EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2012)
L. B. Kara and K. Singh (Editors)

The 1¢ Recognizer: A Fast, Accurate, and Easy-to-Implement Handwritten Gesture Recognition Technique

J. Herold[1] and T. F. Stahovich[2]

[1]Department of Computer Science and Engineering, University of California, Riverside, CA 92521, United States
[2]Department of Mechanical Engineering, University of California, Riverside, CA 92521, United States

Abstract

We present the One Cent Recognizer, an easy-to-implement, efficient, and accurate handwritten gesture recognizer. By applying time series recognition techniques, we have developed a minimally complex technique that is both much faster than and at least as accurate as the Dollar Recognizer. Additionally, the One Cent Recognizer is much easier to implement than the Dollar Recognizer. Our technique is primarily enabled by a simple and novel one-dimensional representation of handwritten pen strokes. This representation is intrinsically rotation invariant, allowing our technique to avoid costly rotate-and-check searches typically employed in prior template-based gesture recognition techniques. In experiments, our technique has proven to be two orders of magnitude faster than the Dollar Recognizer.

Categories and Subject Descriptors (according to ACM CCS): I5.2 [Pattern Recognition]: Design Methodology—Classifier Design and implementation

1. Introduction

Gesture recognition is a primary enabling technology in pen-based computer interfaces and sketch understanding. Gesture recognition is the process by which a user's handwritten pen stroke is recognized as being one of a number of predefined gesture types.

As we show in the related work section, numerous handwritten gesture recognition techniques have been developed. Arguably, the most popular of these is the Dollar Recognizer ($1). This aptly named technique's popularity can be attributed to its ease of implementation, efficiency, and high performance.

In this work, we improve on $1 by introducing the One Cent Recognizer (1¢), a technique that requires less code, is more efficient, and performs at least as well as $1. This technique is enabled by a novel transformation of raw pen stroke data into a one-dimensional feature vector. A simple time series classification technique can then be used to find a feature vector's closest match in a training corpus. This novel transformation is intrinsically rotation invariant, eliminating the need for rotate-and-check searches typically employed in previous approaches, resulting in a substantial speed advantage for the 1¢ technique.

We perform two rigorous train-and-test cross-validation schemes to evaluate the efficiency and accuracy of 1¢ and $1. The first is a user-dependent scheme, which demonstrates each recognizer's performance when the testing and training data are both generated by the same participant. The second is a user-independent scheme, which demonstrates each recognizer's performance when the testing data comes from a participant who generated none of the training data. These tests show that 1¢ is always at least as accurate as $1, and is always two orders of magnitude faster.

In the following section we discuss related work in handwritten gesture recognition as well as time series classification techniques that serve as inspiration for 1¢. In Section 4 we provide the algorithmic details of our technique. The results of the user-dependent and user-independent evaluations are presented in Section 5. We discuss the results and present conclusions in Section 6 and Section 7. Lastly, in the Appendix, we present concise pseudocode for 1¢ that can be used to quickly implement this technique. Additionally, the appendix contains the URL to an open-source implementation of 1¢.

DOI: 10.2312/SBM/SBM12/039-046

2. Related Work

Wobbrock et. al [WWL07] developed \$1, which is so called because of its ease of implementation; the pseudocode for \$1 contains only 72 functioning lines. This handwritten gesture recognition technique comprises two major steps: transformation and recognition. In the first step, a raw stroke is transformed into a template. This transformation begins by resampling a stroke to a fixed number of points such that the distances between all successive pairs of points are equal. Next, the resampled stroke is rotated so that its indicative angle lies at $0°$. Lastly, the points are scaled to a fixed-size square and the points are translated so that their centroid is at the origin. These transformations effectively normalize the pen stroke so that comparisons are scale and position invariant. In the second step, the templatized candidate stroke is compared to a number of training templates in order to find its best match. In order to be rotation invariant, \$1 employs golden section search during this step to determine the best angular alignment of the candidate with each template.

Rubine [Rub91] developed one of the earliest handwritten gesture recognizers in his seminal paper. This technique begins by representing a pen stroke using 13 geometric features, such as the sum of the curvature values at each point of the stroke, the angle at the first point of the gesture, and the size of the stroke's bounding box. Gestures are recognized using a simple linear classifier, which is trained on these features. While this approach achieved good accuracy, it was not rotation invariant.

The Image-Based Recognizer [Kar04] applies popular image recognition techniques. The Image-Based Recognizer converts each candidate stroke into a fixed-size binary bitmap. The distance between two bitmaps is computed using four traditional bitmap distances, e.g., the Hausdorff distance. Similar to \$1, when a candidate template is compared to a training template, a search is employed to find the angular orientation that minimizes the distance to that training template. While this technique is typically more accurate than \$1, it is much more complex in its implementation and is far less efficient; operations on bitmaps are typically $O(N^2)$.

These techniques have several features in common. Most importantly, they are all template-based. The unknown candidate must be compared to each of the templates in the training corpus to determine the best match. Also, \$1 and Rubine's method can recognize only single-stroke gestures. For example, they cannot recognize the letter "t" if it is drawn with two strokes.

While the Image-Based recognizer can recognize multi-stroke symbols, the set of strokes comprising the symbol must be distinguished from the other strokes in the sketch by some other process. Sezgin and Davis [SD05] developed a technique which uses Hidden Markov Models (HMMs) to identify all of the symbols in a sketch without preprocessing. By considering sequences of pen strokes, the HMMs naturally identify multi-stroke symbols. This approach begins by processing strokes using the Toolkit from [SSD06]. This toolkit segments pen strokes into their constituent geometric primitives, such as lines and arcs. The segmented output is then converted into a sequence of discrete observations. A separate HMM is trained for each symbol class. Candidate sequences are classified according to the HMM that best recognizes the candidate sequence, i.e., returns the highest probability using the Forward-Backward algorithm.

While machine learning approaches such as linear classifiers and HMMs, are frequently used for sketch understanding techniques, time series classification techniques [CK05, HMS05, CK05] have not been widely used. This is surprising as pen strokes are a time series of (x,y) values. Xi et al. [XKS*06] have shown that simple first-nearest-neighbor (1NN) Euclidean distance and Dynamic Time Warping (DTW) approaches are often more accurate than more complex time series classification techniques, with DTW being the more accurate of the two. Tappert [Tap82] used DTW to implement a handwriting recognition technique which was used as a benchmark for \$1. It was found that \$1 typically performed as well as DTW and was much more efficient and easier to implement. However, DTW is much more expensive than Euclidean 1NN approaches. Our goal is develop an efficient gesture recognizer using a simple, Euclidean 1NN approach.

The key to using a Euclidean 1NN approach is developing a one-dimensional time series that allows for accurate recognition. The representation used in this paper is based on that found in [XKWMN07] which was used to recognize digitized hieroglyphs. We have significantly modified this representation to handle handwritten gestures. The hieroglyphs always formed a closed contour, which is rarely the case for handwritten gestures. Additionally, while the hieroglyph bitmaps contain no timing information, handwritten gestures do, and this may be leveraged to achieve greater accuracy. Furthermore, the one-dimensional hieroglyph representation is intrinsically sensitive to angular orientation, and thus, just as with \$1, search is required to determine the best angular orientation. As we show in Section 4, our representation is rotation invariant, eliminating the need for such a search.

3. Data Set

We used the pen stroke data from the study described in [HS11] to evaluate the performance of 1¢ and benchmark it against \$1. In that study, an HP TC4400 Tablet PC with a digitizer resolution of 1024x768 pixels was used for data collection. The participants were asked to draw each of the ten symbols in Figure 1 18 times. Thus, there were 180 strokes drawn by each participant, yielding 2,520 strokes in total.

[HS11] reports that the participants were informed that the purpose of the study was to collect data to evaluate the performance of an algorithm. Participants were instructed to “draw naturally with ordinary accuracy,” and to not attempt to “trick or break the system.” Before beginning the exercise, each participant was given a few minutes to practice drawing on the Tablet PC.

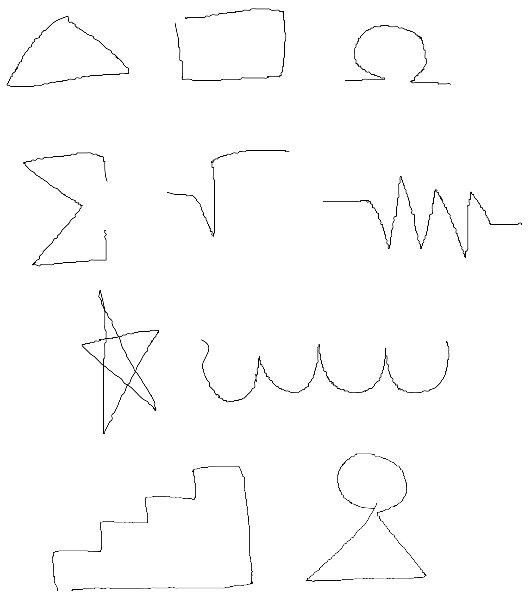

Figure 1: *The ten shapes used to evaluate 1¢: triangle, rectangle, omega, sigma, square root, resistor, star, inductor, stoop, not.*

Each point from the data set is a triple containing an x-coordinate, y-coordinate, and time stamp. To facilitate training and testing of algorithms, each gesture was manually labeled according to its shape.

4. Approach

1¢ comprises two major steps. In the first step, the raw pen stroke data is transformed into a fixed-length, one-dimensional vector which we call a template. In the second step, distances are computed between the candidate and training templates to find the closest match.

4.1. Step 1: Templatizing Pen Stroke Data

4.1.1. Resample

A simple Euclidean distance comparison of two time series requires that both series be the same length. Thus, the first step in templatizing a pen stroke is to resample it to a fixed number of points, N, such that the distance between all pairs of successive points is equal. We accomplish this using the same approach used in [WWL07]. To be consistent with the implementation of \$1 we use $N = 64$ in this paper, although as we show in the results section, any value of N between 16 and 128 is acceptable.

4.1.2. One Dimensional Representation

Next, we transform the three dimensional $(\mathbf{x},\mathbf{y},\mathbf{t})$ data of each stroke into a one-dimensional vector, $\mathbf{d}$. We begin by computing the centroid, c, of the points comprising the stroke, i.e., (μ_x, μ_y). We then compute the Euclidean distance from c to each resampled point in the stroke. The resulting series of distances, $\mathbf{d}$, ordered by time, characterizes the stroke:

$$\mathbf{d} = \{d_1, ..., d_N\} \text{ s.t. } d_i = ||p_i - c|| \tag{1}$$

Because this representation is based on the centroid of the gesture, it is intrinsically rotation invariant. Consider the example in Figure 2: a rotation transformation does not change a point’s distance to the centroid. This allows our technique to avoid expensive search that other methods require to find the optimal angular alignment. Additionally, this transformation is intrinsically position invariant, allowing for strokes drawn anywhere on a page to be compared to each other.

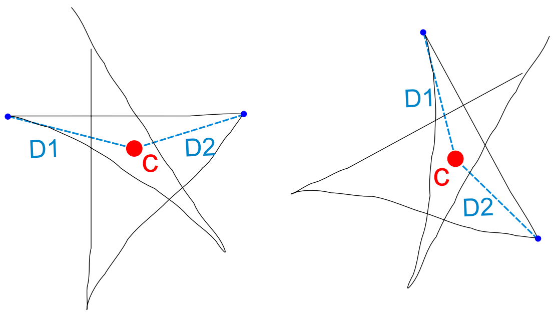

Figure 2: *Rotation invariance of our one-dimensional pen stroke representation. Rotating the star gesture does not change the distance from the centroid (point “C”) to points on the stroke such as points D1 and D2.*

Next, we z-normalize $\mathbf{d}$ to ensure scale invariance:

$$\mathbf{z} = \{z_1, ..., z_N\} \text{ s.t. } z_i = \frac{d_i - \mu_d}{\sigma_d} \tag{2}$$

where μ_d is the average of the stroke’s d_i values, and σ_d is the standard deviation.

This z-normalized distance vector, $\mathbf{z}$, is the template used to represent the candidate pen stroke. Figure 3 illustrates each of the major steps of this templatizing process.

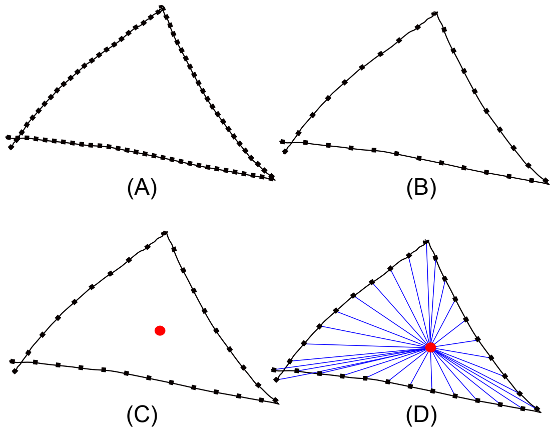

Figure 3: *The templatizing process. The raw candidate pen stroke (A) is first resampled into N points (B). Next the centroid (red point) of the resampled points is computed (C) and the distance from it to every resampled point on the stroke is computed (blue lines). This sequence of distances is z-normalized and the resulting vector is the template for the candidate stroke.*

4.2. Step 2: Comparing Templates

Because all distance vectors are the same length and z-normalized, they may be directly compared. We use the L^2 distance to compare vectors. More formally, the distance between two vectors, $\mathbf{v_1}$ and $\mathbf{v_2}$ is defined as:

$$L^2(\mathbf{v_1},\mathbf{v_2}) = ||\mathbf{v_1}-\mathbf{v_2}||^2 \qquad (3)$$

Using this distance, we apply a simple 1NN approach to recognize gestures. A templatized candidate (unknown) stroke, U, is compared to each training template, T, in the training set. The matching training template, T^*, is the one which minimizes the L^2 distance:

$$T^* = \arg\min_T L^2(U,T) \qquad (4)$$

5. Results

We present here an analysis of the efficiency and accuracy of both \$1 and 1¢. Two separate validation schemes were used, a user-independent scheme and a user-dependent one. In each fold of the user-independent scheme, n examples of each gesture from one participant were used for training, and all gestures from the other participants were used for testing. As there were 14 participants, we performed 14-folds of cross-validation for varying values of n. In each case, 2,340 gestures (13 participants x 180 gestures) were used for testing. The accuracy achieved by both recognizers for varying values of n are shown in Figure 4. Recognition timing results are shown in Table 1.

n	\$1 Tot.	1¢ Tot.	\$1 Ave.	1¢ Ave.
1	5.20	0.16	2.22	0.07
2	10.20	0.21	4.35	0.09
3	15.22	0.25	6.50	0.11
4	20.07	0.30	8.57	0.13
5	25.28	0.38	10.80	0.16
6	29.96	0.41	12.80	0.17
7	35.14	0.47	15.01	0.20
8	40.39	0.54	17.26	0.23
9	45.53	0.57	19.45	0.24

Table 1: *Computation times for \$1 and 1¢ as the number of examples (n) of each gesture used for training is varied. The second and third columns are the total time required by \$1 and 1¢ to recognize all 2340 test gestures (seconds). The last two columns are the average times to recognize a single gesture (milliseconds).*

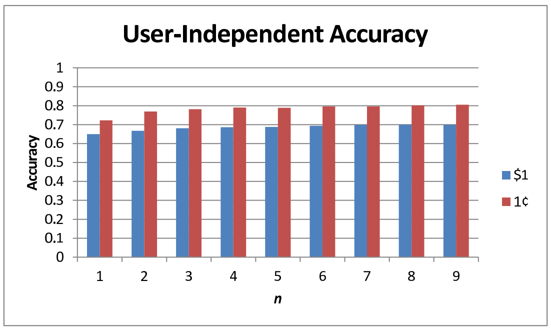

Figure 4: *Average accuracy as a function of the number of training examples (n) for each gesture type for both \$1 and 1¢. In each case the difference between the groups is significant (p < 0.01)*

In the user-dependent scheme, a separate cross-validation is performed for each participant, with testing and training data from only that participant. We performed 18-folds of cross-validation for each participant. In each fold, one example of each gesture type is used for training and the others are used for testing (each gesture was used for training exactly one time). Because each participant drew 180 strokes, there were 170 strokes tested in each fold. Performing 18 folds resulted in 3,060 test recognitions. We averaged the accuracy for each participant across their 18-folds, producing the results in Figure 5.

Figure 6 presents the accuracy for both \$1 and 1¢ for each gesture type. Here accuracy is computed with the user-independent scheme with nine training templates for each

	Induc.	Not	Omega	Rect.	Resist.	Sigma	Sqrt	Star	Stoop	Triang.
Induc.	3245	0	0	0	5	0	26	0	0	0
Not	0	2089	96	435	0	133	22	51	161	289
Omega	3	3	2791	0	238	0	37	0	73	131
Rect.	0	144	33	2822	0	0	5	149	84	39
Resist.	319	0	0	0	2952	0	5	0	0	0
Sigma	153	0	3	0	4	3073	41	0	0	2
Sqrt	533	0	2	0	27	2	2712	0	0	0
Star	0	59	15	325	34	5	0	2776	43	19
Stoop	0	111	1052	374	0	186	11	53	1192	297
Triang.	0	143	182	45	0	104	0	7	63	2732

Table 2: *Per-shape confusion matrix of* 1$^{¢}$ *on the user-independent scheme.*

	Induc.	Not	Omega	Rect.	Resist.	Sigma	Sqrt	Star	Stoop	Triang.
Induc.	3043	0	0	0	27	42	164	0	0	0
Not	33	1217	60	861	23	503	210	59	126	184
Omega	82	36	2755	305	0	13	53	18	1	13
Rect.	0	3	214	2185	0	333	38	0	253	250
Resist.	154	0	0	0	3109	12	1	0	0	0
Sigma	5	1	0	0	0	3260	9	0	0	1
Sqrt	150	73	118	118	0	169	2635	0	0	13
Star	1	680	34	10	191	319	7	1997	25	12
Stoop	0	6	509	1144	0	213	13	6	798	587
Triang.	0	1	264	444	0	338	5	0	316	1908

Table 3: *Per-shape confusion matrix of $1 on the user-independent scheme.*

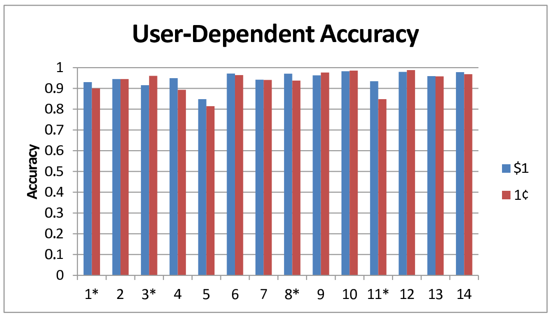

Figure 5: *Accuracy results for each participant under the user-dependent scheme. Accuracy for a participant is averaged across 18-folds of cross-validation. An asterisk indicates that the difference in accuracies of the two methods for a particular participant is statistically significant($p < 0.05$).*

gesture type ($n = 9$). The per-shape confusion matrix of the user-independent evaluation for 1$^{¢}$ and $1 are shown in Table 2 and Table 3 respectively. These tables show how often each shape was recognized as each other shape, which is useful in understanding cases where the two recognizers perform poorly.

	$1	1$^{¢}$	p
$N = 16$	94.8	94.2	0.68
$N = 32$	94.8	93.8	0.55
$N = 64$	94.7	93.4	0.43
$N = 128$	94.7	93.2	0.38

Table 4: *Average accuracy of each technique across varying values of N, the resampling size, using the user-dependent method. The p column represents the p-value of a student t-test of the accuracies of the two techniques.*

Table 4 presents the accuracy of each method across varying values of N. The difference between the accuracy of 1$^{¢}$ for $N = 16$ and $N = 128$ is not significant ($p = 0.61$). Similarly, the difference between the accuracy of $1 for $N = 16$ and $N = 128$ is not significant ($p = 0.94$). Additionally, the difference in accuracies of the two methods for each value of N is never significant ($p > 0.38$). These results demonstrate that the accuracy of either method is insensitive to this parameter and thus any value within this range is acceptable.

We repeated all of the above analyses, rotating the testing templates before recognition for increasing angles of rotation. We found in all such experiments that the rotation angle never affected the accuracy of either $1 or 1$^{¢}$, demonstrating a theoretical rotation invariance. A user study in which participants are asked to draw symbols at varying angles would be required to determine if, in practice, these methods were intolerant of additional transformations caused by drawing gestures at an angle.

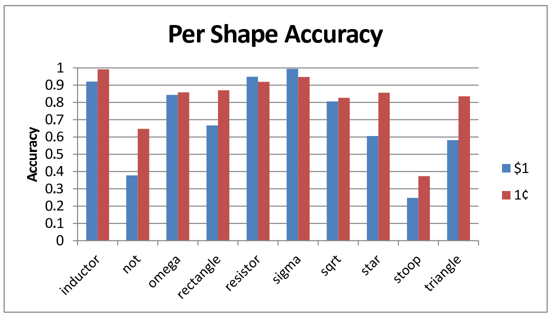

Figure 6: *Average per shape accuracy results from the user-independent scheme with $n = 9$ training templates per gesture type. The difference in accuracy between the two methods is significant for each shape ($p < 0.03$).*

6. Discussion

The user-independent scheme simulates off-the-shelf performance of each recognizer. One might imagine supplying a number of training examples with these recognizers so that a user can deploy them without needing to supply her own training examples. The user-independent evaluation considers the kind of performance that would be achieved in this circumstance. As Figure 4 indicates, user-independent accuracy increases slightly with the number of training templates used. Furthermore, 1¢ performs significantly better than \$1 for this scheme.

Figure 6 provides additional insight into the difference in accuracy of the two recognizers when using the user-dependent scheme. For some gestures, the two recognizers achieve similar accuracy. On others, such as the not and star gestures, 1¢ performs much better than \$1. Empirically, it seems that \$1 often misinterprets gestures that form closed contours as other closed contour gestures, whereas 1¢ is intrinsically well suited for recognizing such shapes.

Table 1 shows the expected result that computation time linearly increases with the number of training templates. Most importantly though, the computation time of 1¢ is consistently two orders of magnitude less than that of \$1.

The user-dependent scheme simulates user-optimized performance for both recognizers. Figure 5 suggests that the two recognizers perform nearly equally well; in only four cases is the accuracy between the two techniques significant, and in three of those cases, \$1 performs better than 1¢.

7. Conclusion

We have presented 1¢, a fast, accurate, and easy-to-implement handwritten gesture recognizer. Our technique applies simple, yet effective techniques adapted from the time series classification literature. The key enabling component of our algorithm is our novel one-dimensional representation of handwritten strokes. This representation is intrinsically rotation invariant, eliminating the need for the expensive search-based angular alignment computation previous template-based gesture recognition techniques have required.

We have evaluated 1¢ on a large database containing 2,520 strokes and compared its accuracy to that of \$1. By applying both user-dependent and user-independent cross-validation schemes, we evaluated accuracy on 75,600 test cases. Our results show that in the user-dependent scheme, both recognizers performed nearly as well as each other. In the user-independent scheme, 1¢ always performs significantly better than \$1. These results suggest that both recognizers are able to take advantage of users' individual drawing styles but 1¢ is more tolerant of drawing styles not encountered in the training corpus. Additionally, our results show that 1¢ is consistently two orders of magnitude faster than \$1. Just as important for developers, 1¢ requires considerably less effort to implement than \$1; 1¢ comprises 37 lines of pseudocode while \$1 comprises 72.

8. Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. 0935239.

References

[CK05] Chen L., Kamel M. S.: Design of multiple classifier systems for time series data. In *Proceedings of the 6th international conference on Multiple Classifier Systems* (Berlin, Heidelberg, 2005), MCS'05, Springer-Verlag, pp. 216–225. 2

[HMS05] Hayashi A., Mizuhara Y., Suematsu N.: Embedding time series data for classification. In *Proceedings of the 4th international conference on Machine Learning and Data Mining in Pattern Recognition* (Berlin, Heidelberg, 2005), MLDM'05, Springer-Verlag, pp. 356–365. 2

[HS11] Herold J., Stahovich T. F.: Speedseg: A technique for segmenting pen strokes using pen speed. *Computers & Graphics 35*, 2 (2011), 250 – 264. 2

[Kar04] Kara L. B.: An image-based trainable symbol recognizer for sketch-based interfaces. In *in AAAI Fall Symposium Series 2004: Making Pen-Based Interaction Intelligent and Natural* (2004), AAAI Press, pp. 99–105. 2

[Rub91] Rubine D.: Specifying gestures by example. *SIGGRAPH Comput. Graph. 25*, 4 (July 1991), 329–337. 2

[SD05] Sezgin T. M., Davis R.: Hmm-based efficient sketch recognition. In *Proceedings of the 10th international conference on Intelligent user interfaces* (New York, NY, USA, 2005), IUI '05, ACM, pp. 281–283. 2

[SSD06] SEZGIN T. M., STAHOVICH T., DAVIS R.: Sketch based interfaces: early processing for sketch understanding. In *ACM SIGGRAPH 2006 Courses* (New York, NY, USA, 2006), SIGGRAPH '06, ACM. 2

[Tap82] TAPPERT C. C.: Cursive script recognition by elastic matching. *IBM J. Res. Dev. 26*, 6 (Nov. 1982), 765–771. 2

[WWL07] WOBBROCK J. O., WILSON A. D., LI Y.: Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes. In *Proceedings of the 20th annual ACM symposium on User interface software and technology* (New York, NY, USA, 2007), UIST '07, ACM, pp. 159–168. 2, 3

[XKS*06] XI X., KEOGH E., SHELTON C., WEI L., RATANAMAHATANA C. A.: Fast time series classification using numerosity reduction. In *Proceedings of the 23rd international conference on Machine learning* (New York, NY, USA, 2006), ICML '06, ACM, pp. 1033–1040. 2

[XKWMN07] XI X., KEOGH E., WEI L., MAFRA-NETO A.: Finding motifs in database of shapes. In *IN PROC. OF SIAM INTERNATIONAL CONFERENCE ON DATA MINING (SDMâĂŹ07* (2007). 2

Appendix - Pseudocode

The following pseudocode has been implemented in JAVA and is available at: https://sourceforge.net/projects/onecentrec/

Step 1 Transforms a candidate stroke into a template. First the stroke is resampled via RESAMPLE, it is then converted to a one-dimensional vector via C-DISTANCE. Last the vector is normalized via Z-NORMALIZE.

```
function RESAMPLE(points, n)
    $I \leftarrow$ PATH-LENGTH(points)/$(n-1)$
    $D \leftarrow 0$
    newPoints $\leftarrow points_0$
    for all point $p_i$ for $i \geq 1$ in points do
        if $D + d \geq I$ then
            $q_x \leftarrow p_{i-1_x} + ((I-D)/d) \times (p_{i_x} - p_{i-1_x})$
            $q_y \leftarrow p_{i-1_y} + ((I-D)/d) \times (p_{i_y} - p_{i-1_y})$
            APPEND(newPoints, $q$)
            INSERT(points, $i, q$)
            $D \leftarrow 0$
        else
            $D \leftarrow D + d$
        end if
    end for
    return newPoints
end function
function PATH-LENGTH(A)
    $d \leftarrow 0$
    for $i$ from 1 to $|A|$ step 1 do
        $d \leftarrow d +$ DISTANCE($A_{i-1}, A_i$)
    end for
    return $d$
end function
function C-DISTANCE(points)                ▷ Equation 4.1.2
    $c \leftarrow$ CENTROID(points)          ▷ Computes($\mu_x, \mu_y$)
    for all point $p$ in points do
        $d \leftarrow$ DISTANCE($c, p$)
        APPEND(distances, $d$)
    end for
    return distances
end function
function Z-NORMALIZE(S)                    ▷ Equation 4.1.2
    $\mu =$ AVERAGE(S)
    $\sigma =$ STANDARD-DEVIATION(S)
    for all $d$ in S do
        $z \leftarrow (d - \mu)/\sigma$
        $APPEND(z, d_z)$
    end for
    return $d_z$
end function
```

Step 2 The following functions comprise all the steps needed to recognize a candidate stroke that has been converted into a template, given a set of training templates.

```
function RECOGNIZE($\boldsymbol{S}$, $\boldsymbol{Templates}$)                ▷ Equation 4.2
    for all template $T$ in $\boldsymbol{Templates}$ do
        $b \leftarrow +\infty$
        $d \leftarrow$ L2($\mathbf{S}, T$)
        if $d < b$ then
            $b \leftarrow d$
            $T^* \leftarrow T$
        end if
    end for
    return $T^*$
end function
function L2($\boldsymbol{S}, \boldsymbol{T}$)                                ▷ Equation 4.2
    $d \leftarrow 0$
    for all $s_i, t_i$ for $i \geq 1$ in $\boldsymbol{S}, \boldsymbol{T}$ do
        $d \leftarrow d + (s_i - t_i)^2$
    end for
    return d
end function
```