Towards a Toolkit for the Development and Evaluation of Sketch Recognition Techniques

Rachel Blagojevic
University of Auckland
Private bag 92019, Auckland,
New Zealand
rpat088@aucklanduni.ac.nz

Paul Schmieder
University of Auckland
Private bag 92019, Auckland,
New Zealand
psch068@aucklanduni.ac.nz

Beryl Plimmer
University of Auckland
Private bag 92019, Auckland,
New Zealand
beryl@cs.auckland.ac.nz

ABSTRACT
We present our prototype software platform for ink recognition algorithm evaluation. Our automated evaluator runs comparative tests of recognisers using the data collected and labelled in the tool. Other functionality includes labelling of groups of strokes and an extensive feature library. Use of such a platform improves efficiency in carrying out sketch recognition research and works towards developing standard benchmarks for evaluating recognisers.

INTRODUCTION
There is no software platform available to compare the numerous digital ink recognisers that have been proposed. Many recognisers have been developed and have reported high accuracy rates, for example [1-6]. However we need standard test platforms available to provide fair comparative evaluations of these recognition techniques under the same conditions, that is using the same training and test data. If we can comparatively evaluate recognisers we can identify strengths and weaknesses and so inform further development and appropriate usage. Such a platform must have access to large amounts of digital ink data that are representative of a wide range of diagram types.

The development of sketch tools has been encouraged by advances in digital pen hardware. Their resemblance to pen and paper make sketch tools ideal for diagramming by supporting ambiguity and quick construction. In terms of early phase designs, this unconstrained environment encourages the flow of creativity resulting in better designs than when using widget based tools [7-10]. In addition to the pen and paper like flexibility, sketch tools benefit from the ease of transmission, replication and archiving offered by computer support. Potential uses include office automation, software design, electronics design, architecture and civil engineering, and education.

With algorithms built into these tools that can "understand" the diagrams, sketch tools have much greater potential. Such automatic recognition allows these sketch tools to support more intelligent tasks such as editing, execution, and conversion of these diagrams. However, current recognition engines lack the flexibility and accuracy levels required for general use. The ambiguity of hand drawn diagrams makes general recognition problems hard to solve.

Recognisers commonly work by measuring features of the hand drawn diagram and using algorithms to classify diagram components based on these features [1, 2, 4-6, 11-17]. Many of these systems have relied heavily on heuristics to determine which features are important for classification and which algorithms are most suited to each problem. Although this has provided a good starting point, it is apparent that there is a need for more rigorous analysis of sketch recognition performance and tuning. Furthermore, advances in recognition techniques require large amounts of digital ink data for training and testing purposes.

There is little ink data publicly available currently and very little support for obtaining and managing such data. We have developed a data manager that supports efficient data collection, single-stroke labelling, and automatic dataset generation for data mining [18]. In this paper we report extensions of our data manager to a general recogniser toolkit. These improvements form the core of a framework for managing sketch data, analysing this data to produce new recognisers, and evaluating recognisers. There are three main extension areas: first extra stroke feature measurements taken from recent sketch literature have been added to enhance the feature set. We have focused particularly on including features that measure stroke context. Second, the single-stroke labelling scheme has been extended to allow us to label groups of strokes. This is essential for testing recognition of multi-stroke diagram components, which is representative of the natural way in which people draw. Finally, a test bed has been developed to allow us to run comparative evaluations of various recognisers based on user defined criteria. Use of such a platform will improve efficiency in carrying out sketch recogniser research and works towards developing a framework for developing and testing recognisers.

The structure of the remainder of this paper is as follows. The next section gives an overview of our earlier data management tool, other ink data collection and labelling tools, and previous work in the area of standard test

platforms used by related research communities. We then proceed to give details of each extension made to our tool including the extra features added, the multi-stroke labelling function, and the test bed for evaluating recognisers. Following this we discuss applications for our tool and limitations to the current developments. We conclude with future work.

RELATED WORK

Our early tool [18] is a data manager for the efficient collection and labelling of sketches and automatic generation of datasets. Using this tool, a researcher first defines the project for which they are collecting data in terms of the types of sketches they wish to collect. Instructions are provided for their participants to follow when constructing the required diagram and labels that may be applied to the particular diagram domain. Data is collected as participants sketch according to the instructions provided. The sketches are then labelled on a single-stroke basis using the labels defined for the particular project. Automatic labelling is also available for text and shape strokes using our divider [16], making the process fast and efficient. Once sketches have been collected and labelled we can analyse various features of the data. We automatically generate datasets consisting of stroke based feature measurements from our existing feature library [16]. Data mining of these datasets using tool such as Weka [19] can reveal significant patterns in sketching and inform further development of recognition algorithms.

Paulson et al [20] designed a tool, SOUSA, for collection and verification of ink data online. User studies for collecting sketches are set up as applets online by researchers. They specify various criteria for the study including the types of sketches they wish to collect. Participants of the study then access the applet online and sketch the required diagrams or symbols according to the instructions given. All data collected is easily accessible for all researchers by downloading from the site. Verification studies can also be set up by the researcher to confirm that participants have drawn what was expected. Participants access a verification study from the website and classify sketches that have been collected according to given categories. Statistics are generated to check that the category given to the sketch through the verification study is consistent with what was expected. This tool does not provide labelling for sketches but instead suggests using other labellers for this as their simple file format does not allow for this extra data.

Wolin et al [21] designed a tool for labelling of ink data using a digital pen. Their tool has three main functions; stroke fragmentation (automatic and manual), stroke grouping, and symbol labelling. Fragmenting is used before labelling as users may draw more than one symbol using a single-stroke and it also helps divide strokes into primitives i.e. lines and arcs. Stroke grouping is for the opposite problem of labelling components made of more than one stroke. Their tool also allows for multiple labels to be applied to a stroke.

There are also tools available for analysing data and comparison of algorithms in a standard environment. In the data mining field, Weka [19] is a widely used workbench which provides access to many machine learning algorithms for data analysis and algorithm development. It includes tools for pre-processing data, feature selection, and testing of algorithms. R [22] is a similar language and environment that has evolved in the field of statistics. It includes functions for statistical data analysis with the use of various classification techniques.

Individually, all of these tools provide very useful features for data management and analysis. However we seek to provide a unified framework for the development of sketch recognition techniques. Such a tool would ensure a fast and more efficient approach to their development and provide an environment for fair comparative evaluations.

EXTRA FEATURES

Features measuring characteristics of a sketch are central to many recognition techniques [1, 2, 4-6, 11-17]. In our previous work [16] we compiled a set of 46 features. These were gathered from related work in sketch recognition and also included features we believed may be useful for recognition and newly available features due to hardware advances e.g. pen pressure.

We have extended this library to include a further 24 features, mostly originating from related work in sketch recognition that were not included in the original feature set. These features include those derived from principal component analysis and stroke fragmentation as well as features relating to spatial context of strokes, stroke size, time, and inter-stroke gaps. More details about these features can be found in the Appendix.

GROUP LABELLING

Our original prototype [18] only allowed labelling of single-strokes. We have now extended this function to allow users to apply labels to groups of strokes, similar to that developed by Wolin et al. [21]. This is important for developing recognisers for multi-stroke shapes or components of diagrams. For example to label a rectangle drawn with four strokes we select the four strokes and check the “Rectangle” checkbox to apply the label to this group. More than one label can be applied to the same stroke group, for example the same group of 4 strokes may be labelled as a “Shape” as well. A stroke can also be part of more than one group of strokes. When a pre-labelled group of strokes is selected the checkboxes corresponding to their labels will be checked to illustrate the label or labels applied to this group. Figure 1 shows

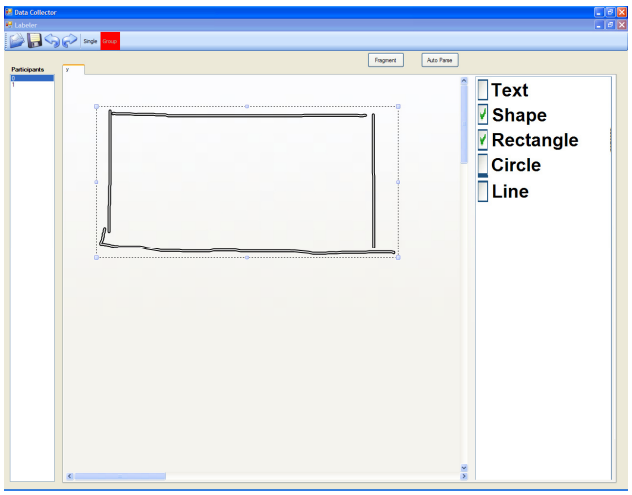

Figure 1. The group labelling interface.

the interface we have developed for labelling groups of strokes.

TESTING FRAMEWORK

Numerous recognition techniques exist for hand drawn diagrams. However, due to the nonexistence of a common data repository for sketched diagrams, statements regarding accuracy of recognisers are not generally comparable. We have extended our original prototype that is used for collecting and labelling data with a platform for recogniser integration and testing. By incorporating the recogniser platform into the data collector, recognisers can be tested and evaluated in parallel, thus a standard test platform is used.

When designing the extension we focused on the following requirements to guarantee easy and effective use:

- Easy integration of recognisers into the framework
- Enabling automatic testing of recognisers
- Providing standard comparative evaluations of recognisers using common data

Although our framework is implemented in C# we aim to be able to easily integrate recognisers regardless of the technology they use. Rather than re-implementing algorithms in C#, which can be extremely time consuming and erroneous, we have chosen to run recognisers externally. This involves serialising stroke information into a temporary XML file on the framework's side and deserialising the file in the language of the target recogniser. A script is used to start the deserialisation and then the recognition process once the stroke information has been obtained. The recognition results amongst other information (e.g. recognition time) are then serialised again by the same script and deserialised on our side when completed. By choosing this approach over a re-implementation, the authenticity of every recogniser is guaranteed, and measurements such as recognition times in the original configuration result in generally comparable results.

The integration of a recogniser into our tool involves two steps:

1. A class implementing the IRecogniser interface has to be added. The interface only specifies the implementation of one method to ensure automated method invocation. An example of such a class is shown in Figure 2.

```
//The namespace must be identical to the recogniser name specified in the xml file
namespace DataCollector.Recognisers.RecogniserName
{
    // every recogniser class must implement the IRecogniser interface
    public class Recogniser: IRecogniser
    {
        public void recognizeBasicShapes(Strokes strokes, RecognizerOptions options)
        {
            //------------------------------ if the recognizers is written in C# ------------------------------

            // initialize recognizer
            RecogniserName recognizer = new RecogniserName();

            foreach (stroke in strokes)
            {
                //actual recognition
                Object recognitionResult = recognizer.recognize(stroke);
            }

            //------------------------------ if the recognizers is NOT written in C# ------------------------------

            // serialize strokes plus optional data
            Serialize(strokes, "path", options.RecognizerInformation);

            // start external process and wait for finish
            Process.Start(process).WaitForExit();

            // Deserialize recognition results
            Object recognitionResult = Deserialize("FileContainingRecognitionResults");
        }
    }
}
```

Figure 2. Example recogniser class implementing the IRecogniser interface.

2. The recogniser name has to be specified in the XML settings file again for automatic class invocation purposes. Other optional parameters including a boolean specifying whether the recognisers needs to be trained or not and the path to the required training set can be part of the XML settings file. More options may be added at a later date as needed. An example of the basic XML settings file required for the recogniser in Figure 2 is shown below in Figure 3.

```
<?xml version="1.0" encoding="utf-8"?>

  <Recognisers >
  <!-- The name of the recogniser must be identical to the namespace of
   the recogniser class-->
  <Recogniser name="RecogniserName"/>

</Recognisers>
```

Figure 3. Example XML settings file for the recogniser specified in Figure 2.

Options other than the recogniser name can also be specified through the user interface (Figure 4); however they are best noted in the settings file for the sake of automation. Currently options cover different aspects of a recogniser such as the ability to recognise shapes consisting of multiple shapes or recognise more than one shape at a time. Additionally, for the sake of automation,

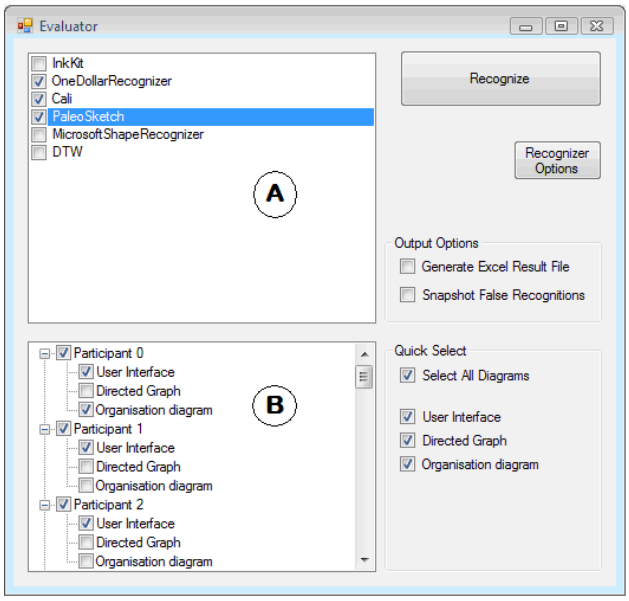

Figure 4. The evaluator interface.

options specific to each recogniser can be specified in the settings file such as switching off certain recogniser modules. This can be useful when comparing recognisers with different sets of basic shape classes. To enable a fair comparison, modules responsible for recognising a basic shape class not available to all recognisers can be automatically switched off.

From the evaluator interface (Figure 4), the user may select from the available recognisers in box A and then select the diagrams that they wish to use as data for the test from box B. There are additional recogniser options (as discussed above) and output options for the results. Once all selections have been made it is a one button click on "Recognise" to run those recognisers on the chosen dataset.

Once the recognition is finished a report on the results of the test is automatically generated. Additionally, thumbnails of misclassified strokes can be generated for viewing. This is to help researchers visually identify patterns that may occur in misclassified instances as a first step to solving these problems.

The generated report first gives general recognition information such as details of the data used for the test, the date and time when the test was performed, and other options used for the test. This is recorded so that the test may be replicated under the same conditions for future comparisons. Furthermore, general statistics including the total number of correctly or incorrectly classified instances are given. Other statistics identify the number of correctly or incorrectly classified instances for each recogniser, for each instance type and for each participant.

DISCUSSION

The purpose of this framework is to enable researchers to run evaluations of their recognisers. However this is not a straightforward task because each recogniser has different goals and constraints. To explore some of these issues we are planning comparative evaluations of existing basic shape recognisers.

Seven basic shape recognisers have been integrated into our tool as shown in Table 1. This table also summarises certain characteristics of each recognition engine. We must consider these differences between recognisers when running our evaluations. For example if we compared the accuracy of CALI [5] and Rubine's [1] basic shape recognisers we would need to consider that Rubine's is constrained by single-stroke shapes whereas

CALI can handle multi-stroke shapes. CALI has a clear advantage given that it supports a more flexible environment.

Name	Recognises single or multi-stroke shapes	Recognises multiple shapes drawn with one stroke	Results in ranked list	Results by confidence value	Technology used
CALI [5]	Multi	No	Yes	Yes	C/C++
Rubine [1]	Single	No	Yes	Yes	
InkKit [17]	Multi	No	Yes	Yes	C#
$1 Recogniser [6]	Single	No	Yes	Yes	
Dynamic Time Warping [6]	Single	No	Yes	Yes	
Microsoft [23]					
PaleoSketch [2]	Single	Yes	Yes	No	Java

Table 1. Summary of basic shape recognisers already integrated into our framework.

An open question remains as to how we can perform fair comparative evaluations of recognisers given that they clearly have different characteristics. The main issues we have identified as important to consider are as follows

1. Accuracy
   The percentage of correct or incorrectly classified instances using the same test dataset.
2. Time
   The time taken for the recogniser to classify a dataset as this reflects its efficiency. This is difficult to measure given that we may not always run the recognisers in the environment they were designed for. We may also like to consider whether it is a lazy or eager recogniser.
3. Recogniser constraints
   The individual capabilities of each recogniser such as its ability to recognise single or multi-stroke shapes, or to recognise multiple shapes drawn with one stroke.
4. Target shape set
   The set of shapes that the recogniser is designed to classify. While many recognisers include "basic shape recognition" there is no agreement on the set of basic shapes.
5. Extensibility of shape set
   Some recognisers have a fixed set of basic shapes while others have an extendable user defined set.
6. Age and Hardware
   The age of the recogniser. Is it fair to test a new system with one designed 10 years ago considering advances in hardware and computing power?
7. Target application
   The recognisers intended use. For example is it designed for recognition of diagrams, command gestures, document analysis, or possibly for a more specialised domain such as just for user interface diagrams?

Many of these factors are interdependent which contributes to the difficulty of running fair comparative evaluations.

Given the issues presented above it is obvious that when comparing recognisers we are unlikely to come to a definitive answer to the question "Which is the best recogniser?" Instead what we can learn are the comparative strengths and weaknesses of each and use this knowledge to inform further development.

It is also clear that we still need human interpretation of results to draw any useful conclusions, especially if we want to learn from failures. Our results report includes thumbnails of misclassified strokes to aid the process of analysing failures by the algorithms. Visualisations like this help us to identify consistent failures quickly and therefore pave the way to the correct solutions. This contributes to the cycle of development for these algorithms that we wish to encourage.

CONCLUSION AND FUTURE WORK

The toolkit presented here advances the development and evaluation of sketch recognition algorithms. The platform includes the group labelling function, the addition of new stroke features, and a framework to integrate recognisers easily into the tool for running comparative evaluations. We are currently considering what constitutes a fair comparative test of recognisers given their differing characteristics and constraints. We are also working on adding more options to the testing framework to help us to support this.

This is an open source project and we encourage other researchers to use and contribute to this project. It is available for download from http://www.cs.auckland.ac.nz/~rpatel/

ACKNOWLEDGEMENTS

The research is partly supported by Microsoft Research Asia.

REFERENCES

[1] Rubine, D.H. *Specifying gestures by example*. in *Proceedings of Siggraph '91*. 1991: ACM.

[2] Paulson, B. and T. Hammond. *PaleoSketch: Accurate Primitive Sketch Recognition and Beautification*. in *Intelligent User Interfaces (IUI '08)*. 2008. New York, USA: ACM Press.

[3] Sezgin, T.M., T. Stahovich, and A.R. Davis. *Sketch based interfaces: early processing for sketch understanding*. in *2001 workshop on Perceptive user interfaces*. 2001. Orlando, Florida: ACM Press.

[4] Yu, B. and S. Cai. *A domain-independent system for sketch recognition*. in *Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*. 2003. Melbourne, Australia: ACM Press.

[5] Fonseca, M.J., C.e. Pimentel, and J.A. Jorge. *CALI: An Online Scribble Recogniser for Calligraphic Interfaces*. in *AAAI Spring Symposium on Sketch Understanding*. 2002: IEEE.

[6] Wobbrock, J.O., A.D. Wilson, and Y. Li, *Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes*, in *Proceedings of the 20th annual ACM symposium on User interface software and technology*. 2007, ACM: Newport, Rhode Island, USA.

[7] Black, A., *Visible planning on paper and on screen: The impact of working medium on decision-making by novice graphic designers.* Behaviour and information technology, 1990. **9**(4): p. 283-296.

[8] Bailey, B.P. and J.A. Konstan. *Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design*. in *CHI 2003*. 2003. Ft Lauderdale: ACM.

[9] Goel, V., *Sketches of thought*. 1995, Cambridge, Massachusetts: The MIT Press.
[10] Yeung, L., B. Plimmer, et al. *Effect of Fidelity in Diagram Presentation*. in *HCI*. 2008. Liverpool.
[11] Bishop, C.M., M. Svensen, and G.E. Hinton, *Distinguishing Text from Graphics in On-Line Handwritten Ink*, in *Proceedings of the Ninth International Workshop on Frontiers in Handwriting Recognition*. 2004, IEEE Computer Society.
[12] Sezgin, T.M., T. Stahovich, and R. Davis. *Sketch based interfaces: early processing for sketch understanding*. in *Proceedings of the 2001 workshop on Perceptive user interfaces*. 2001. Orlando, Florida: ACM Press.
[13] Calhoun, C., T.F. Stahovich, et al. *Recognising Multi-Stroke Symbols*. in *AAAI Spring Symposium on Sketch Understanding*. 2002.
[14] Hammond, T. and R. Davis. *Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams*. in *2002 AAAI Spring Symposium on Sketch Understanding*. 2002.
[15] Li, J., X. Zhang, et al. *Sketch recognition with continuous feedback based on incremental intention extraction*. in *Proceedings of the 10th international conference on Intelligent user interfaces*. 2005. San Diego, California, USA: ACM Press.
[16] Patel, R., B. Plimmer, et al. *Ink Features for Diagram Recognition*. in *4th Eurographics Workshop on Sketch-Based Interfaces and Modeling* 2007. Riverside, California: Eurographics.
[17] Plimmer, B. and I. Freeman. *A Toolkit Approach to Sketched Diagram Recognition*. in *HCI*. 2007. Lancaster, UK: eWiC.
[18] Blagojevic, R., B. Plimmer, et al. *A Data Collection Tool for Sketched Diagrams*. in *Sketch Based Interfaces and Modeling*. 2008. Annecy, France: Eurographics.
[19] Witten, I.H. and E. Frank, *Data Mining: Practical machine learning tools and techniques*. 2nd Edition ed. 2005, San Francisco: Morgan Kaufmann.
[20] Paulson, B., A. Wolin, et al. *SOUSA: Sketch-based Online User Study Applet*. in *Sketch Based Interfaces and Modeling*. 2008. Annecy, France: Eurographics.
[21] Wolin, A., D. Smith, and C. Alvarado. *A Pen-based Tool for Efficient Labelling of 2D Sketches*. in *4th Eurographics Workshop on Sketch-Based Interfaces and Modeling*. 2007. Riverside, CA.
[22] R Development Core Team, *R: A Language and Environment for Statistical Computing*. 2006, R Foundation for Statistical Computing: Vienna, Austria.
[23] Microsoft Corporation, *Ink Analysis Overview*. 2008 [cited 2008; Available from: http://msdn.microsoft.com/en-us/library/ms704040(VS.85).aspx.
[24] Wolin, A., B. Eoff, and T. Hammond. *Short Straw: A Simple and Effective Corner Finder for Polylines*. in *Sketch Based Interfaces and Modeling (SBIM '08)*. 2008. Annecy, France: Eurographics.
[25] Fonseca, M.J. and J.A. Jorge. *Experimental Evaluation of an on-line Scribble Recognizer*. in *Pattern Recognition Letters*. 2001.

APPENDIX: ADDITIONAL INK FEATURES

Feature	Description	Origin
Principle component analysis		
Direction	Direction of the stroke given by the eigenvector of the largest eigen value	[11]
Eigen value ratio	The largest eigen value/ smallest eigen value	
Fragmentation		
Number of Fragments	Number of fragments in a stroke where a stroke is fragmented according its to corners	[11], using Short Straw [24] as a fragmentation technique.
Largest Fragment Length	Arc length of the largest fragment	
Abs Curve of Largest Fragment	The total absolute curvature of the largest fragment	
Largest Fragment Direction	Direction of the largest fragment given by the eigenvector of the largest eigen value	
Long Side of Enclosing Rect of Largest Fragment	The longest length of the largest fragments enclosing rectangle	
Spatial Stroke Context		
# Strokes Horiz Close	The number of strokes horizontally close to the current stroke	New
# Strokes On Same Horiz Plane	The number of strokes on the same horizontal plane as the current stroke	
# Strokes Contained	The number of strokes contained in the current	

	stroke	
# Strokes Similar Height	The number strokes of similar height to the current stroke	
# Strokes Vert Overlapping	The number of strokes vertically overlapping the current stroke	
Inter-Stroke Gaps		
Log time diff from prev	Log of the time between the current and previous stroke	[11]
Log time diff to next	Log of the time between the current stroke and the next stroke	
Log start time from prev	Log of the time from the start of the previous stroke to the start of the current stroke	
Log start time to next	Log of the time from the start of the current stroke to the start of the next stroke	
X Start point diff	The difference in the starting X coordinates of the current stroke to the start of the next stroke	
Y Start point diff	The difference in the starting Y coordinates of the current stroke to the start of the next stroke	
X Diff between strokes	The difference in the X co-ordinate between the current stroke and the next.	
Y Diff between strokes	The difference in the Y co-ordinate between the current stroke and the next.	
Size		
Thinness ratio	The perimeter squared of the stroke's convex hull / area of the stroke's convex hull	[5, 25]
Length:Perimeter ratio	Stroke length / perimeter of the stroke's convex hull	
Length Ratio	Cumulative distance between stroke points/ length from start to end point of a stroke	Adapted from [1]
Time		
Max speed squared	Maximum speed of the stroke squared	[1]