, only one is required and CellWriter can also later augment these samples with regular input. Training takes no more than fifteen minutes and need not be repeated again.

4 Algorithms

After the user has drawn a character in one of the cells and recognition is triggered, the input sample moves through several stages in the recognition process before being finally

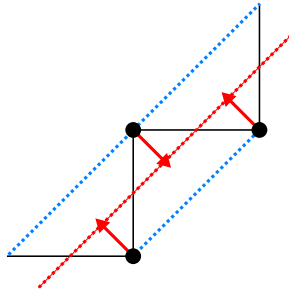


Figure 3: These points exhibit the staircase effect. The blue lines connect every other point and are used in the algorithm to produce the smoothed red line.

assigned a ranked and weighted list of likely matching training samples.

4.1 Digitizer Smoothing

While modern pen digitizers are very accurate, some distortion is still produced. In particular the “staircase effect” that results from snapping the pointer position to the nearest pixel or hardware grid unit (see Figure 3) can complicate segment angle measurements.

The first stage of preprocessing uses a simple smoothing algorithm to remove this kind of distortion. Each point in a stroke is moved halfway toward the straight line formed by connecting that point’s two immediate neighbors. This removes digitizer distortion without significantly altering the intended stroke. For every point \mathbf{b} and its immediate neighbors \mathbf{a} and \mathbf{c} the smoothed point \mathbf{b}' is computed by:

$$\mathbf{b}' = \frac{\mathbf{a} + \mathbf{b}}{2} + \frac{((\mathbf{b} - \mathbf{a}) \cdot (\mathbf{c} - \mathbf{a})) (\mathbf{c} - \mathbf{a})}{2\|\mathbf{c} - \mathbf{a}\|^2} \quad (1)$$

Additionally input is simplified by removing any redundant points that are not a significant distance away from the straight line formed by their neighbors.

4.2 The Preprocessor

The CellWriter preprocessor serves three functions:

1. Disqualify training samples unlikely to score well in primary recognition

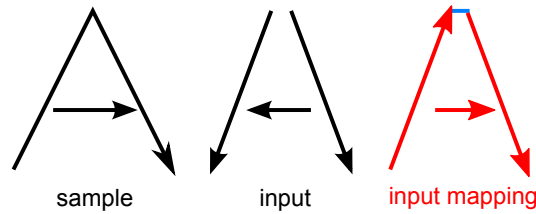


Figure 4: Mapping a three-stroke ‘A’ to a two stroke ‘A’ sample. The input is mapped to the training sample. The first stroke is reversed and glued to the third stroke. The second stroke is also reversed.

2. Find the optimal mapping between the input and training samples
3. Penalize training samples for important differences from the input

Because strokes within a character may be drawn backwards relative to the training sample, in a different order, or connected together, CellWriter employs a corrective process to account for these variations by ‘mapping’ one sample to another. A mapping is always made from the sample with more strokes to the sample with less strokes and describes the order of the strokes, whether any strokes are reversed, and what order to glue together any strokes in the larger sample so that its strokes are correctly mapped to those in the smaller sample. Figure 4 shows how a three-stroke ‘A’ input sample can be made compatible with a two-stroke ‘A’ training sample by the preprocessor mapping.

While finding such a mapping is theoretically a combinatorial problem, in practice these mappings can be found with a fast greedy algorithm without loss of accuracy. The actual implementation of such an algorithm requires a lot of tuning but the basic idea is fairly simple. The preprocessor incrementally builds strokes from the sample with fewer strokes out of strokes from the sample with more strokes. Each stroke is tried as a candidate for addition, in both directions, until a good match is found. If the constructed stroke is still significantly shorter than the target, the process continues by attempting to glue on an additional stroke, otherwise construction moves on to the next stroke. The algorithm can also fail and disqualify the match from further processing.

The stroke distance measure used by the preprocessor is a low-resolution version of the stroke distance algorithm described in the next section with one important difference.

Input: Smaller sample s , larger sample l
Output: Mapped sample m
Data: Sample t , sample t_{best} , i , j , n

```

 $dist \leftarrow 0$ 
 $i \leftarrow 0$ 
 $n \leftarrow 0$ 
 $m \leftarrow \emptyset$ 
while  $i < \text{LengthOf}(s)$  do
  repeat
     $j \leftarrow 0$ 
     $v_{best} \leftarrow \infty$ 
     $t_{best} \leftarrow \emptyset$ 
    while  $j < \text{LengthOf}(l)$  do
      if  $dist + \frac{1}{2}\text{LengthOf}(l[j]) > \text{LengthOf}(s[i])$  then
         $\perp$  continue
       $t \leftarrow m$ 
       $t[i] \leftarrow t[i] + l[j]$ 
       $v \leftarrow \text{PartialMeasure}(t, s)$ 
      if  $v < v_{best}$  then
         $\perp$   $v_{best} \leftarrow v$ 
         $\perp$   $t_{best} \leftarrow t$ 
       $t \leftarrow m$ 
       $t[i] \leftarrow t[i] + \text{Reverse}(l[j])$ 
       $v \leftarrow \text{PartialMeasure}(t, s)$ 
      if  $v < v_{best}$  then
         $\perp$   $v_{best} \leftarrow v$ 
         $\perp$   $t_{best} \leftarrow t$ 
      if  $t_{best} \neq \emptyset$  then
         $\perp$   $m \leftarrow t_{best}$ 
         $\perp$   $n \leftarrow n + 1$ 
      else if  $m[i] = \emptyset$  then
         $\perp$   $m \leftarrow \emptyset$ 
         $\perp$  return
    until  $t_{best} = \emptyset$ 
  if  $n < \text{LengthOf}(l)$  then
     $\perp$   $m \leftarrow \emptyset$ 

```

Algorithm 1: Greedy Preprocessor Mapping

Because the constructed stroke may have more strokes glued onto it later, the stroke built so far is compared only with the equivalent distance on the target stroke rather than the full length of the stroke.

It is convenient to calculate penalties as a by-product of finding the mapping. CellWriter penalizes samples for large vertical displacement to distinguish between samples that differ only in vertical location such as the comma and apostrophe. Long glue distances and many glued segments are also penalized to avoid distorting the sample, by creating a ‘c’ from an ‘=’ by connecting the two horizontal lines for instance.

4.3 Distance Measure

Once the preprocessor has made the training sample and input compatible, the primary recognition engine compares each corresponding pair of strokes in order. Both strokes are sampled to the same number of points with sample size chosen so that no stroke has a segment longer than a small ‘unit’ distance. To compute a ‘distance’ measure between two strokes, each pair of corresponding points at the regular sample intervals is measured with each of two distance functions in turn and the average of the resulting set of values for each function is taken.

The first function is the squared Euclidean distance. While a normal Euclidean distance function is adequate as a distance measure, by removing the square root, a small number large distances can be disproportionately punished versus a set of minor variations. This helps to distinguish very similar letters such as ‘g’ and ‘y’.

Because the two samples might be shifted, an offset vector is computed by calculating the center point of each sample and taking the difference. Each sample’s center point is calculated by summing the center points of each of its component strokes, weighted by stroke length. For a sample stroke S with n points, the center point is computed as

$$\mathbf{S}_c = \frac{\sum_{i=0}^{n-1} \|\mathbf{S}_i - \mathbf{S}_{i+1}\|(\mathbf{S}_i + \mathbf{S}_{i+1})}{2 \sum_{i=0}^n \|\mathbf{S}_i - \mathbf{S}_{i+1}\|}. \quad (2)$$

The squared Euclidean distance function for a point \mathbf{P} and a point \mathbf{Q} with a center offset vector \mathbf{O} can then be efficiently computed as

$$D_E(\mathbf{P}, \mathbf{Q}, \mathbf{O}) = (P_x - Q_x - O_x)^2 + (P_y - Q_y - O_y)^2. \quad (3)$$

The second distance function computes the smallest angle difference between the line segments formed by the current pair of points and the next. The smallest angle difference between points \mathbf{P} and \mathbf{Q} that form segments with angle A_P and A_Q with the next point in their stroke,

$$D_A(\mathbf{P}, \mathbf{Q}) = \begin{cases} |A_P - A_Q| & A_Q \leq A_P, \\ |A_Q - A_P| & A_Q > A_P. \end{cases} \quad (4)$$

Following this procedure strictly would cause synchronization problems with pairs of strokes that have slightly disproportionate segments. By advancing at fixed rate temporally, the measurement position on one stroke would advance ahead of the corresponding position on the other stroke spatially, falsely inflating the distance measurements. To account for these kinds of variations, CellWriter tracks the position on each stroke separately and allows either to speed up or fall behind temporally in order to stay close spatially.

In order to find the optimal assignment of points from one stroke to the other, CellWriter uses a dynamic programming algorithm to compute the lowest sum of measures between the first and second stroke's points. Rather than programming the entire table however, only squares within a small distance, the 'elasticity' parameter in, of the diagonal are computed. The elasticity value will depend on the scale of the unit distance to which the stroke data was uniformly sampled to. For a unit distance of $\frac{1}{32}$ of the cell height, CellWriter uses an elasticity of 2 table cells. This limitation serves both to prevent over-matching and to speed up the algorithm.

The distance measurements of each stroke pair are weighted by their average stroke length and summed to produce the two main ratings for the match. Additional ratings are

Input: Stroke a , stroke b , function $\text{Measure}()$, elasticity

Output: Distance measure v_{lowest}

Data: $\text{table}[\text{LengthOf}(a) + 1][\text{LengthOf}(a) + 1]$, v , m , i , j
 $\min(\text{LengthOf}(a) + 1, \text{elasticity} + 2) \rightarrow j_{\text{end}}$

for $1 \rightarrow j; j < j_{\text{end}}; j + 1 \rightarrow j$ **do**
 $\infty \rightarrow \text{table}[0][j]$
 $2 \times \text{Measure}(a[0], b[0]) \rightarrow \text{table}[1][1]$

for $1 \rightarrow i; i < \text{LengthOf}(a) + 1; i + 1 \rightarrow i$ **do**
 $\max(1, i - \text{elasticity}) \rightarrow j$
 $\infty \rightarrow \text{table}[i][j - 1]$
 $\infty \rightarrow \text{table}[i][j_{\text{to}}]$
 if $i = 1$ **then**
 $j + 1 \rightarrow j$
 $\min(\text{LengthOf}(a) + 1, i + \text{elasticity} + 1) \rightarrow j_{\text{end}}$
 $\text{table}[i - 1][j - 1] \rightarrow v$
 for ; $j < j_{\text{end}}; j + 1 \rightarrow j$ **do**
 $\text{Measure}(a[i - 1], b[j - 1]) \rightarrow m$
 $v + 2m \rightarrow v_{\text{lowest}}$
 $\text{table}[i][j - 1] + m \rightarrow v$
 if $v < v_{\text{lowest}}$ **then**
 $v \rightarrow v_{\text{lowest}}$
 $\text{table}[i - 1][j] + m \rightarrow v$
 if $v < v_{\text{lowest}}$ **then**
 $v \rightarrow v_{\text{lowest}}$
 $v_{\text{lowest}} \rightarrow \text{table}[i][j]$

Algorithm 2: Distance Between Strokes

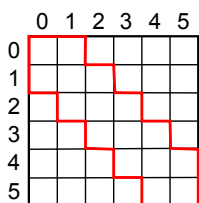


Figure 5: Strokes are compared via a dynamic programming algorithm that finds the shortest distances between their points, limited to a certain elasticity (in red).

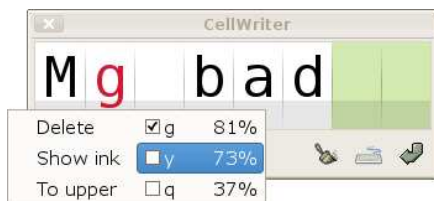


Figure 6: The context menu has been opened on the ‘g’. The letter is highlighted because the recognition match strength over the next candidate was below five percent.

also produced by preprocessor mapping and word context analysis. Each training sample receives one rating from each of these sources.

Since each method of measuring distance has its own scale, the ratings must be normalized before they can be combined into one composite rating for the training sample. For a set \mathbf{R} of n ratings with a mean value of $\bar{\mathbf{R}}$, the normalized value of a rating

$$\text{Normalize}(\mathbf{R}_i) = \frac{\mathbf{R}_i - \bar{\mathbf{R}}}{\max_k(\mathbf{R}_k) - \bar{\mathbf{R}}}. \quad (5)$$

Normalized ratings are then weighted and summed to produce a percentile. Currently only the word context rating is weighted down at one third of normal weight. The percentile is discounted by penalties incurred during preprocessor mapping and the sample with the best rating is chosen as the final match.

CellWriter also allows the user to select from the top five candidates via a context menu. If the match strength between the top two candidates is below five percent, the letter is highlighted to let the user know that the match may be erroneous (see Figure 6).

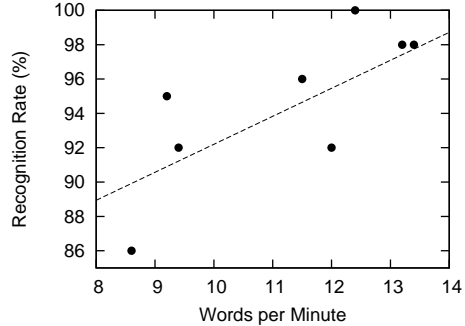


Figure 7: The results of a small informal study suggest a positive correlation indicating that both recognition rate and speed increase with user skill rather than trade off.

4.4 Word Context

To aid recognition of common words, CellWriter can use a list of common words and their expected frequencies if one is available. When a character is written, the letters surrounding it are used to find matching words within the frequency list. The samples of the potential letters are then rated by how frequently they appear. These ratings serve to boost recognition rates but are not a major part of the final score.

This component also suggests whether the character is likely to be a lower- or uppercase letter or a number judging from the preceding character.

5 Performance Evaluation

CellWriter’s performance depends largely on the quality of the training samples and the ability of the writer to produce consistent handwriting. Recognition rates generally improve with more samples and user experience. Since its initial release on August 21 of 2007, many users have reported good results. CellWriter has been successfully used with English, German, Russian, Turkish, Czech, Japanese (kana only), and mathematical symbols.

In a small informal survey of eight CellWriter users, the average recognition rate was reported at 94.6% and average writing speed at 11.2 words per minute. Rather than showing a trade-off between writing speed and recognition rate, the results showed a positive correlation between the two factors (see Figure 7) suggesting that both recognition accuracy

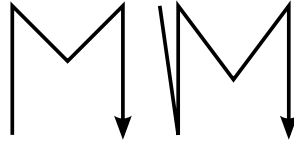


Figure 8: The mapping process cannot account for the retraced stroke in the second ‘M’ sample.

and rate of entry depend on the user’s skill with the program.

6 Future Work

CellWriter is still a relatively young program and there are many opportunities for improvement, either in the generic recognition algorithms or to meet specific language needs.

6.1 Additional Measures

The extendable nature of the ratings system allows additional recognition methods to be easily added to the system. A recognition method would need to produce a match quality rating for every training sample. Like the word context component, individual measure ratings can be weighted to properly tune their contribution to the final composite rating.

6.2 Separate Accents

European users of languages that contain many accented variations of characters need to train each such variation separately even though they often represent an unaccented letter combined with one or more accent marks. The training interface could be adapted to train accent marks and base characters separately and either automatically generate accented samples or adapt the recognition process to look for accents within input characters.

6.3 Preprocessor Mapping

The preprocessor mapping algorithm is not perfect. Most notably, it is incapable of correcting variations within a single stroke (as in Figure 8) even though such variations would

be accounted for if a properly split multi-stroke training or input sample existed. This capability could be added in the future by segmenting strokes at high curvature points.

6.4 Asian Languages

Another issue pertains to training Asian characters. While training Japanese kana is not a problem, Asian languages contain several thousand Kanji characters that are very time consuming to train one at a time with the current interface. Since these characters should not vary significantly from writer to writer, a good solution would be to pretrain these characters from a database.

One option is to integrate with the open source JavaDict program, which features a collection of more than 3000 of the most commonly used Japanese and Chinese Kanji, specifically to recognize Kanji. This approach has been successfully tested for many-stroke Kanji but reported to work poorly with low stroke count characters and kana [10]. Since CellWriter's strengths are in recognizing low stroke count characters, the two systems would complement each other well.

7 Conclusion

CellWriter has been released under the terms of the open source GNU General Public License version 2 and is available from the following sources:

Debian: <http://packages.debian.org/cellwriter>

Ubuntu: <http://packages.ubuntu.com/hardy/gnome/cellwriter>

Source code: <http://risujin.org/cellwriter>

Acknowledgments

The author would like to thank Professor Nikolaos Papanikolopoulos for his guidance as an advisor over the course of the project. The CellWriter project was supported by the Undergraduate Research Opportunities Program at the University of Minnesota.

References

- [1] R. Plamondon, “Online and off-line handwriting recognition: a comprehensive survey,” *Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, Jan 2000.
- [2] S. MacKenzie and L. Chang, “A performance comparison of two handwriting recognizers,” *Interacting with Computers*, vol. 11, pp. 283–297, Jan 1999.
- [3] J. A. Pittman, “Handwriting Recognition: Tablet PC Text Input,” *Computer*, vol. 40, pp. 49–54, Sep 2007.
- [4] C. D. Worth, “xstroke: Full-screen Gesture Recognition for X,” in *Annual Technical Conference*, FREENIX Track, pp. 187–196, USENIX, Apr 2003.
- [5] M. Bennett, “wayV.” <http://www.stressbunny.com/wayv/>.
- [6] S. Kourakis, *On gestural interaction and modular gestures*. PhD thesis, Universitat Pompeu Fabra, Barcelona, Sep 2004.
- [7] S. Hellkvist, “On-line character recognition on small hand-held terminals using elastic matching,” Master’s thesis, Royal Institute of Technology, Stockholm, 1999.
- [8] O. Reinhardt, “Rosetta: Multistroke and full word handwriting recognition for X.” <http://www.handhelds.org/project/rosetta>.
- [9] R. G. Casey and E. Lecolinet, “A survey of methods and strategies in character segmentation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, pp. 690–706, Jul 1996.
- [10] M. C. Ramsey, T. H. Ong, and H. Chen, “Multilingual input system for the web—an open multimedia approach of keyboard and handwriting recognition for chinese and japanese,” *Fifth International Forum on Research and Technology Advances in Digital Libraries*, p. 188, 1998.