EUROGRAPHICS Symposium on Sketch-Based Interfaces and Modeling (2011)
T. Hammond and A. Nealen (Editors)

Continuous Recognition and Visualization of Pen Strokes and Touch-Screen Gestures

P.O. Kristensson[1] and L.C. Denby[2]

[1]School of Computer Science, University of St Andrews, United Kingdom
[2]Cavendish Laboratory, University of Cambridge, United Kingdom

Abstract
We present a technique that enables continuous recognition and visualization of pen strokes and touch-screen gestures. We describe an incremental recognition algorithm that provides probability distributions over template classes as a function of users' partial or complete stroke articulations. We show that this algorithm can predict users' intended template classes with high accuracy on several different datasets. We use the algorithm to design two new visualizations that reveal various aspects of the recognition process to users. We then demonstrate how these visualizations can help users to understand how the recognition process interprets their input and how interactions between different template classes affect recognition outcomes.

Categories and Subject Descriptors (according to ACM CCS): I.5.5 [Pattern Recognition]: Implementation—Interactive systems

1. Introduction

Pen stroke and touch-screen gesture interfaces enable new text entry methods (e.g. [ZK03, KZ04]), new ways of issuing commands (e.g. [AB10, BM08, KZ07, Li09, BZW*09]) and new ways to create sketches (e.g. [ZM06, AN06]). As a result, a variety of recognition algorithms have been developed (e.g. [Li10, AB10, KZ04, WAWL07, Rub91]).

However, a difficult problem with nearly all recognition-based interfaces is that they cause uncertainty among users. Due to noise inherent in human neuromuscular systems and device sensors, users' gestures are bound to be imprecise. Therefore a user's input risks being misrecognized by the system. A particular challenge with recognition errors is that they can appear unintuitive to users, even puzzling (see Kristensson [Kri09] for a recent discussion on this in the context of intelligent text entry methods). It has been observed that users tend to trust systems less if they appear to work mysteriously or behave unpredictably (e.g. [TF99]). Therefore, as a response, researchers have searched for ways to make recognition algorithms more transparent to users.

Long et al. [JLR99] created a gesture design tool based on a statistical recognition algorithm [Rub91]. The tool used confusion matrices to help users understand bottlenecks in the recognition process. Arvo and Novins [AN00] presented a recognizer that continuously detected a few preset shapes from a user's partial trace and then morphed the gesture trace into one of the preset shapes. Bau and MacKay [BM08] presented Octopocus which shows dynamic guides that aid users when gesturing commands. Later, Appert and Bau [AB10] made Octopocus scale-invariant by estimating the scales of gestures *a priori*.

In this work we contribute to the effort in making pen stroke and touch-screen gesture recognition more transparent, and as an effect, perhaps more enjoyable and fascinating to users. We present an algorithm that estimates the posterior probabilities of the user's currently incomplete stroke within a set of template classes. Using this algorithm we can predict a user's intended template gesture based on a partial stroke. It enables us to provide continuous feedback to the user while the user is producing a stroke. We suggest such continuous feedback from the recognizer may help users in understanding how the recognition process works. To this end we demonstrate how we can use the posterior distributions provided by the algorithm to design two new visualizations of the recognition process.

We make the following contributions:

- We present a new template-based incremental recognition algorithm for pen strokes and touch-screen gestures.

- We show that turning angle is a more accurate recognition feature than Euclidean distance. We also demonstrate that by combining both features further small gains in accuracy can be obtained.
- We introduce an end-point bias term and show that this addition to the algorithm is crucial in order for incremental template-based recognition to be as accurate as existing baseline algorithms that recognize complete strokes.
- We describe how the incremental recognition algorithm can be used to design new visualizations that reveal aspects of the recognition process to users.

2. Continuous Recognition

In this section we describe how the recognition algorithm works. The algorithm infers the user's intended template in real-time while the user is producing a stroke.

2.1. Templates and Segments

A template ω_j is a pair (ι, S), where ι is the label (for example *Copy*) and $S = \{\mathbf{S}_i\}$ is a set of segments describing progressively increasing excerpts of the complete template stroke. An individual segment $\mathbf{S} = [\mathbf{s}_1, \mathbf{s}_2, \dots, \mathbf{s}_n]^T$ is a vector of n time-ordered points. The first segment has length l, the second has length $l \times 2$, the next $l \times 3$, and so on, until the last segment describes the complete original template gesture (Figure 1).

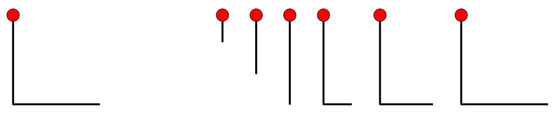

Figure 1: *A complete template (left) and the segments generated from it (right). The red dot indicates the starting position.*

2.2. Recognition

Let $\Omega = \{\omega_k\}$ be the set of templates and let the point vector $\mathbf{I} = [\mathbf{i}_1, \mathbf{i}_2, \dots, \mathbf{i}_n]^T$ represent the complete input pattern with n sample points. We will refer to a partial input of $\mathbf{I}$ with i points $[\mathbf{i}_1, \mathbf{i}_2, \dots, \mathbf{i}_i]^T$ as $\mathbf{I}_i$.

For each new point at index i the system computes the posterior probability for each template $\omega_j \in \Omega$ using Bayes' rule:

$$P(\omega_j|\mathbf{I}_i) = \frac{P(\omega_j)P(\mathbf{I}_i|\omega_j)}{\sum_k P(\omega_k)P(\mathbf{I}_i|\omega_k)}, \tag{1}$$

where $P(\omega_j)$ is the prior probability, $P(\mathbf{I}_i|\omega_j)$ is the likelihood and the denominator is the marginalization term.

If there is no information on which templates are more common than others we use a uniform prior. Otherwise, the prior can be designed to incorporate information about the task. For example, if the recognition algorithm is used to detect gestural commands, certain commands (e.g. *Copy*) are likely more frequently used than others. This information can be provided to the recognition algorithm via the prior. Another example is if the recognition algorithm is used to enable users to enter text. For instance, a letter recognizer such as Graffiti or Unistrokes [GR93] can provide language model information, such as letter frequencies, to the recognition algorithm via the prior. Yet another text entry example is the gesture keyboard [KZ04,ZK03], commercialized as ShapeWriter/Swype/T9 Trace/Flext9. A gesture keyboard enables users to write text by sliding their finger over a touch-screen keyboard. These systems associate touch-screen gestures to individual words. A prior enables such algorithms to use language model information to influence recognition outcomes.

2.3. Likelihood

The likelihood is the probability that partial input $\mathbf{I}_i$ matches a template ω_j:

$$P(\mathbf{I}_i|\omega_j) = P_l(\mathbf{I}_i|\omega_j)E(\mathbf{I}_i|\omega_j). \tag{2}$$

$P_l(\mathbf{I}_i|\omega_j)$ is the likelihood of the user's partial stroke being the best matching partial segment for the template ω_j. This likelihood is found by searching for the segment of the template ω_j that maximizes the likelihood of the distance function D (defined in the next subsection):

$$P_l(\mathbf{I}_i|\omega_j) = \arg \max_{\mathbf{S}_k \in S_j \in \omega_j} D(\mathbf{I}_i, \mathbf{S}_k). \tag{3}$$

$E(\mathbf{I}_i|\omega_j)$ is an end-point detection term we introduced to bias the algorithm towards shorter templates in the case that many templates share similar prefixes. We found that the following formula enabled the best performance:

$$E(\mathbf{I}_i|\omega_j) = 1 + \kappa \exp\left(-\left(1 - D\left(\mathbf{I}_i, \mathbf{S}_{last}\right)\right)^2\right), \tag{4}$$

where $\mathbf{S}_{last}$ is the last segment in the set of segments in ω_j. This last segment represents the complete template. The intuition behind the end-point bias is to take into account whether the user's stroke matches a segment describing a partial template or matches a complete template. If the stroke describes a complete template then it is important that its corresponding template class is prioritized higher than those template classes whose best matches are segments representing incomplete templates.

For example, in Figure 2 the user has articulated a partial stroke (left) that matches either one of two templates' prefix segments, indicated as dashed blue lines (right). Without the end-point bias there is not enough information provided to the recognition algorithm to disambiguate these templates. As a result, the predictions made by the recognition algorithm may fluctuate between both templates arbitrarily. By using an end-point bias, the recognition algorithm can disambiguate among several likely templates so that in the case that the user's input strongly matches a complete template that template will be preferred. There are two free parameters that need to be properly set for this end-point bias to work. The first is κ and the second is one of the variances in the distance function D (described below). These parameters are tuned on a training dataset.

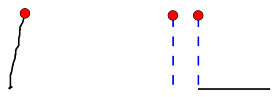

Figure 2: *An example of two segments (right) matching a user's stroke (left). The red dot indicates starting position.*

2.4. Distance Function

The distance function is a combination of the probability estimates of two similarity measures between the point sequences $\mathbf{I}$ and $\mathbf{S}$:

$$D(\mathbf{I},\mathbf{S}) = \exp\left(-\left[\lambda\left(\frac{x_e^2}{\sigma_e^2}\right) + (1-\lambda)\left(\frac{x_t^2}{\sigma_t^2}\right)\right]\right). \quad (5)$$

The first measure (x_e) is the mean Euclidean distance between corresponding points in $\mathbf{I}$ and $\mathbf{S}$. The second measure (x_t) is the mean turning angle between corresponding lines in $\mathbf{I}$ and $\mathbf{S}$. These measures are explained in the next subsection. σ_e and σ_t are variance estimates for both measures. We treat the similarity measures as Gaussians since we assume the sums of point-wise comparisons in the Euclidean distance measure and the sums of line-wise comparisons in the turning angle measure are sums of independent and identically distributed Gaussian random variables. Since the number of summands in the similarity measures tends to be high (since they are related to the number of sampling points) we assume (via the central limit theorem) that the similarity measures themselves are Gaussian.

$\lambda \in [0,1]$ is a mixture weight that controls the relative contribution for both similarity measures. Strictly speaking we can omit λ if we are only interested in the optimal relative contribution of both similarity measures since the ratio of the variances suffices to control this. However, estimating the variance for each feature separately and subsequently finding an optimal mixture weight makes the relative contribution of each feature more transparent and enables us to selectively turn on or off an individual feature without having to retune its individual variance.

2.5. Similarity Measures

There are many choices of similarity measures (or features) for computing a similarity between two point sequences. In this paper we investigate two of the most popular features: Euclidean distance and turning angles. These features have been demonstrated to provide accurate results in a variety of applications (e.g. [KZ04, WAWL07, AB10]).

For both similarity measures, let $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n$ and $\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_n$ be two sequences of points, resampled so that they have an equal number of sampling points.

The first similarity measure is the mean Euclidean distance between all corresponding points:

$$x_e = \frac{1}{n}\sum_{i=1}^{n} ||\mathbf{a}_i - \mathbf{b}_i||. \quad (6)$$

Mean Euclidean distance has been widely used in gesture recognition (e.g. [KZ04, WAWL07]). It requires both point sequences to be normalized. We do this by translating them so that their centroids are at the origin of the coordinate system and by scaling one of them so that the diagonal of the bounding box is unity whilst preserving the aspect ratio.

The second similarity measure computes the mean turning angle between two point sequences using a fixed reference axis:

$$x_t = \frac{1}{n-1}\sum_{i=2}^{n} d_t(\mathbf{a}_i, \mathbf{a}_{i-1}, \mathbf{b}_i, \mathbf{b}_{i-1}), \quad (7)$$

where d_t is the the angular difference in radians for two corresponding line segments that connects the ith and $i-1$th corresponding points.

This similarity measure was originally proposed by Niblack and Yin [NY95] for image-lookup databases and was later used by Appert and Bau [AB10] to predict the scale of gestures for the command selection technique Octopocus [BM08].

2.6. Filtering

We also filter the posterior probabilities with a moving average to make the predictions slightly more stable. We found that a window over the last five predictions provided a good balance between responsiveness and stability.

3. Evaluation

To test the accuracy of the incremental recognition algorithm we set out four research questions. First, does using Euclidean distance and turning angle features result in differences in accuracy? Second, can we obtain a gain in accuracy by combining both of these features? Third, is it possible to enable a template-based algorithm to predict users' intended template classes without sacrificing accuracy when recognizing complete templates? The incremental recognition algorithm performs many more comparisons and therefore risks having an increased number of recognition errors compared to a baseline algorithm that only identifies complete strokes. Fourth, does the introduction of an end-point bias increase the accuracy of the incremental recognition algorithm?

3.1. Method

To answer these research questions we collected in total 2512 gestures from 16 participants recruited from a university campus.

We used a Toshiba R15 Tablet PC with a capacitative pen for collecting stroke data. The 14.1" Tablet PC screen had a resolution of 1024 × 768 pixels and was configured to be in portrait mode.

The stroke sets used were the 16 gestures used to demonstrate the $1 recognizer [WAWL07], ten gestures for particularly difficult-to-recognize words in the gesture keyboard ShapeWriter (now T9 Trace/Flext9) [KZ04], and the 27 Graffiti gestures for the letters A–Z and the space character. We chose these stroke sets because they represent a variety of tasks (input of commands, words, or letters). Participants produced each of the $1 and gesture keyboard stroke sets five times consecutively and each of the gestures in the Graffiti stroke set once. Figure 3 shows these stroke sets.

Before participants produced a stroke they were shown an animation of it so they would know how to draw it. We asked the participants to produce the strokes as quickly and as accurately as possible. No recognition feedback was provided. We then split the collected gestures into two datasets. The first consisted of the first eight participants' strokes and the second consisted of the other eight participants' strokes. We used the first dataset as a training set to tune optimal parameter values for the algorithms. The second dataset was used as the test set for the evaluation of the recognition algorithms.

3.2. Results

We first investigated the accuracy of complete strokes. Table 1 shows the accuracy for the different recognizers. We tested using just one similarity measure (Euclidean distance or turning angle), as well as combining them using optimal mixture weights learned from the training data.

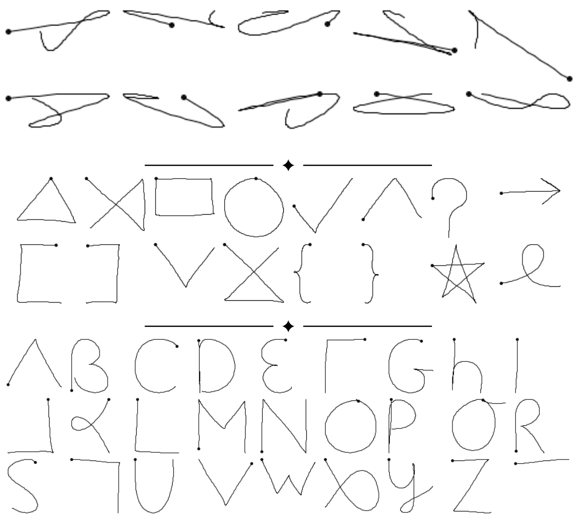

Figure 3: *The stroke sets used for evaluating the algorithms. Top: The gesture keyboard stroke dataset. Middle: The $1 gesture set. Bottom: The Graffiti alphabet.*

Recognizer	Accuracy	Δ
Baseline (E)	91.9%	·
Baseline (T)	94.0%	2.1%
Baseline (E+T)	94.3%	0.3%
Incremental w/o end-point bias (E)	87.2%	·
Incremental w/o end-point bias (T)	91.3%	4.1%
Incremental w/o end-point bias (E+T)	92.1%	0.8%
Incremental with end-point bias (E)	90.8%	·
Incremental with end-point bias (T)	93.9%	3.1%
Incremental with end-point bias (E+T)	94.5%	0.6%

Table 1: *Accuracy and absolute gains in accuracy when recognizing complete gestures. E: Euclidean distance only, T: turning angle only, E+T: combination.*

The baseline was implemented as a standard stroke recognizer, which in our terminology means that each template is only represented by a single segment that consists of the complete template for the stroke. Table 1 shows that regardless of the algorithm, the widely used mean Euclidean distance measure performed notably worse than the mean turning angle measure. Combining the two provided a small but consistent gain for the baseline as well as for both variants of incremental recognition.

Table 1 also shows that introducing an end-point bias was highly effective in retaining the same accuracy as the baseline when recognizing complete strokes. Without the end-point bias, accuracy was consistently lower when performing incremental recognition in comparison to the baseline. The fact that incremental recognition using end-point bias managed to have slightly higher accuracy than the baseline

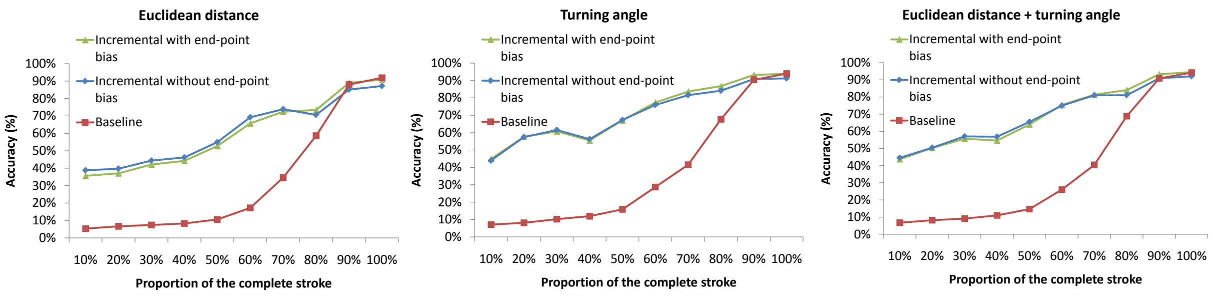

Figure 4: *Accuracy as a function of the proportion of the complete stroke for different combinations of similarity measures. Left: Only Euclidean distance. Middle: Only turning angle. Right: Combination of Euclidean distance and turning angle.*

is impressive considering it had to consider a total of 787 segments compared to only 53 segments for the baseline.

Figure 4 shows accuracy as a function of the proportion of the complete stroke. This provides us with a measure of how well the different recognition algorithms are predicting the user's intended stroke. We used the same parameter settings for the algorithms as in the previous test. The incremental recognition algorithm vastly outperforms the baseline until the user has essentially produced the entire stroke for the complete template. Even when the gesture is complete (i.e. at 100% in Figure 4), the incremental recognition algorithm with end-point bias has slightly higher accuracy than the baseline (0.2%). For the incremental algorithm with end-point bias, the mean accuracy is over 50% after the user has only gestured 20% of the complete template gesture (Figure 4).

4. Visualizations

The incremental recognition algorithm enables us to track how the posterior probabilities for different template classes change as a function of the user's stroke articulation.

Figure 5 plots the posterior probabilities for four template classes as a function of a user's stroke articulation. The bottom of Figure 5 shows the four template classes in the system. The top figure shows how the posterior probabilities vary as the user is gradually producing the complete stroke for template 4. The panels below the plot show versions of the partial stroke at certain numbers of received sampling points.

The plot in Figure 5 demonstrates the effect of the end-point bias in the incremental recognition algorithm. Initially template 1 has the highest probability although all four template classes consist of an almost identical vertical stroke downwards. This is because the end-point bias prioritizes complete strokes over incomplete strokes. However, once the user begins to move right (at the point marked "A" in the plot) the probability for template 1 decreases while the probabilities for templates 2 and 4 increase. Although templates 2 and 4 are both very similar to the user's partially produced stroke, the recognition algorithm prioritizes template 2 as it represents a complete stroke. However, once the user begins to move down (at the point marked "B" in the plot) template 4 becomes the most likely candidate.

Figure 5 demonstrates how the incremental recognition algorithm can provide end-users and software designers with more in-depth knowledge about the recognition process. We now proceed to present two new visualizations that can be integrated into developer tools and interfaces for end-users.

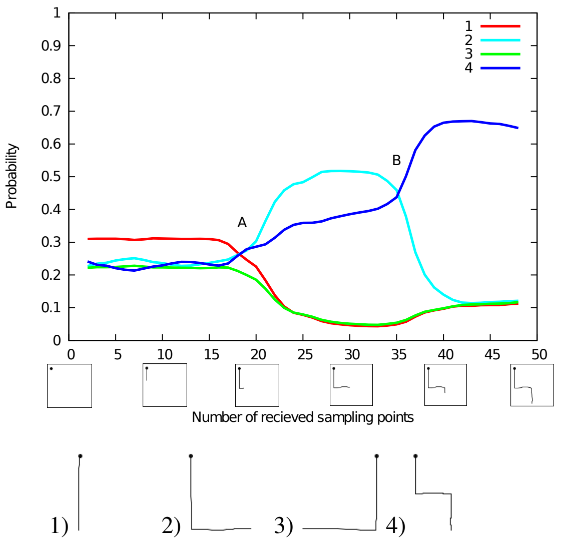

Figure 5: *Above: Probabilities for different template classes as a function of a user's partial stroke for template 4. Below: The four templates. Solid dots indicate starting positions.*

4.1. Linear Prediction

The first new visualization is *linear prediction*. The probability distribution obtained by the incremental recognition algorithm enables us to indicate aspects of the probability landscape at hypothetical points to the user. Linear prediction shows this space in relation to a preset template, such as the current best match, or a particular template indicated by the user.

Figure 6 illustrates linear prediction for a template set consisting of only two templates (shown at the bottom of Figure 6). The user has indicated that the leftmost template should act as the reference template. The top of Figure 6 shows the user's partial gesture in black (the starting point is indicated as a solid dot). When the user pauses, linear prediction begins to sample the probability that the reference gesture would be correctly recognized if the user's next straight stroke segment is at a certain point. If the user continues pausing, the prediction display expands as the algorithm evaluates more and more possible locations. The color of a point indicates whether the reference template would be the best match if the user moved linearly to this position. The color of a point is determined by calculating the proportion of the reference template's probability at a given point in relation to its closest competitor. The most probable points are in green and the least probable points are in red. The visualization in Figure 6 indicates to the user that a stroke to the right will match a partial or complete version of the reference template. If the user moves down the outcome is undetermined. If the user moves left the reference template will not be the top match.

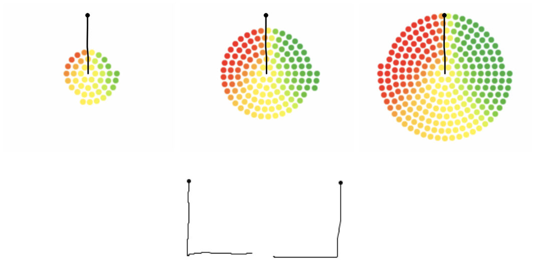
Figure 6: *Above: An example of linear prediction. Below: The templates in the system when the figure above was generated. Solid dots indicate starting positions.*

Figure 7 demonstrates linear prediction visualization on a more realistic dataset. Figure 7 was generated by using the $1 gesture set [WAWL07] (see Figure 3). In the figure the user is attempting to produce a stroke corresponding to the reference template 12 in the $1 gesture set (shown below Figure 7; also cf. Figure 3). As the user is producing the stroke the green and red areas show nearby points that are likely to result in the reference template being accurately and inaccurately classified respectively. When the stroke is completed the visualization indicates to the user that the stroke is complete. The red ring around the user's last sampling point means that if the user continues to produce a stroke, that stroke will not be correctly classified as the reference template, no matter which direction the stroke turns.

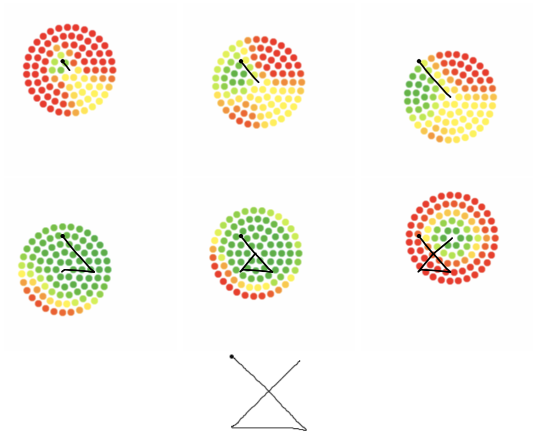
Figure 7: *Above: A real-world example of linear prediction using the $1 gesture set. Below: The reference template. Solid dots indicate starting positions.*

Figure 8 illustrates linear prediction with a varying reference template. Here the system changes the reference template whenever a new template becomes the most probable. In Figure 8 the user is attempting to produce a stroke corresponding to template 12 in the $1 gesture set (indicated below Figure 8 as 12*; also cf. Figure 3). Initially, in the top-left frame, template 2 is the most probable and is therefore acting as the reference template. In the next top frame template 12 (the intended template) is the most probable template and therefore the reference template changes to it. In the top-right frame, template 11 is the most probable template and the reference template changes again. Thereafter the user's intended template 12 becomes the most probable template throughout the rest of the user's production of the stroke. The difference between using static or varying reference templates is evident by comparing the top-right frame from Figure 7 with the top-right frame from Figure 8.

4.2. Preview

The second visualization is *preview*. The incremental recognition algorithm makes it possible to visualize a preview of the current templates that are most likely to correspond to the user's partial stroke. Figure 9 shows the user's trace in black with the starting position indicated by a solid dot. In the beginning, several templates have prefix segments that match the user's partial stroke. The ones that have the highest probabilities are overlaid onto the user's stroke and alpha blended proportionally to their posterior probabilities. The templates that are represented by these overlays are first recognized invariant of scale and translation. We thereafter scale and translate them so that they achieve a best fit in relation to the user's partial stroke. This scale-independence assumption can be observed in Figure 9, in which templates

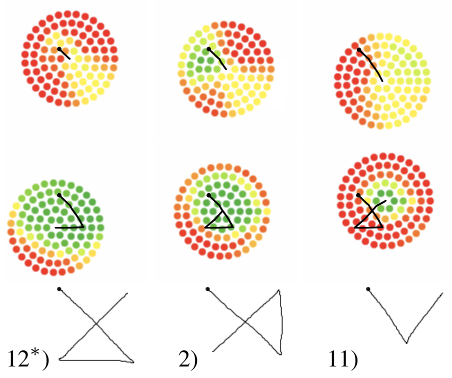

Figure 8: *Above: A real-world example of linear prediction using the \$1 gesture set with a varying reference template that is set to the currently most likely template. Below: The reference templates. Solid dots indicate starting positions.*

are predicted and consequently visualized in different scales to the user.

The overlays are first translated so that the first point of the template corresponds to the first point in the user's partial gesture. We then subdivide the template into partitions divided up according to high curvature points in the template. Recall that for each recognized template a particular segment (call it $\mathbf{S}^*$) matched the user's partial gesture the best. We now find the partition of the template that is the closest to $\mathbf{S}^*$ using the distance function defined earlier in this paper. The best matching partition of the template is then scaled to fit the user's partial stroke. Since we have now obtained the proper scale we can overlay the entire template onto the user's partial stroke.

Figure 10 shows another example of preview. The user is producing the stroke for the letter "M". The stroke set in Figure 10 consists of all the 27 templates in the Graffiti alphabet.

On the surface preview is similar to the scale-invariant command selection technique Octopocus [BM08, AB10]. Octopocus is an interface that uses dynamic guides to help users explore and learn a well-separated command vocabulary. In contrast to Octopocus, our incremental recognition algorithm and accompanying visualization performs a complete pattern match against all templates each time a new sampling point is received from the user. Further, our templates can be arbitrarily defined polylines. Therefore, our technique can be used for purposes other than for issuing commands. For instance, it could be used to enable users to write words using gesture keyboards [KZ04, ZK03] by only articulating partial gestures for words. It could also be used to enable users to enter prefixes of arbitrary free-form gestures or gesture keyboard commands [KZ07]. It may also assist users of certain sketch recognition systems which attempt to recognize individual strokes in a sketch as the user is producing them.

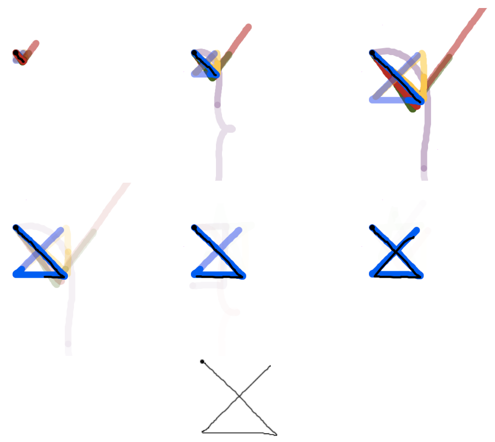

Figure 9: *Above: An example of overlay with correct scale estimation for arbitrary strokes (\$1 stroke set). The black trace shows the user's partial stroke developing during six frames. Below: The template the user is producing a stroke for. Solid dots indicate starting positions.*

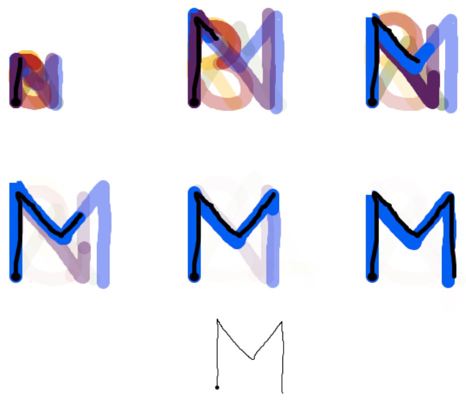

Figure 10: *Above: An example of overlay with correct scale estimation for arbitrary strokes (Graffiti stroke set). The black trace shows the user's partial stroke developing during six frames. Below: The template the user is producing a stroke for. Solid dots indicate starting positions.*

5. Discussion and Conclusions

In this paper we first presented a new probabilistic algorithm for continuous recognition of pen strokes and touch-screen gestures. Our algorithm extends existing template matching algorithms [KZ04, WAWL07, NY95, AB10]. We investigated two popular features, Euclidean distance [KZ04, WAWL07] and turning angle [NY95, AB10]. We found that the turning angle measure consistently resulted in higher accuracy than Euclidean distance (see Table 1). Since many recognizers use the latter measure we urge their designers to reconsider this choice. We also suggest that template-based incremental recognition algorithms should introduce an end-point bias. Our evaluation showed that introducing this bias resulted in incremental recognition being equally accurate as a baseline recognition algorithm that only recognizes complete strokes. This was despite the incremental recognition algorithm having to compare many more subsegments. Without the end-point bias, incremental recognition provided lower accuracy than the baseline (see Table 1).

We presented two new visualizations that take advantage of the incremental recognition algorithm. The first visualization is *linear prediction*. It uses the incremental recognition algorithm to visualize whether the user's ongoing stroke will be recognized as a reference template at hypothetical future positions. It may enable users who design stroke templates to better understand characteristics of the template sets they are designing. It has previously been observed that designers find it very difficult to understand how the recognition process affects the design of a template set [JLR99]. The second visualization is *preview*. It overlays the currently most likely templates that match the user's partial stroke onto the user's pen or finger trace. This technique may be useful to enable users to only articulate a partial stroke for an intended action. For instance, it may enable users of gesture keyboards [ZK03, KZ04] to write faster by only articulating short prefixes of gestures for common words. These are just a few examples of how the incremental recognition algorithm can be applied. We hope the algorithm will find its way into a wide variety of pen stroke and touch-screen gesture interfaces.

The source code for the incremental recognition algorithm is released under an open source license and can be downloaded from:

`http://pokristensson.com/increc.html`.

Acknowledgements

We thank K. Vertanen for his assistance. This work was supported by the Engineering and Physical Sciences Research Council (grant number EP/H027408/1). L.C. Denby's internship was funded by a donation from Nokia.

References

[AB10] Appert C., Bau O.: Scale detection for a priori gesture recognition. In *Proceedings of the 28th ACM Conference on Human Factors in Computing Systems* (2010), ACM Press, pp. 879–882. 1, 3, 7

[AN00] Arvo J., Novins K.: Fluid sketches: continuous recognition and morphing of simple hand-drawn shapes. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology* (2000), ACM Press, pp. 73–80. 1

[AN06] Arvo J., Novins K.: Fluid sketching of directed graphs. In *Proceedings of the 7th Australasian User Interface Conference* (2006), Australian Computer Society, Inc., pp. 81–86. 1

[BM08] Bau O., Mackay W.: Octopocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology* (2008), ACM Press, pp. 37–46. 1, 3, 7

[BZW*09] Bragdon A., Zeleznik R., Williamson B., Miller T., LaViola Jr. J. J.: Gesturebar: improving the approachability of gesture-based interfaces. In *Proceedings of the 27th ACM Conference on Human Factors in Computing Systems* (2009), ACM Press, pp. 2269–2278. 1

[GR93] Goldberg D., Richardson D.: Touch-typing with a stylus. In *Proceedings of the 17th ACM Conference on Human Factors in Computing Systems* (1993), ACM Press, pp. 80–87. 2

[JLR99] Jr. A. L., Landay J., Rowe L.: Implications for a gesture design tool. In *Proceedings of the 17th ACM Conference on Human Factors in Computing Systems* (1999), ACM Press, pp. 40–47. 1, 8

[Kri09] Kristensson P. O.: Five challenges for intelligent text entry methods. *AI Magazine 30*, 4 (2009), 85–94. 1

[KZ04] Kristensson P. O., Zhai S.: SHARK^2: a large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology* (2004), ACM Press, pp. 43–52. 1, 2, 3, 4, 7, 8

[KZ07] Kristensson P. O., Zhai S.: Command strokes with and without preview: using pen gestures on keyboard for command selection. In *Proceedings of the 25th ACM Conference on Human Factors in Computing Systems* (2007), ACM Press, pp. 1137–1146. 1, 7

[Li09] Li Y.: Beyond pinch and flick: Enriching mobile gesture interaction. *IEEE Computer 42* (2009), 87–89. 1

[Li10] Li Y.: Protractor: a fast and accurate gesture recognizer. In *Proceedings of the 28th ACM Conference on Human Factors in Computing Systems* (2010), ACM Press, pp. 2169–2172. 1

[NY95] Niblack W., Yin J.: A pseudo-distance measure for 2d shapes based on turning angle. In *Proceedings of the 2nd International Conference on Image Processing* (1995), IEEE Press, pp. 352–355. 3, 7

[Rub91] Rubine D.: Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques* (1991), ACM Press, pp. 329–337. 1

[TF99] Tseng S., Fogg B.: Credibility and computing technology. *Communications of the ACM 42* (1999), 39–44. 1

[WAWL07] Wobbrock J., A.D. Wilson A., 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* (2007), ACM Press, pp. 159–168. 1, 3, 4, 6, 7

[ZK03] Zhai S., Kristensson P. O.: Shorthand writing on stylus keyboard. In *Proceedings of the 21st ACM Conference on Human Factors in Computing Systems* (2003), ACM Press, pp. 97–104. 1, 2, 7, 8

[ZM06] Zeleznik R., Miller T.: Fluid inking: augmenting the medium of free-form inking with gestures. In *Proceedings of Graphics Interface 2006* (2006), Canadian Information Processing Society, pp. 155–162. 1