CALI: An Online Scribble Recognizer for Calligraphic Interfaces

Manuel J. Fonseca and **César Pimentel** and **Joaquim A. Jorge**
Department of Information Systems and Computer Science
INESC-ID/IST/Technical University of Lisbon
R. Alves Redol, 9, 1000-029 Lisboa, Portugal
email: mjf@inesc-id.pt, pimentelcesar@hotmail.com, jorgej@acm.org

Abstract

CALI is a fast, simple and compact online recognizer that identifies Scribbles (multi-stroke geometric shapes) drawn with a stylus on a digitizing tablet. Our method is able to identify shapes of different sizes and rotated at arbitrary angles, drawn with dashed, continuous strokes or overlapping lines. We use temporal adjacency to allow users to input the most common shapes in drawing such as triangles, lines, rectangles, circles, diamonds and ellipses, using multiple strokes. We have further extended this approach to identify useful shapes such as arrows, crossing lines and unistroke gesture commands and have developed a library of software components to make this software generally available. The recognition algorithm uses Fuzzy Logic and geometric features, combined with an extensible set of heuristics to classify scribbles. More recently we developed a trainable version of the recognizer to allow users to easily add new shape classes to the initial core set. Evaluation results show recognition rates over 97% for the non-trainable and 95% for the trainable version.

Introduction

User interfaces are developing away from the classical WIMP (windows, icons, mouse and pointing) paradigm. Among the new generation of intelligent systems there is a broad class based on new modalities such as sketching, hand-drawing and stylus input gestures, which we call calligraphic interfaces. These seem natural, because people use diagrams to think, to represent complex models and ideas, to reason about and discuss designs in domains as diverse as architecture, engineering and music. Despite the "graphical" in the current generation of graphical user interfaces, these instances are characterized by discrete interaction modalities such as selecting items from menus, pointing to things on the screen and pressing buttons. The drawing paradigm, so powerful and so useful for humans, goes largely underutilized.

Our aim is to develop simple components to help people use computers in more engaging and "natural" modes. To this end, we have developed a simple recognizer capable of identifying the most common constructs and gestures used in drawings.

The rest of the paper is organized as follows. First we describe related work about calligraphic interfaces and gesture recognizers. Then we present our non-trainable gesture recognizer, CALI. We identify the geometric features used by the recognizer, we explain how Fuzzy Logic is integrated in the recognizer and how it models ambiguity and finally, we present CALI architecture. The next section describes a trainable version of the recognizer. Finally, we describe our experimental evaluation and we present some conclusions and future work.

Related Work

The idea of calligraphic interfaces is not new. In 1963, Sutherland presented Sketchpad the first interactive system that used one light pen to draw diagrams directly over the screen surface. Nevertheless, due in part to ergonomic problems with the light pen and the invention of the mouse in 1964, current graphical interfaces relegated pen-based interactions to specific CAD applications. Things changed with the appearance of the first pen computers in 1991, even though pens were used mainly to input text through handwriting recognition.

The Newton system (Kounalakis & Menuez 1993) , one of the first hand-held pen-based computers, incorporates handwriting, shape and gesture recognizers. While the shape recognizer only handles uni-stroke, axis-aligned shapes, the gesture recognizer is more suitable for text editing than for drawing schematics.

Our work is based on a first approach to recognize schematic drawings developed at the University of Washington (Apte, Vo, & Kimura 1993), which recognized a small number of non-rotated shapes and did not distinguish open/closed, dashed or bold shapes. Zhao (1993) describes an interactive editor based on the StateCharts (Harel 1987) formalism. Although the editor uses a mouse and direct manipulation, many of the ideas described by Zhao express an approach based on diagram recognition guided by syntax. Gross (1996) describes a system fundamentally based on sketches that are partially interpreted for use in architecture drawings, using a recognizer substantially simpler and less robust than ours. Although this recognizer is trainable, the number of identified shapes is apparently smaller than ours. Landay (1996) uses a recognizer developed by Rubine (1991) to sketch graphical arrangements of bidimensional

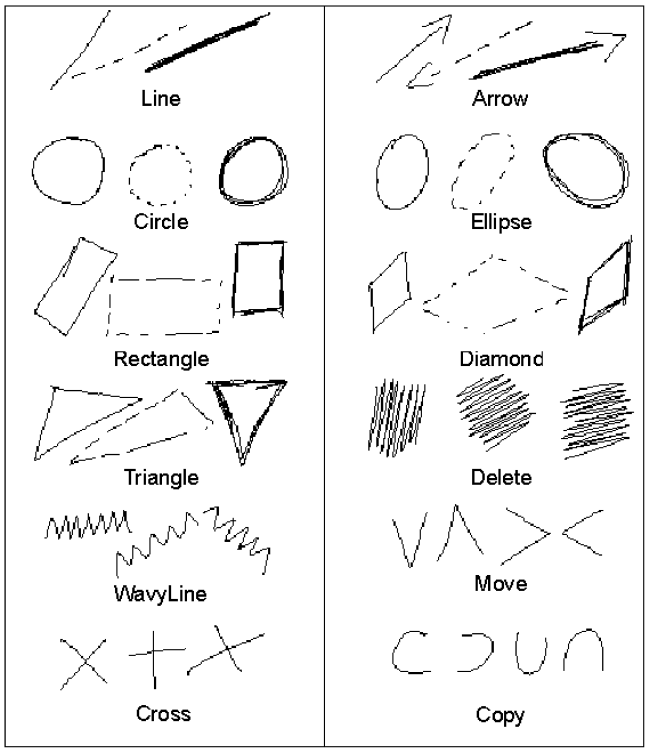

Figure 1: Shapes identified by the recognizer.

documents. Rubine's recognizer is a trainable single-stroke gesture recognizer that uses a classic linear discriminator-training algorithm. One main advantage of our approach over Rubine's is that our method allows the user to draw scribbles without any restriction and as close as possible to the intended shapes. This way, we can recognize shapes without the limitation of being single-stroked (i.e., without having to rule out crosses or arrows).

Other authors have proposed more complex methods, involving neural networks (Ulgen, Flavell, & Akamatsu 1995), to identify a small number of geometric shapes (rectangles, squares, circles, ellipses and triangles). These shapes are recognized independently of size and rotation, but they cannot be drawn using dashed or bold lines. Even though the authors claim good performance for their method, the paper does not present clear measurements to back their claims.

Non-Trainable Recognizer

The recognition method used in CALI is based on three main ideas. First, we use entirely global geometric properties extracted from input shapes, because we are interested in identifying geometric entities. Second, to enhance recognition performance, we use a set of filters either to identify shapes or to remove unwanted shapes using distinctive criteria. Third, to overcome uncertainty and imprecision in shape sketches, we use fuzzy logic (Bezdek & Pal 1992) to associate degrees of certainty to recognized shapes, thereby handling ambiguities naturally.

This algorithm recognizes elementary geometric shapes,

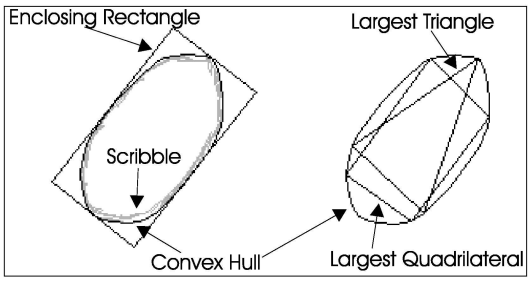

Figure 2: Polygons used to estimate features.

such as `Triangles`, `Rectangles`, `Diamonds`, `Circles`, `Ellipses`, `Lines` and `Arrows`, and five gesture commands, `Delete`, `Cross`, `WavyLine`, `Move` and `Copy`, as depicted in Figure 1. Shapes are recognized independently of rotation, size or number of strokes.

The set of shapes selected and presented in Figure 1 are the basic elements to allow the construction of technical diagrams such as electric or logic circuits, flowcharts or architectural sketches. These diagrams also require distinguishing between solid, dashed and bold depictions of shapes in the same family. Typically, architects will use multiple overlapping strokes to embolden lines in sketches, a mechanism commonly used in drawing packages. We are therefore interested in recognizing different renderings of a given shape as illustrated in Figure 1. We are just interested in collecting qualitative differences, not in obtaining precise values for attributes, without regard to different linestyles and widths.

The algorithm works by collecting strokes from a digitizing tablet. A stroke is the set of points from pen-down to pen-up. We collect strokes into scribbles until a set time-out value is reached. Recognized scribbles are classified as shapes. A shape associates a scribble with a class (e.g. `Triangle`) and a set of geometric attributes, such as start- and end-points and bounding box. The recognizer works by looking up values of specific features in fuzzy sets associated to each shape. This process may yield a list of plausible shapes ordered by degree of certainty. If the recognizer cannot associate any shape to the scribble, it will return the Unknown shape, together with some basic geometric attributes, such as linestyle and "openness".

Geometric Features

After collecting each scribble from the digitizing tablet, we compute the convex hull (ch) of its points, using Graham's scan (O'Rourke 1998). We then use this convex hull to compute three special polygons. The first two are the largest area triangle (lt) and quadrilateral (lq) inscribed in the convex hull (Boyce & Dobkin 1985). The third is the smallest area enclosing rectangle (er) (Freeman & Shapira 1975) (see Figure 2). Finally we compute the area and perimeter for each special polygon, to estimate features and degrees of membership for each shape class.

To select features that best identify a given shape, we built percentile graphics for each feature. These graphics illustrate the statistical distribution of feature values over the dif-

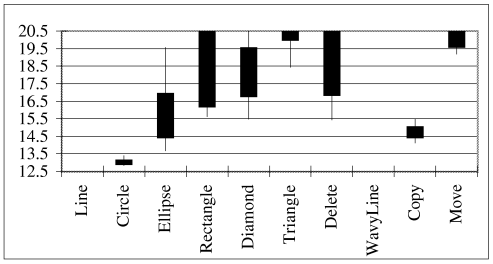

Figure 3: Percentiles for the P_{ch}^2/A_{ch} ratio.

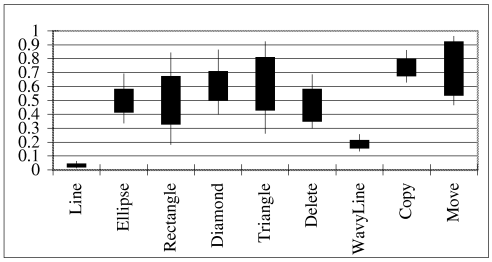

Figure 4: Percentiles for the H_{er}/W_{er} ratio.

ferent classes, extracted from sample drawings. For each shape, the solid bar spans the 25% to 75% percentiles, while the line extends from 10% to 90% of all observed values of a given feature.

Our initial selection of features takes into account specific properties of shapes to identify. Associated with these features we infer fuzzy sets from training data, which express the allowable values of a feature for a given shape. Usually one feature alone is not enough to distinguish shapes, yielding incorrect classifications. We then add extra features (with corresponding fuzzy sets) to prevent unwanted classifications, yielding more complex rules as needed. The main features we use are ratios between perimeters and areas of the special polygons described above.

Figures 3 and 4 show the percentile graphics for features P_{ch}^2/A_{ch} and H_{er}/W_{er}. The first feature, *Thinness* ratio, where A_{ch} is the area of the convex hull and P_{ch}^2 is its perimeter squared, is used to distinguish `Circles` from other shapes. The thinness of a circle is minimal, since it is the planar figure with smallest perimeter enclosing a given area, yielding a value near 4π (see Figure 3). In this figure the thinness values for `Lines` and `WavyLines` lie outside the range of values indicated. We chose not to indicate these values in order to make the range of values for `Circles` more visible.

We identify `Lines` using the aspect ratio, which compares the height of the enclosing rectangle (H_{er}) with its width (W_{er}). The H_{er}/W_{er} ratio will have values near zero for lines and bigger values for other shapes (see Figure 4). The rest of the features were selected using a similar method of analyzing all percentile graphics, to identify unique traits for each shape.

Deriving Fuzzy Sets from Training Data

To choose the "best" fuzzy sets describing each shape class we use a training set developed by three subjects, who drew each shape thirty times; ten times using solid lines, ten times using dashed lines and ten times using bold lines. Based on this training set we define several ratios, combining among others the area and perimeter of the polygons described above.

Each shape is defined by several fuzzy sets, some of which identify the shape while others serve to avoid "wrong" results. We do not use the same number of features for all shapes. Some shapes require no more then one or two features. In general we tend to avoid using "too many" fuzzy sets to characterize a shape. This requires careful data analysis and experimentation to find the "right values", that is, those yielding higher recognition rates and less misrecognitions (false positives). The simplest case is that of a line, which is distinguished by thinness alone.

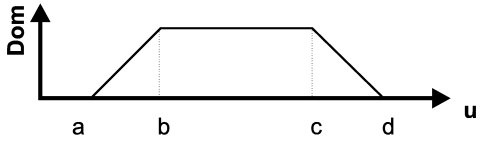

Figure 5: Definition of a fuzzy set.

A fuzzy set is defined by four values, as depicted in Figure 5. After selecting the features for each shape, we compute these four values based on the percentiles. Values `b` and `c` correspond to the 10% and 90% percentiles respectively, and `a` and `d` to the "minimum" and "maximum" after we have identified outliers in each distribution. Removing these outliers minimizes confusion between different shape families, which rises from overlap in distributions. Thus, there is a tradeoff in designing fuzzy sets from statistical data. If `a` and `d` are set very wide apart the recognition rate increases, as well as the number of false positives. "Narrower" values decrease the number of false classifications at the cost of a much lower recognition rate. Abe (1996) discuss an automated procedure for collecting this information using activation and inhibition hyper-boxes. We chose to generate rules manually, since their approach is not sufficiently flexible for our purposes.

Deriving Results from Fuzzy Sets

After collecting input points from the tablet and computing the special polygons from scribble data, the recognizer calculates the degree of membership for each shape class. This degree is the result of `AND`ing together degrees of membership for the relevant fuzzy sets.

In the following paragraphs we exemplify how to build rules that classify shapes. The first rule identifies `Lines` while the second defines `Diamonds`.

```
IF Scribble IS VERY THIN
THEN
     Shape IS A Line
```

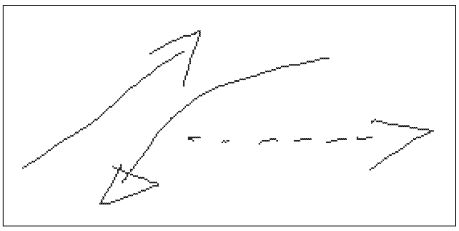

Figure 6: Different types of Arrows.

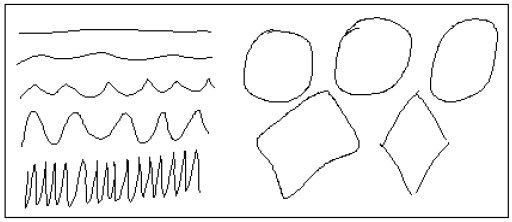

Figure 7: Ambiguity cases among shapes.

```
IF A_lt/A_lq IS LIKE Diamond AND
   A_lq/A_ch IS NOT LIKE Ellipse AND
   A_lq/A_er IS NOT LIKE Bold Line AND
   A_lq/A_er IS NOT LIKE Rectangle
THEN
     Shape IS A Diamond
```

The rule on the top is the simplest rule used in our recognizer. It ascertains that "very thin" ($H_{er}/W_{er} \simeq 0$) scribbles should be classified as Lines. A more complicated rule recognizes Diamonds, where AND denotes the conjunction of fuzzy predicates $\mu_x(f \text{AND } g) = \min(\mu_x(f), \mu_x(g))$ and NOT is defined by $\mu_x(\text{NOT } f) = 1 - \mu_x(f)$.

The algorithm distinguishes between Solid, Dashed and Bold styles after identifying "basic" shapes. This enables us to treat linestyle as an attribute orthogonal to shape, making the design of our recognizer more modular and easier to add different shapes in the future.

Re-segmentation

An approach based entirely on global geometric properties has some limitations. Even though Arrows or Crosses cannot be recognized using this approach, it would be useful if our recognizer identified them, since they are commonly used in most diagram notations. In order to recognize these shapes we must look for new properties that characterize them. Among those, we can consider the small number of strokes, or the existence of a stroke that uniquely identifies the shape, or a distinct spatial relation between strokes. For example Arrows are built of a variable set of strokes terminated by a last stroke which is either a Triangle or a Move shape, as shown in Figure 6. Finally, a Cross consists of two intersecting strokes (that must be Lines).

The next two rules show how we classify these shapes. The first rule identifies Arrows while the second defines Crosses.

```
IF NumStrokes >= 2 AND
   (LastStrk IS LIKE Triangle OR
   LastStrk IS LIKE Move)
THEN
     Shape IS A Arrow


IF NumStrokes == 2 AND
   FirstStrk IS LIKE Line AND
   SecondStrk IS LIKE Line AND
   FirstStrk INTERSECT SecondStrk
THEN
     Shape IS A Cross
```

We perform the analysis of these new properties by resegmenting the original scribble and by applying the recognition process to some specific strokes, e.g. the last stroke in the Arrow. Re-segmentation allows recognizing new gestures that could not be identified using just geometric properties.

Ambiguity

Considering the shapes identified by the recognizer, we present four special cases which can yield ambiguous results. Ambiguity exists between Lines and WavyLines, WavyLines and Deletes, Circles and Ellipses, Diamonds and Rectangles. These cases are presented in Figure 7.

Humans solve this natural ambiguity between geometric shapes, by identifying more than one shape and making the final distinction based on the surrounding context or using feedback from others.

The recognizer described in this paper deals with ambiguity between shapes in a similar way, i.e. when it can not uniquely identify a geometric shape, it returns a list of plausible candidates. The application can then choose the best

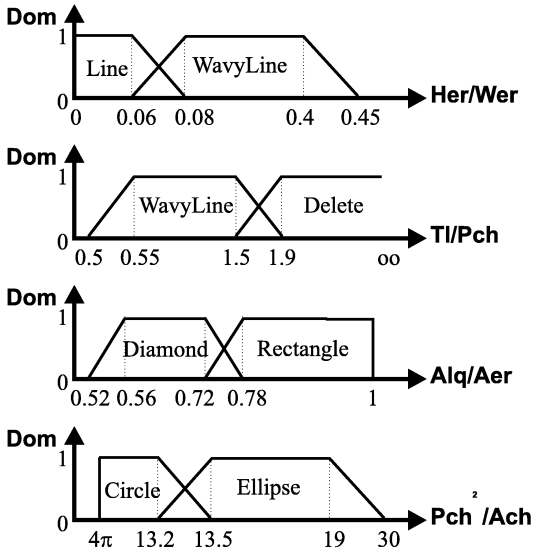

Figure 8: Fuzzy sets representing the ambiguity cases supported by the recognizer.

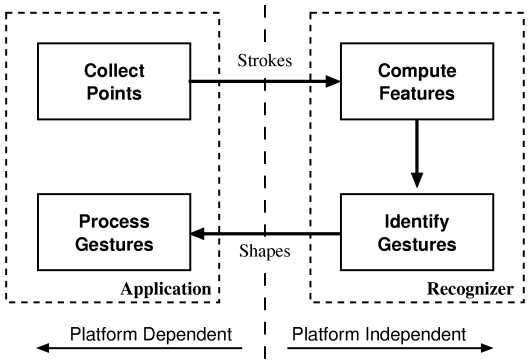

Figure 9: A simplified version of the CALI architecture

candidate using context information.

The ambiguities between shapes are modeled naturally using fuzzy logic to associate degrees of certainty to recognized shapes. Figure 8 illustrates corresponding fuzzy sets for the ambiguous cases shown in Figure 7.

Architecture

The CALI library was developed to be platform independent. Actually, there are two packages available (Fonseca & Jorge 2000), one for Linux and another for MS Windows.

Figure 9 shows the main blocks of the recognizer as well as the blocks to develop on the application side. One of the blocks, on the application side, is responsible for collecting the individual points of the strokes, while the other is responsible for receiving and manipulating the shapes returned by the recognizer. The code developed on the application side is machine dependent.

The first block of the recognizer receives strokes from the application and computes the corresponding geometric features. The second identifies the correct shape based on the values computed before. The recognized shapes are inserted in a list, order by degree of certainty, and returned to the application.

Class Description The following lines describe the main interface classes exposed to clients of the CALI library (see Figure 10).

CIRecognizer Main component of the library that interacts directly with calligraphic applications identifying hand drawn scribbles. This class implements a recognizer of geometric shapes and gesture commands based mainly on geometric information.

CIGesture Defines all the recognized entities, shapes and commands. The objects of this class have the original scribble and the recognition degree of certainty associated to them.

CIShape A Shape is a special case of gesture, that models all geometric shapes (Line, Circle, Rectangle, etc.). All instances of this class have attributes, like open, dashed or bold, and a geometric definition (a set of points).

CICommand A special case of gesture, but without attributes or geometric definition. Commands do not have a visual representation and usually trigger an action.

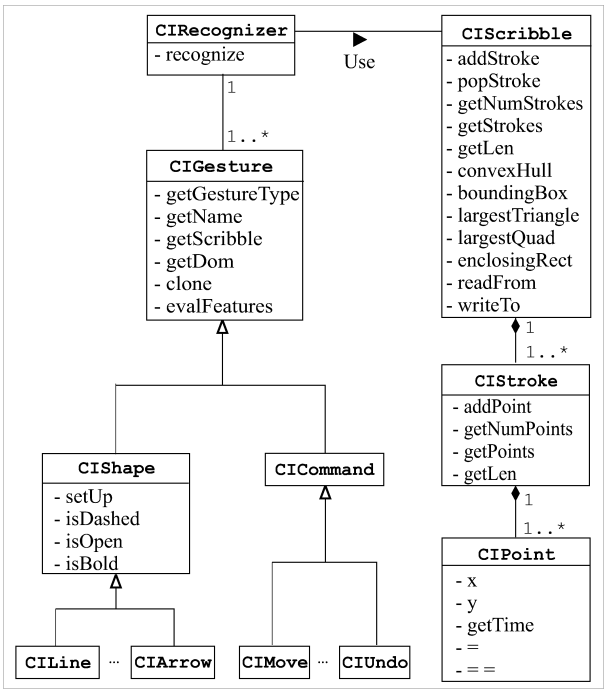

Figure 10: Class diagram

CIScribble This class represents a scribble, which is built from a set of strokes. From a scribble we can compute some special polygons, like the Bounding Box, the Convex Hull, the Largest Triangle, etc., used during the recognition process.

CIStroke Defines a stroke composed by a set of points. It has methods to add points, to know the number of points, to get the points and to compute the total length of the stroke.

CIPoint Models a bidimensional point with a time stamp.

Trainable Recognizer

While the simple approach we used on the non-trainable recognizer achieved fairly high recognition rates, it does not allow users to specify new gestures, requiring hand-coding and careful tuning to add new primitives, which is a tedious and cumbersome task. This considerably limits the flexibility of CALI in accommodating new gestures and interaction primitives. The trainable recognizer we present in this section is a considerable advance from the one described above. Instead of limiting ourselves to using the pre-defined fuzzy logic based method for a fixed number of different stroke classes, it allows us to teach and use new classes of gestures.

Before implementing the trainable version of our recognizer, we evaluated three different training algorithms. We chose three general approaches, K-Nearest Neighbors (KNN) (Cover & Hart 1967), Inductive Decision Tree (ID3) (Quinlan 1986; 1993) and Naïve Bayes (NB) (Domingos & Pazzani 1996), for their simplicity, flexibility and robust-

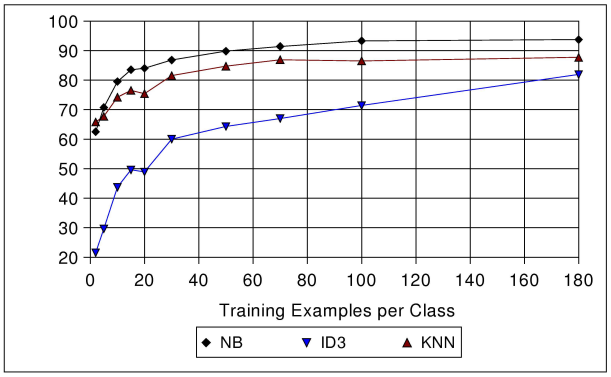

Figure 11: Learning evolution.

Drawn \ Recognized	Line	Arrow	Triangle	Rectangle	Diamond	Circle	Ellipse	Delete	WaviLine	Copy	Move	Cross	Unknown
Line	**97.7**			0.5				1.4	0.2				0.2
Arrow	0.5	**90.2**	0.3						0.3				8.7
Triangle	0.3	0.8	**97.8**						0.6				0.6
Rectangle	0.2	0.5		**96.9**	1.4		0.2						0.7
Diamond		0.8		8.4	**87.8**		0.3	0.3	0.5				1.9
Circle				0.3		**97.2**	2.5						
Ellipse						1.1	**97.6**	0.5					0.8
Delete					0.4			**98.9**					0.7
WavyLine	2.0							2.7	**94.6**				0.7
Copy										**99.6**	0.4		
Move			3.0								**94.8**		2.2
Cross												**97.0**	3.0

Figure 12: Confusion matrix (values in percentages).

ness.The first algorithm, KNN, revealed good classification efficiency and a fast training time. However, large training sets placed considerable demands in both computer memory and classification time required. The second, ID3, presented average classification efficiency and a very fast classification response, due to its low computational complexity. Finally, the Naïve Bayes algorithm was very fast in training and classifying, without sacrificing much on classification efficiency. We quickly noticed that the Naïve Bayes algorithm presents the highest classification efficiency (around 95% when the number of training examples is high). This combined with very acceptable training and classification complexity, makes this algorithm the best suited for our trainable recognizer.

The Naïve Bayes algorithm uses probabilities estimated from the training set. For a given instance (gesture), it computes the probability of each feature value belonging to each class. Then it combines these probabilities, to discover to which class the instance most likely belongs. This procedure assumes that there are no statistic dependencies between features. While generally this is not the case, the simplification is often acceptable and frequently we do not have enough information about dependencies between features. However, this algorithm requires discretizing features values. Even though the method cannot handle symbolic features, its classification efficiency in recognizing our gestures was the highest among the algorithms we have tried.

Besides the recognition rate, we also evaluated the algorithm learning ability. We wanted to know how many examples of a gesture were required to "teach" that gesture. To this end, we plotted recognition rate against training set size for the twelve solid shapes mentioned above. Relatively speaking, we would say that Naïve Bayes exhibited acceptable classification efficiency with 50 examples per class, and tended to stabilize around 100 examples per class, as we can see in Figure 11.

Experimental Results

In this section we describe the experimental evaluation and recognition rates achieved by both recognizers. The experiments were geared at measuring "positive" recognition rather than "negative" rejection. The rationale being that in interactive applications it should be "ease" to correct recognition errors, but user acceptance is related to positive recognition.

Non-Trainable Recognizer

To evaluate the recognition algorithm, we asked nine subjects to draw each multi-stroke shape 40 times, using solid, dashed and bold lines, 30 times each uni-stroke shape and a simple Entity/Relationship diagram with 22 shapes. All these drawings yield a total of 4068 shapes. Subjects were told that the experiment was meant to test recognition, so they didn't try to draw "unnatural" shapes.

We used a Wacom PL-300 LCD digitizing tablet and a cordless stylus to draw the shapes. We gave a brief description of the recognizer to the users, including the set of recognizable shapes. We also told them about the multi-stroke shape recognition capabilities and the independence of changes with rotation or size. Novice subjects had a short practice session in order to become acquainted to the stylus/tablet combination. During the drawing session the recognizer was turned off in order not to interfere with data collection and to avoid any kind of adaptation from the user.

The recognizer successfully identified 95.8% of the scribbles drawn considering just the first shape identified. It is fast: each scribble requires, on average, less than 50 ms (using a Pentium II @ 233 MHz) to be recognized, from feature vector computation to final classification.

A cursory analysis of the confusion matrix, shown in Figure 12, reveals that `Diamonds` are often confused with `Rectangles`, and have the lowest recognition rate. `Arrows` are other shape which exhibits low recognition rate. The former is due to the ambiguity between `Rectangles` and `Diamonds` that favors `Rectangles` and the latter is due to incorrect drawing (single-stroke) of arrows by users. We can also identify other cases of confusion between shapes, such as `Circles` with `Ellipses`, `Moves` with `Triangles` and finally `WavyLines` with `Lines` and with `Deletes`. In fact, the confusion between these shapes is both an acceptable and *intuitive* behavior.

Since ambiguity is one of the main characteristics of our recognizer, we prefer to consider the top three shapes identified instead of just the most likely one. Using this, when we

		Recognized											
	Shapes	Arrow	Circle	Cross	Copy	Delete	Diamond	Ellipse	Line	Move	Rectangle	Triangle	WavyLine
Drawn	Arrow	**84.8**		5.6			1.6		5.6		0.8		1.6
	Circle		**94.0**			2.0		4.0					
	Cross	5.9		**94.1**									
	Copy		1.2		**91.7**			6.0			0.6		0.6
	Delete					**89.1**			2.2	2.2			6.5
	Diamond						**98.7**					1.3	
	Ellipse		14.0					**86.0**					
	Line								**99.2**				0.8
	Move									**100**			
	Rectangle			2.0			2.0		2.0		**86.0**		8.0
	Triangle											**96.1**	3.9
	WavyLine								1.7				**98.3**

Figure 13: Confusion matrix (values in percentages).

take this route, the recognition rate increases to 97%.

Trainable Recognizer

To train and evaluate the different learning algorithms described above we used a set of sample data collected using a Wacom PL-300 LCD digitizing tablet and a cordless stylus. We asked 22 subjects to draw each multi-stroke shape twenty times using solid lines, ten times using dashed and ten times using bold style. We also asked them to draw 30 times each uni-stroke shape. All these drawings yield a total of 9944 shapes. For evaluation proposes we selected a subset of the data collected. The confusion matrix shown in Figure 13 presents the recognition results from the Naïve Bayes algorithm considering the twelve solid shape classes. We used 2610 scribbles to train the recognizer and 959 other scribbles (from a different set of users) to evaluate it. The recognizer successfully identified 93.3% of the scribbles drawn.

As shown by the confusion matrix, sometimes `Arrows` are classified as `Crosses` and vice-versa. This happens because these scribbles have two strokes. The number of strokes is one of the most important features for distinguishing `Arrows` and `Crosses`. `Arrows` are also sometimes misclassified as `Lines` due to their geometrically similar shape. `Copies` are seen as `Ellipses` when they are drawn as a nearly closed shape. If `Deletes` are made a bit longer than usual, there may be some confusion with `WavyLines`. Many of the `Ellipses` are classified as `Circles`, a very intuitive failure mode. Eight percent of the `Rectangles` are taken as `WavyLines` (perhaps the worst misclassification) because of the similarities between their enclosing rectangles.

Finally, we wish to explain that "strange" misclassifications such as mistaking a `Rectangle` for a `WavyLine` can be easily solved by adding appropriate extra features. Such features should present distant values on each gesture class. The training algorithm will automatically handle the task of using that feature to better distinguish the classes at stake. The features we have chosen have proven useful for these twelve classes, and since these are relatively diverse, we believe the recognizer will behave well for a broad variety of classes. In summary, our recognizer performs considerably well, presenting an overall classification efficiency of 93.3% in this experiment.

In general, our observations confirmed that recognition rates increase with training set size and decrease with number of classes. Indeed, a separate experiment with more samples (6022) and a different test set, exhibited 95.1% recognition rate for the NB approach when ignoring the styles.

Conclusions and Future Work

We have described two approaches to recognize multi–stroke geometric shapes. The non-trainable recognizer is simple, fast and robust. The trainable version is more flexible, in that adding new classes is simpler, at the cost of somewhat lower performance. Moreover, both the non-trainable and trainable recognizers, have good recognition rates, 95.8% and 95.1% respectively.

The non-trainable recognizer uses fuzzy rules to classify geometric shapes and introduced re-segmentation to identify higher-level patterns such as `Arrows`. Additionally, it uses fuzzy logic to model ambiguities between shapes. The trainable recognizer, uses a simple training algorithm, Naïve Bayes, which provides good performance and classification rate.

While the non-trainable recognizer is already stable, the trainable version is in a preliminary phase, but with promising classification rates (95.1%). However, the trainable classifier requires many samples to achieve this performance. While this can be perfected, the amount of work required is considerably less than the hand tuning required to add new shapes to the non-trainable recognizer.

Our intent was to provide more a means to support calligraphic interaction rather than a totally robust and "foolproof" approach to reject shapes outside the domain of interest. An in-depth description of both approaches can be found in (Fonseca & Jorge 2001) and (Pimentel, Fonseca, & Jorge 2001), while the source code is publicly available at `http://immi.inesc-id.pt/cali`.

The high recognition rates and fast response characteristic of both recognizers make them very usable in interactive applications.

We are currently porting the non-trainable recognizer to JAVA and we also plan to use a simple indexing structure for high-dimensional data (Fonseca & Jorge 2002) as the method to perform KNN. This way we expect to achieve a better performance with the KNN algorithm than we achieved with the Naïve Bayes. Additionally, we could select a new set of features more suited for training algorithms than the ones currently in use, which were selected for use with classification rules and decision trees.

Acknowledgments

This work was funded in part by the Portuguese Foundation for Science and Technology, project 34672/99 and the European Commission, project SmartSketches IST-2000-28169.

References

Abe, S., and Lan, M.-S. 1996. Effi cient Methods for Fuzzy Rule Extraction From Numerical Data. In C. H. Chen., ed., *Fuzzy Logic And Neural Networks Handbook*. IEEE Press. 7.1–7.33.

Apte, A.; Vo, V.; and Kimura, T. D. 1993. Recognizing Multistroke Geometric Shapes: An Experimental Evaluation. In *Proceedings of the ACM (UIST'93)*, 121–128.

Bezdek, J. C., and Pal, S. K. 1992. *Fuzzy Models for Pattern Recognition*. IEEE Press.

Boyce, J. E., and Dobkin, D. P. 1985. Finding Extremal Polygons. *SIAM Journal on Computing* 14(1):134–147.

Cover, T., and Hart, P. 1967. Nearest neighbor pattern classifi cation. *IEEE Transactions on Information Theory* 13:21–27.

Domingos, P., and Pazzani, M. 1996. Beyond independence: Conditions for the optimality of the simple bayesian classifi er. In *Proceedings of the 13th International Joint Conference on Machine Learning*, 105–112.

Fonseca, M., and Jorge, J. 2000. CALI : A Software Library for Calligraphic Interfaces. INESC-ID, available at `http://immi.inesc-id.pt/cali/`.

Fonseca, M. J., and Jorge, J. A. 2001. Experimental Evaluation of an on-line Scribble Recognizer. *Pattern Recognition Letters* 22(12):1311–1319.

Fonseca, M. J., and Jorge, J. A. 2002. A simple indexing technique for variable–high–dimensional data. Technical report, INESC-ID.

Freeman, H., and Shapira, R. 1975. Determining the minimum-area encasing rectangle for an arbitrary closed curve. *Communications of the ACM* 18(7):409–413.

Gross, M. D. 1996. The Electronic Cocktail Napkin - A computational environment for working with design diagrams. *Design Studies* 17(1):53–69.

Harel, D. 1987. StateCharts: A visual formalism for complex systems. *Science of Computer Programming* 8:231–274.

Kounalakis, M., and Menuez, D. 1993. Defying gravity: The making of newton. Beyond Words Publishing Co.

Landay, J. A. 1996. *Interactive Sketching for the Early Stages of User Interface Design*. Ph.D. Dissertation, Carnegie Mellon University, Computer Science, Pittsburgh - USA.

O'Rourke, J. 1998. *Computational geometry in C*. Cambridge University Press, 2nd edition.

Pimentel, C. F.; Fonseca, M. J.; and Jorge, J. A. 2001. Experimental Evaluation of a Trainable Scribble Recognizer for Calligraphic Interfaces. In *Proceedings of the Fourth Int. Workshop on Graphics Recognition (GREC'01)*.

Quinlan, J. R. 1986. Induction of decision trees. *Machine Learning* 1(1):81–106.

Quinlan, J. R. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Rubine, D. H. 1991. *The Automatic Recognition of Gestures*. Ph.D. Dissertation, Carnegie Mellon University, Computer Science, Pittsburgh - USA.

Sutherland, I. E. 1963. Sketchpad: A Man-Machine Graphical Communication System. In *Spring Joint Computer Conference*, 2–19. AFIPS Press.

Ulgen, F.; Flavell, A.; and Akamatsu, N. 1995. Geometric shape recognition with fuzzy fi ltered input to a backpropagation neural network. *IEEE Trans. Inf. & Syst.* E788-D(2):174–183.

Zhao, R. 1993. Incremental Recognition in Gesture-Based and Syntax-Directed Diagram Editors. In *Proceedings of the INTERCHI'93*, 95–100.