delim $$

On-Line, Alphabet-Independent, Gestural Recognition Using Probabilistic Properties

Gary Newell
Department of Computer Science
University of Arizona, Tucson AZ 85721

Abstract

We present an approach to alphabet-independent gestural recognition which differs from existing techniques such as curve-matching in that it attempts to approximate the functionality of the feature-based approaches while allowing the advantages which result from alphabet independence. Our approach uses a probabilistic model to analyze an alphabet based on user supplied samples and to effectively describe the symbols as a collection of properties or attributes which are either absent or present in the particular symbol with a known probability. These properties are then used to test an unknown symbol and a simple cost function is used to determine the recognition result.

KEYWORDS: Gestural Recognition, Feature Analysis, Probabilistic Recognition.

1 Introduction

Many existing gestural recognition techniques assume that the designer has a prior knowledge of the alphabet in question. These algorithms are often feature-based approaches which attempt to encode the symbols of the alphabet as a set of attributes or features which are present in the symbol (e.g. closures, cusps, dots, etc.) [1,3,4,8,11,12,16,26]. Other existing techniques tend towards less alphabet dependence, such as the signal processing techniques, in particular curve-matching algorithms [6,18,27,28]. Although often relatively effective, these techniques may require extensive computation and may also result in extended program tuning in order to distinguish between particularly ambiguous symbols.

Feature analysis algorithms for the recognition of hand-drawn symbols have the advantage of allowing the algorithm designer to select specific features on which to base the recognition of an unknown symbol. She may choose to include tests which are effective in distinguishing only a very small subset of the given alphabet if ambiguity amongst those symbols is detected either in advance of the original design or later during the testing phase of development. These techniques also have the advantage of being easily modeled for computation, often in the form of a binary decision tree or as a finite state machine [8,11,16,26]. However, this required prior analysis has obvious drawbacks as well. For example, this approach is frequently alphabet dependent and thus requires a designer who is not only familiar with the purpose of the given alphabet but who is most likely experienced in its use. This assumption of knowledge also hinders quick prototyping and thus testing of gestural interface systems.

As a result of a continuing research effort to design a system that would integrate a variety of different recognition techniques in order to support the quick prototyping, performance analysis, and automatic generation of recognition algorithms for on-line gestural recognition algorithms, we examined the possibility of designing an algorithm which could approximate the behavior of feature recognition while at the same time being alphabet independent. Our system requires that all techniques be able to operate in an alphabet independent manner and be able to extract enough information from a limited training process to support their effective operation. These requirements are met by curve-matching algorithms. Also, many temporal-based approaches [9,11,12,22] can be easily adapted to operate on user-supplied training data (e.g. pen-tip motion, sequence of zones, sequence of direction changes). These requirements, however, are not as easily addressed in the case of feature-based techniques with their inherent assumption of knowledge with respect to the given alphabet.

For example, suppose that our alphabet consisted of the digits 0..9 and we wished to design a feature analysis algorithm to distinguish all of the symbols effectively. We could examine the alphabet and design a decision tree which would be adequate for effective recognition. But how can we do this quickly and effectively without examining the alphabet in advance? In particular, how can we produce a set of features which is adequate to distinguish the entire alphabet without some foreknowledge of the features actually present in the collection of symbols?

2 Property-based Recognition

Our algorithm defines a fixed set of *properties* which are attributes that are either absent or present in any particular instance of a drawing of some particular symbol of the alphabet. For example, one property might be the presence or absence of an intersection of strokes in the symbol. Another might be whether or not there are more than 3 such intersections in the symbol as drawn or whether the symbol contains a dot or not. Many properties are actually classes of attributes. That is, they are a set of properties whose size is dependent upon the data entered during the training stage of algorithm execution. For example we might examine whether or not a symbol's height to width ratio is greater than some variable parameter whose value or values are set based on the analysis of the data entered by the user in training. Figure 1 presents the data gathered for the height/width ratio of an alphabet consisting of 10 symbols. This data consists of the mean ratio along with a value for the standard error on the ratio measured over all user drawn samples. When examined for clustering amongst the distribution, three distinct intervals are identified and the result is three distinct property tests to determine within which cluster an unknown symbol lies. This same height/width ratio could result in a number of intervals other than three for a different alphabet or even for the same alphabet as drawn by a different user.

There are some obvious problems which arise from this approach to predefining the properties to be tested on a given alphabet. Since all of the properties and property classes are defined a priori (only the parameters to some specific properties will change as described above) there are likely to be some properties which are either consistently useless in distinguishing a particular alphabet or extremely inconsistent for a particular symbol. We may have a property which holds for all symbols all of the time and thus does nothing to help us recognize an unknown symbol. Such a property must be identified and its effect on recognition should be eliminated or minimized. It is also possible that we may have a property $PR sub i$ which is found to be present in a symbol 50% of time and absent 50% of the time. This property is in effect useless in distinguishing the symbol, as its presence or absence is seemingly unpredictable. It is due to these problems that we chose to use a probabilistic approach to recognition with properties.

We consider each property not as a deterministic binary attribute which is either present or absent in a symbol but rather as an attribute which occurs in an instance of a drawn symbol with some given probability. This probability is determined by testing user drawn samples of the symbols against the properties and determining the likelyhood that a property will hold for the given symbol. Figure 2 shows an example of the symbol "A" as drawn 10 times, along with the number of intersections present in each instance of the symbol. We can see that the user has drawn the symbol with 2 intersections 8 out of the 10 times. Thus we would classify the symbol "A" as one for which the property *"Contains Two Intersections"* has probability 0.8 of holding true and probability 0.2 of not being present.

We can now build a table indexed by the symbols of the alphabet as rows and the predefined properties as columns and assign each cell of the table a probability that the indexed property holds for the indexed symbol, based on the examination of training data. If a cell in this *Property Table*, $Prop\_table[S_i , PR_j]$ were to contain the value 0.85, then it indicates that symbol S_i was drawn with property PR_j present 85% of the time. In an effort to avoid confusion, figure 3 presents a small property table representing a user's rendering of a 6 symbol alphabet with 5 active properties. As can be seen from the table, some properties are consistent in their presence or absence from a given symbol while others are not. For example, we can see that the SYM_2 was drawn with property PR_2 occurring 100% of the time and that SYM_1 was never drawn with property PR_4. On the other hand, we can see that SYM_0 was drawn with property PR_0 only 50% of the time.

As was mentioned earlier, we may find that a particular property is not effective in distinguishing the symbols of a given alphabet. That is, the property may tend to be present in almost all training samples of all symbols or it may be absent from almost all of the samples. As we shall see, this problem becomes important when we attempt to determine the correct recognition result. For this reason we attempt to get a measure of the properties ability to distinguish the given alphabet by taking the standard deviation of each column of the table and associating this weight with the particular property. These column weights, W_j are represented in figure 3 in parenthesis under the column titles. Those columns with a low standard deviation are less likely to be effective in discriminating between symbols of the alphabet, while those with higher standard deviations tend to partition the alphabet in a more effective manner.

The other problem which we identified was that of a property which, for a particular symbol, is effectively useless. That is, it occurs or fails to occur about half of the time and thus cannot be used to accurately predict whether or not the symbol matches the unknown. We address this problem by designing our recognition calculations to effectively eliminate or lessen the input of such a property for the particular symbol in question.

Both of these problems are illustrated in the property table of figure 3. In this table, we can see that the column for property $PR sub 1$ indicates that all of the symbols contained the property in every sample. Thus when an unknown symbol is entered, the presence or absence of property $PR sub 1$ gives us no new information with respect to which symbol is likely to match the unknown. If we examine the row of the property table associated with SYM_0, we note that property $PR sub 0$ occurs 50% of the time the symbol is drawn. Although this property is an effective tool for measuring the other symbols, it is of no use for SYM_0 as its absence or presence in the symbol is equally likely.

Once the property table is complete and the column weights are calculated, we need only determine a process by which, after an unknown symbol $S sub u$ is entered by the user, the appropriate alphabet symbol is selected for recognition. The approach that we have taken is to evaluate a *cost* function for each symbol of the alphabet based on the likelyhood that the symbol matches the given unknown. In order to carry out this calculation we first examine the unknown to determine which properties are present in it as well as which properties are absent from it. When this information is complete, we can calculate $cost(S sub i)$ for all symbols in the alphabet as follows:

```
# Initialize Costs to 0.0
  forall $S sub i$ in alphabet do
     cost[i] := 0.0

  forall properties $PR sub j$ do
     forall symbols $S sub i$ do
       if $PR sub j$ is present in unknown symbol $S sub u$ then
           cost($S sub i$) <- cost($S sub i$) + (prop_tab[$iˆ,ˆj$] - 0.5) * $W sub j$
       else
           cost($S sub i$) <- cost($S sub i$) + (( 1.0 - prop_tab[$iˆ,ˆj$]) - 0.5) * $W sub j$
```

After the cost function has been evaluated, we simply select the symbol with the highest result as having matched the unknown $S sub u$. The property weights $W sub j$ are used to both increase the impact of properties found to have a high level of distinguishability and to lower the impact of those which do not. One can see that a property found to be present in all instances of training samples for all symbols of the alphabet would result in a weight of 0 (the standard deviation) and thus effectively eliminate the impact of that particular property in the calculation. Also, if a property is found to be present in a symbol at a rate of 50% of the time the cost increment again evaluates to 0 and the property is effectively ignored for the symbol under consideration. The result of this calculation is that properties which distinguish the alphabet well and also occur with ether a high probability or a low probability in a particular symbol have the largest impact on the symbol's final cost value. Those properties which yield little information have only a slight effect on the symbol's final cost value. Figure 4 presents an example of the cost calculations for an unknown symbol from the 6 symbol alphabet used to create the property table and property weights of figure 3. The cost calculations of figure 4 assume that the user has entered a symbol $S sub unknown$ which was found to have properties $PR sub 0 ˆ,ˆ PR sub 1$ and $ PR sub 2$. However, properties $PR sub 3$ and $PR sub 4$ failed to hold for the symbol. After the costs for all symbols have been calculated, SYM_4 is chosen as the recognition result.

3 Properties

Properties are defined as attributes of a symbol which are either present or absent in any particular hand-drawn instance of the symbol. *Property tests* examine the data of an unknown symbol entered by the user and determine if these binary properties are present or absent in the unknown.

We are continuing to experiment with new property types and expect that further examination of a variety of statistical analysis techniques will result in a number of useful properties which have yet to be considered. We also hope to explore the area of *functional attributes* [5,17,19] in an effort to increase the number of property classes. Currently we are working with a basic set of properties as listed in figure 5. The prototype implementation of this algorithm described in the next section uses only a subset of these properties but extensions are currently under way. We are particularly interested in examining properties which we feel have yet to be explored adequately in the area of online gestural recognition. For example, extending the usual 3 dimensions of data (x coordinate, y coordinate, time stamp) to include pen-tip pressure information may prove to be an effective tool in distinguishing some symbols in a user-dependent recognition system. Acceleration patterns of the pen-tip may also be a useful tool for a user-dependent system. Such pressure and acceleration information has proven to be useful and effective in signature-verification systems [2, 14].

As our collection of properties grows, so does the resulting computational effort in calculating the recognition result. Thus we may wish to identify properties which are not effective and eliminate them from any consideration in the recognition process. This would be a relatively easy process to carry out as it can be done after the training stage and before any actual recognition begins. This does, however, present a problem in that experience indicates that users tend to adapt and/or alter their writing style to a particular system and thus a user may find that, for example, using 3 strokes to draw the symbol "A" rather than 2 results in better recognition rates. If we preserve the entire set of properties we allow more freedom for the user to make changes in her style and yet still find the system effective. This issue of user-adaptability was also a focus of this project. We have implemented the algorithm in an attempt to allow the system to adjust to changes in the user's writing style. This is done by updating the property table after each correct recognition or after the correction of a misrecognition. This allows the system to improve over time as the user adapts to both the interface and the hardware.

One way to achieve this goal is to attempt to use only properties which require relatively little computational effort [25]. We have attempted to limit all property tests to computational complexities which will allow us to combine a large number of these relatively small tests. The majority of tests are of linear complexity with respect to either the number of points in the unknown symbol or to the number of stroke segments in the symbol. This limitation should allow for a large number of properties (necessary for larger alphabets) while still allowing effectively real-time response rates for recognition.

4 Prototype Implementation and Experimental Results

The current prototype implementation is written in X-Icon, an X Window System extension to the Icon programming language [15], and runs on a SUN Sparcstation 1. Handwritten data is entered on an opaque Wacom 420D tablet using a cordless, pressure sensitive stylus.

a)

b)

Figure 6. a) True Strokes with Raw points. b) Segmented strokes.

The prototype algorithm uses only 6 property classes: True stroke count, segmented stroke count, center of gravity, height to width ratio, intersection counts, and closed-region counts.

A *true stroke* is defined as the sequence of points from pen-down to pen-up.

A segmented stroke is a sequence of connected line segments which approximates the original true stroke curve. Figure 6 gives examples of true strokes and the corresponding segmented strokes. The particular segmenting algorithm used in the prototype is an extended version of a technique described by Dannenburg and Amon [7] which combines their approach and a variety of other filtering techniques to produce a smoothed segment sequence with de-hooking and dot reduction [29].

In order to use the center of gravity of the symbol as an effective property, we divide the bounding box of the symbol into 5 areas and determine in which area the center of gravity lies. This has proven to be a relatively consistent and useful property in testing.

The height to width ratio tends to be a consistent property but one which often has a minimal ability to distinguish symbols within a given alphabet. For example, the number "1" is easily distinguished from the other digits using this value but the standard error measure results in the other digits tending to have over-lapping ranges of acceptable values for this property.

We detect and count an intersection every time the pen-tip crosses a line previously drawn in the symbol. Upon detection of an intersection, we determine if the intersection has resulted in the creation of one or more closed regions in the symbol.

As was mentioned earlier in this paper, this algorithm was designed to be used as a component algorithm in a much larger system currently under construction (tentatively titled the *INCA* system) which attempts to integrate the results from a variety of distinct recognition techniques into a single recognition result. Initial experimental results from the system indicate that unlike algorithms which are expected to stand alone in recognition, those used as composite algorithms can be useful and effective with recognition rates of well below those of traditional techniques. With this in mind we tested the prototype algorithm on a variety of alphabets ranging in size from 8 to 15 symbols using 60 samples of each symbol from 3 separate users. The results are listed in figure 7. Testing was carried out by using the first 20 samples of a symbol as training data and the final 40 samples for testing recognition. The alphabets consisted of the digits 0..9, 15 lower-case characters, and a set of 8 mathematical symbols.

The current implementation does not perform its own inter-symbol segmentation but rather relies upon an explicit signal to mark the end of one symbol and the start of the next. This is a design choice based on the fact that the algorithm is intended to be placed into a much larger system where segmentation is already performed for all of the component algorithms.

In order to support the continual update of the property table, misrecognition is handled using the *Tap-Correction* approach described by Goldberg and Goodisman [10]. This approach has proven very effective, as the correct symbol is found to have at least the second highest $cost()$ result in over 91% of our test cases.

When a symbol has been recognized correctly or after misrecognition has been identified and corrected, the algorithm uses a Bayesian approach to update the property table to ensure that it reflects the algorithms performance accurately at any point in time. This allows the algorithm to adapt to the user and improve its performance.

5 Conclusions and Future Work

We feel that our experimental results are promising and offer an indication that this technique may be an effective approach to on-line recognition of hand-drawn symbols. We expect that as the quality of our property tests improve and as we extend and improve the cost function we may see results which are comparable to a variety of existing techniques. In particular, we expect that with little extension to the current system, we will be able to achieve our initial goal of approximating the functionality of feature analysis in an alphabet-independent manner which can be used in the INCA system to assist in recognition.

We are currently underway in the implementation of this algorithm in C++ to run with ARTKIT (Arizona Retargetable Toolkit) [13] in order to allow it to be introduced into the larger on-line recognition system. A number of new properties and variations of the cost function are also being implemented. We believe that the addition of new properties is essential to this technique if it is to succeed on alphabets with more than 15-20 symbols, but these properties must be carefully designed so as not to lose the real-time recognition that is essential to a quality gestural interface.

We are also examining the possibility of using *extended properties*. These properties simulate entire recognition techniques, such as curve-matching, temporal analysis, or other approaches, in an effort to design algorithms of these types which are likely to be more robust at an early stage of design due to their composition with simple properties that will address the few ambiguities that often occur in initial implementations of these techniques. Because of the inherent complexity of many recognition techniques, it is essential to carefully select the simple property tests with which to combine them, in order to preserve real-time recognition.

References

[1] AMIN, A., KACED, A., HATON, J., MOHR, R. "Hand written Arabic character recognition by the I.R.A.C. system", *Proc. 5th Int. Conf. on Pattern Recognition*, 1980, pp.729-731.

[2] AMMAR, M., YOSHIDO, Y., FUKUMURA, T., "A New Effective Approach for Off-Line Verification of Signatures by Using Pressure Features", *8th Intern. Conf. on Pattern Recognition,* IEEE, Oct. 1986, pp.566-569.

[3] BERNSTEIN, M. "A method for recognizing handprinted characters in real-time". in L.N. Kanal, editor, *Pattern Recognition,* Washington DC: Thompson, 1968, pp.109-114.

[4] BERTHOD, M., AHYAN, S. "On-line Cursive Script Recognition: A Structural Approach with Learning", *Proc. 5th Int. Conf. on Pattern Recognition*", 1980, pp.723-725.

[5] BLESSER, B., KUKLINSKI, T., SHILLMAN, R. "Empirical tests for feature selection based on a psychological theory of character recognition", *Pattern Recognition*, vol. 8, pp.77-85, 1976.

[6] BURR, D.J. "Designing a handwriting reader", *Proc. 5th Inter. Conf. on Pattern Recognition*". 1980, pp. 715-722.

[7] DANNENBURG R. B., AMON, D. "A Gesture Based User Interface Prototyping

System". *UIST'89*, 1989, ACM, pp.127-132.

[8] FRISHKOPF, L., HARMON, L. "Machine Reading of Cursive Script", in C. Cherry editor, *Information Theory (4th London Symposium)*, London England: Butterworths, 1961, pp.300-316.

[9] GAINES, B., MCKELLAR, I., DINGER, W., FAST, S., FOWLES, B., FRACCARO, M., JOLIVET, G., MALUDZINSKI, A. "Some Experience in the Real-Time Processing of Handwriting". *7th Inter. Conf. on Pattern Recognition*, IEEE, Vol. 1, 1984, pp.630-634.

[10] GOLDBERG, D., GOODISMAN, A. "Stylus User Interfaces for Manipulating Text", *UIST'91*, ACM, pp.127-135.

[11] GRONER, G.F. "Real-time Recognition of Handprinted symbols", in L.N. Kanal, editor, *Pattern Recognition*, Washington DC: Thompson, 1968, pp.103-108.

[12] HANAKI, S., YAMAZAKI, T. "On-Line recognition of handprinted Kanji characters", *Pattern Recognition*, vol. 12, 1980, pp.421-429.

[13] HENRY, T., HUDSON, S., NEWELL, G. "Integrating gesture and snapping into a user interface toolkit". *UIST'90*, 1990, ACM, pp.112-122.

[14] HERBST, N., LIU, C. "Automatic Verification of Signatures by means of acceleration patterns", *Proc. IEEE Comp. Soc. Conf. Pattern Recognition and Image Processing*, June 1977, pp.331-336.

[15] JEFFEREY, C.L., "X-Icon: An Icon Window Interface". Tech. Report TR 91-1c, Dept. of Computer Science, University of Arizona, Tucson Arizona, 85721, January 1991.

[16] KIM, J. "Gesture Recognition by Feature Analysis", Tech Report RC12472, IBM T.J. Watson Research Center, IBM Corp., PO BOX 218, Yorktown Heights, NY, 10598, Jan. 1987.

[17] KONDO, S., ATTACHOO, B. "Model of Handwriting Process and its Analysis", *8th Inter. Conf. on Pattern Recognition*, IEEE, Oct. 1986, ' pp.562-565.

[18] KRUSKAL, J. "An overview of sequence comparison: time warps, string edits, and macromolecules", *SIAM Review*, vol. 25, April 1983, pp.201-237.

[19] NAUS, M., SHILLMAN, R. "Why is a Y not a V: a new look at the distinctive features of letters", *Jour. of Experimental Psychology: Human Perception and Performance*, vol. 2, 1976, pp.394-400.

[20] PAUSCH, R., WILLIAMS, R.D. "Tailor: Creating Custom User Interfaces Based on Gesture", *UIST'90*, ACM, pp.123-134.

[21] PITTMAN, J.A. "Recognizing Handwritten Text", *CHI'91 Conference Proceedings*,1991, ACM, pp.271-275.

[22] POWERS, V. "Pen Direction Sequences in character recognition", *Pattern Recognition*, vol. 5, 1973, pp.291-302.

[23] RHYNE, J.R., WOLF, C.G. "Gestural interfaces for information processing applications". Tech. Report RC12179, IBM T.J. Watson Research Center, IBM Corp., P.O. BOX 218, Yorktown Heights, NY 10598, Sept. 1986.

[24] RHYNE J. "Dialogue Management for Gestural Interfaces", Tech. Report RC12244, IBM T.J. Watson Research Center, IBM corp., P.O.BOX 218, Yorktown Heights, NY 10598, Oct. 1986.

[25] RUBINE, D. "Specifying Gestures by Example". *Computer Graphics* Vol. 24, No. 4, 1991, pp.329-337.

[26] SHRIDAR, M., BADRELDIN, A. "A Tree Classification Algorithm for Handwritten Character Recognition", *7th Int. Conf. on Pattern Recognition*, IEEE, August 1984, pp.615-618.

[27] TAPPERT, C.C., KURTZBERG, J. "Elastic matching for handwritten symbol recognition", *Proc. IBM Int. Conf. Image Processing and Pattern Recognition*, 1978.

[28] TAPPERT, C.C. "Cursive Script Recognition by Elastic Matching". *IBM Research and Development*, Vol. 26, No. 6, Nov. 1982, pp. 765-771.

[29] TAPPERT, C.C., SUEN, C., WAKAHARA, T. "ON-line Handwriting Recognition - a survey", Tech. Report RC14045, IBM T.J. Watson Research Center, IBM Corp., PO BOX 218, Yorktown Heights, NY 10598, DEC. 1987.