Handwritten Gesture Recognition for Gesture Keyboard

R. Balaji, V. Deepu, Sriganesh Madhvanath, Jayasree Prabhakaran

Handwritten Gesture Recognition for Gesture Keyboard

R. Balaji[1]*, V. Deepu, Sriganesh Madhvanath and Jayasree Prabhakaran*

Hewlett-Packard Laboratories, Bangalore, India
{deepuv,srig, jayasree.prabhakaran}@hp.com

Abstract

Gesture Keyboard (GKB) is a novel method of text input for syllabic scripts whose success and acceptance is critically dependent on the reliability of handwritten gesture recognition. In this paper, we describe the solution we have developed for the Devanagari Gesture Keyboard. A data driven approach is adopted for the recognition of basic shapes corresponding to components of gestures. Script specific rules are then employed to combine basic shapes into gestures. These rules are captured externally in an XML configuration file so that the system may be adapted for different scripts easily. Evaluation of the solution using gesture data collected from novice users shows a gesture recognition accuracy of 97% for supported writing styles. The paper concludes with next steps.

Keywords: Gesture Recognition, Text Input, Devanagari, Indic scripts

1. Introduction

Text input of Indic scripts poses a unique challenge, because of the large number of syllabic characters formed by combining consonants and vowel diacritics (called *matras*). The Gesture Keyboard (GKB) solves this problem in a novel way with an inexpensive digitizing tablet peripheral device [1].

The keyboard allows the user to enter isolated vowels, base consonants and some symbols by tapping with the stylus on their respective locations in the layout printed and pasted on the digitizing tablet. For the input of syllabic characters corresponding to consonant-vowel combinations, the user can write (gesture) the corresponding *matras* at the consonant location using the stylus. The gesture is recognized and combined with the base consonant to form the syllabic character as depicted in Figure 1. Since handwriting input is constrained to a relatively small number of gestures, much higher recognition accuracies can be achieved in principle, compared to the unconstrained input of characters or words.

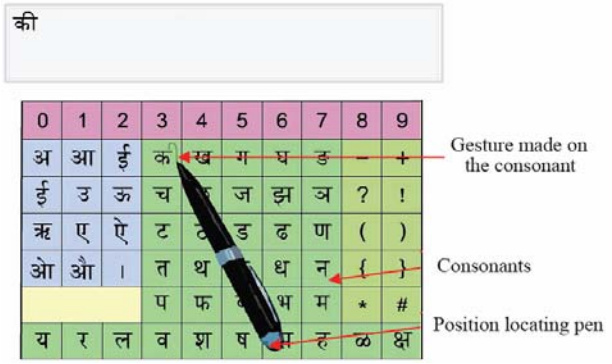


Figure 1. Gesture Keyboard for Devanagari

The Devanagari GKB supports a set of 19 Devanagari *matras* in different writing styles (a total of 23 gestures) and one special gesture (strike-through) as shown as Figure 2. Each gesture is composed of one or more pen strokes and is ideally written at a specific position relative to the glyph of the base consonant. It is well known that with gesture or handwriting based interfaces, recognition accuracy is a critical determiner of success or failure in the marketplace. The objective of our effort was to recognize these 24 gestures with accuracy exceeding 95% - which is the acceptance threshold indicated by user studies[6].

Figure 2. 24 gestures supported by Devanagari GKB

[1] ECE, Purdue University, This work was carried out while the author was in HP Labs India

2. Gesture Keyboard Architecture

The architecture of GKB is shown in Figure 3. The activity of gesture recognition is divided across two major components known respectively as the Controller and the Shape Recognition Engine (SRE). This division is motivated by the observation that the different gestures are in turn composed of different spatial arrangements of a core set of single-stroke shapes which we call *g-strokes*. The *g-strokes* corresponding to the Devanagari gestures of Figure 2 are shown in Table 1. The SRE, recognizes these simple single-stroke shapes, while the Controller assembles the identified *g-strokes* into complete gestures.

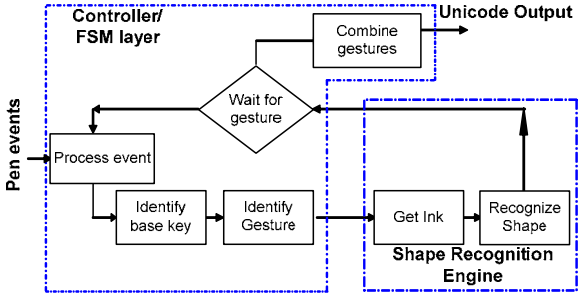


Figure 3. Gesture Keyboard Architecture

While on the face of it, the recognition of a small number of gestures in constrained styles appears to be a simple problem, it is made challenging by a number of issues. Writers seldom write like the ideals shown in Figure 2 and Table 1, with the result that different *g-strokes* are often indistinguishable in shape and position. The inexpensive tablet hardware introduces other sources of error, such as inaccurate registration of pen-down and pen-up events (related to the sensitivity of the pen-tip switch), the unfamiliar and slippery plastic surface, the lack of visual feedback (the pen does not ink!), and errors in sensed position resulting from the writer's grip and resulting tilt of the pen, as well as from the sensing circuitry. There is also ambiguity of another kind - two *g-strokes* made consecutively on the same consonant may correspond to a single two-stroke gesture, or two single-stroke gestures, depending on the time elapsed between them, relative to the speed of writing.

In the following sections, the SRE and Controller are described in detail. However we first describe the critical activity of handwritten gesture data collection and annotation that produced the *g-stroke* data for training the SRE, and for evaluating the reliability of Gesture Recognition.

3. Data Collection and Annotation

Devanagari gesture data was collected from 135 school children between the ages of 7 and 12 from a school in Bangalore. A specially created desktop PC application was used for data collection, along with the standard AceCad® digitizing tablet used by GKB, with all but three consonants at different corners of the layout (cha, na, ya) masked out. After capturing the writer profile (age, gender, handedness, familiarity with Devanagari, and so on), the writer was led through a calibration process aimed at estimating the pen tilt for the specific writer. Subsequently the writer was prompted to write each gesture in turn on each of 3 consonants using the tablet, and the ink logged in the standard UNIPEN format.

The gesture data was manually validated and annotated at the lab using a second tool. The objectives of annotation were (i) to verify correctness of input, and identify noise, spurious strokes & unsupported styles, and (ii) for each gesture sample, capture the style, component ID, and *g-stroke* ID of each constituent stroke. Some of the flags used to annotate *g-strokes* included *Spurious stroke, Wrong Orientation, Noisy, and Unsupported g-stroke*.

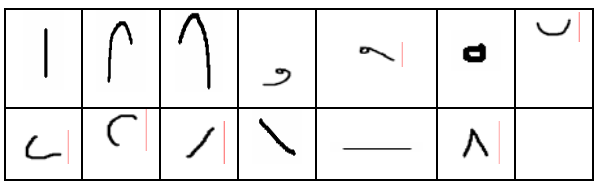
Approximately two-thirds of the gesture data collected (100 writers) was set aside for design and training of gesture recognition algorithms, and the remainder was used for testing and evaluation of accuracy.

Being manual in nature, the process of validation and annotation is prone to human error. Following the first pass of annotation, a clustering algorithm was used to group *g-strokes* with the same label further, based on shape similarity. Singleton clusters found in this process were likely to be noise or mislabeled. Another approach used to find annotation errors was to train the SRE and classify the training data samples themselves. Incorrectly recognized training samples were frequently noisy or otherwise mislabeled. Suspect samples discovered by these processes were manually inspected and reannotated in a second pass.

4. Shape Recognition Engine

As already mentioned, the SRE is intended to recognize single stroke shapes. During the training phase, samples of each pattern class (in this case, *g-stroke*) acquired earlier are preprocessed, and features extracted. A model or reference data (statistical or otherwise) corresponding to each class is computed and stored. During the recognition phase, a test sample is classified as one of the known pattern classes using the stored models.

Table 1. *g-strokes* for Devanagari GKB



In the context of Devanagari gesture recognition, it was found that the supported styles of gestures could be decomposed into a set of 13 *g-strokes* (Table 1). The SRE used in the present version of GKB uses two different recognizers in combination for the recognition of these *g-strokes*. The first uses the actual

(x,y) points constituting the stroke as features, whereas the other uses global features of the stroke as a whole. The two recognizers share many preprocessing operations which are captured in a common preprocessing module.

4.1. Preprocessing

During the preprocessing process, the *g-stroke* obtained from the Digitizer Abstraction Layer as a variable length sequence of (x,y) pen-positions is passed through a series of transformations to remove noise and unwanted variations among samples:

- **Thinning** is the process of removing duplicate points resulting from the user pausing in the midst of writing.
- **Smoothing** is the process of removing random and high frequency noise from the data, resulting from either the capture device or unsteadiness of the hand.
- **Dehooking** is the process of removing "hook" artifacts from delayed registration of pen lifts or slippage of the pen when it is first placed on the tablet.
- **Orientation normalization** is the process by which all strokes, regardless of the direction of writing, are normalized to go in a certain standard direction.
- **Size normalization** is designed to overcome the variability in the size and position of the handwritten shape across writers, and results in all traces being translated to the origin and scaled to a standard size while preserving aspect ratio.
- Finally, **equidistant resampling** resamples each stroke at equal intervals in space along its trajectory, and removes variability from speed variations while writing and across writers**.** The resampling is performed such that a constant number of points are obtained from any trace.

4.2. Subspace-based Recognition

The first recognition scheme used by the SRE is based on Principal Component Analysis (PCA) [2]. In this scheme, the x-y coordinates after preprocessing are used directly as features for classification, resulting in a fixed-length feature vector representing the *g-stroke*.

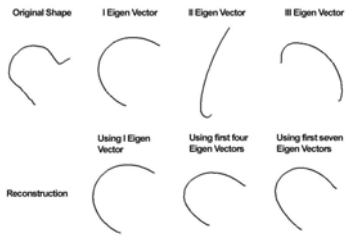


Figure 4. Eigenvector decomposition of a *g-stroke* class

Each pattern class (i.e. *g-stroke*) is modeled statistically by a subspace based on the Principal Components computed from training samples of the class (Figure 4). To obtain the Principal Components, the Eigenvectors of the covariance matrix extracted from the training samples are computed. The Eigenvectors corresponding to the *n* largest Eigenvalues in descending order (where *n* is a tunable parameter) form a subspace (called Eigenspace) representing the class, and are stored as the model for the class.

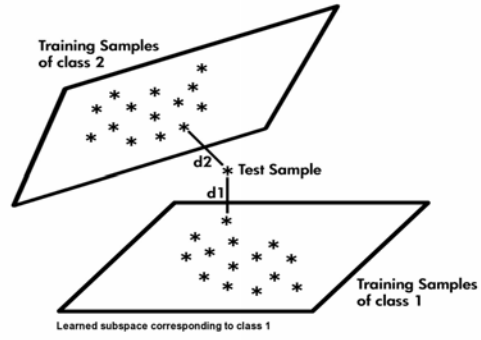


Figure 5. Classification using Eigenspaces

When a new *g-stroke* is to be recognized, the Weighted Mahalanobis Distance [3, 4] of its feature-vector from the Eigenspaces of each of the shapes is computed (Figure 5). The *g-stroke* IDs corresponding to these Eigenspaces are arranged in ascending order of the computed distance from each of these spaces. The top N *g-stroke* ids along with confidence values are returned as the result of shape recognition.

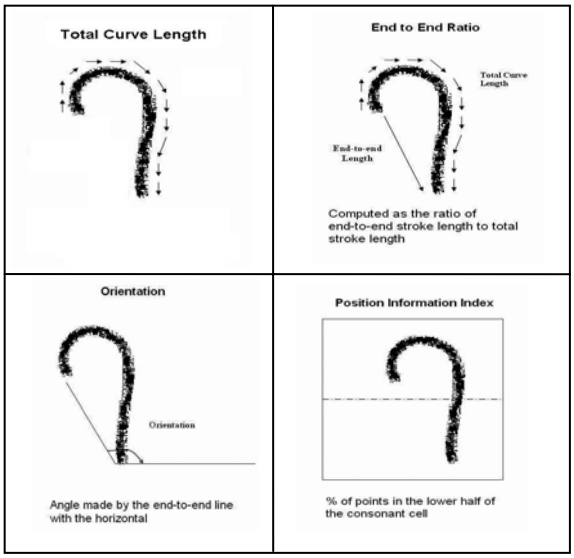


Figure 6. Global features extracted from *g-strokes*

4.3. Global Shape Recognition

The second recognition scheme employed by the SRE uses a set of simple features that characterize the overall shape of the *g-stroke* (Figure 6). Total Curve Length, as the name implies, is the length of the raw *g-stroke* before normalization. End-to-End Ratio is the

ratio of the distance between the stroke endpoints, to the total curve length. Another feature used is the orientation of the stroke with respect to the horizontal. The Position Information Index is the fraction of points in the upper half of the consonant "cell" of the GKB layout. Finally, the total swept angle is computed as the sum of angles subtended by consecutive line segments in the piecewise linear approximation of the stroke.

For a given class, the values of these features are computed from the training samples, and stored as the model for the class. Given a *g-stroke* to be recognized, the global features are computed from the *g-stroke* and the k-nearest neighbors are computed from among the stored prototypes of all the classes, using an Euclidean distance metric. These N most probable classes among these are returned as the output of shape recognition.

4.4. Combination and Final Results

The results from the two shape recognizers are combined using the Borda Count Combination scheme [5], a rank-based combination scheme wherein the final rank for a given pattern class is computed by summing the ranks returned by each of the recognizers. The top N results after rank combination are returned by the SRE as the final result of shape recognition.

The subspace-based shape recognizer, Global shape recognizer, and Borda Count combination leverage the *Lipi Toolkit* [7] – a toolkit for Online Handwriting Recognition created at HP Labs India to simplify the creation of recognizers, and their integration into pen-based applications. The toolkit provides a library of common preprocessing operations, in addition to Makefiles and build scripts for building recognizers as dynamic link libraries. The standard shape recognition interface exposed by these recognizers simplifies their training, evaluation and subsequent integration into the GKB application.

5. Controller

The Controller's role in gesture recognition is one of aggregating and mapping *g-stroke* IDs to the actual gesture IDs depending on their position relative to the cell and other *g-strokes*, and inter-stroke timing information. The Controller is essentially an event processor, and its behavior is determined by a Finite State Machine (Figure 7).

In the figure, each transition is annotated as *Trigger Event/Action(s)*. Further, the notation "^" is used to represent any action that generates an event. The FSM has two states – *Ready* and *Wait*. The machine is initialized to the *Ready* state. On receiving a *Tap* event, the Controller identifies and returns the base consonant corresponding to the cell in the GKB layout on which the tap occurred. However, when the received event is a *g-stroke* event (distinguished from a tap using length, duration and other attributes of the stroke), the *g-stroke* is recognized using the SRE, and the resulting *g-stroke* IDs along with their confidence values are stored into a *trellis*, after applying certain language-specific combination rules.

5.1. Language-specific Rules

These language-specific rules specify how a *g-stroke* may be "refined" into multiple gesture IDs when the same *g-stroke* ID could map to different gesture IDs depending on position. A simple positional classifier is used within the Controller to disambiguate these gestures, based on the distance of the *g-stroke* from the "ideal" gesture position. A weighted combination of the confidences from shape recognition and positional classification is used to compute the final confidence for each hypothesized gesture. However since the computed position of a gesture tends to differ from its "ideal" position because of inaccuracies in writing, as well as the tilt of the pen, position is only given a small weight relative to shape in the weighted combination. The result of this step is a set of *refined* gesture IDs that are distinct in both shape and position. The language-specific rules also specify the valid combinations of *g-strokes*/gestures to form other valid gestures. Since these relations and mappings are script-specific, a standard XML interface is provided for defining them, thereby facilitating adaptation of the GKB for new languages and scripts.

5.2. Trellis

Each column in the trellis table represents the set of possible gesture IDs (interpretations) of the gesture at a given time. A relative position rule is applied to every refined *g-stroke* that forms a part of the multi-stroke gesture before storing it into the trellis table. This is meant to ascertain that the stroke occurs in the correct position relative to the hypothesized multi-stroke gesture. A Timer event is triggered if the stroke is either part of a valid gesture combination, or not the last stroke of the multi-stroke gesture.

On a Timer event, the FSM moves from its Ready state to Wait state, expecting the next stroke of a valid combination (i.e. multi-stroke gesture). If the recognized stroke does not belong to a valid combination, the Timeout event is triggered, signifying the end of input for the current gesture.

At this point the best path in the trellis (corresponding to the most plausible interpretation) is chosen based on the accumulated confidences of the paths in the Trellis. This also causes the FSM to transition to its Ready state and the Controller to return the syllable corresponding to the base consonant + the recognized gesture.

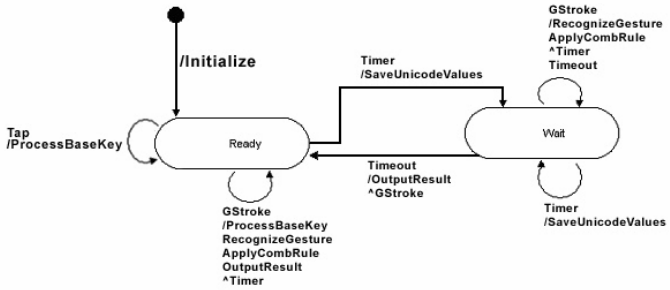


Figure 7. Finite State Machine used by GKB Controller to process events

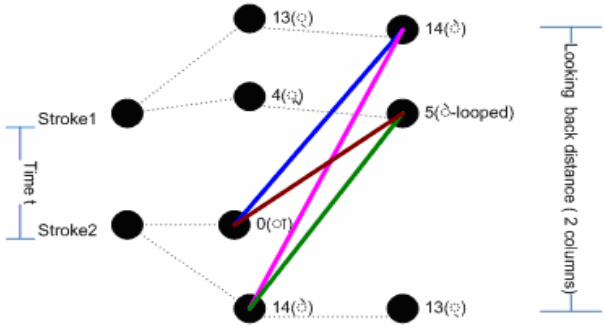


Figure 8. Snapshot of Trellis showing different interpretations corresponding to an "ै (AI)" *matra*

Figure 8 shows the Trellis after the user has gestured an "ै (AI)" gesture in Devanagari script. The dashed lines show the multiple choices returned by the SRE and refined by the positional classifier. The solid transitions are the valid combinations leading to the formation of multi-stroke gestures.

6. Status

A robust implementation of the various components described has been integrated into Gesture Keyboard which has since been licensed and now available as a packaged product. We have evaluated gesture recognition accuracy for the input of Devanagari at both *g-stroke* level (SRE) and gesture level (Controller) on the test data (35 writers). As shown in Table 2, mean top-choice accuracies both for individual *g-strokes* as well as entire syllables are of the order of 97%. Most of the residual errors in gesture recognition result from highly similar training samples of different *g–stroke* classes, and the ambiguity in position as explained earlier. Another source of errors is the continued presence of a small number of mislabeled, noncompliant and noisy data in the training set, which adversely affects the accuracy of subspace-based recognition.

We also evaluated the performance of Linear Discriminant Analysis (LDA). While the *g-stroke* level accuracies are comparable (96.9%), both memory and time complexities are higher than those for subspace recognition Experiments with first time users have confirmed that GKB v1.0 is a usable solution for Devanagari data entry, is readily accepted by the users without the need for much training, and does not cause anxiety on account of recognition accuracy. The GKB architecture is being adapted for two additional Indic languages.with their own scripts – Tamil and Kannada – and initial experiments

Table 2. *g-stroke* and Gesture level accuracies on Devanagari Test data.

g-stroke ID	TOP1 Accuracy (%)	TOP2 Accuracy (%)
0	96.77	99.35
1	100.00	-
2	92.98	100.00
3	94.74	100.00
4	98.32	100.00
5	100.00	-
6	94.55	97.27
7	98.25	100.00
8	98.15	100.00
9	100.00	-
10	94.12	99.16
11	96.83	100.00
12	95.53	98.19
AVG	**97.14**	**99.57**

Gesture ID	Gesture	TOP1 Accuracy (%)
0	Halant	95.12%
1	AA matra	94.94%
2	E matra	96.81%
3	EE matra	93.81%
4	U matra	98.92%
5	Big U matra	100.00%
6	Ru matra	99.00%
7	AE matra	99.02%
8	AI matra	94.74%
9	O matra	95.08%
10	AU matra	100.00%
11	Chandra	97.20%
12	Bindu	98.28%
13	Chandra Bindu	93.90%
14	Aha matra	96.51%
15	Reph	92.78%
16	Half-Ra	93.16%
17	Nukta	98.67%
18	Strike-through	96.67%
	AVG	**96.56%**

indicate that similar accuracies obtained for these scripts

7. Summary and Next Steps

In this paper, we described our solution for the creation of highly accurate gesture recognition for the Gesture Keyboard, initially described in [1]. The next steps are oriented towards improving accuracy to even higher levels, extending support to more natural styles of writing, and demonstrating the generality of the solution on other scripts.

To obtain higher levels of accuracy especially in the presence of more writing styles (and hence more *g-stroke* classes), we have started to explore pairwise classification based on Support Vector Machines, and other features in addition to the ones described. Another important research direction is writer adaptation, which in certain usage scenarios has the potential to significantly improve accuracy, while allowing greater flexibility in terms of allowed writing styles. A related investigation is into less data-intensive recognition schemes with a view to reducing the scope and burden of manual data collection and annotation required for high accuracy.

A major reason for misrecognitions yet to be adequately addressed is the influence of the tilt of the pen on the recorded pen position. In the present GKB, a simple two-point calibration method is used to model the way the user is holding the pen and correct for pen tilt. This needs to be revisited and improved upon, from the perspectives of both usability and accuracy.

Finally, there is also the need for a scheme for rejection of unsupported gestures based on recognition confidences and other attributes. The present implementation tends to return the nearest match among the supported gestures, and a rejection mechanism would allow improved feedback to the user regarding unsupported styles, and consequently faster adaptation of the user to the styles that are supported.

Acknowledgements

We gratefully acknowledge the contributions to the Gesture Keyboard project from various colleagues - too numerous to list here - from HP Labs, HP GDIC, and other collaborating organizations. Significant contributions to the work described here came from Mudit Agrawal, Thanigai Murugan and Venkata Rajesh M. We also thank Shekhar Borgaonkar and A. Prashanth of HP Labs for the original GKB concept, and subsequent technical contributions and inputs.

References

[1] Ashish Krishna, Rahul Ajmera, Sandesh Halarnkar and Prashant Pandit, Gesture Keyboard - User Centered Design of a Unique Input Device for Indic Scripts, *HCI International-2005*, Las Vegas, Nevada, USA, July 22-27, 2005.

[2] Deepu V. and Sriganesh Madhvanath, Principal Component Analysis for Online Handwritten Character Recognition, *17th Intl Conf. Pattern Recognition (ICPR 2004)*, Cambridge, United Kingdom, August 23-26, 2004.

[3] H. Mitoma, S. Uchida, and H. Sakoe, Online Character Recognition Using Eigen-Deformations, *9th International Workshop on Frontiers in Handwriting recognition (IWFHR–9)*, Tokyo, Japan, October 26-29, 2004 pp 3-8.

[4] Richard O. Duda, Peter E. Hart and David G. Stork, *Pattern Classification* , John Wiley &Sons, NY, 2000.

[5] Tin K.H., Jonathan H. and Sargur N. Srihari, Decision Combination in Multiple Classifier Systems, *IEEE Transactions of Pattern Analysis and Machine Intelligence*, 16(1), Jan 1994, pp 66-75.

[6] Mary LaLomia, User Acceptance of Handwritten Recognition Accuracy, *Conference Companion on Human Factors in Computing Systems*, Boston, USA, April 24 - 28, 1994 pp 107-108.

[7] Sriganesh Madhvanath, Deepu Vijayasenan and Thanigai Murugan Kadiresan, LipiTk: A Generic Toolkit for Online Handwriting Recognition,. *10th International Workshop on Frontiers in Handwriting Recognition (IWFHR-10)*, La Baule, France, Oct 2006