SketchTester: Analysis and Evaluation of Calligraphic Gesture Recognizers

Abílio Costa
Dep. de Engª Informática, ISEP - IPP
R. Dr. António Bernardino de Almeida, 431, Porto
amfcalt@gmail.com

João P. Pereira
Dep. de Engª Informática / GECAD, ISEP - IPP
R. Dr. António Bernardino de Almeida, 431, Porto
jjp@isep.ipp.pt

Resumo
As interfaces caligráficas apresentam uma forma natural para utilizadores interagirem com aplicações. Dado que o núcleo de uma interface caligráfica é o seu reconhecedor, existe a necessidade de avaliação de diversos reconhecedores antes de optar pela utilização de um. Neste artigo apresentamos uma avaliação de três reconhecedores caligráficos: o reconhecedor de Rubine, o reconhecedor de $1 e o reconhecedor CALI. A avaliação foi realizada com base em amostras caligráficas reais, desenhadas por 32 participantes, com um conjunto de símbolos selecionados para utilização num trabalho futuro. Para além disto, discutimos também alguns aperfeiçoamentos realizados à implementação dos reconhecedores e que ajudaram a obter taxas de reconhecimento superiores. No final, o CALI obteve a melhor taxa de reconhecimento com 94% de sucesso, seguido do reconhecedor de $1 com 87% e finalmente pelo reconhecedor de Rubine com 79%.

Abstract
Sketch-based interfaces can provide a natural way for users to interact with applications. Since the core of a sketch-based interface is the gesture recognizer, there is a need to correctly evaluate various recognizers before choosing one. In this paper we present an evaluation of three gesture recognizers: Rubine's recognizer, CALI and the $1 Recognizer. The evaluation was done using real gesture samples drawn by 32 subjects, with a gesture set arranged for use in a future work. We also discuss some improvements to the recognizers' implementation that helped achieving higher recognition rates. In the end, CALI had the best recognition rate with 94% accuracy, followed by $1 Recognizer with 87% and finally by Rubine's recognizer with 79%.

Keywords
Gesture recognition, Calligraphic interfaces, Rubine, CALI, $1 Recognizer.

1. INTRODUCTION

Using pen and paper to draw or sketch something in order to express an idea is very common and also very natural for us. By using this concept in user interfaces one can make the interaction process more natural and spontaneous.

In the future, we aim to develop a programing library to aid in the creation of applications for two-dimensional physics simulations in which the user interacts directly with the scene using a "pen and paper" style interaction. Thus, instead of selecting from a menu which objects compose the scene to be simulated, the user can simply draw the objects directly in the scene. We hope that developing a library that integrates a calligraphic interface and a physics simulation engine will provide a boost for developers to create new applications around this concept, be they for educational purposes, like an application used for teaching physics to students using an interactive whiteboard, or for entertainment purposes, such as a physics-based game where the user draws parts of the scene in order to reach a goal, in the same genre as Crayon Physics Deluxe [Purho09]. These are only two examples of a wide range of possibilities.

The library will support three gestures to draw primitives and other three to define relations between primitives. The first three gestures are used to draw rectangles, triangles and circles, which can be created by drawing these symbols directly. To establish relations between primitives the user can draw a zigzag to connect two primitives with a spring, a cross to pin a primitive over another and a small circle to connect one primitive over another with a rotation axis. Since both the circle primitive and the rotation axis relation use the same gesture[1], we only have in fact five gestures to recognize, presented in Figure 1. Given that the cross is the only gesture that cannot be drawn with only one stroke, we opted to replace it with an alpha, which is an intuitive single-stroke representation of a cross. We chose to use only single-stroke gestures because besides meeting the needs of our library it makes the interaction simpler, since using gestures formed with multi-strokes will force the user to specifically signalize when a gesture is completely drawn or, if

[1] The identification of whether the system should recognize a circle primitive or a rotation axis relation is done by analyzing the size of the gesture and whether or not it is drawn over two existing gestures.

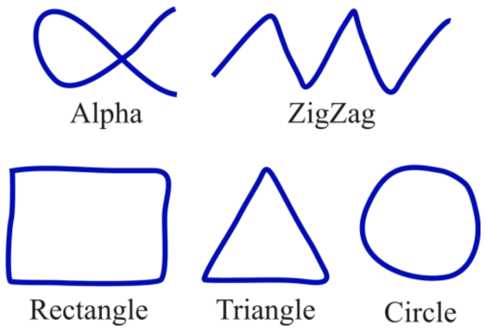

Figure 1 – Set of gestures used in our work

using a timer approach, to draw all the gesture's parts within a specific time and wait for the recognition to happen, which may lead to user frustration.

Given the importance of having good gesture recognition, since the user must feel the interaction to be as natural and unrestrictive as drawing with a pen and a paper, we conducted an evaluation of various gesture recognizers in order to select the one that best fits our needs. In this evaluation we have done two sessions to collect samples of the five gestures drawn by various subjects, in order to put the recognizers to test with a wide range of data. This paper describes that evaluation in detail, along with various considerations to achieve higher recognition rates.

In the next section we present an overview of the related work done in the gesture recognition field. This is followed by a description of the application we developed to test gesture recognizers and the implementation of these recognizers. We then present how the evaluation of the gesture recognizers was conducted and discuss its results. Finally we propose potential future developments of this work and present our conclusions.

2. RELATED WORK

Given the potential of automatic sketch recognition, a lot of work has been done in order to develop recognizers capable of dealing with the intrinsic ambiguity of hand-drawn sketches. Since there is a great variety of sketch recognition algorithms, it is only natural that there's also diversity in their characteristics. For example, some recognizers only work with single-stroke sketches, while others are oriented towards multi-stroke sketches. Also, whether or not the recognizer can identify sketches independently of their orientation, scale, and drawing order can greatly affect its usefulness in some domains. Another important characteristic is if the recognizer can be trained with new gestures, meaning that it can be easily expanded, or if its gestures are hardcoded, which makes it difficult to change its gesture set to fit a new domain.

Rubine's recognizer [Rubine91], a trainable gesture recognizer, classifies each gesture using a linear classifier algorithm with a set of distinct features. Rubine specifies 11 static geometric features, such as sin/cosine of the initial angle of the gesture, distance between the first and last points, total gesture's length, among others. Rubine also defined two dynamic features: the maximum speed of the gesture and its duration. The recognizer is very flexible since features can be easily added or removed to make the recognizer fit the application needs. For example, [Plimmer07] shows how to improve the recognition and make it independent of gesture's size by removing features that involve absolute sizes and adding new ones that use ratios instead. For the training process, Rubine's recognizer calculates the features of the training templates of each gesture class[2] and computes the weights of those classes based on their features. As pointed by the author, the recognizer requires about 15 training templates per gesture class to be effective, which can make the training process a time consuming task. To recognize a gesture, the algorithm makes a linear combination of the gesture's features and the weights of each class. The class which maximizes that combination is selected as the one that the gesture belongs to. The major limitations of Rubine's recognizer are its sensibility to the drawing direction, scale and orientation and being unable to identify multi-stroke sketches. Pereira et al. [Pereira04] made some modifications to Rubine's recognizer in order to make the algorithm accept multi-stroke sketches, but only when drawn with a constant set of strokes as pointed out by [Stahovich11]. The authors also present a way to make the algorithm insensitive to drawing direction, by doing the recognition twice: first with the original sketch and then with an inverted sketch.

CALI [Fonseca02] is an easy to use multi-stroke recognizer that uses Fuzzy Logic and geometric features to classify gestures independently of their size or orientation. Instead of an individual algorithm, CALI is a complete library that can be easily built into an application. CALI separates gestures into two types: shapes and commands. Shapes can be drawn (and recognized) using solid, dashed and bold[3] lines, while commands are only recognized with solid lines. The recognizer defines a set of geometric rules or features to identify each gesture, like its thinness, aspect ratio, and many others. For example, a Line shape is characterized by being "very thin". In addition to the global geometric features, some gesture classes are also characterized by local features such as the sub-gestures that compose the gesture or whether it has intersections. For example, the Cross command is identified by having two intersecting Line shapes. When an input gesture enters the recognition process its features are computed and checked against each defined gesture rules, using fuzzy sets to find the degree of membership to each rule and therefore to each class. Since CALI is a non-trainable recognizer, adding new gestures is not an easy task, involving hand-coding and analysis of which features characterize and distinguish the gesture. To solve this limitation the authors also present a trainable recognizer and compare three training algorithms: K-Nearest Neighbors, Inductive Decision Tree and Naïve Bayes, the latest being the one with highest training efficiency. Nevertheless, the trainable recognizer has a lowest recognition rate and requires numerous training templates for each gesture class.

[2] A gesture class represents a unique gesture, but can be made from multiple representations of that gesture, i.e. multiple templates.

[3] Bold lines are made from multiple overlapping solid strokes.

In [Wobbrock07], Wobbrock et al. present the $1 Recognizer which aims to be easy to understand and quick to implement. It is insensitive to scale and orientation of sketches, but is sensitive to the drawing direction. One major advantage of $1 Recognizer is the simplicity to add support for new gestures, requiring only one training template per gesture class in most cases. The algorithm has basically four major steps, the first three being applied to both the training templates and the input gesture (the one that is to be recognized), and the fourth step only to the input gesture. The first step is to resample the point path, using simple linear interpolation, so that every gesture (including the training templates) has the same number of points. This enables a direct point-to-point comparison between input gesture and training templates, independently of drawing size and speed. The second step is to rotate the gesture to an orientation that is optimal for matching and thus reduce the recognition time later. This rotation is made based on the angle between the centroid of the gesture and the gesture's first point. The third step is to scale the gesture non-uniformly to a square and translate its centroid to the origin (0,0). The fourth and last step, which is only applied to input gestures, is where the actual recognition happens. The input gesture is compared to each training template to find the average distance between corresponding points and, based on that distance, a score is calculated. The training template with the biggest score is the one that, according to the recognizer, matches the input gesture. Templates with lower score can be used to deal with ambiguity, serving as alternative matches. When the algorithm is computing the average path-distance between an input gesture and a training template, the input gesture is rotated using the Golden Section Search algorithm to find the angle in which that distance is minimized. The authors also explain how to make the recognizer sensitive to scale or orientation, for some or all gesture templates.

In order to solve some of the limitations of the $1 Recognizer, such as not being able to recognizing multi-stroke gestures, sensitiveness to the drawing direction, and problems recognizing uni-dimensional gestures such as lines, Anthony et al. extended it and created the $N Recognizer [Anthony10]. The algorithm starts by computing all the possible combinations of stroke orders and directions for each multi-stroke gesture serving as training template and creates a single-stroke gesture for each combination, by connecting the individual strokes with the order and direction of that combination. These single-stroke gestures are used for comparison with the input gesture, using the same process as the $1 Recognizer, since multi-stroke input gestures are also transformed into single-stroke gestures by connecting their individual strokes by the order they were drawn. The transformations used in $1 Recognizer, such as point resampling, rotation to find the optimal orientation, and translation of the centroid to the origin are also applied by $N to every combined single-stroke gesture. Despite the improvements over the $1 Recognizer, $N has problems recognizing gestures made with more strokes than defined in the training templates. Also, it is not well suited to recognize "messy" gestures like a scratch-out, commonly used for erasing-like actions.

Lee et al. [Lee07] present a trainable graph-based recognizer that is insensitive to orientation, scale and drawing direction and is able to recognize multi-stroke gestures. The recognizer uses statistical models to define symbols, which makes it deal with the small variations associated with hand-drawn gestures naturally. Each gesture is represented by an attributed relational graph, in which nodes depict the type of primitive (line or arc) and its relative length[4]. The edges of the graph represent the geometric relationships between primitives, characterized by the number of intersections, the intersection angle and the intersection location. Gestures are segmented into individual primitives using a technique based on the drawing speed [Stahovich04], meaning that errors in the segmentation process will propagate to the recognition process. When an input gesture arrives, the recognizer compares it to each trained gesture class and computes a dissimilarity score based on six error metrics, each one with a different weight on the resulting score. This dissimilarity score is then converted to a similarity score which is used to identify the gesture class that classifies the input gesture. Since the same gesture can be drawn with varying number of primitives and drawing orders, comparing input gestures and training templates is not straightforward and presents a graph matching problem. To solve this, the authors evaluate and propose five approximate matching techniques. For the training process, an average attributed relational graph is created for each gesture class, by averaging the graphs of multiple training templates. One limitation of this approach is that all training templates of a gesture class must be drawn with a consistent drawing order or consistent orientation.

Vatavu et al. [Vatavu09] present a trainable recognizer that uses elastic deformation energies to classify single-stroke gestures. The recognizer is naturally insensitive to gesture scale and orientation, since the same gesture has similar curvature functions independently of the drawing orientation or size, but is sensitive to drawing direction and starting point within the gesture. To classify a gesture, the recognizer computes its curvature function, based on trajectory analysis, and calculates the alignment cost to each gesture class to find the one that minimizes that cost. Computing the curvature function for each class is done by averaging the functions of multiple training templates for that class.

In [Sezgin05], the authors present a multi-stroke sketch recognizer, based on Hidden Markov Models (HMM), that is capable of recognizing individual sketches in complex scenes even if the scene is not yet completed, i.e. while it is being drawn, and without the need to pre-segment[5] it. On the other hand it can only recognize sketches in their trained orientations, thus being sensitive

[4] The length of the primitive in relation to the total gesture's length.

[5] Pre-segmenting a scene means isolating individual sketches or gestures in the scene.

to orientation. The algorithm first creates discret observation sequences of the scene by identifying various geometric primitives such as sloped lines, horizontal/vertical lines, polylines, among others, with the aid of the Early Sketch Processing Toolkit [Sezgin01]. Then, for each trained HMM, it computes the likelihood for various subsections of the scene given that HMM. Using these likelihoods, it builds a graph in which the shortest path gives the most likely segmentation of the scene, that is, the individual sketches in the scene. Finally, it classifies each segment (the individual sketch) by finding the HMM that maximizes the probability of generating that segment. Since the recognition relies on the stroke order of the trained templates, it is not well suited for domains where the stroke ordering cannot be predicted. Also, because HMMs are suited for sequences, it cannot recognize single-stroke sketches, unless they are pre-segmented.

PaleoSketch [Paulson08] is a low-level non-trainable sketch recognizer for single-stroke primitives. When recognizing a gesture, it starts with a pre-recognition phase, where it removes consecutive duplicate points, cleans drawing noise from the beginning and the end of the gesture, and computes various graphs and values that characterize that gesture. After the pre-recognition process is done, the actual recognition is made. PaleoSketch uses multiple individual sub-recognizers, where each indicates whether or not the input gesture matches a given primitive. Since a gesture can be recognized by more than one sub-recognizer and since each sub-recognizer only returns whether the stroke matches or not, without any "matching score", the results are passed to a hierarchy function for sorting. This function uses a corner finding algorithm to find the minimum number of lines to correctly describe the input gesture and compares that to the minimum number of lines defined for each sub-recognizer's primitive, ordering the results accordingly. PaleoSketch is insensitive to orientation, scale and drawing order, but can only recognize low-level primitives. For more complex gestures (such as rectangles or triangles), one needs to add a higher-level recognizer on top of PaleoSketch.

The need to compare and evaluate the performance of various sketch recognizers is not something new. In [Schmieder09] a toolkit to automatically evaluate recognition algorithms is presented. In addition to sketch data collection and labeling, the toolkit allows the integration of multiple trainable and non-trainable recognizers, which can be tested simultaneously. After testing, the toolkit outputs the results in the form of a Microsoft Excel file or screenshots of the gestures that have been misclassified. As a proof of concept, the authors also present and discuss experimental results of the evaluation of six gesture recognizers: CALI, Microsoft Ink Analyser, $1 Recognizer, Rubine's recognizer with the extended features used in InkKit, PaleoSketch, and a recognizer using Dynamic Time Warping techniques.

3. IMPLEMENTATION

In order to test the sketch recognition algorithms, we developed SketchTester, an application that enables us to rapidly prototype and incorporate recognition algorithms. With this application we are able to individually test each algorithm against drawn gestures and immediately see the recognition results (recognized gesture, recognition score, and other alternative matches with lower score). It also provides a graphical interface to add/remove training templates to/from each trainable recognizer, automatically saving them to hard-disk in a recognizer-specific file. As shown in Figure 2, to collect sketch samples from subjects SketchTester offers a window that specifies what gesture should be drawn and, after the gesture is submitted, shows the recognition result of each recognizer. The application keeps asking for random gestures until a predefined number of samples of each gesture is collected, point at which the recognition rates of each recognizer are presented. Each of the submitted sample gesture and corresponding results are saved to a file, so that we can analyze it later.

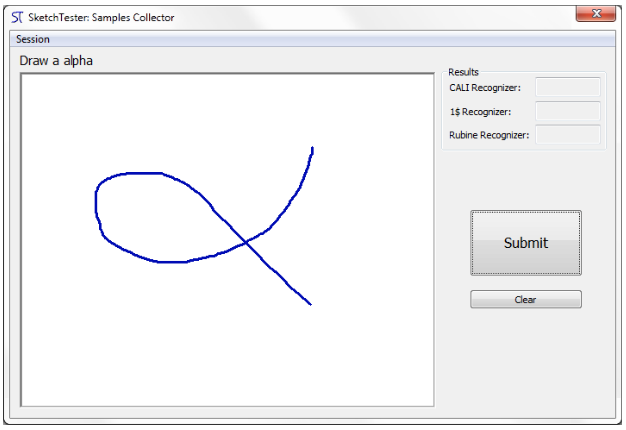

Figure 2 – Interface used to collect gesture samples from subjects

SketchTester also incorporates functionality to review collected gesture samples, save them as bitmap files, and extract data from them, such as total number of correctly recognized gestures. Another important feature of SketchTester is the possibility to reprocess multiple files of collected samples. This was particularly useful since we made some improvements to the recognizers after the collection of samples from subjects. By reprocessing those samples, their recognition results were updated according to the improved recognizers.

In SketchTester we implemented three popular recognizers: the $1 Recognizer, Rubine's recognizer, and CALI. The algorithms of the first two recognizers were implemented according to the descriptions given by their authors in [Wobbrock07] and [Rubine91], respectively. In the case of CALI, since it exists in the form of a library, it was only necessary to integrate it with SketchTester. Another important note is that we didn't implement any rejection for gestures with low score, since we always want a result even if it is a low-scored match.

As regards to training templates, $1 Recognizer was first trained with 2 templates for each gesture while Rubine's recognizer was trained with 15 templates for each gesture. Also, since $1 and Rubine's recognizer are sensitive to the drawing direction, when adding a new training template the application automatically creates a copy of the template but with inverted drawing direction, adding it to the same gesture class as the original template in the case of $1 or to a new class in the case of Rubine's recognizer

CALI was also subject to changes. First we disabled the recognition of gestures we don't need, such as the "copy" gesture, lines, arrows, and some others. Also, since we don't need to discriminate circles from ellipses, we always classify an input gesture as a circle whether it is recognized by CALI as circle or ellipse. The same logic applies to rectangles and diamonds, where they are both classified as rectangles. By removing unneeded gestures from the recognizer and grouping similar gestures we expect to increase the recognition success. Furthermore, we added support for the alpha gesture in CALI by hard-coding a new gesture class in which we defined two features that where selected based on empirical observations: the ratio between the area of the largest quadrilateral and the area of the convex hull (A_{lq}/A_{ch}), and the ratio between the perimeter of the largest quadrilateral and perimeter of the convex hull (P_{lq}/P_{ch}). The fuzzy sets associated with these features are presented in Figure 3. These two features alone were not enough, because most of the times CALI would classify an input alpha gesture as both alpha and zigzag[6]. To prevent alphas from being misrecognized as zigzags we defined that zigzags can't have any intersections, and that alphas must have an intersection situated at more than 10% away from the limits of the gesture, meaning that each of the alpha's tails must make more than 10% of the gesture.

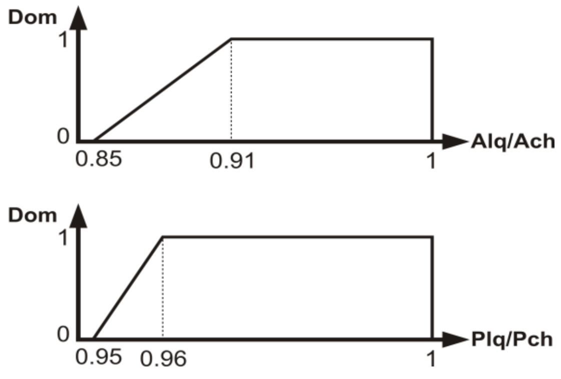

Figure 3 – Fuzzy sets defined for the alpha gesture

After analyzing the samples collected from subjects, we found that the implemented recognizers could be improved. In Rubine's recognizer we added three more classes (plus three "duplicates" with inverted drawing direction) per gesture to contemplate different orientations. Figure 4 exemplifies how the rectangle gesture would be represented in each of the four classes. Before the inclusion of new classes to represent various orientations, only the orientation represented in the top-left side of Figure 4 was present.

[6] The zigzag gesture is originally called WavyLine in CALI.

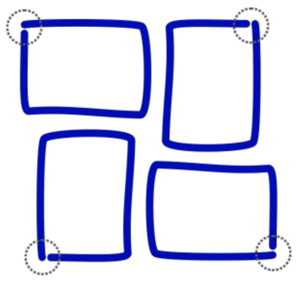

Figure 4 – Representation of four classes for the rectangle gesture contemplating different orientations; the circles mark start and end points

For the $1 Recognizer, one possible improvement we found was adding training templates describing rectangles and triangles starting at the middle of an edge, since the first version only had templates starting at vertices. We also added training templates with different starting vertices for these two gestures in order to contemplate every drawing possibility, but without overdoing it, since the algorithm is insensitive to drawing orientation and so we don't need templates of the gesture starting in every vertex.

We have also improved CALI implementation, by setting the minimum size of each of the alpha's tails to 7% instead of 10%. Also, we defined that a zigzag can have intersections as long as the distance along the gesture between the two intersecting points is less than 13% of the total gesture's length. Finally, the distance along the gesture between the two intersecting points in an alpha must be more than 20%.

The rationale behind these improvements will be described in the next section of this paper.

4. EVALUATION

In our effort towards finding the recognizer which best fits our purpose and has the greatest recognition rate we collected 1550 gesture samples from 32 subjects.

4.1 Method

We conducted two sessions to collect gesture samples, each session with 16 subjects. Using the SketchTester application and following a guide document[7], each subject was asked to draw 10 samples of every of gesture, for a total of 50 samples per subject which provided us with 1550 gesture samples[8]. Both sessions were conducted at our institution where 9 subjects used the institution's desktop computers with traditional mice. The remaining 23 subjects had personal laptops and about 25% of them used the laptop's built-in touchpads and the remaining subjects used conventional mice. All the subjects were MSc students.

Since our evaluation aims to test the recognition of single-stroke gestures with no drawing restrictions, we asked the subjects to freely draw the five gestures presented in Figure 1 with diversified sizes, orientations and shapes.

[7] http://dei.isep.ipp.pt/~i060687/guiao_recolha_caligrafica.pdf

[8] 1550 and not 1600 because two subjects provided less than 50 samples each.

4.2 Results

As previously described, we made some improvements to the recognizers and their training templates after analyzing the first recognition results of the collected gesture samples. Then, we reprocessed these samples with the improved recognizers. In this section we will present the recognition results obtained before and after the improvements and discuss how these improvements affect the recognition rates.

To make sure the collected samples fit the domain in which we intend to use the recognizers, we first cleaned them by removing samples that do not match the requested gesture (e.g., the subject was asked to draw a rectangle and drew a circle) and samples drawn incorrectly (e.g., gestures drawn with multiple overlapping lines or gestures that don't represent any of the five required gestures). As expected, cleaning the samples enhanced the recognition rates, especially with CALI whose improvement reached 11%. This enhancement is shown in Figure 5, with a side-by-side comparison of the recognition rates of the three algorithms before and after cleaning the samples, with CALI achieving the highest rates.

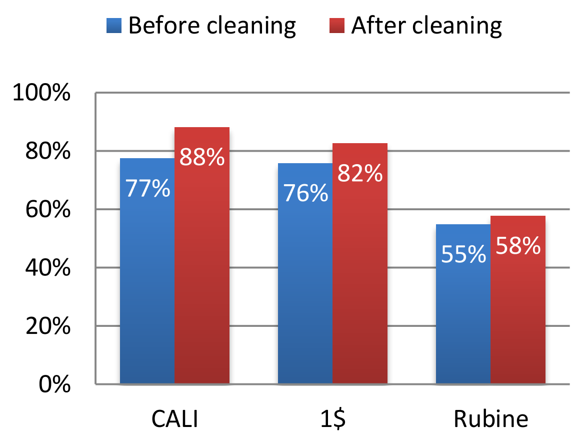

Figure 5 – Recognitions rates before and after cleaning the collected sample gestures

In order to understand why Rubine's recognizer obtained such low recognition rates we need to examine the individual recognition rates of each gesture. As presented in Figure 6, the recognition rates of the triangle and circle gestures are 26% and 27% respectively, which is extremely low and far from the remaining gestures' rates. After reviewing individual samples it was clear that what was affecting the recognition of these gestures was the diversification in drawing orientations. We also noticed that users have a tendency to draw rectangles starting with the top-left vertex, which explains why that particular gesture was not suffering much with the orientation sensitivity problem, since the training samples had that same orientation. Interestingly, the drawing orientation had a low impact on the recognition of the zigzag gesture, mainly because it is very distinctive from the remaining gestures in terms of features, and since we did not implement gesture rejection, it is recognized even with a low recognition score.

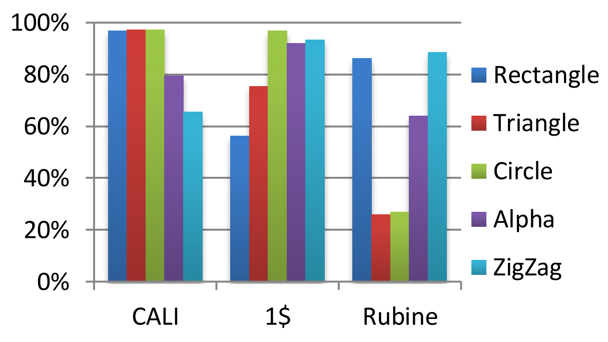

Figure 6 – Recognition rates of each gesture with each recognizer

To increase Rubine's recognizer recognition rates we needed to overcome the problem of rotation dependence. As described earlier, the solution was to add new classes representing each gesture in different orientations. As show in Figure 7, this greatly increased the recognition rates and, despite a slight decrease in the recognition of the rectangle gesture, the overall recognition rate with Rubine's recognizer was improved to 79%, against the previous 58%.

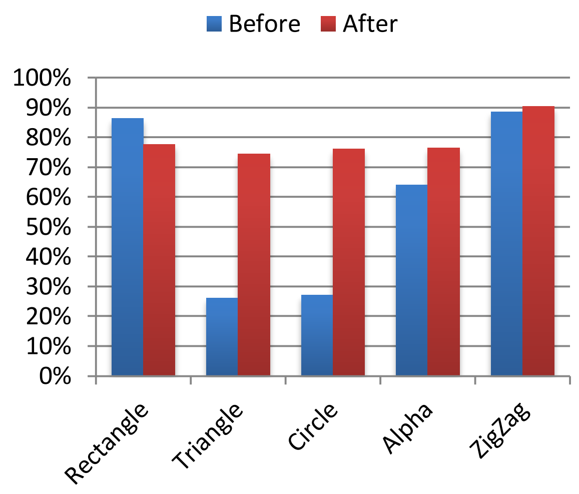

Figure 7 – Recognition rates for each gesture with Rubine's recognizer, before and after adding gesture classes representing multiple orientations

Regarding the $1 Recognizer, the most problematic gestures were rectangles and triangles, with recognition rates of 56% and 75% respectively. We found that both gestures needed training templates with starting points at different vertices and also at the middle of edges. We also found that a training template representing right triangles was needed. After these additions to $1 Recognizer's training templates the recognition rates of rectangles and triangles were improved and, despite a decrease in the recognition rates of circles and alphas, the global recognition rate of the algorithm was improved to 87% against the previous 82%. We then tried to improve the recognition of circles and alphas, since they were affected in these changes, but found no success. Figure 8 shows the recognition rates for each gesture before and after the addition of the new training templates.

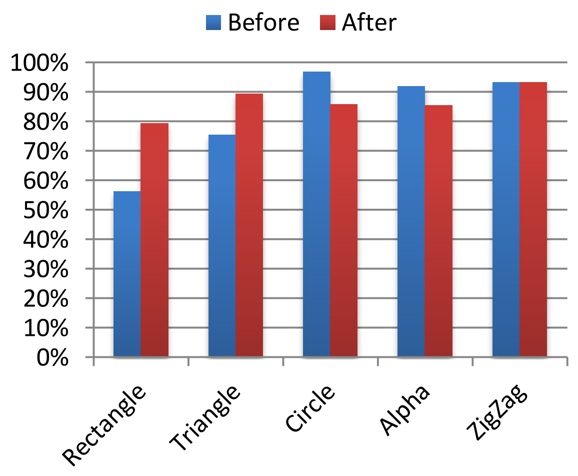

Figure 8 - Recognition rates for each gesture with $1 Recognizer, before and after improving its training templates

Despite CALI already reaching a fairly high recognition rate of 88%, analyzing the results for individual gesture rates suggests that it could be further improved. As show in Figure 6, the alpha and zigzag gestures have low recognition rates when compared to the other gestures, which may be due to the changes we've made to the original CALI source code to include the alpha gesture, discussed previously. Further investigation showed that many alphas with short tails were simply not recognized. After detailed examination of the collected gestures we found that 7% was the optimal value as the minimum relative size of the gesture's tails. In relation to the zigzag gesture, we found that many subjects made small intersections when drawing zigzags and since we first specified that zigzags must not have intersections, many were recognized as alphas. After inspecting the collected samples, we specified that zigzags can have intersections but only when the two intersecting points are not more than 20% away from each other. We also updated the alpha gesture so that the distance between the two intersecting points must be more than 20% of the total gesture's length. As presented in Figure 9, these changes to CALI had a positive impact on the recognition of alphas and zigzags, without affecting the other gestures.

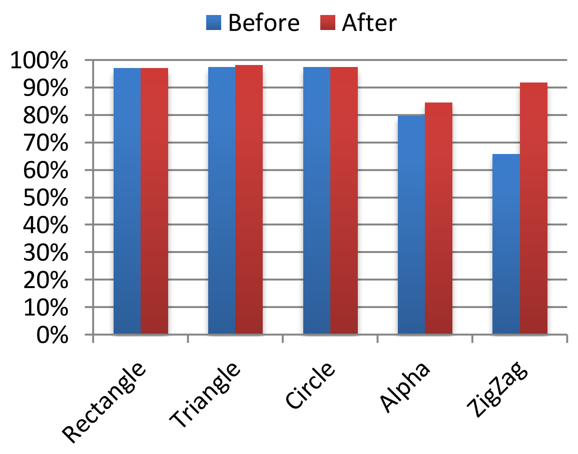

Figure 9 – Recognition rates for each gesture with CALI, before and after updating the recognition source code of the alpha and zigzag gestures

By analyzing the first recognition results of the samples collected from subjects, we were able to identify flaws in our implementation of the recognizers. As show in Figure 10, our attempts to correct these flaws where successful and all the recognizers reached higher recognition rates, with CALI achieving the best rates, followed by $1 and then by Rubine's recognizer. Also, Figure 10 shows that even after improving the recognizers, the respective standard deviations are high, which is caused by a significant disparity in the individual recognition rates of each gesture. This suggests the possibility that the recognizers can still be improved. If we were able to improve the recognition of the gestures with lowest rates, the overall recognition rate would increase and the standard deviation would decrease, meaning that the recognizer would be recognizing all the gestures in a balanced way.

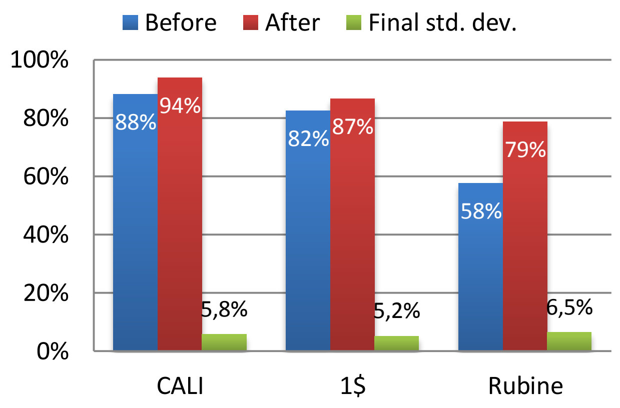

Figure 10 – Overall recognitions rates before and after improving the recognizers, along with the standard deviation of the final rates

In [Schmieder09], to show the potential of their automated recognizer evaluation toolkit, the authors conducted an experiment in which they evaluate six recognizers, including the same three we evaluated, with three basic single-stroke gestures[9]: circle, rectangle and line. In their experiment, Rubine's recognizer had the best recognition with a 96% success rate, followed by $1 Recognizer with 89% and CALI with 84%. While at first these results can seem to contradict our evaluation's results, with CALI and Rubine's recognizer inverting positions, they can be easily explained. While we used the original feature set in Rubine's algorithm, they implemented the extended feature set used in InkKit [Plimmer07], which explains why Rubine's recognizer achieved such high recognition rates. In respect to CALI, they consider circles and ellipses as independent gestures, unlike our evaluation where we don't need to differentiate these two gestures and consider both as one. In their results, there have been 94 circles misclassified as ellipses, in a total of 730 evaluated gestures. If we consider these 94 ellipses as being correctly classified, effectively merging circles and ellipses, the recognition rate for CALI increases to 96%. If we also merge rectangles and diamonds it rises to 98%. These

[9] There's also a second experiment which we won't cover on this paper because it is done with Entity Relationship (ER) diagrams instead of basic gesture shapes.

rates are close to those obtained in our evaluation. Regarding $1 Recognizer, both evaluations yield similar results. Finally, it is important to notice that since they used fewer gestures than us it is normal that they obtained higher recognition rates, as the misclassifications tend to increase with the number of gestures.

5. FUTURE WORK

In the future we could implement more recognizers in SketchTester, and even improve Rubine's features as described in [Plimmer07], in order to also evaluate them against the collected samples. The inclusion of other gestures could also be subject of study if the library is extended beyond five gestures.

An evaluation of the recognizers with gestures collected using touchscreens or interactive whiteboards would also be an interesting evaluation, since these are the kind of devices that most benefit from calligraphic interfaces.

Since the improvements to the recognizers were made and evaluated using the same set of gesture samples, it would be important to re-evaluate these improvements with new samples, in order to confirm that they are valid not only for our sample set but also to generic samples.

Finally, although the most relevant result of recognition is the gesture with the highest score, a study considering the first two or three high-score results could be relevant in cases where the application presents a list of alternative matches to solve ambiguity.

6. CONCLUSION

In this work we've collected sample gestures from various subjects and evaluated three popular gesture recognizers to find the one that best fits in the interaction layer of our physics simulation library. We've also presented some insights on how the implementations of these recognizers can be improved to yield better results. Also, despite the specificity of the tested data, our work can serve as a base to others exploring gesture recognizers.

To conclude, both CALI and $1 are good candidates for our library since both achieved high recognition results. Also, we are confident that if the improved features described in [Plimmer07] where implemented in Rubine's recognizer, it could have results as good as the results of the other two recognizers. Nevertheless, since CALI archived higher rates it shall be selected to integrate our library.

7. ACKNOWLEDGMENTS

We wish to thank all the participants in the recognizer evaluation sessions. We are also grateful to Manuel J. Fonseca for providing us the CALI library source code.

8. REFERENCIES

[Anthony10] Lisa Anthony, Jacob O. Wobbrock. 2010. A lightweight multistroke recognizer for user interface prototypes. In *Proceedings of Graphics Interface 2010* (GI '10), 245-252.

[Fonseca02] Manuel J. Fonseca, César Pimentel, and Joaquim A. Jorge. 2002. CALI: An online scribble recognizer for calligraphic interfaces. In *AAAI Spring Symposium on Sketch Understanding*, 51-58.

[Lee07] WeeSan Lee, Levent Burak Kara, and Thomas F. Stahovich. 2007. An efficient graph-based recognizer for hand-drawn symbols. In *Computers & Graphics 31*, 554-567.

[Paulson08] Brandon Paulson and Tracy Hammond. 2008. PaleoSketch: accurate primitive sketch recognition and beautification. In *Proceedings of the 13th international conference on Intelligent user interfaces* (IUI '08), 1-10

[Pereira04] Pereira, J. P., Branco, V. A., Jorge, J. A., Silva, N. F., Cardoso, T. D., and Ferreira, F. N. 2004. Cascading recognizers for ambiguous calligraphic interaction. In *Eurographics Workshop on Sketch-Based Interfaces and Modeling.*

[Plimmer07] Beryl Plimmer and Isaac Freeman. 2007. A toolkit approach to sketched diagram recognition. In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...but not as we know it* (BCS-HCI '07), vol. 1.

[Purho09] Petri Purho. 2009. Crayon Physics Deluxe. `<http://crayonphysics.com/>`

[Rubine91] Rubine, Dean. 1991. Specifying gestures by example. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (SIGGRAPH '91), 329-337.

[Schmieder09] Paul Schmieder, Beryl Plimmer, and Rachel Blagojevic. 2009. Automatic evaluation of sketch recognizers. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling* (SBIM '09), 85-92.

[Sezgin01] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. 2001. Sketch based interfaces: early processing for sketch understanding. In *Proceedings of the 2001 workshop on Perceptive user interfaces* (PUI '01), 1-8.

[Sezgin05] Tevfik Metin Sezgin and Randall Davis. 2005. HMM-based efficient sketch recognition. In *Proceedings of the 10th international conference on Intelligent user interfaces* (IUI '05), 281-283.

[Stahovich04] Thomas F. Stahovich. 2004. Segmentation of pen strokes using pen speed. In *Proceedings of 2004 AAAI fall symposium on making pen-based interaction intelligent and natural*, 152-158.

[Stahovich11] Thomas F. Stahovich. 2011. Pen-based Interfaces for Engineering and Education. In *Sketch-based Interfaces and Modeling*, 119-152.

[Vatavu09] Radu-Daniel Vatavu, Laurent Grisoni, and Stefan-Gheorghe Pentiuc. 2009. Gesture Recognition Based on Elastic Deformation Energies. In *Gesture-Based Human-Computer Interaction and Simulation*, vol. 5085, 1-12.

[Wobbrock07] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. 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* (UIST '07), 159-168.