The University of Auckland

Thesis Consent Form

This thesis may be consulted for the purpose of research or private study provided that due acknowledgement is made where appropriate and that the author's permission is obtained before any material from the thesis is published.

I agree that the University of Auckland Library may make a copy of this thesis for supply to the collection of another prescribed library on request from that Library; and

1. I agree that this thesis may be photocopied for supply to any person in accordance with the provisions of Section 56 of the Copyright Act 1994

Name (print): Rachel Venita Patel.....

Student ID: 9812400.....

Signature: .....

Date: 5 February 2007 .....

Exploring better techniques for diagram recognition

Rachel Venita Patel

Under supervision of Dr Beryl Plimmer and Prof John Grundy

A thesis submitted in fulfillment of the requirements for the degree of
Master of Science in Computer Science
The University of Auckland, Feb 2007

Abstract

The objective of this research is to improve recognition of hand-drawn diagrams to allow the benefits of designing on pen and paper to be combined with the benefits of a computer. Most diagramming sketch tools have ignored the need for text recognition and concentrate solely on identifying shapes. However it is obvious that we need to address text recognition as well, because writing is an important component of most diagrams. An accurate method of automatically separating text and shapes is needed to allow each group to be recognised by algorithms specific to their type. It is essential that this division of text and shapes be automatic to ensure minimal interruption and distraction to the user when designing diagrams, preserving the flexibility of using pen and paper. Therefore we have focused our research on improving the existing methods of dividing text and shape strokes.

Our review of literature shows that sketch recognition is composed of two elements, *features* and *algorithms*. We concentrate on the first step of recognition by identifying features to distinguish between text and shape strokes successfully as a way to improve the divider, a component of recognition that automatically groups text and shape strokes together. Most past research identify features simply based on observation and have no statistical evidence of their significance to the recognition domain. We use a more rigorous approach applying a formal statistical analysis technique to determine the features that are significant to classifying text and shape strokes.

We compose a set of features that may be able to contribute to a divider and analyse these features using a tree-based partitioning technique. This analysis results in the identification of a statistically significant feature set for dividing text and shape strokes. The feature set is used to implement a divider and is subsequently evaluated against the Microsoft Tablet OS divider and the InkKit divider. Our new divider classifies shape and text strokes more accurately than the Microsoft and InkKit dividers with an error rate of 16.2% as opposed to an average error rate of 37.1% for the other dividers. Therefore the process of stroke division is more accurate and contributes to an improvement in the recognition of sketches.

Acknowledgements

To my best friend, my darling Marko, for his enduring patience, support and love.

To Sharina for her distraction always timed to perfection, for three hour lunches, encouragement and motivational speeches!

For Mum and Dad for encouraging me do this in the first place, glad you made me, and glad I listened to you eventually.

My supervisors Beryl and John, thank you for the advice and guidance encouraging me every step of the way and challenging me do things I've never done before!

To all the people who have helped shove me in the right direction, couldn't have done it without your encouragement and faith in me.

To my lord and saviour Jesus Christ through whom all things are possible!

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

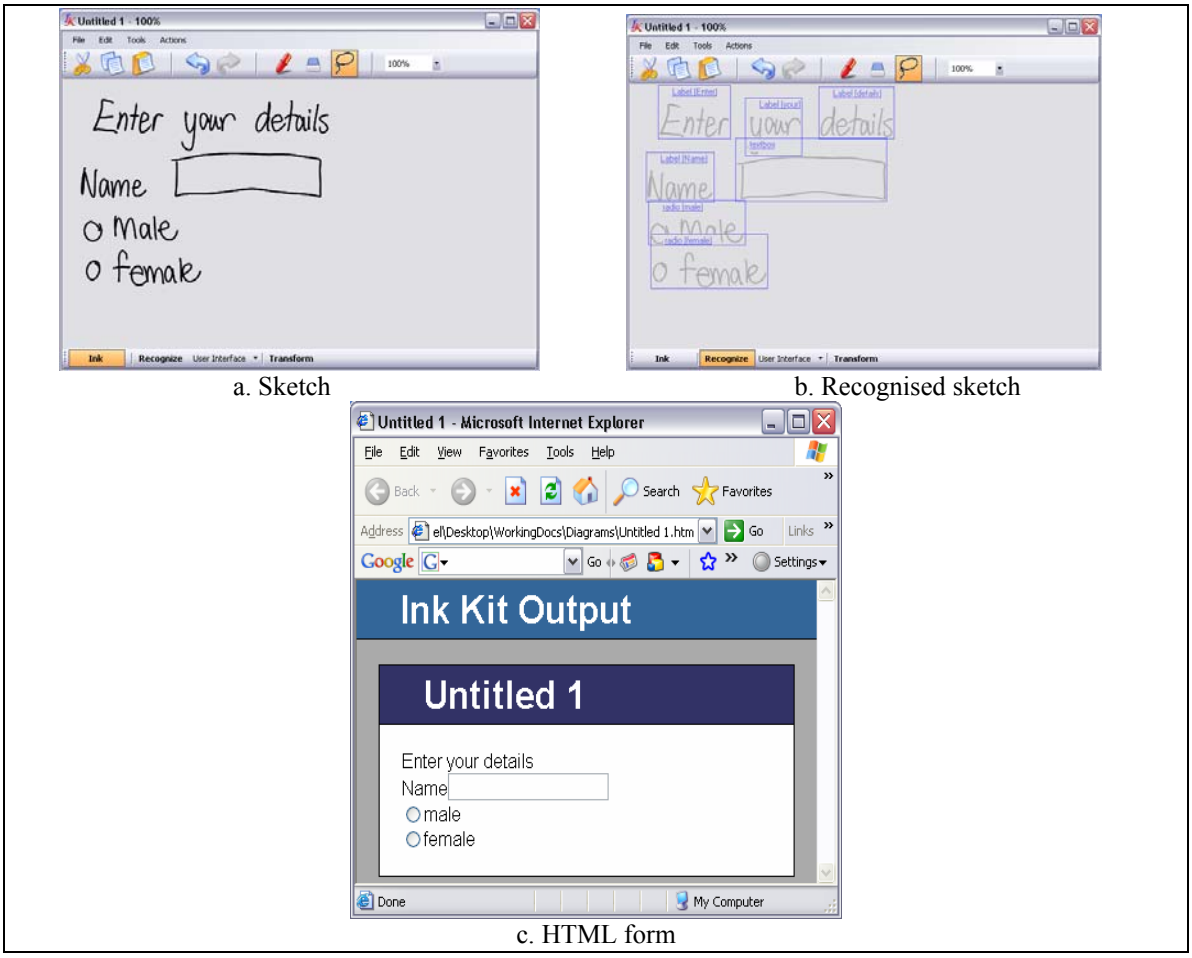
Computers can be used to produce formal diagrams: software designs, user interface designs, CAD specifications, hierarchy charts and so on. However, in the beginning of a project it can be better to use pen and paper (Black 1990; Goel 1995). The reason is that these simple, human-centric design tools are far more flexible which encourages creativity and in turn this produces better designs in the end (Black 1990; Goel 1995). However a computer offers easy editing and distribution features and greater formality to the look of a diagram and so sometimes the important pen and paper design stage is overlooked. Rather than design diagrams on paper first and reproduce them on a computer later we tend to go straight to the computer in an effort to avoid the inefficiency of transferring the diagram to the computer.

How can we combine the benefits of pen and paper and the functionality provided by the computer without having to waste time and effort reproducing a diagram? One solution to this problem would be to have the ability to sketch diagrams directly onto a computer screen and have them accurately translated into formalised representations; thus combining the ease of designing using pen and paper with the benefits of a computer (Bailey and Konstan 2003; Plimmer and Apperley 2003). As well as formalising a diagram, sketch recognition can allow sketches to be animated and executed providing a interaction level far above pen and paper (Davis, Agrawala et al. 2003; Thorne, Burke et al. 2004).

The Tablet PC supports sketching of characters and recognition of these but has much weaker support for general diagram drawing and accurate recognition. InkKit (Chung, Mirica et al. 2005; Young 2005; Plimmer, Tang et al. 2006; Freeman and Plimmer 2007)

is a sketch tool for the Tablet PC that is designed to bridge the gap between pen and paper and computers. It is used as a foundation for this research. InkKit allows you to sketch any type of diagram. Its recognition engine then identifies each component of the diagram and transforms it into a formal diagram in a specified format such as HTML or a Word drawing object. An example of a user interface design drawn using InkKit is shown in Figure 1a, a recognised sketch is shown in b where blue boxes outline and label each component of the diagram and its formalised version in HTML is shown in Figure 1c.

Sketch tools must have reliable and accurate recognition, and this requires the use of important sketch features to guide recognition algorithms. This research project is a search for sketch features to improve existing sketch recognition techniques. It focuses on distinguishing between text and shapes and grouping them appropriately.



a. Sketch

b. Recognised sketch

c. HTML form

Figure 1. User Interface Design

This chapter presents the motivation behind this research, states the primary thesis objectives, gives an outline of the remainder of the thesis and provides definitions of commonly used terms.

1.1. Motivation

Sketching diagrams with a stylus directly onto a Tablet PC is a new way of diagramming vastly different from the traditional mouse based interfaces that are currently in use. It allows us to interact with the computer in a way that imitates the use of pen and paper (Freeman and Plimmer 2007). Sketch tools serve to enhance this interaction for diagramming by providing automatic recognition, beautification and translation of hand drawn diagrams and even allowing animation and execution of these designs. A critical part of sketch tools is being able to reliably recognise sketched diagram components. While considerable progress has been made on sketch recognition, it is still error prone and few sketch tools attempt to recognise both writing and drawing.

Most diagrams require recognition of both words and shapes at the same time; however this is a difficult problem to tackle. Figure 1a, shows a typical user interface design that can be used to highlight this challenge. For example, what is it that makes the circles in front of the words “Male” and “Female” circles indicating radio buttons and not the letter ‘o’? Geometrically a radio button is the same as the letter ‘o’ in most cases, however there may be other features such as the distance of the radio button from other strokes or a difference in the time taken between drawing the radio button and other strokes in contrast to the time taken between drawing a letter ‘o’ and other letter strokes that are surrounding it. Such features may allow sketch tools to distinguish between text and shape strokes successfully allowing the sketch to be recognised and transformed into a formalised version accurately as shown in Figure 1b.

The implementation of this research is embedded into the InkKit sketch tool. The existing InkKit system automatically divides the ink into words and shapes, and then recognises the words using the Tablet Operating System recogniser and shapes using a variant of

Rubine's algorithm (Rubine 1991). Once the basic shapes and words are recognised they are combined back together to suggest the most probable diagram components. Section 2.2 describes InkKit in more detail. This method of dividing the words and shapes automatically is preferred over using a modal interface requiring users to explicitly specify how their actions with the stylus should be interpreted, which often interrupts and distracts users from their tasks doing little to preserve human's natural sketching approaches used on pen and paper (Plimmer 2004). The aim of our research is to greatly improve the algorithm that divides the ink into either text or drawing as this is the area where we can expect improvements in diagram recognition (Young 2005).

Most sketch recognition mechanisms measure various features of the strokes in a sketch to guide its recognition algorithms. However what is lacking is a definitive, principled set of the most significant features that can be used to provide an accurate division of the shape and text strokes in a sketch. Therefore our goal is to find these features in order to improve division of shape and text strokes and in doing so improve the accuracy of sketch recognition as a whole.

1.2. Thesis Objectives

For sketch tools to support intelligent interaction they must contain reliable and accurate recognition algorithms. The primary objective of this research is to investigate how improvements can be made to existing sketch recognition techniques for diagrams. One crucial component of the recognition process is to divide text and shape strokes so that each can be recognised separately using algorithms that are best suited to their type. Therefore the key objective of this research is to improve the divider of text and shape strokes. Fulfilling this goal will in turn improve sketch recognition.

Most sketch recognition methods rely on quantifiable features of ink strokes to direct recognition. This is demonstrated by the review of related work in Section 2.1. For that reason we have focused our investigation on identifying the features of strokes that are

most useful to distinguish text from shape strokes. This is accomplished by statistically analysing quantifiable features to identify a set of significant features to improve the divider and therefore improve recognition of hand drawn diagrams.

Therefore the first step is to identify a feature set comprised of the features of strokes that may be significant to division; this includes measures of pressure, time, intersections, size, curvature and Tablet OS recognition values. The majority of these features come from an examination of those used by previous work in sketch recognition however we also include features that can now be accessed due to newly available hardware such as pressure and others we thought of.

A set of hand drawn diagrams are collected from participants using InkKit on Tablet PC's. The diagram set includes basic shapes and text, complex shapes, composite shapes and various combinations and ordering of shapes and text. It is important that these diagrams represent characteristics typical to most diagrams as measurements of the feature set are taken from these examples and subsequently used to determine which features are significant to dividing text and shape strokes.

The sketches are processed to obtain the data on each feature. This involves calculating each feature in the feature set for each stroke in the diagrams collected. This is done automatically within InkKit and saved to a spreadsheet. Additional features are added manually for analysis purposes, this includes classifying each stroke as either a TEXT or SHAPE stroke as base data for the statistical analysis. All the data is then collated into one dataset ready for analysis.

The statistical analysis of this dataset is done using a tree-based partitioning technique (Breiman, Friedman et al. 1984; Venables and Ripley 2002). This examines the distribution of the measurements for the features from the dataset and finds the optimal value of the feature where the observations can be split so that most text strokes lie in one group and most shape strokes are in the other. A completely accurate partition of the dataset is impossible when dealing with highly variable data such as this. Therefore the

most significant features are those that can split the dataset with the least amount of misclassified strokes, i.e. with a minimal amount of text strokes in the shape group and a minimal number of shape strokes classified as text. A statistical software package, R (R Development Core Team, 2006), is used to perform the analysis which subsequently constructs a binary classification tree containing all the significant features and the values of these features that are used to partition the strokes into text and shape groups (Venables and Ripley 2002).

The resulting classification tree identifies the features as being significant to dividing text and shape strokes. As some features require data from their next or previous stroke, the tree is pruned to deal with some special case strokes, the first, last or only stroke in a diagram as they do not have a next or previous stroke.

An alternative approach is investigated to deal with these special cases. A modified dataset with extreme values substituted in for missing data on the special case strokes is created and analysed using the same statistical technique. The classification tree resulting from this analysis, referred to as the Version II divider, identifies further features as significant to text and shape stroke division.

Both the Version I and II dividers are implemented within InkKit. A preprocessing step to division to group strokes that may be apart of the same shape or word is also implemented. This allows us to determine if classifying groups of related strokes as one rather than classifying single strokes independently has any benefit to division.

The new dividers are evaluated against InkKit's existing divider and the Microsoft divider. The original diagram set collected as data for the feature analysis is used for the evaluation as well as a new diagram set of examples that were not included previously. The evaluation establishes which divider is the most accurate and therefore if the features identified by our analysis improve the division of text and shape strokes. If the new dividers are in fact more accurate then we can conclude that they are an improvement on current sketch recognition techniques.

The key contributions of this research include:

- The identification of a set of ink stroke features that may be beneficial to divide shapes and text and could also be used for refining other steps in the recognition process.
- The use of a formal statistical technique to identify significant features.
- More accurate division of shape and text strokes as a consequence of using the identified feature set.
- An improvement to general sketch recognition techniques resulting from a more accurate divider.

1.3. Thesis Outline

The remainder of this thesis is organised as follows.

Chapter 2 provides an overview of the background to this research. It begins with a review of related sketch recognition work including an overview of tools that support both text and shape recognition and the approaches used to divide the two groups and also those that do not allow for any character recognition. This follows with a detailed description of some key sketch tools and the feature sets that they employ to guide their recognition processes including a summary of some of the recent work that has taken advantage of newly available pressure data in pen-based environments. A description of the InkKit diagramming sketch tool is also given here, focusing on the features and algorithms used by its recognition engine.

Chapter 3 proceeds to detail the methodology that is used to identify the most significant features of sketches to aid division of strokes. This includes a discussion of the process of feature discovery to identify the feature set for analysis; the method of data collection that is followed; an explanation of the formal statistical analysis technique, tree-based partitioning, that is used to find the distinguishing features of text and shape strokes and constructs a tree accordingly; and the plans for implementation of the divider and evaluation of its performance.

The feature set is described in Chapter 4. This is broken into the categories of features measuring an aspect of pressure, time, intersections, size, curvature, Tablet OS recognition values and other features used as base data for analysis. A detailed description of each feature is given as well as the reasons why they were chosen.

The feature analysis is presented in Chapter 5. It begins by revealing the way in which data is collected which includes a description of the diagram set chosen to be used as example sketches and the participants that drew these sketches; then how the diagrams are processed to produce a dataset. The statistical analysis of the dataset is detailed, concluding with the results which are the set of distinguishing features of shapes and text and a classification tree that is used to combine these features.

The implementation of the new divider represented by the classification tree is described in Chapter 6, which includes an overview of the technology used for implementation; an outline of the basic division process using a classification tree; the strategy used to deal with special cases of stroke classification for the divider; an introduction of a second classification tree, the Version II divider, that was produced through the statistical analysis of a modified dataset; details of a preprocessing technique involving stroke grouping before division and the experiences gained from implementing the dividers.

Evaluation of the resulting dividers of shape and text strokes is presented in Chapter 7. After giving an overview of the evaluation method used, this describes the evaluation results of each divider in terms of their performance when dividing the original diagram set that was used for the statistical analysis. These results are followed by the evaluation of the dividers when tested using a new diagram set of examples that are not included in the training diagram set.

Chapter 8 presents a discussion of the research undertaken and its role in the sketch recognition field. We conclude in Chapter 9 and present directions for future work.

1.4. Definition of Terms

Term	Definition
Stroke	"A user pen-down, pen-move, pen-up sequence" (Plimmer 2004)
Gesture	"A stroke that involves an action" (Plimmer 2004) such as delete, edit, select etc.
Ink	"A general term for any hand-drawn symbols / letters/ words that remain on the sketch" (Plimmer 2004)
Text	A single character or word
Shape	A graphical symbol other than text
Stroke Feature	A measurement of a particular characteristic of a stroke
Divider	A recognition component that separates text and shape strokes into two groups
Division	The process of separating text and shape strokes into two groups
Sketch	A hand drawn diagram or collection of strokes
Decision / Classification tree	A binary tree containing rules at each node and a type class at each leaf. Here these rules are used to determine the type of a stroke as either a SHAPE or TEXT stroke

Table 1. Definition of Terms

Chapter 2

Background

This chapter provides background information regarding sketch recognition techniques. Section 2.1 reviews previous work in diagram recognition to establish what approaches exist and identify the features of strokes that are considered important in solving recognition problems. Section 2.2 describes the InkKit diagramming tool that is used as a foundation for the implementation and evaluation of the text and shape divider resulting from this research.

2.1. Related work

This section reviews the techniques that are currently used in sketch tools for recognition. It discusses diagramming tools that provide recognition of both shapes and text, and those that only provide shape recognition. It then proceeds to review the recognition approaches of various sketch tools in depth identifying the features and algorithms that are used.

Most previous work in sketch recognition concentrates on basic shape and gesture recognition, where gestures are control commands for the computer such as the editing actions delete, move and copy. Very few sketch recognition tools attempt to recognise both text and shapes. Freeform (Plimmer 2004) is one tool that provides a modal interface to separate text and shapes. A modal interface requires the user to explicitly specify how their actions should be interpreted, for example in Freeform the user indicates whether they want to write or draw by selecting either the writing or drawing icon. This allows the application to easily identify shape strokes as those drawn while the drawing icon is

selected and text strokes as those drawn when the writing icon is selected. The shapes are then recognised with a variant of Rubine's algorithm (1991) and text is matched to a vocabulary.

InkKit is a sketch tool that aims to support non-modal text and shape recognition by automatically dividing text and shape strokes. This is done by examining various distinguishing features of these strokes and proceeding to recognise shapes with a variant of Rubine's algorithm (1991) and text with the Tablet PC text recognition engine, see Section 2.2 for a full description of InkKit. The features used by InkKit's divider are determined through statistical analysis (Young 2005). Another sketch recognition system, Tahuti (Hammond and Davis 2002), allows text to be written in its UML class diagrams and uses location and size features to separate text from drawings before recognition of the diagram, see Table 6 for these feature descriptions. There is no formal method of feature selection mentioned; we assume features are chosen by visual inspection. It does not perform text recognition, it simply ignores the text strokes and relies on text entered using the keyboard.

MathPad^2 (LaViola and Zeleznik 2004), a tool for recognition of hand written mathematical expressions has a non-modal interface which allows some text and shape recognition. It uses the Microsoft Tablet OS Divider to divide shape and text strokes. However they acknowledge that this divider needs further research to be able to accurately solve the complex problem of stroke division. Lank (2003) describes a framework that is used for mathematical equations, molecular diagrams and UML. This supports recognition of both shapes and text however he does not give any details as to how he divides the strokes before recognition. Sumlow (Chen, Grundy et al. 2003), a sketch tool for UML diagrams, also has a non modal interface for text and shape recognition. Shapes are recognised first and then dashed lines are automatically added in certain parts of the recognised shape to indicate the places where text may be written. The text is grouped according to proximity and subsequently recognised. Other annotations may also be added but this text is not recognised.

Sketch recognition tools that do not support text recognition include user interface design tools Silk (Landay and Myers 1996), Demais (Bailey and Konstan 2003), Denim (Hong and Landay 2000; Lin, Newman et al. 2000) and SketchiXML (Coyette, Faulkner et al. 2004); the UML diagramming tool Knight (Damm, Hansen et al. 2000); a tool for architecture design The Electronic Cocktail Napkin (Do and Gross 2001); the domain independent application LADDER (Hammond and Davis 2003) which has a specified grammar that is used for recognition; and a recognition system that uses Zernike moments and template matching (Hse and Newton 2004; Hse and Newton 2004; Hse, Shilman et al. 2004). Additional sketch recognition applications are described in more detail below.

Although recognition algorithms can differ, ranging from statistical pattern matching (Rubine 1991) and fuzzy logic (Jorge and Fonseca 1999; Fonseca and Jorge 2000; Fonseca and Jorge 2001; Fonseca, Pimentel et al. 2002) to the use of Bayesian networks (Alvarado and Davis 2004; Liao, Wang et al. 2005), one commonality is the identification of distinguishing features of strokes in a sketch to guide the recognition process. It is these features that are of most interest here as they form the basis of recognition and may provide a contribution to a set of features that are useful in distinguishing text from shapes in a divider.

Rubine's work (1991) in the area of gesture recognition is worthy of close inspection as it is used by numerous sketch recognition systems (Landay and Myers 1995; Landay and Myers 1996; Damm, Hansen et al. 2000; Lin, Newman et al. 2000; Chen, Grundy et al. 2003; Plimmer and Apperley 2003; Plimmer 2004) including InkKit (Chung, Mirica et al. 2005; Young 2005; Plimmer, Tang et al. 2006; Freeman and Plimmer 2007). A gesture here is identified as a single stroke shape drawn as a control instruction for the system, for example a cross gesture indicates a delete instruction. The main advantage of Rubine's recognition method is that gestures may be recognised using drawn examples. The disadvantage of this algorithm is that it requires gestures to be drawn using single strokes (Young 2005).

A two step recognition method is used:

1) Calculate a feature vector, f , for the unclassified gesture, then
2) Classify the gesture using the feature vector f by matching it to one of the possible types of shape classes. This is done using a linear discriminator based on the example shapes.

Rubine chose the feature set using empirical analysis. However he used the following four conditions to guide his choice (Rubine 1991; Rubine 1991):

1) The number of features should be small as long as recognition is still accurate because it is more efficient with a small number of features and shape classes.
2) Calculation of each feature should be efficient enough so that the size of a gesture has no effect.
3) Features should be meaningful so that they can also be used for information to aid processes other than recognition.
4) The distribution of a feature should be similar to a Gaussian distribution as this provides the optimal conditions for the classifier.

The 13 features Rubine identifies for use in the feature vector are described in Table 2, with some illustrated in Figure 2. Rubine found that features measured using discrete counts rather than some continuous measurements are not as effective. He states that a possible reason for this is that his fourth condition is met more closely by continuous features rather than discrete features as the amount of error is smaller. Features 1, 2, 6 and 7 from Table 2 involve measurements of angles using the sine or cosine of the angle rather than the angle itself. This is so that these features also follow his fourth condition and remain more continuous in nature. However, notice that features 4, 9, 10 and 11 use the raw angle value. Feature 4 is the bounding box angle which is always between 0 and $\pi/2$. Features 9, 10 and 11 are calculated using an arctangent which ensures that the angles are always between $\pm\pi$. The feature set is fully extensible, where additional features can be added if the current set is not sufficient.

Features	Description	Calculation
f_1 Cosine of initial angle	Cosine of the gestures initial angle, see Figure 2.	$f_1 = \cos\alpha = \frac{(x_2 - x_0)}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}}$
f_2 Sine of initial angle	Sine of the gestures initial angle of the gesture, see Figure 2.	$f_2 = \sin\alpha = \frac{(y_2 - y_0)}{\sqrt{(x_2 - x_0)^2 + (y_2 - y_0)^2}}$
f_3 Length of bounding box diagonal	The gestures bounding box diagonal line length. See Figure 2.	$\sqrt{(x_{max} - x_{min})^2 + (y_{max} - y_{min})^2}$
f_4 Angle of bounding box diagonal	The gestures bounding box diagonal line angle. See Figure 2.	$f_4 = \arctan \frac{y_{max} - y_{min}}{x_{max} - x_{min}}$
f_5 Distance between first and last point	Distance from the first point of the gesture to the last point of the gesture. See Figure 2.	$f_5 = \sqrt{(x_{p-1} - x_0)^2 + (y_{p-1} - y_0)^2}$
f_6 Cosine between first and last point	Cosine of the angle between the first and last point of the gesture. See Figure 2.	$f_6 = \cos\beta = (x_{p-1} - x_0) / f_5$
f_7 Sine between first and last point	Sine of the angle between the first and last point of the gesture. See Figure 2.	$f_7 = \sin\beta = (y_{p-1} - y_0) / f_5$
f_8 Total gesture length	Length of the gesture.	Let $\Delta x_p = x_{p+1} - x_p$ and $\Delta y_p = y_{p+1} - y_p$ $f_8 = \sum_{p=0}^{P-2} \sqrt{\Delta x_p^2 + \Delta y_p^2}$
f_9 Total angle traversed	The gestures total angle. See Figure 2.	Let $\theta_p = \arctan \frac{\Delta x_p \Delta y_{p-1} - \Delta x_{p-1} \Delta y_p}{\Delta x_p \Delta x_{p-1} + \Delta y_p \Delta y_{p-1}}$ (see f_8 calculation) $F_c = \sum_{p=1}^{P-2} \theta_p$
f_{10} $\sum$ |angle at each point|	Sum of the absolute value of the angle at each point of the gesture. See Figure 2.	$f_{10} = \sum_{p=1}^{P-2} \lvert\theta_p\rvert$ (see f_9 calculation)
f_{11} $\sum$(angle at each point)^2	Sum of the squared value of the angle at each point of the gesture. This is used to distinguish between smooth and sharp gestures. See Figure 2.	$f_{11} = \sum_{p=1}^{P-2} \theta_p^2$ (see f_9 calculation)
f_{12} Maximum speed (squared)	The maximum speed squared of the gesture. This is a dynamic component that does not have to	Let $\Delta t_p = t_{p+1} - t_p$ where t is time $f_{12} = \max^{P-2} \frac{\Delta x_p^2}{} + \frac{\Delta y_p^2}{}$

	be used for some applications	$\sum_{p=0} \Delta t_p^2$
f_{13} Total duration	Total duration of the gesture. This is a dynamic component that does not have to be used for some applications	$f_{13} = t_{P-1} - t_0$ where t is time

Table 2. Rubine's Feature Set from (Rubine 1991)

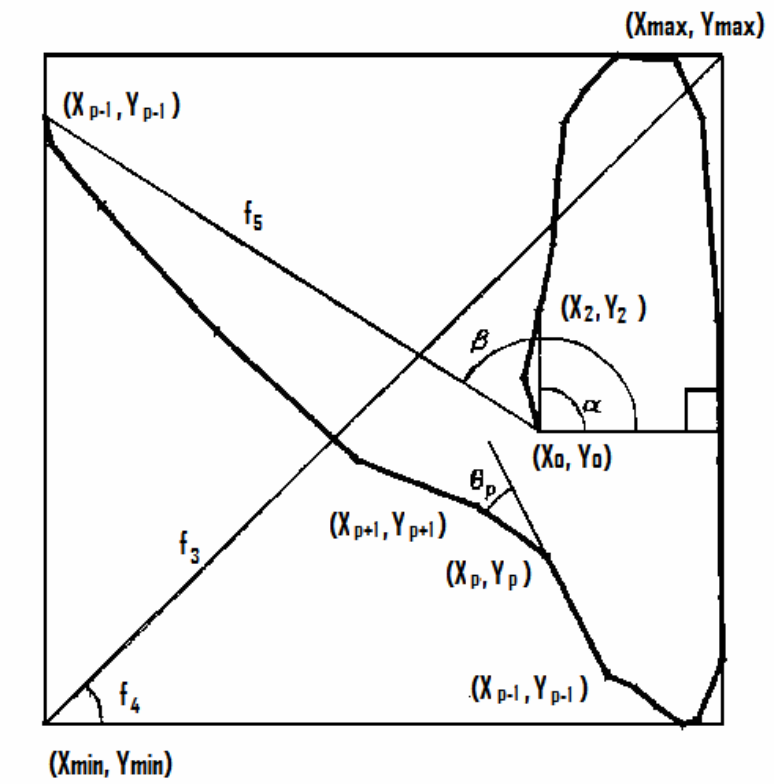


Figure 2. Rubine's Features Diagram from (Rubine 1991)

Figure 3 illustrates each step of Rubine's algorithm. On the left there are the drawn example shape classes representing all the possible shapes that a gesture could be. The examples are used to calculate weights *w* for each feature. First an average feature vector $\bar{f}$ for each shape class is calculated using all the features in Table 2 then the common covariance matrix Σ is constructed from the sample covariance matrix of each shape class calculated from each average feature vector $\bar{f}$. This allows the weights *w* for each feature to be calculated. On the right hand side there is a user drawn stroke of a circle that is waiting to be recognised. The strokes feature vector *f* is calculated, again using the features described in Table 2. The feature vector is used in conjunction with the weights

w to generate a probability score for each shape class. The highest probability is for the circle class therefore the gesture is recognised as a circle.

Rubine's evaluation of this recognition algorithm shows a recognition rate of 96.8% for a recogniser with 30 shape gesture classes and 100 training examples per class, and a recognition rate of 95.6% with 10 example gestures per class. Others have reported lower rates for shape recognition when using Rubine's algorithm where Young (2005) had a 84% recognition rate and Plimmer (2004) found its rate to be 86%. These less reliable results illustrate the difficulty in creating recognisers that can be applied as successfully to new datasets as they are to the training data.

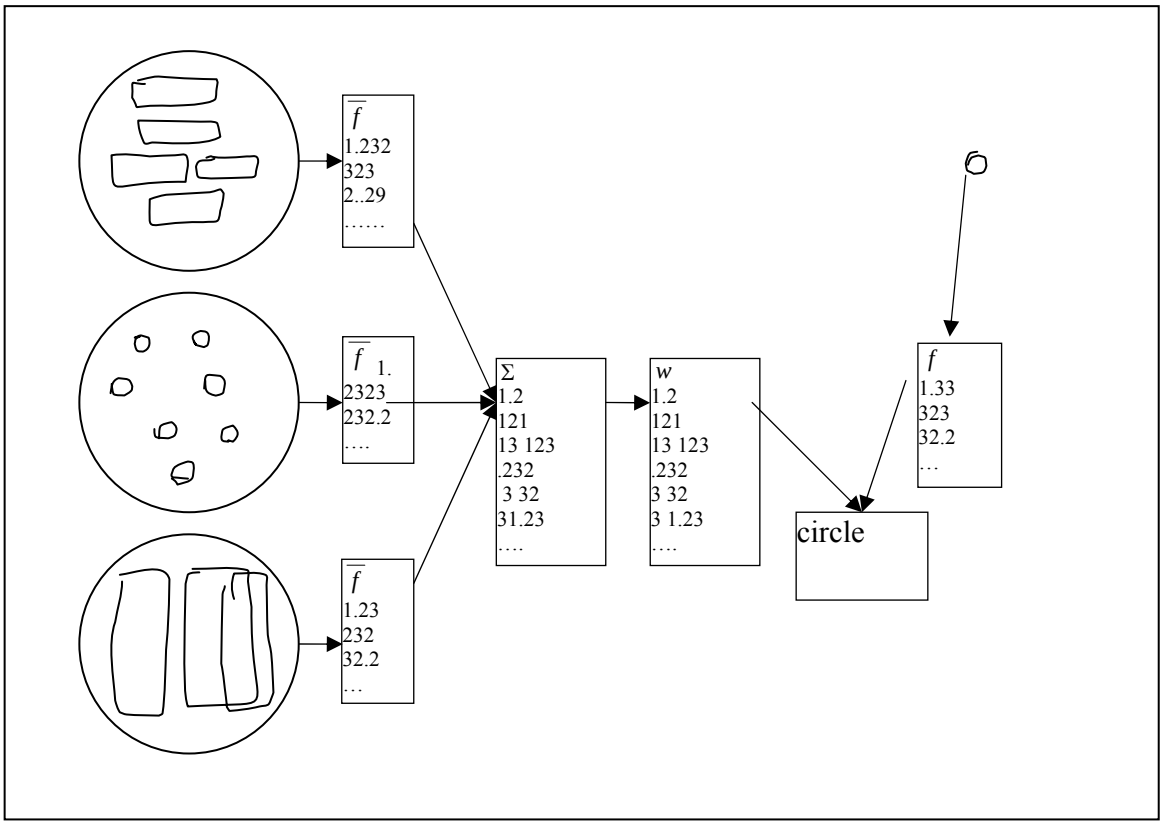


Figure 3. Graphical Representation of Rubine's Algorithm from (Plimmer 2004)

Sezgin & Stahovich et al (2001) describe a system for recognition of mechanical engineering design sketches, in particular the early processing of these sketches. These sketches consist of basic shapes however there is no fixed set of shapes that can be used here. Three phases to early processing of the sketches are used, approximation,

beautification and basic recognition. The approximation phase involves fitting lines and curves to the sketch, the aim being to reduce the stroke to a simpler description. This is done by using features of stroke speed, curvature and direction to detect vertices in the strokes and construct lines and curves based on these vertices, see Figure 4 for an illustration of the direction, curvature and speed of a square.

Stroke speed is found to decrease when drawing a corner and therefore is useful in detecting these corners, Figure 4 illustrates this where each minima in speed values corresponds to a corner of the square. However as this data can be noisy it is combined with stroke direction data and curvature to further confirm the presence of a vertex. A description of these features is also given in Table 3. Average based filtering is used on the speed and curvature data to further reduce effects of noise in data where a threshold is used to limit the parts of the data that are considered as possibly containing a vertex. For example only areas of the stroke that are already at a low speed and a high curvature are examined. The actual threshold is determined empirically using the mean for curvature and 90% of the mean for speed. One problem encountered with using speed data is that for shapes with short edges such as a thin rectangle there may not be a large enough increase in speed on the short edge to allow for detection of a minima in speed for the corners, however the curvature data can make up for this.

Features	**Description**
Speed	Location of minima of speed indicates the location of vertices, illustrated in Figure 4.
Curvature	This is the change in direction with respect to arc length where the maxima of absolute value of curvature indicate vertices, illustrated in Figure 4. This is calculated as the absolute value of the angle between tangent to the curve at a point and the x axis / the distance travelled along the curve.
Stroke direction	A change in stroke direction can indicate a vertex when combined with curvature and speed information, illustrated in Figure 4.
Euclidean distance	Using original stroke data the Euclidean distance and arc length between two vertices is measured to compare with each other to determine if a line or curve should be fitted between those vertices. If they are 1:1 it should be a line, if it is >1 a curve is fitted.
Arc length	

Table 3. Features for Vertex Detection (Sezgin, Stahovich et al. 2001)

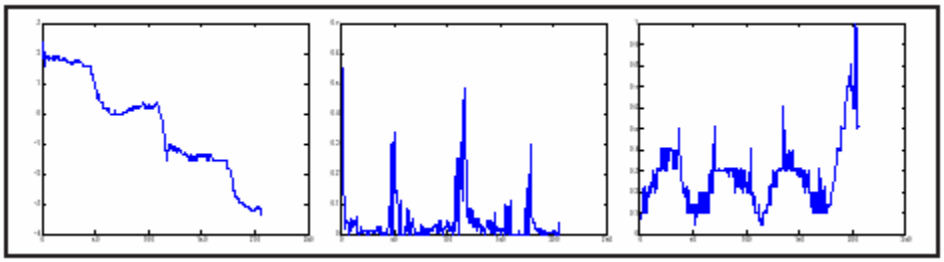


Figure 4. Direction, Curvature and Speed Graphs for a Hand Drawn Square from (Sezgin, Stahovich et al. 2001)

Once the vertices have been detected there can be either lines or curves fitted to connect each vertex. Using a comparison of the Euclidean distance between 2 vertices to the arc length between those same vertices it can be determined if a curve is appropriate. This comparison should be 1:1 for lines and >1 for curves. Curves are then approximated using Bezier curves. The features summarised in Table 3 are determined by empirical observation.

The second phase of beautification tidies up the results of the approximation phase. The last phase, basic recognition, uses templates to recognise the basic shapes that exist. For example a rectangle is made up of 4 lines therefore a stroke with four lines detected in the approximation phase is most likely a rectangle. Evaluation of a system employing this recognition technique shows 96% success rate in recognition.

Calhoun & Stahovich et al (2002) describe a shape recognition system that uses the approximation phase above (Sezgin 2001; Sezgin, Stahovich et al. 2001) to segment the strokes into lines and arcs and then employs semantic networks using stroke features for basic shape recognition. The system is trained with example shapes where each of the examples for a particular shape are studied to determine the characteristic features of the shape, the number of lines and arcs that the shape is made up of and the relationship between these segments. Therefore a general semantic network is constructed to represent this shape, this is known as the shapes definition. The nodes in the network are either lines or arcs and the links are the relationships between them. The features that are considered as forming relationships between the segments are described in Table 4, also there are general features measured for each shape class in Table 5. There is no mention

of the method used to identify these features. Figure 5 illustrates the semantic network for a square. To recognise a user drawn shape, first segment the strokes into lines and arcs then construct a semantic network for the strokes. This network is then compared with each shape's definition network. The shape whose definition fits the strokes network with the least amount of error is considered as the shape that the strokes represent, and therefore the strokes are recognised.

Features	Description
Intersections	The existence of intersections between arcs or lines
Intersection location	The relative location of an intersection
Angle at intersections	The angle between intersecting lines
Parallel lines	The existence of parallel lines

Table 4. Features Identified as Relationships Between Lines and Arcs (Calhoun, Stahovich et al. 2002)

Features	Description
Type	Line or Arc
Length	Length of strokes measured in pixels
Relative length	Length of strokes measured as a proportion of the total of all of the stroke lengths in the shape. For example the relative length of each side of a square would be 25%
Slope	Gradient of the line, for lines only
Radius	The radius of an arc, for arcs only

Table 5. General Features of each Shape Class (Calhoun, Stahovich et al. 2002)

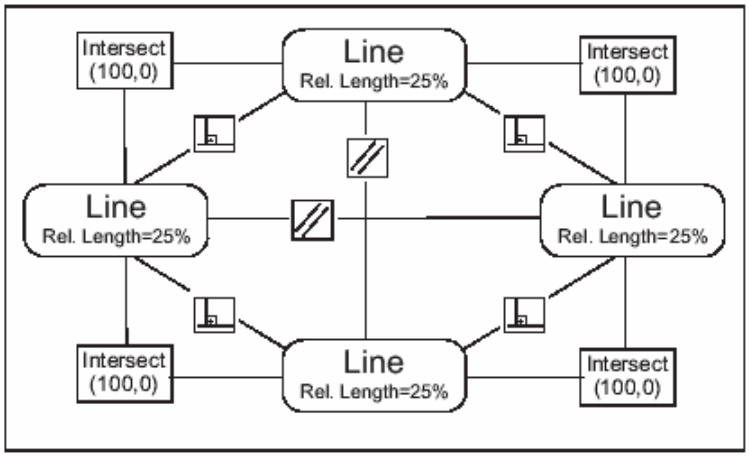


Figure 5. The Semantic Network Definition for a Square from (Calhoun, Stahovich et al. 2002)

Object	Feature	Description	
Rectangle	Number of lines	There must be at least 4 line segments	
	Endpoints	The start and end of the strokes must be close together	
	Parallel lines	After all line segments with adjacent endpoints and similar slope are joined together there must be exactly 4 line segments left. Parallel and perpendicular lines are then tested.	Must be 2 sets of parallel lines
	Perpendicular lines		Adjacent lines must be perpendicular
Ellipse	Single stroke	Must be composed of a single stroke as circles are almost always drawn in this manner	
Arrow	Bounding box	These features are used to help classify each part of the arrow as part of the arrow shaft or head.	
	Slope		
	Y-intercept		
	Length	The length is used to double check if the classifications of each line segment as part of the arrow shaft or head is correct. Each line segment should be > ½ of the ideal length of the section (shaft or arrow head) of the arrow that it is thought to represent.	
Editing "move" command	Cursor time	The cursor must be held over a sketch with minimal movement for > 0.5 seconds.	
Editing "delete" command	Intersections	A class or arrow is deleted if horizontal, vertical or diagonal lines cross over it > 4 times.	
Text	Size	Text is not recognised but is classified as text.	Text must be small in comparison to the size of classes
	Location		Text must be inside or near a class

Table 6. Tahuti Features from (Hammond and Davis 2002)

Tahuti (Hammond and Davis 2002) is a multi-stroke sketch recognition system for UML class diagrams. It relies on features of strokes for recognition. A multi-layer framework is used beginning with pre-processing, then selection, recognition and identification of strokes. Pre-processing uses stroke processing techniques outlined by Sezgin & Stahovich et al (2001) who detect vertices and allow strokes to be fitted to lines and arcs. Here strokes are fit to an ellipse, line, polyline or a complex shape. Selection involves grouping strokes that are most likely to be part of the same object using spatial and temporal rules. The recognition stage uses various algorithms to find the possible interpretation of a collection of strokes into shapes or editing actions. A different algorithm exists for each

of rectangle, ellipse, arrow and editing actions based on each of their features. These are the only shapes considered here as Tahuti is domain specific to UML class diagrams. Some of the features used to recognise each shape class are described in Table 6. It is not clear how this feature set is compiled. The identification stage is responsible for selecting a final interpretation for the strokes. Sumlow (Chen, Grundy et al. 2003) uses a similar approach for recognition of UML diagrams.

Features	Description
1. Direction	The direction for each point is calculated for vertex detection where changes in direction indicate a vertex. The following formula is used, where n denotes the nth point, and x, y are the coordinates of the point $d_n = \arctan(\frac{y_{n+1} - y_n}{x_{n+1} - x_n})$
2. Curvature	The curvature, which is the change in direction with respect to path length, is calculated for each point of a stroke for vertex detection. Points with high curvature indicate a vertex. The following formula is used to calculate curvature, which is basically the absolute value of the angle between tangent to the curve at a point and the x axis, φ, divided by the distance travelled along the curve, D $c_n = \frac{\lvert \sum \varphi (d_{i+1} - d_i) \rvert}{D(n-2, n+2)}$
3. Feature Area Verification	The feature area for lines, points and arcs are illustrated in Figure 6 by the shaded regions. The standard area is the area of a fitted line or circle to a stroke. For example if the ratio of feature area: standard area is < 1 the stroke is confirmed as being a line segment. Confirmation of a circle matches the feature area of a circle to the standard area of the fitted circle.
4. Direction slope	The direction graph for a circle/ellipse can be fitted to a line with a slope of 2π / number of points in the stroke. A stroke is considered as a probable circle/ellipse if a line of this slope can be fitted to its direction graph. For an arc a line with a slope < 2π / number of points should be able to fit its direction graph.

Table 7. Features used by (Yu and Cai 2003)

Yu and Cai (2003) present a domain independent sketch tool for shapes. Recognition of these shapes is carried out in two stages, imprecise stroke approximation and then the post-process. Imprecise stroke approximation matches strokes to primitive shapes e.g. Lines, arcs, circles, ellipses and helixes using various geometrical features of strokes as shown in Table 7. These features are determined empirically. It first detects vertices using

features 1 and 2 in Table 7, a similar process to Sezgin & Stahovich et al (2001) and then approximates lines and curves according to the vertices using features 3 and 4. The post-process takes the primitives and finds any relationships between them, gets rid of unnecessary lines or arcs and recognises the primitives that remain as basic shapes such as rectangles, triangles and arrows. For example a rectangle is recognised as having 4 lines, 2 that are parallel and adjacent lines with an angle between them > 80 degrees. Evaluation of this system shows recognition rates between 70% and 98% for different types of shapes.

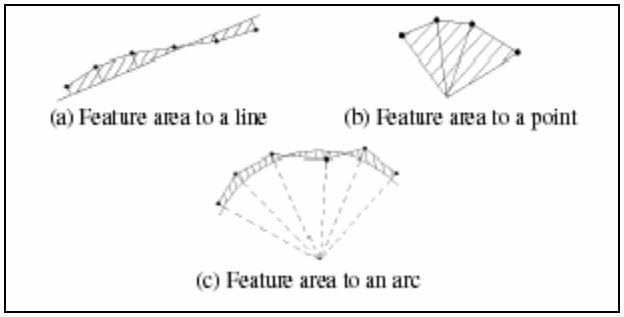


Figure 6. Feature Area Examples from (Yu and Cai 2003)

Features	**Description**
Local distance	Distance between points $D_i = \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}$
Angle	Angle, A_i, between the X orthoaxis and the maximum point from adjacent points P_{i-1} and P_i where the positive direction is anti-clockwise.
Curvature	$C_i = A_{i-1} - A_i$
Curvature point	$CP_i = (\cos(C_i), \sin(C_i))$ The curvature points of clockwise ellipse are distributed around $(\cos(-\alpha), \sin(-\alpha))$, for multi-lines (1,0) and for anti-clockwise ellipse around $(\cos(\alpha), \sin(\alpha))$
Curvature class	$CC_i = k \mid D(M_k, CP_i) <= D(M_k, CP_i)$ where $j.k \in (0, 1, 2), j \neq k$ and D is the Euclidean distance.

Table 8. Features Identified by (Li, Zhang et al. 2005)

A basic shape recognition method presented by Li & Zhang et al (2005) claims it can be used to solve efficiency and coverage problems in sketch recognition. Strokes are drawn and when there are enough sample points in a stroke to fill a lag window features are calculated from the points, described in Table 8, to find the user intention, which is either

to draw a clockwise elliptic arc, a multi-line, or an anti-clockwise elliptic shape. It is not clear as to the origin of these features. If when new points are added, the previously recognised shape does not fit the new points, the algorithm reviews its classification. Probabilities are calculated with these features to determine which class of shape the user is drawing using fitting error, multi-line fitting probability and weighted curvature class.

CALI (Jorge and Fonseca 1999; Fonseca and Jorge 2000; Fonseca and Jorge 2001; Fonseca, Pimentel et al. 2002) is a pen-based system that can recognise multi-stroke basic shapes and gestures. It uses a combination of geometric features, fuzzy logic and heuristics to perform shape classification. It is able to recognise triangles, rectangles, diamonds, circles, ellipses, lines, wavy lines, arrows, crosses and the gestures delete, move and copy. They may be drawn using solid, bold, or dashed lines. It uses temporal adjacency rules to group strokes into sets for each shape, where strokes are considered part of the same shape if drawn before a timeout value is reached. Shapes are recognised independent of rotation, size, number of strokes and drawing style.

The geometric features are based on 3 special polygons identified from the strokes convex hull illustrated in Figure 7

1. Largest area triangle within the convex hull,
2. Largest area quadrilateral within the convex hull
3. Smallest area enclosing rectangle of the convex hull

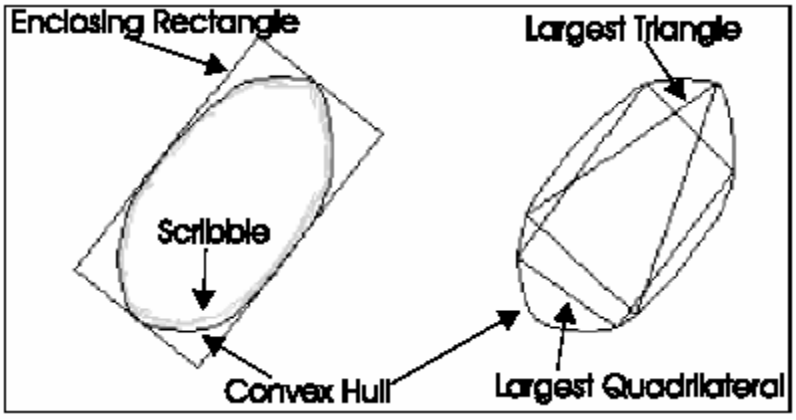


Figure 7. Polygons used to Estimate CALI Features from (Fonseca and Jorge 2000; Fonseca, Pimentel et al. 2002)

Features	Description
Thinness ratio P_{ch}^2 / A_{ch}	Perimeter squared of the convex hull / area of convex hull. This is used primarily to identify circles as they have a very low value for this ratio. Copy and Undo commands also have a low thinness ratio, slightly higher than a circle.
Aspect ratio H_{er} / W_{er}	Height of enclosing rectangle / width of the enclosing rectangle. It is used to identify straight lines, for which the ratio is near zero and wavy lines for which the ratio is a little larger as it is fatter in width. This ratio is much larger for other shapes.
A_{ch} / A_{er}	Area of the convex hull / area of enclosing rectangle. A rectangle's area of its convex hull is almost the same size as the area of its enclosing rectangle; therefore this ratio should be near 1.
A_{lq} / A_{er}	Area of the largest quadrilateral / area of enclosing rectangle. A rectangle's area of the largest quadrilateral is almost the same size as the area of its enclosing rectangle; therefore this ratio should be near 1.
A_{lt} / A_{lq}	Area of the largest triangle / area of largest quadrilateral. For a diamond this ratio is between 0.5-0.6, and larger for other shapes.
A_{lq} / A_{ch}	The area of the largest quadrilateral / area of the convex hull. This is close to 0.7 for ellipse shapes; all other shapes have a larger value.
T_l / P_{ch}	The total gesture length / perimeter of the convex hull. The total length of a delete gesture is much larger than the perimeter of its convex hull. For copy and undo commands the ratio is much smaller. This ratio is also large, >1.5, for bold lines.
A_{lt} / A_{ch}	Area of the largest triangle / area of the convex hull. Triangles and the move gesture's area of its triangle is almost the same size as the area of its convex hull, therefore this ratio should be near 1 and smaller for other shapes
P_{lt} / P_{ch}	Perimeter of the largest triangle / perimeter of the convex hull. Triangles and the move gesture's perimeter of its triangle is almost the same size as the perimeter of its convex hull, therefore this ratio should be near 1 and smaller for other shapes
W_{bb} / H_m	The width of the bounding box / the absolute value of horizontal movement. For a wavy line this is near 1.
$P_{ch} * N_s / T_l$	Perimeter of the convex hull * the number of strokes / the total gesture length. This feature can be used to distinguish dashed lines from solid lines, where the ratio is large for dashed shapes due to a greater number of strokes and a smaller stroke length.
Hollowness	This distinguishes filled from non-filled shapes. A triangle with the same barycentre as the largest triangle, but only 60% of its size is calculated. If any points of a shape lie within this triangle the shape is filled, this is expected for a delete gesture, otherwise it is hollow, and this would be true for bold lines.
P_{lq} / P_{ch}	Perimeter of the largest quadrilateral / perimeter of the convex hull.
A_{lt} / A_{er}	Area of the largest triangle / Area of the enclosing rectangle.
V_m / H_{bb}	Vertical movement / height of the bounding box.
N_s	The number of strokes for the shape or gesture. For crosses and arrows if

	the number of strokes is >= 2 it is one feature in favour of the gesture being a cross or an arrow.
Is Intersecting	The presence of intersecting strokes. For crosses, if the strokes intersect it is one feature in favour of the gesture being a cross.

Table 9. CALI Features (Fonseca and Jorge 2000; Fonseca, Pimentel et al. 2002)

The features mainly consist of different combinations of ratios of the area and perimeter of these polygons, described in Table 9. The features were chosen using percentile graphics for each possible feature which show the statistical distribution of feature values for different shape classes. The data for these graphs are taken from sample sketches collected representing each shape class. This is one of the few feature sets that is determined statistically.

Example sketches are used to train the system and develop fuzzy sets for each shape class which characterise the class using different features that are significant to the particular shape class. The strokes can be classified by calculating how similar the stroke is to each shape class using these fuzzy sets and choosing the shape class with the greatest similarity. The system can distinguish between solid, dashed and bold lines after basic shape recognition. Evaluation of CALI shows that 93.3% of sketches are correctly classified.

A method of classifying sketch strokes for shapes is proposed by Qin (2005) with the use of adaptive thresholds and fuzzy knowledge. Various features of strokes are measured for classification purposes; these are described in Table 10. It is not clear how these features were chosen. A unique approach here is that there is a weighting applied to features based on the speed of the stroke as strokes drawn more slowly are considered to be more precise.

SketchREAD (Alvarado and Davis 2004) is a multi domain diagram recognition engine which constructs Bayesian networks for stroke interpretation. It describes shapes using a hierarchical shape description language noting various features of the shape such as the primitive shapes that it is made up of and the relationships between these primitive shapes e.g. intersections, angles, length proportions. For example an arrow is composed

of 3 lines, the shaft and two lines for the arrow head. Features of the arrow include the arrow head lines being shorter in length than the shaft and there is an acute angle between the arrow head lines and the shaft.

Features	Description
Linearity	If the distance between the beginning and the end of the stroke divided by the accumulative chord length between the two points is >0.98 then a stroke is considered linear. Note for a perfect line this should be 1.
Inflection points	Strokes with inflection points indicate that there is a change in convexity between convex and concave and therefore can be classified as a freeform curve.
Direction	Changes in direction of a stroke also can be used to confirm freeform curves
Self intersections	Can indicate freeform curves such as loops.
Arc length	If the stroke is thought to be a loop, any arc lengths that are outside of the closed segment of the loop should have an arc length >20% of the strokes total arc length otherwise the stroke is most probably a circle.

Table 10. Features used by (Qin 2005)

Some recent work has taken advantage of the availability of pressure data in pen based environments. Li & Hinckley et al (2005) have tested the use of pressure as a mode switching technique to switch between sketching and gesturing commands to a computer e.g. delete, move or copy commands. A system called LEAN (Ramos and Balakrishnan 2003; Ramos, Boulos et al. 2004) has developed pressure widgets that are used to control various tasks such as target selection (Ramos, Boulos et al. 2004), mode switching and control of continuous parameters such as size by mapping the desired size to the amount of pressure applied by the stylus to the screen (Ramos and Balakrishnan 2003). Nakai & Sudo et al (2002) use pressure information to assist in recognition of Japanese characters. It uses substroke HMM to recognise characters with a feature vector, the features are described in Table 11. The Designers Apprentice (Hutton, Cripps et al. 1997) is a pen-based system for recognition of sketched engineering drawings. It uses pen pressure to distinguish between thick and thin lines where thick lines are drawn for object outlines and thin lines are used for annotation.

Features	Description
Velocity	Velocity vector between 2 points of a stroke calculated by $\sqrt{dx^2 + dy^2}$ where (dx, dy) is the difference between two points next to each other.
Direction	The direction of the velocity vector.
Pen pressure	Pen pressure sampled at a certain interval, this will help to identify pen-up and pen-down events.
Pen pressure time-derivative	This is a regression coefficient of pen pressure.

Table 11. Features Identified by (Nakai, Sudo et al. 2002)

The literature review has uncovered that there are several undeveloped areas in the research regarding sketch recognition. Firstly there are very few tools that allow for both shape and text recognition. The tools that do support both are either modal interfaces or limited in functionality. However sketch tools need to increase their flexibility if they are to draw the benefits of interaction modelling pen and paper (Black 1990; Goel 1995); (Bailey and Konstan 2003; Plimmer and Apperley 2003). InkKit is one tool designed to provide this flexible sketching environment; it is described in detail in the next section.

It is also evident that sketch recognition is comprised of two main elements, stroke *features* and *algorithms* to combine these features. These features measure aspects of a strokes curvature, size, time, intersections and also use these aspects to detect relationships between strokes. Considering that stroke features are such an important part of recognition there is little evidence of the use of formal methods to identify features that are significant to the recognition domain, most rely on empirical observation.

2.2. Review of InkKit

InkKit (Chung, Mirica et al. 2005; Young 2005; Plimmer, Tang et al. 2006; Freeman and Plimmer 2007) is a diagramming sketch tool for the Tablet PC. It is used as a foundation for this research providing a place to implement and evaluate our new text and shape divider. It is fully extensible allowing you to sketch any type of diagram. Its recognition engine then identifies each component of the diagram and transforms it into a formal diagram in a specified format such as HTML or a Word drawing object. InkKit is

developed for the Microsoft Tablet OS in Visual Studio .Net using C#. This allows good access to the Ink SDK which supports ink data and text recognition however it restricts InkKit's use to this OS and hardware (Young 2005).

InkKit is currently able to recognise organisation diagrams, directed and non-directed graphs, user interfaces, UML class diagrams, venn diagrams and musical notes. Recognition of these diagrams is example based, where user drawn examples are used to train the recogniser. Once recognised the diagram remains in sketch form and is labelled with the names of the components that are recognised. The interface allows the user to correct recognition results should errors occur. Recognised diagrams can be transformed into formalised versions in a chosen format, for example user interface diagrams can be transformed into HTML or java code, as shown in Figure 1, organisation diagrams can be transformed into Word drawing objects or gif image formats, illustrated in Figure 8. Full editing functions, erase, select, cut, copy, paste, undo, redo and move are also available when sketching.

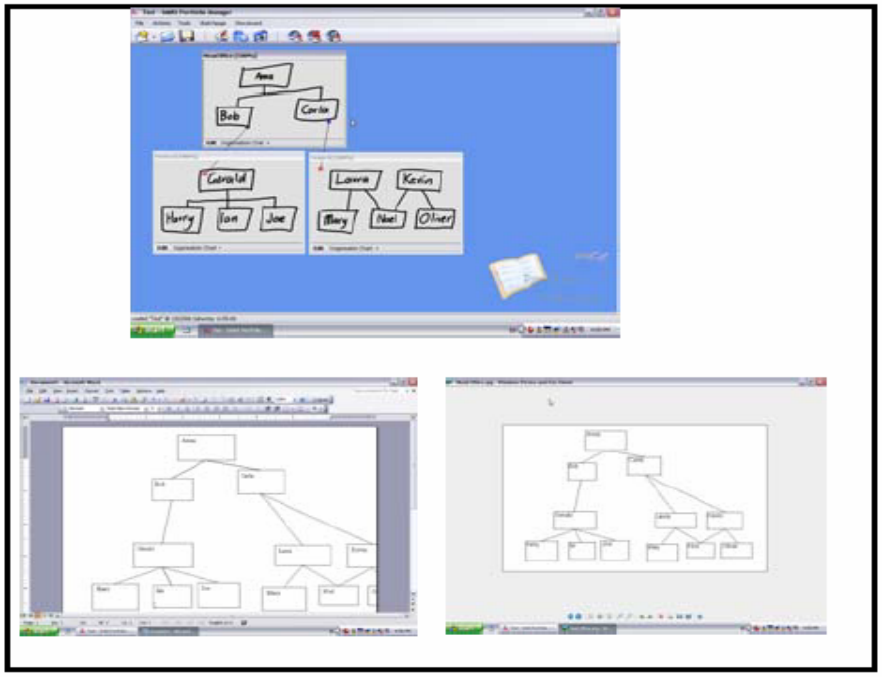


Figure 8. **InkKit Organisation Diagram Transformed into Microsoft Word and a GIF**

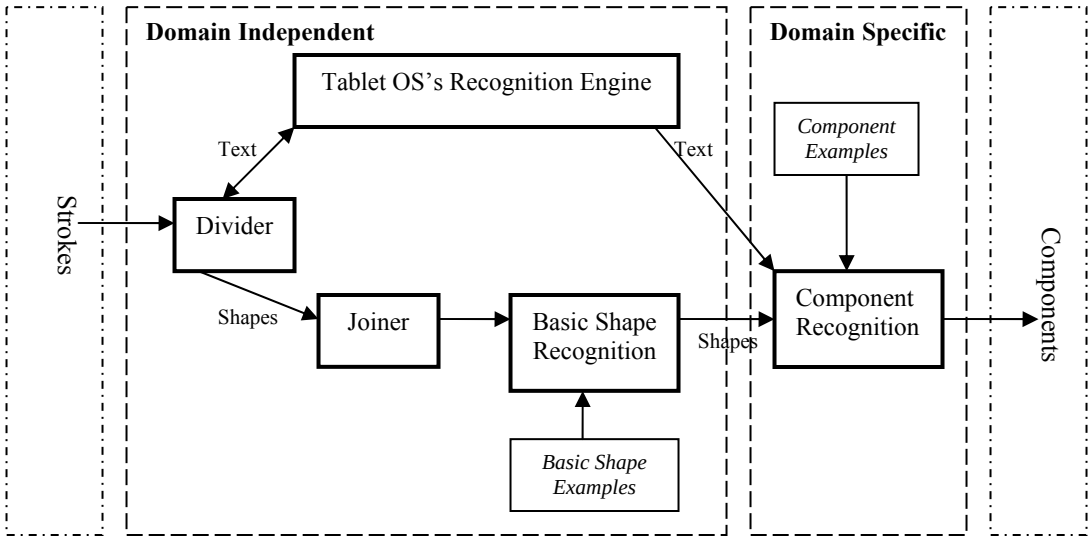


Figure 9. Architecture of InkKit's Recognition Engine

InkKit's recognition engine can be broken into five main steps, illustrated in Figure 9. They work together to take raw ink strokes from a diagram and infer the diagram components that these strokes represent. It begins with the divider, where the strokes are separated into probable text and shape strokes. The text strokes are then recognised by the Tablet OS recognition engine. The shape strokes are passed to the joiner to group together strokes that are likely to belong to the same shape before entering the basic shape recognition phase. Basic shape recognition involves taking the shape strokes and inferring the basic shapes that they represent using basic shape examples. Once the text and basic shapes are recognised they are combined back together so that component recognition can be executed, with the help of domain specific component examples, and finally the diagram components are identified. Each step and the stroke features used for the step are described in more detail as follows. Table 12 lists all the features used by InkKit's recognition. This information is gained through observation of InkKit's code and from Young's report (2005).

#	Features	Description	Origin	Use
1	Tablet OS text probability	Tablet OS text recogniser probability of the stroke being text, levels of probability are strong, intermediate or poor.	Microsoft Tablet OS Ink SDK	Divider (phase 1)

2	Amount of ink inside	Amount of ink inside the strokes bounding box. This is calculated by counting the number of points of the stroke that are inside the bounding box.	(Young 2005)	
3	Cosine of initial angle	Cosine of the initial angle of the stroke.	(Rubine 1991) see Table 2, Section 2.1 for details	Divider (phase 1) & Basic Shape Recognition
4	Sine of initial angle	Sine of the initial angle of the stroke.		
5	Angle of bounding box diagonal	Angle of the bounding box diagonal.		
6	Distance from first to last point	Distance from the first point of the stroke to the last point of the stroke.		
7	Cosine from 1st to last point	Cosine of the angle between the first and last point of the stroke.		
8	Sine from 1st to last point	Sine of the angle between the first and last point of the stroke.		
9	Total angle	Total angle traversed by the stroke.		
10	$\sum$ |angle at each point|	Sum of the absolute value of the angle at each point of the stroke.		
11	$\sum$(angle at each point)^2	Sum of the squared value of the angle at each point of the stroke.		
12	Length of the bounding box diagonal line.	Bounding box diagonal length		
13	Total length/bounding box diagonal length	Length of the stroke divided by the length of the bounding box diagonal	Adapted from (Rubine 1991) see Table 2, Section 2.1 for details	
14	Perimeter to area	Ratio of perimeter to area of the strokes convex hull.	CALI (Fonseca and Jorge 2000; Fonseca, Pimentel et al. 2002) –see Table 9, Section 2.1 for details	
15	Convex hull area ratio	Ratio of the area of the convex hull to the area of the enclosing rectangle of the stroke		
16	Score of surrounding	Divider phase 1 scores of the strokes close by. Strokes		

	strokes	considered to be close by have either their bounding boxes intersecting with this strokes bounding box or are a short distance from this stroke.	(Young 2005)	Divider (phase 2)
17	Baseline similarity	This is to help establish the existence of words – find average bottom and height of current strokes and close strokes, then generate a probability of the strokes being on the same baseline by examining how far close strokes are from the average baseline.		
18	Distance between strokes	This is a measure of closeness to other stroke groups. Add horizontal distance between bounding boxes of stroke groups to the current strokes.		
19	Relationship	Possible relationships that are considered are parent (enclosing), child (enclosed), intersecting, close or no relationship.		Component Recognition
20	Width : height	Ratio of the width to the height of the bounding box.	Adapted from CALI (Fonseca and Jorge 2000; Fonseca, Pimentel et al. 2002) –see Section 2.1 for details	
21	Smallest enclosing rectangle / sum of areas	Area of the smallest enclosing rectangle around the whole component / the sum of all the enclosing rectangle areas of each individual shape making up the component.		
22	Convex hull area / smallest enclosing rectangle area	Ratio of the area of the convex hull to the area of the smallest enclosing rectangle.	CALI (Fonseca and Jorge 2000; Fonseca, Pimentel et al. 2002) –see Section 2.1 for details	
23	Width : height of enclosing rectangle	Ratio of the width to the height of the enclosing rectangle.		
24	Number of component shapes	The number of shapes that make up the component.	(Young 2005)	

Table 12. Stroke Features used in InkKit's Recognition Engine

2.2.1 Divider

The task of the divider is to look at the features of the ink strokes and recognise which strokes are text strokes and which are shape strokes. It is important to do this so that InkKit can perform text and shape specific recognition in later phases. This method of dividing the text and shapes is preferred over using a modal interface to separate the two as it preserves human's natural sketching approaches used on pen and paper (Plimmer 2004). The division is done in two phases, first calculating a probability score of a stroke as being either text or a shape based on its own features and then extending this by considering the score given to strokes that are close by.

The first phase of division takes the features of strokes described in Table 12, features 1-15, and uses these features to calculate a probability score that indicates if the stroke is a text or shape stroke. Rubine's algorithm (1991), detailed in Section 2.1, is used to combine features 3-15 from Table 12 and calculate a probability score accordingly. The weights for each feature used by this algorithm are calculated from a predefined set of shape and text examples. This score is added to the score generated from features 1 and 2.

The second phase of division involves combining the probabilities from phase one with the score of neighbouring strokes, this is feature 16 from Table 12. Strokes whose bounding boxes intersect are considered neighbouring strokes. This process was based on the premise that if the strokes next door are most probably text strokes then the current stroke has a higher chance of also being a text stroke, and the same for shape strokes. Also features 17 and 18 contribute to the probability score generated.

If the final probability score of a stroke is > -3 then the stroke is classified as text and is passed to the Microsoft Tablet OS recognition engine for text recognition. It is unknown as to why the value "-3" has been chosen by Young (2005) . The remaining strokes are classed as shape strokes and are passed to the joiner and consequently basic shape recognition.

2.2.2 Tablet Operating System Text Recognition

The Tablet OS recognition engine is passed the collection of strokes that are considered to be text by the divider; it is used to recognise the text symbols that they represent. After text recognition only those strokes representing strings with a length > 1 letter are actually considered to be text strokes and the rest are passed back to the joiner for basic shape recognition. This is to ensure basic shapes such as circles are not mistaken for text (Young 2005). This text recognition engine is accessed through the Ink SDK via C# .Net.

2.2.3 Joiner

All shape strokes are given to the joiner so that multi-stroke shapes can have their strokes grouped together. This is important for basic shape recognition as Rubine's algorithm (1991), the method of recognition used here, is designed for single stroke gestures. Once the stroke joining process is complete all the joined up strokes are ready for basic shape recognition.

2.2.4 Basic Shape Recognition

The basic shape recognition phase is responsible for taking the joined up shape strokes and identifying the basic shapes that they each represent using a variant of Rubine's algorithm (1991). Basic shapes include lines, circles, rectangles and triangles. Features 3-15 from Table 12 are calculated to construct a feature vector for each shape stroke. Feature 13 is altered from what is in Rubine's original algorithm to be a ratio of total gesture length to the bounding box diagonal length; Rubine just uses the gesture length. Also features 14 and 15 are added to the algorithm from CALI (Fonseca and Jorge 2000; Fonseca, Pimentel et al. 2002). They are considered useful as they are constant no matter what the shapes orientation is (Young 2005). Two of Rubine's original features are not included, the total duration and maximum speed squared of the stroke, see Table 2.

Rubine's algorithm, detailed in Section 2.1, is used to classify a stroke as a basic shape by calculating a probability score indicating how well the stroke fits each basic shape class based on the feature vector and a weighting for each feature. The weights are computed using the user drawn basic shape examples. The probability score for each basic shape class that the stroke could represent are ranked and the basic shape class that has resulted in the highest score is considered as that which the stroke represents. For example if we calculate the probability of a stroke as being a line, circle, rectangle and triangle and find that the highest probability out of all of these is for a line, then the stroke is recognised as a line. The recognised basic shapes can now be passed to the component recognition phase along with the recognised text so that domain-specific diagram components maybe inferred.

2.2.5 Component Recognition

The strokes have all been recognised as either basic shapes or text at this point without regard to the domain of the diagram. Component recognition combines the shapes and text together and seeks to recognise the diagram components that they represent based on the diagram's domain. The first step involves detecting the relationships that exist between the recognised shapes and text based on the bounds of the strokes (Feature 19 from Table 12) and computing probabilities of these relationships. The probability that each recognised shape or text element is a part of each possible diagram component class is then calculated using the probabilities of relationships. Features 20 to 24 from Table 12 are also calculated for each recognised element. The Hidden Markov Model algorithm (Rabiner 1990) is applied to these features to calculate the probability that each possible combination of elements belongs to a component class using the user drawn component shape examples. These probabilities are combined together to give the final probability for each combination of shapes belonging to each component class. The highest probabilities indicate the best diagram component matches for each stroke and therefore the diagram is recognised.

2.3. Summary

The review of related work in Section 2.1 outlined various approaches to diagram recognition that are used. It shows that there are very few recognition engines that are able to recognise shapes and text together, most have concentrated on shapes. It also highlights the fact that the majority of diagram recognition methods are based on the measurement of stroke features in conjunction with algorithms to guide recognition. These features commonly measure stroke size, curvature, time and intersections. However, very few have used formal statistical methods to determine the set of features that should be used by their sketch tool.

InkKit is a sketch tool for diagrams that allows recognition of both shapes and text. Like many others, it relies on features of strokes in all phases of its recognition engine including its current divider of shape and text strokes. InkKit is more successful for recognising user interface and UML diagrams, but other diagram types have been tried. Young suggests that the divider component of InkKit's recognition engine needs more work (Young 2005).

This leads us to the conclusion that the best place to start to improve sketch recognition is with the divider and the best way to improve shape and text division is to search for the most distinguishing features that can be used to perform this step as accurately as possible. The following chapter outlines the methodology used to identify these features.

Chapter 3

Methodology

In the previous chapter we found that there are a range of methods used to solve the problem of sketch recognition. One commonality these methods share is the specification of various features of a sketch which are measured to aid recognition. This is also evident in the recognition engine of InkKit. However what is lacking is a definitive set of the most significant features that can be used to provide an accurate division of the shape and text strokes in a sketch. Therefore our goal is to find these features in order to improve division of shape and text strokes and in doing so improve the accuracy of sketch recognition as a whole.

This chapter provides an overview of the approach used beginning with the investigation of possible features, how data is collected and analysed and lastly the implementation and evaluation plans for the resulting divider.

3.1. Feature Discovery

The first step of this project is to identify all the possible features that could be useful in distinguishing between text and shape strokes in a sketch. The origin of these features is from related work in sketch recognition; features from newly available hardware e.g. pen pressure on the Tablet PC and additional features we have thought of.

Many features have been documented by past work to be significant to solving various sketch recognition problems as presented in Section 2.1. Section 2.2 shows that several of

these features already appear in InkKit's recognition engine. Therefore it would be in our interest to include these features in this investigation.

Newly available hardware will also allow us to consider features that have not been widely studied before such as pen pressure and tilt. Some preliminary analysis is required for these new features to establish exactly what measurements are most useful and possibly significant to the division problem. This involves collecting a small number of sketches of text and shapes and empirically determining if there are any patterns in pen pressure or tilt that could be useful features for division.

We also consider features that we believe may be distinguishing features of text and shapes that have not been used specifically in sketch recognition work before.

By looking at features from previous work in sketch recognition and investigating new features our aim is to compile a comprehensive list of features as this will give us more chance of finding those that are most significant to the division of text and shape strokes.

3.2. Data Collection

We want to determine which features from the list are the most significant to dividing text and shape strokes. To be able to test this we need to collect measurements of each feature from sketches and compile these measurements into a dataset.

Before the data can be collected we must decide on a set of diagrams to be used for the experiment. It is important that this set display the most common characteristics of diagrams and include basic shapes, text, combinations of shapes and text and complex shapes. This is important because if we apply the feature set to a realistic group of diagrams we are more likely to achieve a more accurate result of the features that are widely applicable.

The next step is to get people to sketch the set of diagrams using InkKit on Tablet PC's. We need to collect diagrams sets from as many people as possible to avoid any individual variation that may occur in their sketching. Once these diagrams are collected the dataset can be compiled by calculating all the features identified for each stroke of a diagram. The measurement of these features should be based on each stroke as we are looking at how we can differentiate the diagram strokes into drawing or text strokes. The resulting dataset will then be ready for analysis.

3.3. Analysis

Once all the sketches are collected and processed into a dataset they are analysed to determine the features of strokes that are significant and should be used to divide text and shape strokes. A formal technique for this analysis will provide a clear and accurate view of the degree of significance the features have to the division problem.

One way of finding the significant features is to employ a statistical partitioning technique (Breiman, Friedman et al. 1984; Venables and Ripley 2002). This involves taking a dataset and finding which features can be used to split the data into the required groups most accurately, i.e. into text or shape strokes, based on each observations measurement of the features. An example of a possible partition using the bounding box width of a stroke is illustrated in Figure 10.

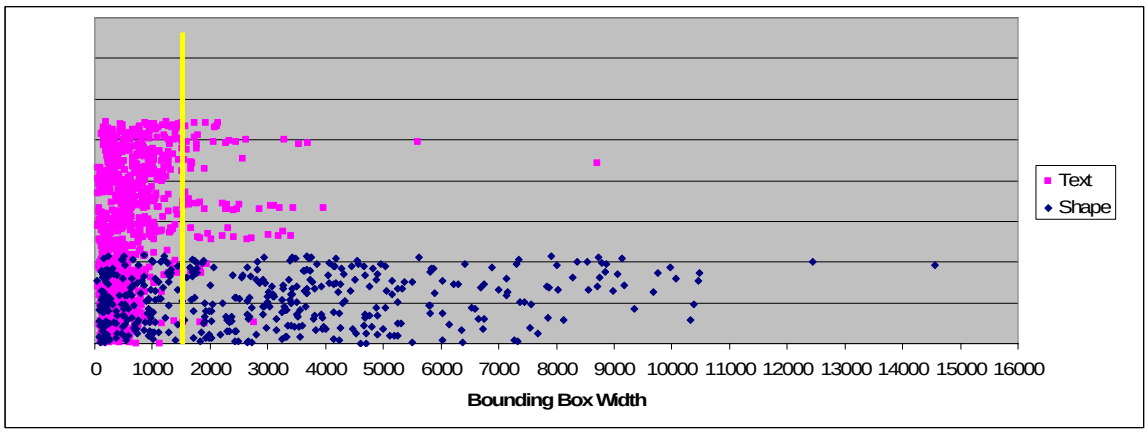


Figure 10. Possible Partition for Bounding Box Width Feature

The bounding box width values for all the observations in an example dataset are shown, where each observation has been manually categorised into the text and shape strokes that they are known to be. The yellow line shows a possible split of the feature where most of the text stroke observations lie to the left of the line with a bounding box width approximately <1500 HIMETRIC units and most shape stroke observations lie to the right hand side of the line with a bounding box width approximately >1500 HIMETRIC units. We can also see that there is no possible partition that will completely split the dataset into the two groups by itself as there is overlap in values where a small number of text observations lie to the right of the partition where most shape strokes are and a small number of shape stroke observations lie to the left of the split where most text strokes are. This overlap represents those strokes that would be misclassified if this feature was used as the sole condition to decide which strokes are text and which are shape strokes.

The aim is to find the most optimal position for a split to be made so that there are a minimal amount of misclassified strokes. If this is done for all features in the feature set, using the observations in the dataset, then the features that most accurately split the data into text and shape stroke groups, with the least amount of misclassified strokes, will be identified as the significant features for division of text and shape strokes.

On advice from a statistician, Dr. Ross Ihaka, a tree-based partitioning technique is used to analyse the dataset, to identify the significant features, and consequently construct a binary classification tree (Breiman, Friedman et al. 1984; Venables and Ripley 2002).

The use of trees is a common and effective way to assist in decision making problems, our decision being whether to classify a stroke in a sketched diagram as either a text or shape stroke. A binary classification tree has decision variables at each node, which correspond to the most significant features found and the optimal partition of that feature, and a classification label at each leaf, which is either TEXT or SHAPE in our case.

A simple example of a binary classification tree is illustrated in Figure 11 where the decision variables are bounding box width and average pressure. Note that if the result of

the decision variable is true then the left hand side branch of the tree is followed; otherwise take the right hand side path. For this example, according to the optimal partition established through statistical analysis, if a strokes bounding box width is >=1500, we would follow the left hand side branch, then if the strokes average pressure value is <50 the stroke would be classified as a shape stroke.

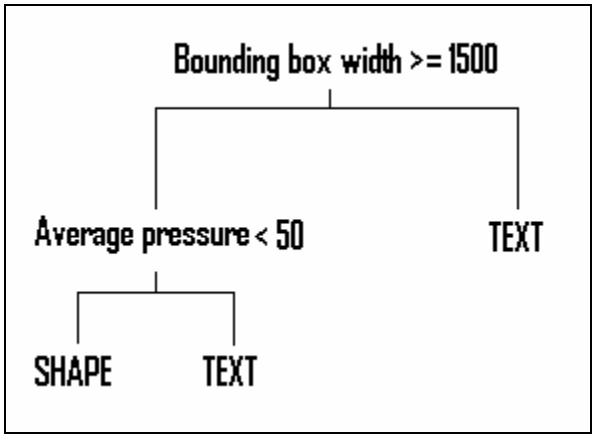


Figure 11. Simple Binary Classification Tree

The advantage of constructing a tree with all the significant features is that the optimal combination of features is identified that can work together to accurately classify a stroke. As illustrated in Figure 10, using one feature alone to split a dataset can result in a large number of misclassified observations, however if we use other features to continue to split each partition of the data we will end up with a more accurate result. Tree-based partitioning techniques most often use a one-step look-ahead approach to constructing a tree. This involves choosing a partition of a feature that is optimal not only to the current node but to the next node as well. Therefore the process of constructing a tree is as follows:

1. Find the optimal split of all observations in the dataset for each feature using a one-step look ahead.
2. Choose the feature that has a split with the least misclassified observations and is an optimal choice for the next split as well. Add this feature to the tree.
3. Repeat from step 1 for each partition of the dataset until the dataset has been classified satisfactorily.

Employing a tree-based approach to partitioning will allow us not only to identify significant features to division, but also provide us with the most optimal combination of features for implementation. It will provide a result that is fast, simple and intuitive to implement and use.

3.4. Implementation

Once the significant features have been identified through formal statistical analysis a divider can be implemented in InkKit. InkKit's existing divider will remain, and the Microsoft divider will also be implemented for evaluation purposes.

All three dividers will run in parallel in the InkKit recognition engine. However the recognition process will be stopped at the end of division for all dividers, where basic shape, text and component recognition phases will not be executed (refer to Figure 9 to see the architecture of InkKit's recognition engine). This is so that each divider can be judged only by its immediate results and can not be altered by subsequent recognition phases.

The basic division process that will be implemented is shown in Figure 12. We begin with a sketched diagram drawn using InkKit on a Tablet PC. Data from the ink strokes is captured such as the stroke point coordinates, time stamps for each stroke, pressure for each stroke point etc. so that the important stroke features for division can be calculated. A decision tree is then used to classify each stroke as either TEXT or a SHAPE based on the features that have been calculated. This completes the division process for a diagram.

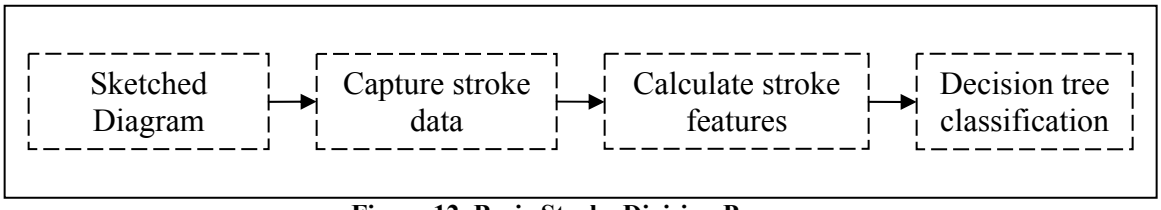


Figure 12. Basic Stroke Division Process

3.5. Evaluation

As was mentioned in Section 3.4, the new divider based on a classification tree will be implemented in InkKit. Alongside this will be InkKit's existing divider and the Microsoft divider. The new divider will be evaluated against the two other dividers to determine which performs better in terms of its ability to accurately divide shape and text strokes.

The training set of diagrams, mentioned in Section 3.2, that are collected for analysis will be used for evaluation. Each diagram will be processed by the new, old and Microsoft dividers in InkKit and divided into probable text and shape strokes.

When all the diagrams have been processed by each divider and the necessary information gathered we can evaluate their accuracy. Not only can we compare the accuracy of each divider but we can also view results according to diagram type and by each participant (the authors of each diagram set). Looking at the accuracy of division for each diagram type may enable us to infer some features of diagrams that have not been acknowledged. Accuracy of division may also be evaluated for each person to learn if there is any particular division method that shows the most variation in accuracy between people. All of this information will allow us to present a full comparison and evaluation of the accuracy of our divider.

In following the steps presented here beginning with discovering possible divider features, collecting data for these features, analysing the data using formal statistical techniques and finally implementing and evaluating a resulting divider, we hope to improve the division of text and shape strokes therefore enhancing our ability to provide accurate and reliable recognition of sketched diagrams.

Chapter 4

Feature Set

Chapter 3 outlined the methodology for identifying significant features of strokes to use in a text and shape divider. This consists of composing a feature set of possible stroke features, then collecting data on each feature from sketches, analysing that data using a tree-based partitioning technique and finally implementing and evaluating the new divider.

This chapter concerns the first phase of our project, to identify a feature set. All the possible features that may be helpful in distinguishing text from shape strokes are compiled together to form the feature set. They are from related work in sketch recognition, the discovery of features available from new hardware as described in Section 3.1, and also result from our own thoughts as to what may be distinguishing features of text and shape strokes. Some features identified in the related work in Section 2.1 are not included in this feature set as they are either very similar to features that are used or measure the same characteristic of a stroke or they are not considered relevant to classifying text and shape strokes. Here we describe the feature set composed during the feature discovery phase of our research.

46 features are selected in all. They are grouped into six categories; pressure, time, intersections, size, curvature and Tablet OS recognition values. Each category of features is described below with the rationale for the choice of those features.

An attempt was made to measure pen tilt, however this feature, while found to be supported by the Tablet PC and the Ink SDK, is not supported by the current Wacom driver (Chesnut 2002).

4.1. Pressure

Recent advances in hardware for Tablet PC's have allowed us to capture the pen pressure applied to the tablet screen. This feature could be of interest if we can detect some difference in pressure applied when drawing shapes and when writing text, for example we may apply more pressure when writing as this could require a more precise use of the pen.

Some preliminary analysis is carried out to observe the nature of pen pressure data and empirically detect any possible patterns that may be helpful in distinguishing text from shape strokes. This involves collecting a small number of sketches of diagrams with text and shape strokes and graphing the pressure at each point of these strokes. Pressure values can be accessed for each point of a stroke through the Stroke object in the Tablet PC's Ink SDK. It is unclear what the units of measurement for pressure are from the Ink SDK.

An example showing the pressure values for two text strokes and a shape stroke is shown in Figure 13. We can observe that pressure seems to increase with the duration of a stroke regardless of it being a shape or a word stroke. However shape strokes seem to be more stable in nature where the pressure increases at the beginning of the stroke then stabilises and eventually decreases at the end when the pen is lifted off the tablet. Of most interest is that it is apparent that there are more oscillations in pressure for text strokes than in drawing strokes. These observations contribute to the inclusion of the four features described in Table 13.

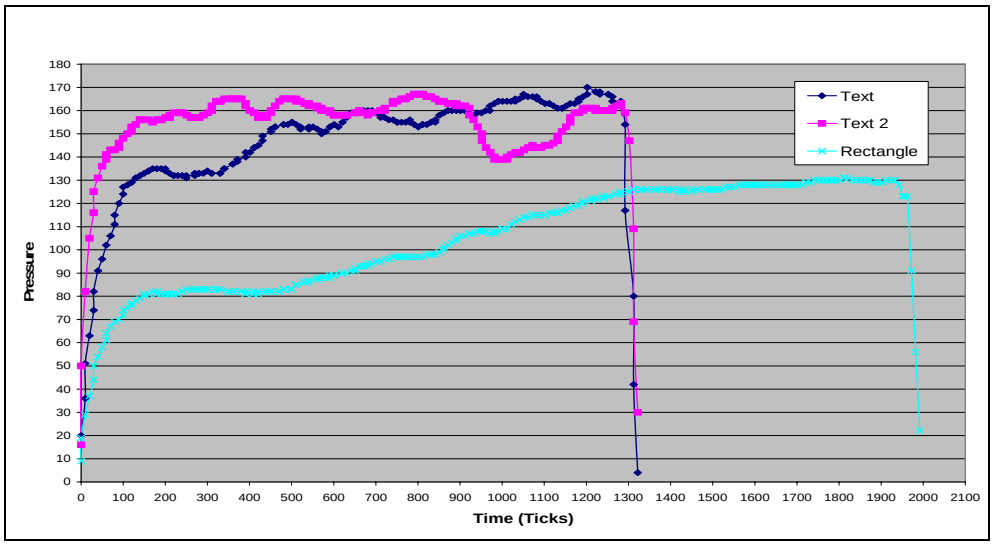


Figure 13. Pen Pressure over Time for Text and Shape Strokes

	Feature	Description	Origin
1	Maximum pressure	Maximum pressure value for the stroke.	Adapted from (Nakai, Sudo et al. 2002)
2	Minimum pressure	Minimum pressure value for the stroke.	
3	Average pressure	Mean average pressure of the stroke.	
4	# Pressure minima	Number of minima in the pressure values for the stroke, this excludes the minima that occur at the beginning and end of the stroke for pen up/down events.	New

Table 13. Pressure Features

The minimum, maximum and average pressure features are included to determine whether there are significant differences in these values for text and shape strokes.

The number of minima in pressure values over time for each stroke is of interest as it is observed during the preliminary analysis of data that in some cases there are more minima for text strokes than for shapes strokes as illustrated in Figure 13. This data is not smoothed as we want a raw count of the number of minima with no expectations as to how extreme the decrease in pressure is as long as the pressure increases thereafter.

4.2. Time

Features measuring the timing of strokes are identified by Rubine (1991) for gesture recognition and Sezgin and Stahovich et al (2001) for vertex detection when recognising shapes, see Section 2.1 for a summary of their work. In regards to division of text and shape strokes, stroke timing may be able to provide important information to aid this process mainly due to the smaller cognitive shift in thinking that we expect when writing which may lead to faster times and speeds for text strokes than shape strokes. Table 14 describes the features of time that are identified followed by the rationale behind each choice of feature.

	Feature	Description	Origin
5	Total duration	Total duration of the stroke from pen up to pen down.	(Rubine 1991)
6	Maximum speed	Maximum speed when drawing the stroke.	Adapted from (Rubine 1991)
7	Minimum speed	Minimum speed when drawing the stroke.	
8	Average Speed	Mean average speed when drawing the stroke.	
9	Time from last stroke	The time between the current stroke and the previous stroke in the sketch. Not applicable to the first stroke of a diagram.	New
10	Time till next stroke	The time between the current stroke and the next stroke in the sketch. Not applicable to the last stroke of a diagram.	
11	Speed from last stroke	Speed (distance/time) between the current stroke and the previous stroke in the sketch. Not applicable to the first stroke of a diagram.	
12	Speed to next stroke	Speed (distance/time) between the current stroke and the next stroke in the sketch. Not applicable to the last stroke of a diagram.	
13	# Speed minima	The number of extreme minima in the speed values for the stroke, this excludes the minima that occur at the beginning and end of the stroke for pen up/down events.	Adapted from (Sezgin, Stahovich et al. 2001)

Table 14. Time Features

Timing data for each point of each stroke can be accessed through the Tablet PC's Ink SDK using the "TimerTick" property of the Stroke object; this provides the local timing of each point within the stroke. In addition each stroke can be manually time stamped using a Stroke objects extended property to store the time of each stroke in relation to other strokes. Time is measured in milliseconds here.

The total duration of a stroke could vary between text and drawings however it is considered an important measurement as there may be some significant difference apparent that could help in division. The minimum, maximum and average speed of a stroke is measured to determine whether there are significant differences in these values for text and shape strokes.

The time from the last stroke, time till the next stroke, speed from the last stroke and the speed till the next stroke have been included to test if a significant difference exists in the time and speed between text strokes and between shape strokes. It is expected that there is a smaller time and a faster speed between text strokes than between shape strokes due to a smaller cognitive shift required when writing as we have a flow in our thinking where it does not require much pause to decide what we must write next unless we have problems spelling the word.

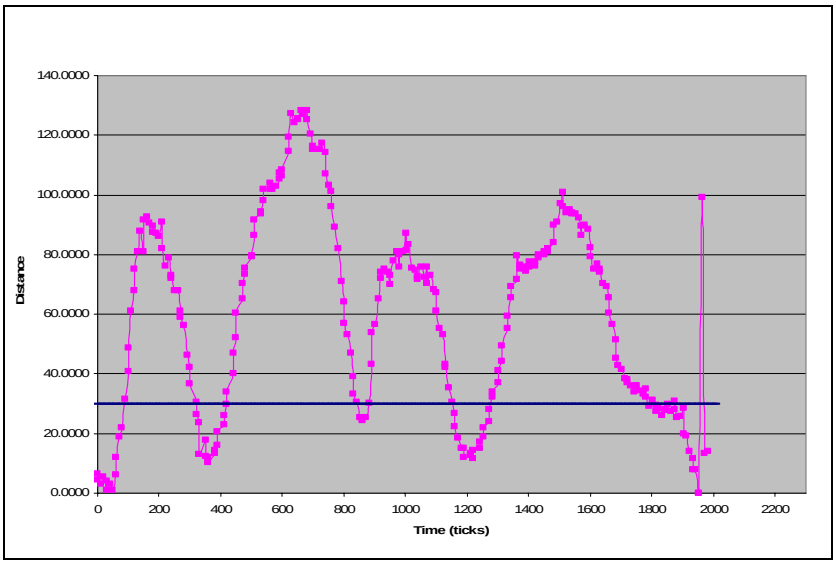


Figure 14. Speed Graph for a Rectangle Stroke Showing Three Minima below the Ceiling Line
(Excluding pen up and pen down minima at the beginning and end of the stroke)

We measure the number of extreme minima in speed values for a stroke as it is known that pen speed is slower when drawing corners (Sezgin, Stahovich et al. 2001), see Section 2.1 for details. Therefore measuring the number of times the pen slows down by a significant amount will allow us to determine the approximate number of sharp corners in the stroke. There may be a difference in this number for shape and text strokes especially if we are dealing with cursive writing for which we would not expect a large number of sharp corners.

Smoothing is applied to these values using a moving average so that only the extreme minima are included. Also the average based filtering method used by Sezgin and Stahovich et al (2001) is used where a ceiling is applied to the minima that will be accepted. The data is noisy so a ceiling will ensure that we only count minima in areas where the speed is already very low therefore focusing on global rather than local minima. The ceiling is determined empirically here where only minima occurring in the bottom fifth of all values for that stroke are counted. Note that this feature can also be categorised as relating to the curvature of strokes. Figure 14 shows a graph for the speed of a rectangle stroke. A ceiling is illustrated by the blue line, below this ceiling are the 3 extreme minima indicating the corners of the rectangle, this excludes the minima at the beginning and end of the stroke which are the pen up and down points.

4.3. Intersections

Intersections of strokes have been used in various work related to sketch recognition in the past. For the division problem we recognise that the nature of intersecting strokes for text and shape strokes may differ. Table 15 describes the features of stroke intersections that have been identified followed by why each feature has been chosen. Figure 15 also illustrates these intersection features in a typical diagram.

All the intersection data can be obtained through the Tablet PC's Ink SDK from the Stroke object.

	Feature	Description	Origin
14	# Self intersections	Number of points where the stroke intersects itself. See points A, B and F in Figure 15.	Adapted from (Qin 2005)
15	# Endpoint self intersections	Number of self intersections at the endpoints of the current stroke. See points A and F in Figure 15.	
16	# Other self intersections	Number of self intersections that are not at the current stroke's endpoint. See point B in Figure 15.	
17	# Other intersections	Number of points of intersection of the current stroke with other strokes (excluding self intersections). See points C, D and E in Figure 15.	Adapted from (Calhoun, Stahovich et al. 2002)
18	Total # intersections	Total number of intersections (includes self intersections). In Figure 15 for the rectangle on the left there are a total of 2 intersections at points A and C.	
19	# Other strokes intersecting	Number of other strokes that intersect the current stroke (excluding itself). In Figure 15 for the rectangle on the left there is only 1 other stroke intersecting it at point C.	Adapted from (Hammond and Davis 2002); (Fonseca, Pimentel et al. 2002)
20	Total # strokes intersecting	Number of strokes that intersect the current stroke (including itself). In Figure 15 for the rectangle on the right there are a total of 2 strokes intersecting it at points D and F.	

Table 15. Intersection Features

It is expected that there are more self intersections, other intersections and therefore more intersections in total for text strokes than for drawing strokes particularly if it is cursive writing.

Self intersections at the end points of a stroke is expected to be more typical of drawing strokes where sides of shapes are joined together at ends to close a shape. Therefore the number of endpoint self intersections could be larger for shape strokes and the number of other self intersections (not at the endpoints) may be larger for text strokes.

The number of strokes that intersect the current stroke, including and excluding itself will be measured as there could be a significant difference in values for text strokes and drawing strokes.

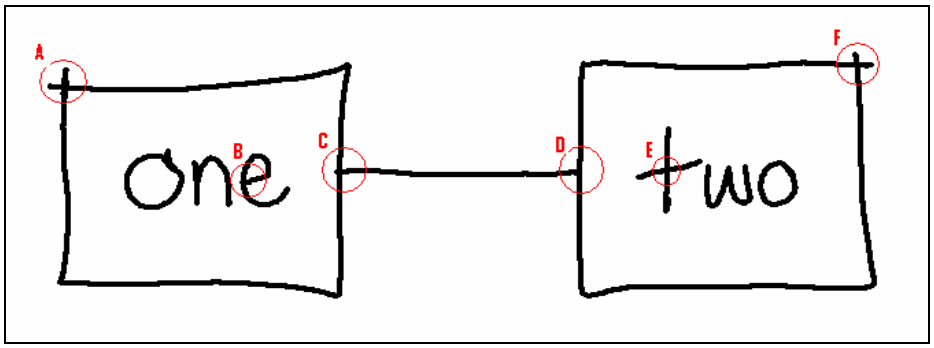


Figure 15. Example of the Types of Intersections that can occur in a Diagram.
Points A, B and F are all self intersections, where A and F are endpoint self intersections. Points C and D and E show intersections with other strokes.

4.4. Size

Previous work in sketch recognition has recognised the significance of features of size as shown in Section 2.1. We believe that size could be significant to the divider problem in particular as text in diagrams is typically smaller than shapes. Features from previous work have been included in our feature set; these are all described in Table 16. Following this are the reasons behind each choice of feature.

The size features can be calculated using the coordinates of each point in a stroke obtained through the Stroke object in the Ink SDK. Features are measured using HIMETRIC units.

The length of a stroke could vary between text and drawings however it is considered an important measurement as there may be some significant difference apparent that could help in division.

The height, width and area of the bounding box of a stroke are considered important to measure as text is typically smaller than shapes in a diagram. These features could provide helpful information to a possible divider.

	Feature	Description	Origin
21	Length	Total length of the stroke.	(Rubine 1991)
22	Bounding box area	Area of the bounding box of the stroke.	Adapted from Tahuti (Hammond and Davis 2002) and CALI (Fonseca, Pimentel et al. 2002)
23	Bounding box height	Height of the bounding box of the stroke.	
24	Bounding box width	Width of the bounding box of the stroke.	
25	Distance from last stroke	Distance the pen travels between the current stroke and the previous stroke. Not applicable to the first stroke in the sketch.	Adapted from InkKit (Young 2005)
26	Distance to next stroke	Distance the pen travels between the current stroke and the next stroke. Not applicable to the last stroke in the sketch.	
27	Amount of ink inside	Amount of ink inside the strokes bounding box. This is calculated by counting the number of points of the stroke that are inside the bounding box.	InkKit (Young 2005)
28	Bounding box diagonal length	Length of the bounding box diagonal line, refer to Figure 2 and Table 2 for details.	(Rubine 1991)
29	Distance from first to last point	Distance from the first point of the stroke to the last point of the stroke; refer to Figure 2 and Table 2 for details.	
30	Total length/bounding box diagonal length	Length of the stroke divided by the length of the bounding box diagonal.	Adapted from (Rubine 1991);
31	Perimeter to area	Ratio of perimeter to area of the strokes convex hull, refer to Table 9 and Figure 7 for details.	CALI (Fonseca, Pimentel et al. 2002)
32	Convex hull area ratio	Ratio of the area of the convex hull to the area of the enclosing rectangle of the stroke, refer to Table 9 and Figure 7 for details.	
33	Rectangle ratio	Ratio of strokes enclosing rectangle width to height, refer to Table 9 and Figure 7 for details.	
34	Width to height	Ratio of the strokes bounding box width to height.	Adapted from CALI (Fonseca, Pimentel et al. 2002)

Table 16. Size Features

The distance to the next stroke and distance from the last stroke are included to give some indication of the position of this stroke in terms of the strokes drawn before and after it. The distances are expected to be very small when between two text strokes in the same word and larger when between shapes.

The measure of the amount of ink inside the bounding box is included as it is expected that there is more ink inside the bounding box of a text stroke than a shape stroke (Young 2005).

Features 28 to 34 are included in the feature set primarily because they are features used by InkKit's current recognition engine. See Chapter 2 for further details of these features.

4.5. Curvature

Features indicating the curvature of a stroke may help in distinguishing text from shape strokes. The features measuring stroke curvature are described in Table 17.

Rubine's work (1991) in particular recognised the significance of using the information from various angles of a stroke to improve sketch recognition, see Chapter 2 for more details. These features have also been successfully used by InkKit's recognition engine and therefore have been included in this feature set as features 35 to 42.

The number of bezier and polyline cusps in a stroke will be collected for the possibility of a pattern emerging that could tell us more about text and shape strokes. It is hypothesised that text strokes could have on average a greater number of cusps than shape strokes due to the curved nature of writing. They are calculated automatically as data for the Ink SDK's Stroke object.

	Feature	Description		Origin
35	Cosine of initial angle	Cosine of the initial angle of the stroke, refer to Figure 2 and Table 2 for details.		(Rubine 1991)
36	Sine of initial angle	Sine of the initial angle of the stroke, refer to Figure 2 and Table 2 for details.		
37	Angle of bounding box diagonal	Angle of the bounding box diagonal; refer to Figure 2 and Table 2 for details.		
38	Cosine from 1st to last point	Cosine of the angle between the first and last point of the stroke, refer to Figure 2 and Table 2 for details.		
39	Sine from 1st to last point	Sine of the angle between the first and last point of the stroke, refer to Figure 2 and Table 2 for details.		
40	Total angle	Total angle traversed by the stroke; refer to Figure 2 and Table 2 for details.		
41	∑ |angle at each point|	Sum of the absolute value of the angle at each point of the stroke refer to Table 2 for details.		
42	∑(angle at each point)^2	Sum of the squared value of the angle at each point of the stroke; refer to Table 2 for details.		
43	# Bezier cusps	Number of bezier cusps. Cusps indicate points where the direction of the stroke has changed; therefore Bezier cusps are the cusps for a Bezier curves control points for the stroke (Anonymous 2007)	For example take the letter "L". This letter has three cusps (Bezier and polyline) one at the start and end of the letter and one at the corner	New
44	# Polyline cusps	Number of polyline cusps. Cusps indicate points where the direction of the stroke has changed; therefore polyline cusps are the cusps of the points of a stroke. (Anonymous 2007)		

Table 17. Curvature Features

4.6. Tablet OS Recognition Values

The Microsoft Tablet Operating System has its own text recognition engine that can provide the features listed in Table 18. These features are included in our feature set on

the hope that the predictions they provide are accurate enough to be able to distinguish text strokes from shape strokes in our divider.

	Feature	Description	Origin
45	Tablet OS text probability	Tablet OS text recogniser probability of the stroke being text, levels of probability are strong, intermediate or poor.	Microsoft Tablet OS
46	Tablet OS prediction	Tablet OS text recogniser best prediction of the text symbol that the stroke represents.	

Table 18. Tablet OS Recognition Features

4.7. Others

The following features, listed in Table 19, are not automatically inferred from the stroke, and therefore not typical of those described in previous categories. However they are necessary for analysis and are manually added to the dataset.

We need to specify a person ID to allow us to compare individual's sketches and determine if there is any significant individual variation upon evaluation. The stroke ID must be noted as a reference point for the previous and next ID features. The ID of the stroke that came before and the stroke that came after is recorded to enable us to find out if the classification of the strokes next door after division, as either a text or drawing stroke, has any bearing on the classification of the current stroke.

The question is can we use a second iteration of the division process to look at each strokes classification and infer something about the type of the strokes coming before or after it based on its previously predicted classification. This could be an advantage in classifying letters in the middle of words where the stroke before and after it are classified as text and therefore this could increase the probability of the current stroke being text. As mentioned in Section 2.2, InkKit's second phase of its divider currently uses the initial classification of strokes close by to further confirm the probability of the

current stroke based on the premise that strokes that are close together are more likely to be of the same type (Young 2005).

The known type of stroke as either a TEXT or SHAPE stroke is a basic requirement for analysis to confirm if our dividers prediction is correct and is therefore added to the feature set. The stroke name on the other hand is not vital but may provide some clues in later evaluation for patterns of what kind of strokes in diagrams are more accurately divided than others.

	Feature	Description
47	Person ID	The ID number of the person who draws the diagram.
48	Stroke ID	The ID number of the stroke.
49	Type	The type of stroke it is known to be by observation, either text or shape.
50	Stroke name	The stroke name indicating the part of the diagram this stroke represents, for example “rectangle”, “circle”, “text”.
51	Previous ID	The Stroke ID of the stroke that came before the current stroke. Not applicable to the first stroke in a diagram.
52	Next ID	The Stroke ID of the stroke that came after the current stroke. Not applicable to the last stroke in a diagram.

Table 19. Other Features

4.8. Summary

The features compiled here have come from literature in the area of sketch recognition, from new hardware capabilities and our own thoughts as to features that may be significant to dividing text and shape strokes. They can be described by the categories of size, curvature, pressure, intersections, time, Tablet OS recognition values and others. These features conclude our feature set and provide a basis for the following analysis, described in the next chapter, to determine which of these features are significant to the division of text and shape strokes.

Chapter 5

Analysis

The feature set specified in Chapter 4 serves as the focus of our analysis to find the features that are significant to classifying shapes and text. The 46 features are from previous work in sketch recognition, from our own thoughts of possible significant features and also include features now available from new hardware. They are primarily measures of size, curvature, pressure, intersections, time and the Tablet OS recognition engine values.

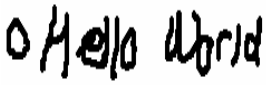
The feature set is analysed to discover which of those features provide the greatest contribution to dividing text and shape strokes in a diagram. In following with the methodology set out in Chapter 3, this Chapter describes the experiment beginning with the collection and processing of data from a range of sketched diagrams then analysing this data using a tree-based partitioning technique that consequently constructs a classification tree containing the most significant features for division of text and shape strokes. This process allows us to draw conclusions as to which features are most significant to division.

5.1. Data Collection and Processing

The first step to performing a full analysis of the feature set described in Chapter 4 is to collect data in the form of sketched diagrams from as many people as possible. The following section will describe the types of diagrams that are collected, how sketches are collected and from whom and the processing of the sketches into a dataset ready for formal statistical analysis.

5.1.1. Diagram Set

In compiling a set of diagram types we look for examples of shapes and text that represent those typical of most diagrams and therefore allow the most significant features of strokes for division to be identified. This set includes basic shapes and text, complex shapes, composite shapes and various combinations and ordering of shapes and text. The chosen diagram set consists of nine diagrams which are presented in Table 20 along with the reasons why each was chosen.

Shape Description	Example Sketch	Rationale for Choice
Circle		This is an example of a basic shape that appears in most diagrams and therefore it is important that the features chosen account for this type of shape.
Button: rectangle with the label "Button" inside it.		The rectangle and label here is an important example of basic shapes and text that appear in many diagrams and therefore it is important that the features identified in analysis are based on such an example. The diagram as a whole represents an important combination of shapes and text with the containment of words within shapes that is so often seen in many types of diagrams.
Text: "Hello how are you", without punctuation.		Basic text diagrams are vital to the diagram set to provide some comparison of the features important to text and those important to shape division.
Radio-text: radio button with the label "Hello world" next to it.		The circle and label here is an important example of basic shapes and text that appear in many diagrams and therefore it is important that the features identified in analysis are based on such an example. The diagram as a whole is an example of a familiar combination of shapes and text in diagrams. Two variations of the radio button were drawn, the radio-text and text-radio diagrams, to demonstrate variations in order that each component was drawn in.
Text-radio: same as radio-text above but the label is written before radio button (however the spatial ordering is the same).		

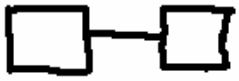
Combo box: rectangle with a triangle inside.		The rectangle and triangle here give examples of basic shapes that commonly appear in diagrams and therefore it is important that the features chosen account for these types of shapes. The diagram also represents a composite shape with a smaller shape within a larger shape.
Resistor: spiked line, from an electrical diagram.		This complex shape was added for its uniqueness in its corners and similarity to text because of its shape. It is an example that could easily be recognised as text however one obvious difference is that it usually has no intersections whereas text often does. It will provide a challenge to feature selection to ensure that it can successfully be classified as a shape.
Hexagon: six-sided polygon.		This complex shape was added to represent polygons with more than 4 sides that may appear in some diagrams.
Connector: two rectangles with a line connecting them in the middle.		The rectangles and line here give examples of basic shapes that commonly appear in diagrams and therefore it is important that the features chosen account for these types of shapes. The diagram also represents a composite shape with deliberate intersections.

Table 20. The Diagram Set

5.1.2. Participants

Sketches are gathered from 26 people, each person completing the set of 9 diagrams shown in Table 20. The diagrams are sketched on a Tablet PC using the InkKit application. The participants are between the ages of 13 and 60 years old, a large number of them coming from a computer science background. Most do not have previous experience sketching on a Tablet and are given a few minutes to familiarise themselves with using the pen to sketch. Each participant is given a piece of paper with sample drawings of each diagram in the required set and are asked to sketch the complete set in one sitting. When writing any text they are asked to omit any punctuation, also when

drawing the text-radio diagram they are instructed to draw the label first and then the radio button. If they have any further questions they are answered.

5.1.3. Data Processing

The sketches are processed within InkKit. This involves programmatically calculating and recording all the stroke features that are identified in Chapter 4 for each sketch, the C# code used to calculate these features is given in Appendix A. Most of the feature set can be accessed directly from the C# stroke data structure however special consideration is made for time and pressure data. In order to collect pressure values for each point in the stroke this option is explicitly enabled as a packet property. For time data, each stroke is manually time stamped and added to the stroke as an extended property. Also for some of the features requiring data from the previous or next stroke, if this data is not available, as in the case of single stroke diagrams, and the first or last stroke of a diagram the feature is simply marked as not applicable for that particular stroke.

The feature data for each sketch is recorded in the form of an Excel spreadsheet, as illustrated in Figure 17. The spreadsheet data for each sketch is eventually collated into one final dataset ready for statistical analysis with 1519 strokes in total. 432 of these strokes are known to be shape strokes and the remaining 1087 strokes are text. Each stroke is manually categorised as SHAPE or TEXT as base data for the statistical analysis. Additional information, described in Table 19, is also manually added to the dataset for each observation.

Therefore the following steps are carried out for data collection and processing:

1. Each participant draws the diagram set on a Tablet PC using InkKit like that shown in Figure 16.
2. Each sketch is processed by InkKit where features are automatically measured and written to a spreadsheet.
3. Features from the category “Others” (described in Section 4.7) are manually entered into each spreadsheet.

4. All spreadsheets for each sketch drawn by each participant are collated into a final dataset to be used for the analysis; a snapshot of this dataset is shown in Figure 17.

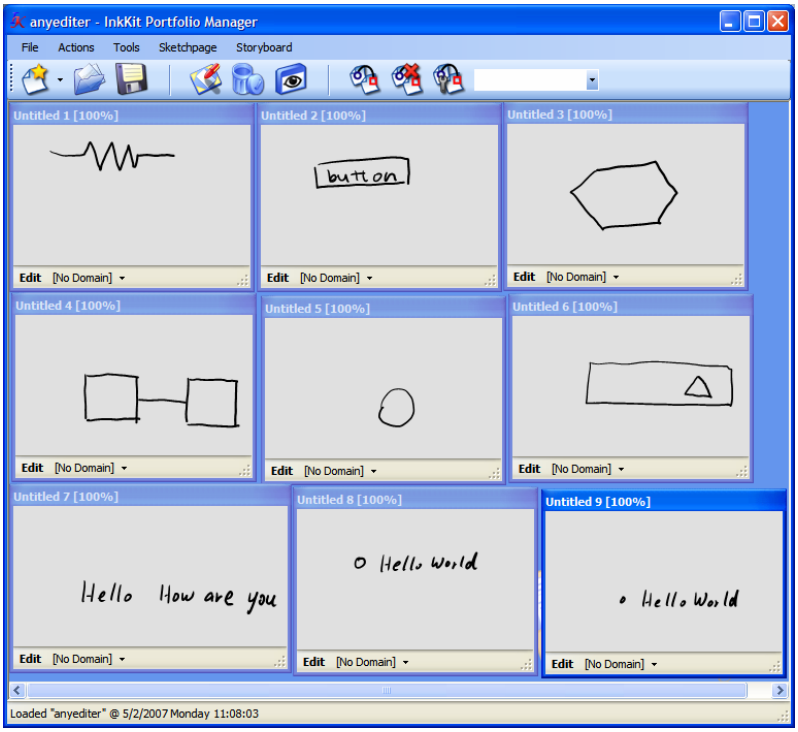


Figure 16. Screen shot of a typical participant's sketches of the diagram set

Microsoft Excel - DataSet-update11

File Edit View Insert Format Tools Data Window Help

J5 370.32418230518

	A	B	C	D	E	F	G	H	I
1	Type	Stroke	Max Pressure	Min Pressure	Avg Pressure	Min Speed	Max Speed	Avg Speed	Speed Minima
2	shape	button-rect1	132	8	118.5460993	0	57035.0676	19670.1583	3
3	shape	button-rect2	146	0	126.44	0	91443.9569	25034.0691	0
4	text	button-text1	128	2	89.59537572	0	36674.2356	17400.8752	4
5	text	button-text2	108	6	62.14606742	0	46173.578	9180.95899	2
6	text	button-text3	83	12	59.52	1285.6487	39319.2001	20511.665	1
7	text	button-text4	109	4	79.45945946	3000.0023	31144.8179	18535.4841	0
8	text	button-text5	117	5	80.27906977	2999.99951	55172.4479	25637.5752	1
9	text	button-text6	139	14	115.9605263	999.999836	38626.4095	18650.9095	1
10	shape	resistor	154	3	123.8242123	0	81584.2988	18902.2368	6
11	shape	hexagon	155	11	133.3863299	0	84148.6642	21430.3731	4

Figure 17. Feature Data Captured for Example Strokes

Six features are not included in the analysis. The features Person ID, Stroke ID and Stroke Name are not required as they hold information that is required for later evaluation and are not intended for use in distinguishing strokes. The features Previous ID and Next ID are not used here as it is recorded for use in a possible second iteration of analysis that

could be done in the future to find out if the classification of the next or previous stroke has a significant bearing on the classification of a stroke. The Tablet OS Prediction feature unlike most features is a discrete measurement of the predicted symbol that the stroke could be, therefore it has too many possible values and is not helpful to the analysis.

5.2. Statistical Analysis

A tree-based partitioning technique is used to identify the significant features for dividing text and shape strokes and as a result constructs a binary classification tree as described in Section 3.3. This analysis is performed using the rpart function (Venables and Ripley 2002) within the R statistical package (R Development Core Team, 2006), pre-release Version 2.5.0. The dataset is used as training data for the rpart function which applies the tree-based partitioning technique to identify the significant features to use as decision variables in a binary classification tree and determines how to split those features so that they can accurately classify strokes as either text or shape strokes.

In order to do this, first rpart takes a random sample of observations from the dataset to create a smaller independent dataset. The larger dataset acts as the training set for the process, and the independent dataset is used later for evaluation of the resulting classification tree constructed using the training set.

For each feature rpart is provided with all the example strokes feature data from the observations in the training set e.g. length, average speed, number of intersections and average pressure value etc, and the known classification of each stroke i.e. SHAPE or TEXT. It goes through each of the possible features that could be used as decision variables in the tree and finds the best partition of the observations in the dataset. The best partition is chosen by minimising a measure of purity called the Gini index which is a measure of the misclassification rate for that partition (Venables and Ripley 2002). The splitting process is continued until the number of observations at each leaf is < 20 or they are all known to be either SHAPE or TEXT strokes.

The tree obtained by the recursive splitting process suffers from a phenomenon known as "over-fitting". It is constructed to describe the training data, rather than the underlying process which generated the data. A better description of the underlying process and hence better classification of data generated by that process is obtained by pruning the tree. The tree is pruned by minimising the cost-complexity measure:

$$C(T, \alpha) = R(T) + \alpha |T|$$

over the sub-trees of the maximal generated tree. Here R(T) is the misclassification rate for a given (sub)tree and |T| is the size of the tree. The misclassification rate R(T) is estimated by 10-fold cross-validation as explained in (Breiman, Friedman et al. 1984).

Because cross-validation is used, the values of C(T, alpha) are random estimates of the true underlying cost-complexity measure. This randomness makes it impossible to identify the value of alpha which minimises the underlying C(T, alpha).

Although the minimal cost-complexity tree cannot be identified, a good classification tree can be obtained by using the "one standard error rule", see (Breiman, Friedman et al. 1984). The tree with minimal estimated cost-complexity is identified and progressively smaller trees are considered until a tree is identified whose estimated cost-complexity is greater than the minimum observed cost complexity plus one standard error. This tree is typically smaller than the true best classification tree, but it still provides a good classifier for new observations.

Figure 18 illustrates the value of the estimated cost-complexity measures for each tree size constructed from the analysis of our data. The value of alpha is usually small so this is essentially the estimated misclassification rate. There is a statistical measure of variability associated with each point (the standard error). The vertical line through the point gives +/- one SE limits.

The one SE rule says: find the minimum cost complexity measure; in this case it is for the tree of size 12. Take this value plus one standard error (the top of the vertical line through

the point) and draw a line horizontally across at that value. Now work down through smaller tree sizes until you reach a cost-complexity which lies above the line. In this case it is for a tree size of 8 as illustrated by Figure 18.

Underneath the point is the value of alpha (the complexity parameter, hence cp) which makes this the optimal tree. This is used to prune the tree to a size of 8. Therefore for our analysis the best tree is one pruned to 8 nodes.

See Appendix B for the R code that is used for the analysis.

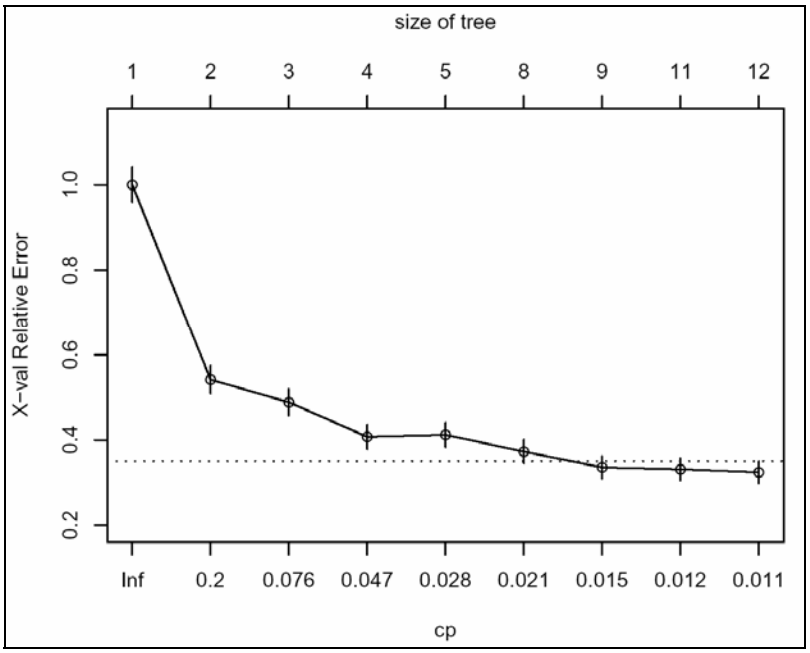


Figure 18. Cost Complexity Parameter Values for each Possible Tree Size

5.3. Results

The results of the statistical feature analysis are presented in this section. This includes the classification tree produced from the analysis and the classification accuracy statistics of each feature in the tree when used to classify the training dataset of strokes.

Figure 19 shows the classification tree constructed from the dataset using a tree-based partitioning approach. Eight different features of strokes, named in each node of the

classification tree, are identified from the feature set as being significant to dividing shape from text strokes. Each feature is examined below in greater detail to determine if any predicted patterns in the features, put forward in Chapter 4, are confirmed.

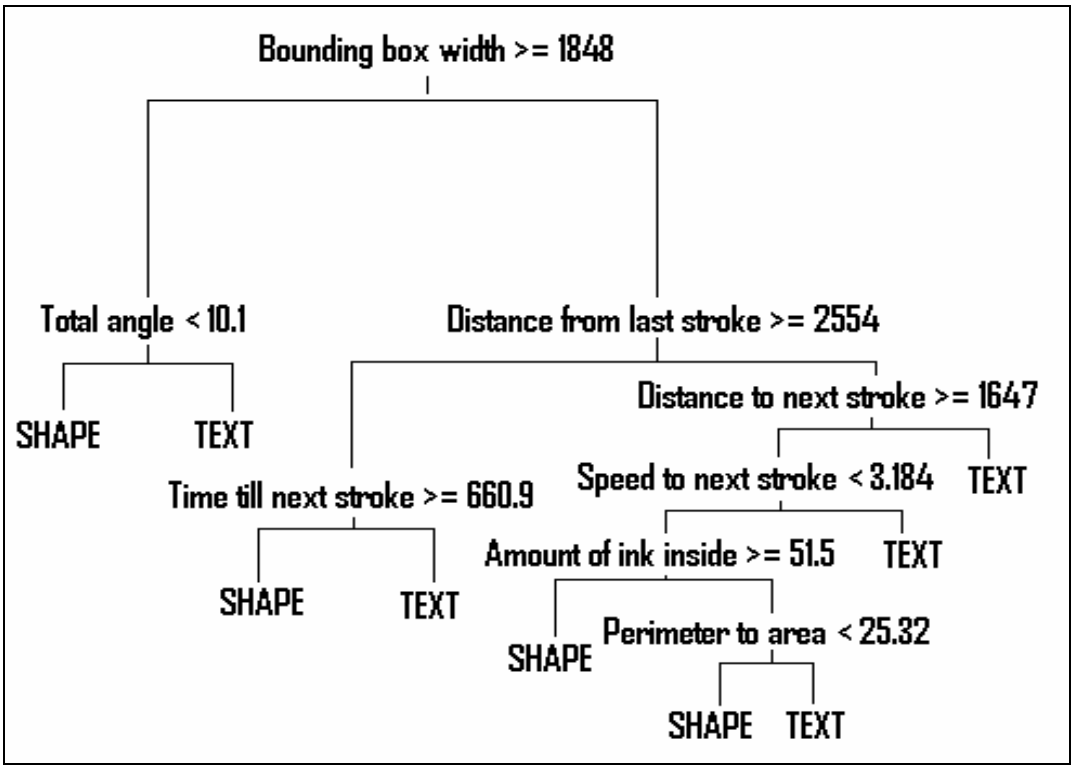


Figure 19. Classification Tree Version I

The relative importance of each feature can be inferred from their position in the tree. The bounding box width feature is clearly the most significant as it is the first node in the tree which all strokes must pass through. At the lower levels of the tree are the features amount of ink inside the bounding box and the perimeter to area ratio. These features are not as significant to dividing strokes as comparatively only a small proportion of strokes are classified using these features. However, only these eight features from the 46 in the feature set are chosen indicating that they are all significant.

As we can see in Table 21, the majority of these features are measuring some element of size, as 5 out of 8 of the features come from this category. Features of time have also registered their importance as making up 2 of 8 of the significant features. Also one feature of curvature is included as important to the division of shape and text strokes. The

features have come from InkKit (Young 2005), Tahuti (Hammond and Davis 2002), CALI (Fonseca, Pimentel et al. 2002) and Rubine's algorithm (1991) and two are new features.

Name	Category	Origin
Time till next stroke	Time	New
Speed till next stroke		
Bounding box width	Size	Adapted from Tahuti (Hammond and Davis 2002) and CALI (Fonseca, Pimentel et al. 2002)
Perimeter to area		CALI (Fonseca, Pimentel et al. 2002)
Distance from last stroke		Adapted from InkKit (Young 2005)
Distance to next stroke		
Amount of ink inside		
Total angle	Curvature	(Rubine 1991)

Table 21. Significant Features and their Categories and Origin

Accuracy statistics for the percentage of strokes correctly classified as either shape or text strokes for each feature in the classification tree are presented in Table 22 along with the bounding conditions of each feature for grouping shape and text strokes. They are generated by the partitioning technique to ensure the most optimal tree is constructed. They are based on the number of strokes that enter the particular node of the tree in question.

These statistics help confirm some of the hypotheses formed in Chapter 4 for the significant features. For example for the bounding box feature 85% of strokes with a bounding box width >=1848 are correctly classified as shape strokes and 86% with a measurement less than this bound are correctly classified as text strokes. This shows strong evidence that the width of the bounding box for a shape stroke is greater than that of a text stroke according to the direction of the bounds shown for text and shapes.

Similar conclusions are shown in Table 22 for all other features which support the hypothesis that are given in Chapter 4. The only feature that does not behave as expected is the amount of ink inside the strokes bounding box, as it is thought that text strokes

have a greater amount of ink inside its bounding box. However considering that we expect that the typical bounding box and length of a shape stroke to be greater than that of a text stroke, this could be why the result shows more ink found inside shape strokes. This could be more accurate if it is calculated as a ratio to the length of the stroke or the size of the bounding box.

Using a tree-based partitioning technique, from a feature set of 46 stroke features, we have identified the above 8 significant features to distinguish between text and shape strokes in a sketched diagram. These features can now be implemented as a divider within InkKit and evaluated to determine if this new divider is in fact an improvement from the existing InkKit divider and the Microsoft divider.

Features	Shape		Text		Conclusion
	Bounds	% Correct	Bounds	% Correct	
Bounding box width	>= 1848	85%	< 1848	86%	Width of text < shapes
Distance from the last stroke	>= 2554	71%	< 2554	90%	Distance between text < shapes
Distance to the next stroke	>= 1647	39%	<1647	94%	
Time till the next stroke	>= 660.9	89%	<660.9	82%	Time between text < shapes
Speed to the next stroke	< 3.184	63%	>3.184	93%	Speed between text > shapes
Total angle	< 10.1	94%	> 10.1	97%	Total angle of text > shapes
Perimeter to Area	< 25.32	79%	>25.32	88%	Perimeter to area ratio for text > shapes
Amount of ink inside	>= 51.5	88%	<51.5	64%	Amount of ink inside the bounding box of text < shape

Table 22. Accuracy Statistics for each Feature

Chapter 6

Implementation of Dividers

The feature analysis in the previous chapter has identified the features that are significant to dividing strokes using a method of partitioning data which consequently constructs a classification tree. Eight features are chosen and combined together as part of the tree illustrated in Figure 19. The implementation of the resulting tree representing our divider is described here.

The new divider is implemented within InkKit; Figure 20 shows the architecture of InkKit's recognition engine. The process of stroke recognition begins with the user sketching a diagram using InkKit on a Tablet PC. The strokes of the diagram are then divided into probable text and shape strokes by the divider. This is where the new divider is added. Following this division the text strokes are recognised by the Tablet OS text recogniser and the shapes strokes, after being joined together, are passed to the basic shape recognition component of the recognition engine. The recognised shapes and text are then combined back together during the component recognition phase to recognise the domain specific components of a diagram. Thus the diagram is recognised.

This chapter outlines the chosen technology that is used to implement the divider. It describes the implementation of the first decision tree, the Version I divider, the resulting formulation of a second decision tree, the Version II divider and a comparison of these trees. It also introduces a pre-processing step of stroke grouping that is used. Finally the experiences gained from implementing such an application are shared.

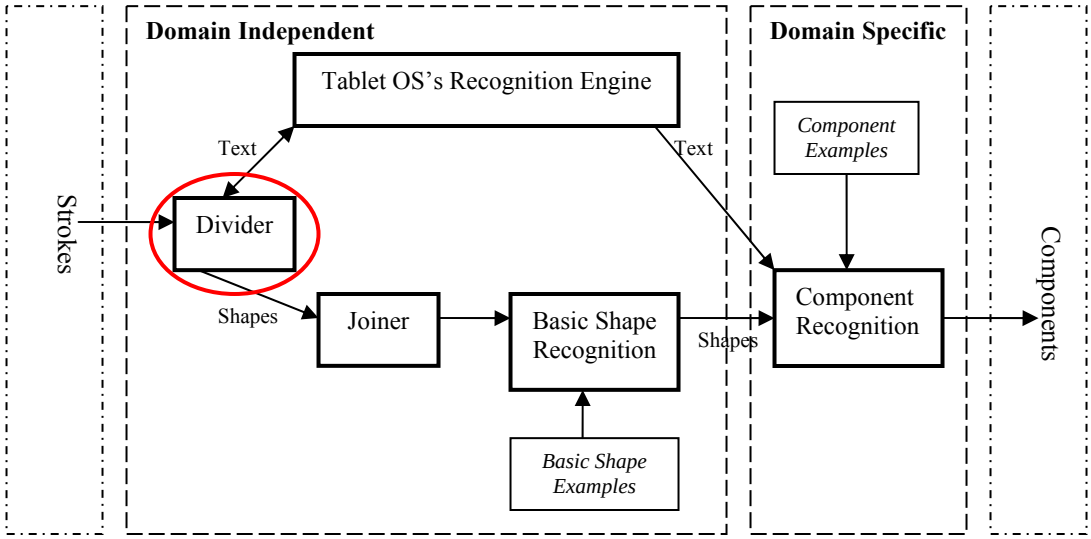


Figure 20. Architecture of InkKit's Recognition Engine
The new dividers are implemented in the circled region.

6.1. Technology Choices

The divider is implemented within InkKit rather than a stand alone application for several reasons. InkKit has an architecture which is clearly separated into components making it convenient to add in new functionality and also if the new divider is found to be more successful than InkKit's existing divider then it will be incorporated into the recognition engine, therefore designing it to fit InkKit is of benefit for later development stages. In addition, we want to ensure that any variables that may exist in an application, such as user interface factors like the size of the drawing pane or the control of the ink is consistent for the new divider and InkKit's existing divider when evaluating their performance against each other.

As mentioned in Section 2.2, InkKit uses C# in Visual Studio .Net and is developed for the Microsoft Tablet OS. The Tablet PC is used as this provides an excellent hardware platform for such development. The Tablet PC uses the Microsoft Tablet OS which provides the Ink SDK for ink support. Access to the Ink SDK is vital to our application. The Ink SDK provides all the basic ink data support such as classes for Ink and Stroke objects which hold all the information about each ink stroke in the diagrams drawn. Data such as the coordinates of each point in the stroke and time values are all required for

feature calculation and are easily obtained from the Stroke objects. It also allows access to the Microsoft text recognition engine which includes the Microsoft divider, used as a comparison for our divider evaluation. C# is the easiest .NET language to use in conjunction with the Ink SDK and therefore is the obvious choice. The only drawback is that this application can only be used with the Tablet OS and Tablet PC hardware (Young 2005).

6.2. Classification Tree: Version I

6.2.1. Overview of Division Process

A divider using the classification tree described in Section 5.3 is implemented within InkKit. This section will use an example to explain how this tree is implemented to carry out the stroke division process.

The division process is simply a collection of if/else statements which check the values of the feature at each node of the tree and proceeds from there, see Figure 23 for pseudo-code of the divider. Figure 21 shows an example organisation diagram that we use to show how the tree divides diagram strokes into text and shape groups. Two strokes in the diagram are considered here, a shape stroke marked in red and a text stroke marked in blue. Their stroke feature values are given in Table 23. Using these values we can trace the route that is taken through the tree and determine the classification of these strokes based on this divider. If the condition at each node is found to be true the left branch is taken otherwise if the condition is false the right branch is followed. The routes taken through the classification tree for each stroke are shown in Figure 22, marked in red and blue and are also shown in corresponding colours through the pseudo-code in Figure 23.

We begin by considering the text stroke (blue). The first node of the tree checks if the strokes bounding box width is greater than 1848, this strokes value is 109 therefore the right branch is followed to the next node. This node checks if the strokes distance from

the last stroke is >= 2554, this strokes value is 748 therefore the right branch is followed once again to the next node. This node checks if the strokes distance to the next stroke is >= 1647, this strokes value is 601, less than the bounding condition therefore the right branch is taken to a leaf which classifies the stroke as a text stroke. The tree is now terminated.

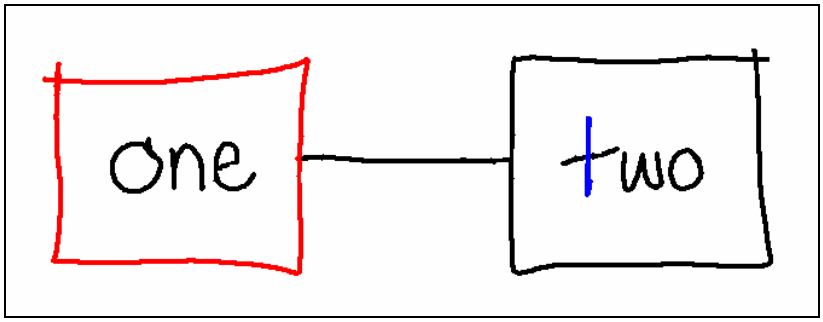


Figure 21. Example Organisation Diagram

Feature Description	**Text Stroke Values**	**Shape Stroke Values**
Bounding box width (HIMETRIC units)	109	2184
Distance from the last stroke (HIMETRIC units)	748	1771
Distance to the next stroke (HIMETRIC units)	601	1896
Speed to the next stroke (milliseconds/HIMETRIC unit)	0.91	2.49
Amount of ink inside (number of points)	25	101
Perimeter to area ratio (HIMETRIC units)	4.8	465
Total angle (degrees)	2.1	4.5
Time till the next stroke (milliseconds)	661	761

Table 23. Example Stroke Data

For the shape stroke (red) the bounding box width is compared to the bounding condition, as its value of 2184 is >1848 the left branch is taken through the tree. Here it arrives at the node for the feature total angle. The bounding condition for this node is found to be true as the shape strokes total angle is 4.5, which is <10.1 therefore the left branch is followed where the tree terminates and the stroke is successfully classified as a shape stroke.

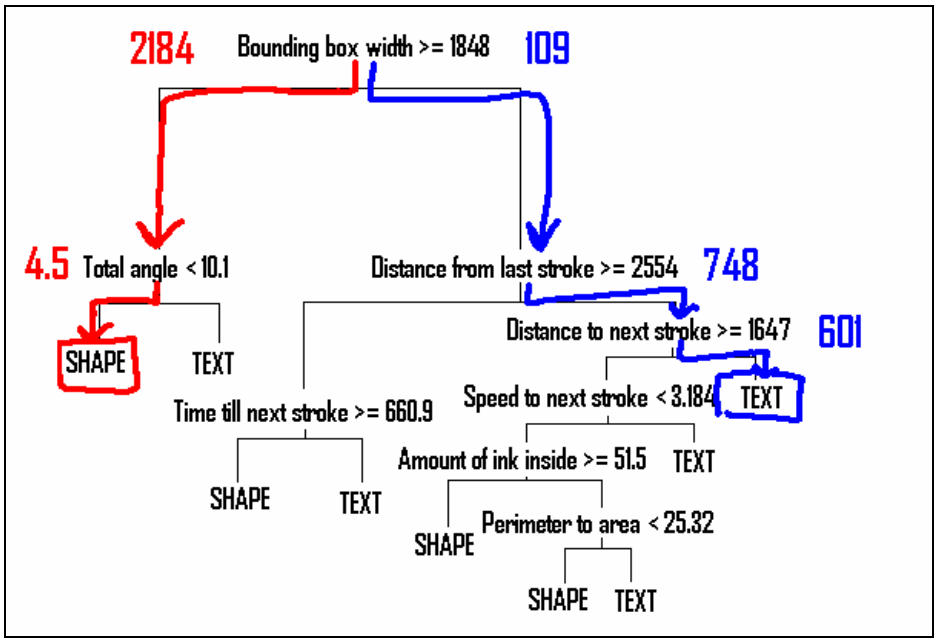


Figure 22. Example Stroke Path through the Version I Tree

```
If (bounding box width >= 1848){
        If(total angle <10.1)
                return SHAPE
        Else
                return TEXT
}
Else(
        If( distance from the last stroke >= 2554){
                If (time till the next stroke >= 660.9)
                        return SHAPE
                Else
                        return TEXT
        }
        Else{
                If(distance to the next stroke>=1647){
                        If( speed to the next stroke < 3.184){
                                If (amount of ink inside >= 51.5)
                                        return SHAPE
                                Else{
                                        If (Perimeter to area < 25.32)
                                                Return SHAPE
                                        Else
                                                Return TEXT
                                }
                        }
                        Else
                                Return TEXT
                }
                Else
                        Return TEXT
        }
}
```

Figure 23. Pseudocode for the Version I Divider

6.2.2. Handling Special Cases

The classification tree, shown in Figure 22, includes features that require data from previous and next strokes, e.g. the distance to the next stroke. Therefore we deal with the following special cases in the implementation by excluding the features that are not able to be calculated because of unavailable previous or next stroke data.

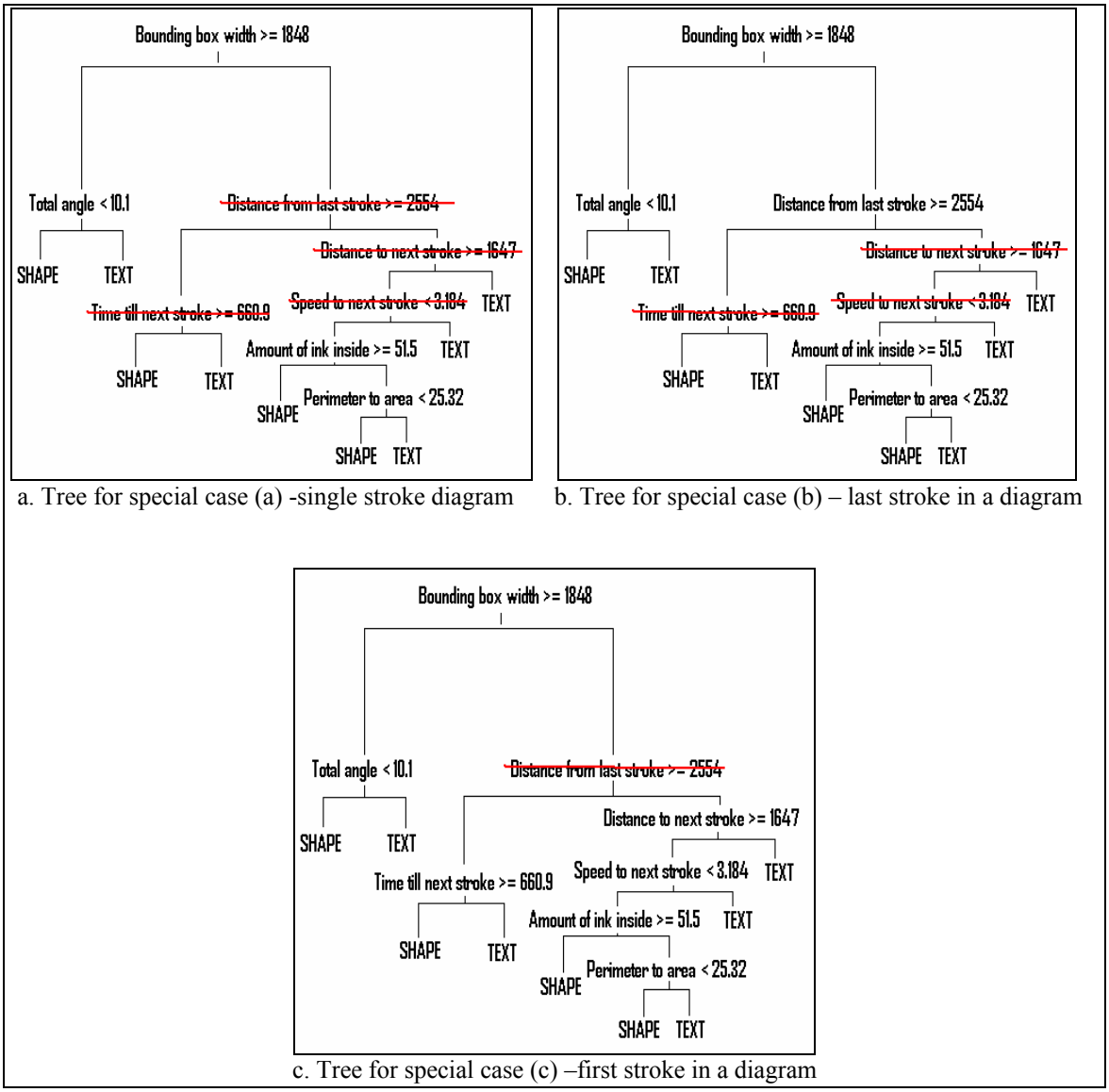
Special cases:

a. No previous and next stroke available e.g. a single stroke diagram
b. No next stroke e.g. for the last stroke in the diagram
c. No previous stroke e.g. for the first stroke in the diagram

The trees for each special case are illustrated in Figure 24 where each feature that is not able to be calculated is crossed out and the remaining features are used to classify the given stroke.

The most important issue when pruning the original classification tree to deal with the special cases is to ensure its accuracy is maintained. Therefore we include as many of the original trees features and ensure that the structure of the original tree is preserved as much as possible.

However an alternative approach is now considered to produce a tree not affected by the special cases. This is done by considering these special cases right from the beginning of the feature analysis. The original dataset is modified by substituting in values for the missing data for the special case strokes. A second feature analysis on this modified dataset is then performed via the tree-based partitioning technique. This produces a second decision tree which is described in the next section. This new tree is evaluated to determine if dealing with special cases in this way produces a more accurate division result.



a. Tree for special case (a) -single stroke diagram

b. Tree for special case (b) – last stroke in a diagram

c. Tree for special case (c) –first stroke in a diagram

Figure 24. Version I Classification Trees for each Special Case

6.3. Classification Tree: Version II

Another approach to handling the special cases described in the previous section is described here. The aim is to create a tree that can be used for all strokes, unlike the Version I tree that must be pruned of some features for special case strokes. By using only one tree for all strokes all the significant features identified by the statistical analysis can be used to classify the strokes.

To create this tree first the original dataset is modified to substitute in values for missing data of the special case strokes, then the tree-based partitioning technique is used once again, as described in Sections 3.3, to analyse the dataset, identifying the significant features for dividing text and shape strokes and constructing a classification tree using these features. The modifications to the dataset, the results of the analysis of the dataset and the implementation of the Version II tree are detailed below.

6.3.1. Dataset Modifications

To create a second version of the tree that deals with the special cases as well as the general strokes in a diagram the dataset is modified. Some observations of features in the original dataset are missing for the first and last strokes of a sketch for example the distance from the last stroke or the distance to the next stroke features. To deal with these missing observations extreme values are substituted into the dataset. The features that are affected are the distance/time/speed from the last stroke and the distance/time/speed to the next stroke. For the missing values for the features distance from the last stroke, distance to the next stroke, time from the last stroke and time to the next stroke, +1million is entered as we expect a positive infinity value (an extremely large distance or time) if the next or previous stroke does not exist in these cases. For the features speed from the last stroke and speed to the next stroke, -1million is substituted in for the missing values as we expect a negative infinity value (an extremely slow speed) here if a previous or next stroke does not exist. The dataset is now ready for analysis.

6.3.2. Results

The modified dataset is analysed using the tree-based partitioning technique resulting in a second version of the classification tree shown in Figure 25. See Appendix B for the R code used for the analysis. Six stroke features are identified as significant in this tree to dividing shape from text strokes. One feature, bounding box width, is used in two separate nodes of the tree.

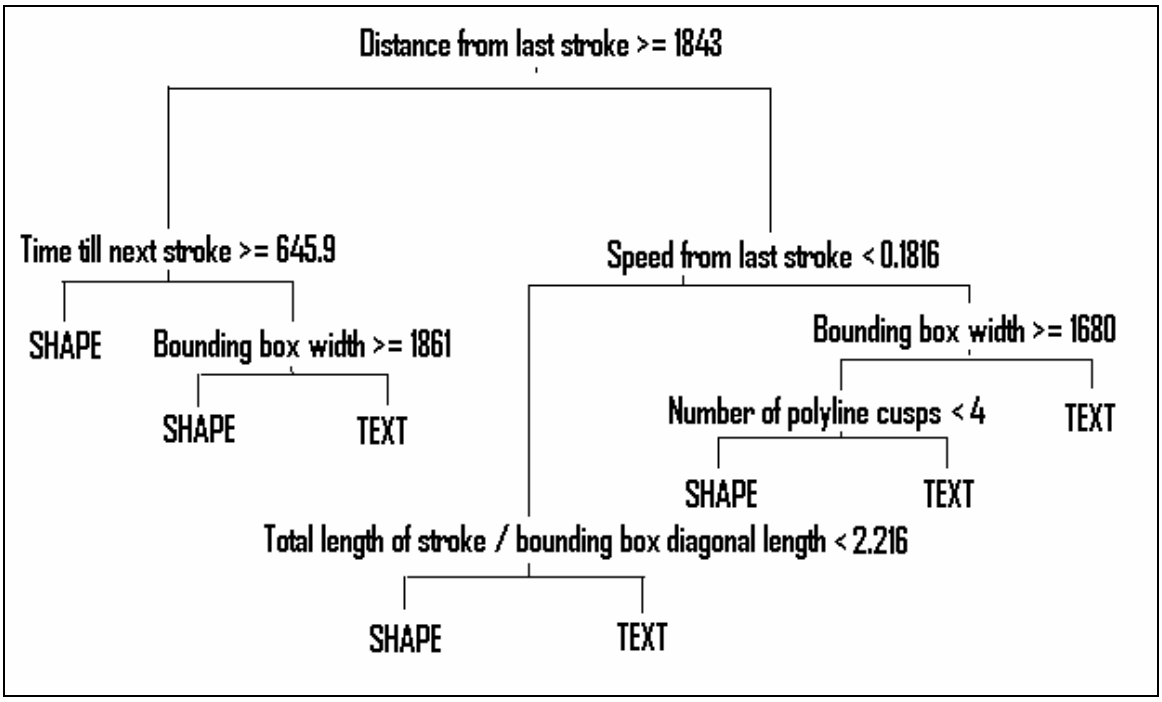


Figure 25. Classification Tree Version II

Name	Category	Origin
Total length/bounding box diagonal length	Size	Adapted from (Rubine 1991)
Bounding box width		Adapted from Tahuti (Hammond and Davis 2002); CALI (Fonseca, Pimentel et al. 2002)
Distance from last stroke		InkKit (Young 2005)
Time till next stroke	Time	New
Speed from last stroke		
Number of polyline cusps	Curvature	

Table 24. Significant Features and their Categories and Origin for the Version II Tree

The relative importance of each feature can be inferred from their position in the tree. The distance from the last stroke feature is clearly the most significant as it is the first node in the tree which all strokes must pass through. At the lowest level of the tree is the number of polyline cusps which is not considered as significant when compared to the others as only a small proportion of strokes are classified using this feature. The fact that the bounding box width is in two nodes of the classification tree marks the feature as very important.

As we can see in Table 24, the majority of these features are measuring some element of size, as half of the features come from this category. Features of time are also important, making up a third of the significant features. One feature of curvature is included as well. In terms of the origin of each feature Table 24 shows that half of the features are new features, and the other half are from Rubine (1991), Tahuti (Hammond and Davis 2002), CALI (Fonseca, Pimentel et al. 2002) and InkKit (Young 2005).

Each feature in the Version II tree is examined below in greater detail to determine if any predicted patterns in the features, put forward in Chapter 4, are confirmed. Accuracy statistics for the percentage of strokes correctly classified as either shape or text strokes for each feature in the classification tree are presented in Table 25 along with the bounding conditions of each feature for grouping shape and text strokes. The accuracy statistics shows the strength of support for the direction of each bounding condition reflected in the conclusions that are drawn. The conclusions in Table 25 are all in support of the hypotheses for these features that were given in Chapter 4.

Features	Shape		Text		Conclusion
	Bounds	% Correct	Bounds	% Correct	
Distance from the last stroke	>= 1843	76%	< 1843	92%	Distance between text < shapes
Time till the next stroke	>= 645.9	92%	<645.9	84%	Time between text < shapes
Speed from the last stroke	< 0.1816	75%	> 0.1816	94%	Speed between text > shapes
Bounding box width (left sub-tree)	>= 1861	80%	<1861	95%	Width of text < shape
Bounding box width (right sub-tree)	>=1680	41.3%	<1680	96.9%	
Number of polyline cusps	< 4	74%	>4	94%	Number of polyline cusps for text > shapes
Total length/bounding box diagonal length	< 2.216	92%	> 2.216	86%	Total length to bounding box ratio for text > shapes

Table 25. Accuracy Statistics for Version II Tree Features

The accuracy statistics in Table 25 are generated by the partitioning technique to ensure that the most optimal tree is constructed. They are based on the number of strokes that enter the particular node of the tree in question.

Through the analysis of a modified dataset using the tree-based partitioning technique, six out of a feature set of 46 stroke features are identified as being distinguishing features of text and shape strokes in sketched diagrams.

6.3.3. Implementation of Version II Tree

Like the Version I classification tree, shown in the pseudo-code in Figure 23, the Version II tree is a simple collection of if/else statements. The only difference in the Version II tree's implementation is the way in which it handles special case strokes, i.e. the only, first or last stroke in a diagram. To accommodate these special cases, the same extreme values that are substituted in for the dataset are used for the implementation of these features. Table 26 shows the values that are substituted for each feature affected by the special cases which correspond to the values used in the modified dataset.

Feature	Substituted Value
Distance from the last stroke	+1 million
Speed from the last stroke	-1 million
Time till the next stroke	+1 million

Table 26. Substituted Values for Special Case Features

Once again using the organisation diagram in Figure 21 as an example we can demonstrate how the Version II tree works. All the feature values for the text stroke marked in blue in the diagram and shape stroke marked in red are shown in Table 27.

Considering the text stroke (blue) first, we begin again at the node which checks the distance from the last stroke. Figure 26 marks the path taken for the stroke according to the feature values in Table 27, in blue corresponding. The text stroke has a value of 748, less than the bounding condition therefore the right branch of the tree is taken. The speed from the last stroke is surveyed to see if it is < 0.1816, at a value of 0.91, greater than

0.1816, the right branch is taken again. Here the bounding box width is compared and since the text strokes bounding box width is not >=1680 the right hand side branch is followed and terminates at a leaf node where the stroke is successfully classified as a text stroke.

Feature Description	Text Stroke Values	Shape Stroke Values
Distance from the last stroke (HIMETRIC units)	748	+1 million
Time till the next stroke (milliseconds)	661	761
Speed from the last stroke (milliseconds/ HIMETRIC unit)	0.91	-1 million
Bounding box width (HIMETRIC units)	109	2184
Total length/bounding box diagonal length (HIMETRIC units)	2.34	1.02
# Polyline cusps	2	6

Table 27. Example Stroke Data for Version II Features

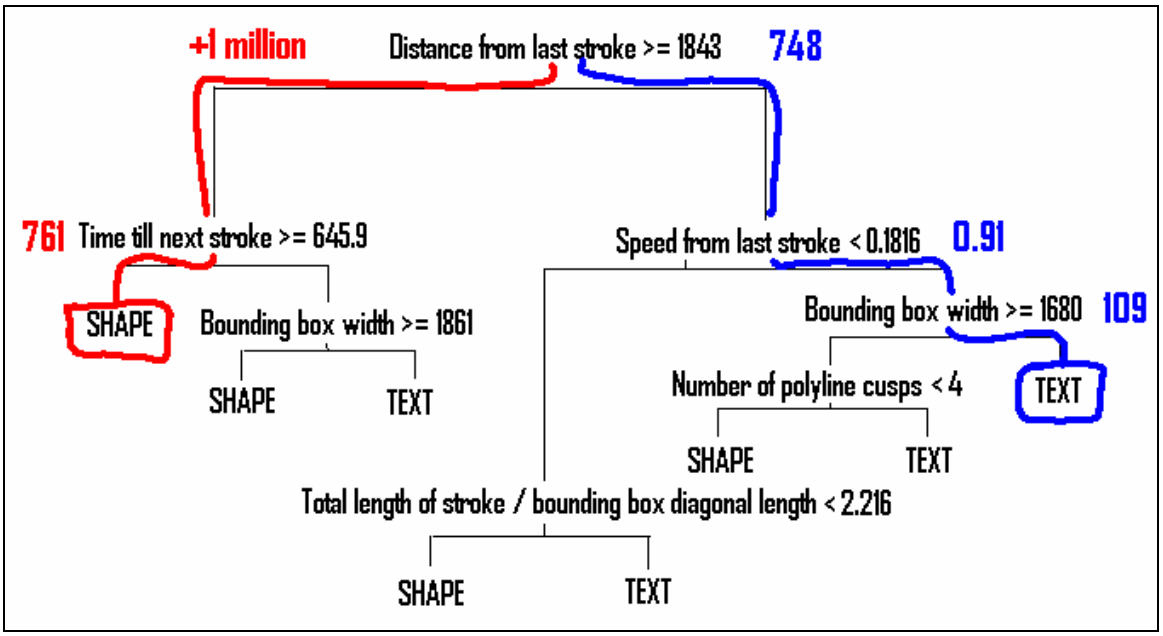


Figure 26. Example Stroke Path through Version II Tree

For the rectangle shape stroke (red), we treat this as the first stroke in the diagram here therefore values must be substituted in for features requiring a previous stroke. The first node in the tree checks if the strokes value for the feature distance from the last stroke is >=1843. The shape stroke has no previous stroke therefore the value +1 million is used

here which is greater than 1843 so we follow the left branch of the classification tree. Here we see if the time till the next stroke is >= 645.9, our strokes value is greater at 761 causing the left branch to be taken once again where we arrive at a leaf node which successfully classifies the stroke as a shape.

6.4. Comparison of Decision Trees

Two sets of features, Version I and Version II, are identified as useful to divide shape and text strokes. There are some commonalities in these features sets that are discussed here.

The features shown in Table 28 are those that are included in both Version I and Version II classification trees. This suggests that these features are considered highly significant to solving the division problem. The common direction of the bounding conditions also shows us that shape strokes typically have a larger bounding box width, a greater distance from the last stroke and a greater time to the next stroke than text strokes.

Feature	Version I bounds	Version II bounds
Bounding box width	>= 1848	>= 1861; >= 1680
Distance from last stroke	>= 2554	>= 1843
Time till next stroke	>= 660.9	>= 645.9

Table 28. Common Features of Version I and II Trees

Category	Version I Features	Version II Features
Size	Bounding box width Distance from last stroke Distance to next stroke Amount of ink inside Perimeter to area	Bounding box width Distance from last stroke Total length/bounding box diagonal length
Time	Time till next stroke Speed till next stroke	Time till next stroke Speed from last stroke
Curvature	Total angle	Number of polyline cusps

Table 29. Categorisation of Features for Version I and II Trees

We also draw conclusions from the category that each of the features come from, shown in Table 29. Both feature sets contain features from the size, time and curvature

categories. Size appears to hold the most importance making up at least half of the features for the Version I and II feature sets. The features from the time category are interesting in that they are almost identical for each tree, where the time till the next stroke and speed between strokes is shared, the only difference being that Version I takes the speed to the next stroke and Version II uses the speed from the last stroke. There is also one feature in each feature set which comes from the curvature category. Analysis of the categorisation of each feature set also shows that features of pressure, intersections and the Tablet OS recognition predictions are not significant to dividing text and shape strokes.

Origin	**Version I Features**	**Version II Features**
New	Time till next stroke Speed till next stroke	Time till next stroke Speed from last stroke Number of polyline cusps
InkKit (Young 2005)	Distance from last stroke Distance to next stroke Amount of ink inside	Distance from last stroke
(Rubine 1991)	Total angle	Total length/bounding box diagonal length
Tahuti (Hammond and Davis 2002);	Bounding box width	Bounding box width
CALI (Fonseca, Pimentel et al. 2002)		
	Perimeter to area	

Table 30. Origin of Features for Version I and II Trees

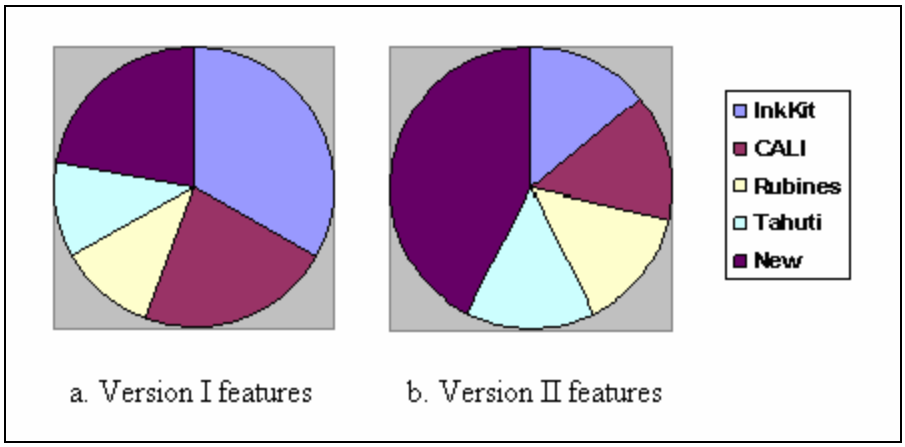


Figure 27. Origin of Features

The origin of the features for both dividers is also similar, as shown in Figure 27 and Table 30. They are from InkKit (Young 2005), CALI (Fonseca, Pimentel et al. 2002), Rubine's work (1991), Tahuti (Hammond and Davis 2002) and two are new to sketch recognition. Almost the same proportion of features for both dividers originates from Rubine's work and Tahuti. It is not surprising that both feature sets share similar origins as they are trained with similar data.

6.4.1. Summary of Features

Due to their identification by both Versions I and II classification trees, the most significant features found to distinguish between text and shape strokes are bounding box width, distance from the last stroke and the time till the next stroke. The three features confirm the following hypotheses:

- Shape strokes are typically larger in the width of their bounding box than text strokes.
- Shape strokes on average have a greater time till the next stroke when compared with text strokes.
- Shape strokes on average have a greater distance from the last stroke than text strokes.

Other features in the two feature sets also confirm the following hypotheses:

- The typical speed between shape strokes is slower than the speed between text strokes.
- On average there are less polyline cusps in shape strokes than text strokes.
- On average shape strokes have a smaller total angle than text strokes.
- On average shape strokes have a smaller perimeter to area ratio of their convex hull than text strokes.
- On average shape strokes have a smaller total length to bounding box diagonal length ratio than text strokes.

The only feature that does not behave as expected is the amount of ink inside the bounding box. It is thought that text strokes have a greater amount of ink inside its bounding box but the reverse is found when used in the classification tree. However considering that we expect that the typical bounding box and length of a shape stroke is greater than that of a text stroke, this could explain why the results show more ink found inside shape strokes. It may be more accurate if it is calculated as a ratio to the length of the stroke or the size of the bounding box.

Features of size, time and curvature are important in both feature sets, indicating that a significant difference exists in the size, time and curvature between text and shape strokes.

One other interesting point about the features identified is that all but one, the number of polyline cusps, are continuous measures rather than discrete counts. As mentioned in Section 2.1, Rubine (1991) states that features based on discrete counts are subject to greater error therefore he prefers continuous measurements which support his normality criteria for features. This appears to be confirmed by the feature analysis here.

6.5. Pre-processing Strokes by Grouping

As well as trying to identify the most significant features to improve the division of text and shape strokes, we thought that grouping strokes that are likely to be a part of the same word or shape as a pre-processing step before division may help. Here we describe why and how strokes are grouped for the divider implementation.

InkKit joins strokes together after the division process before it comes to determining the basic shape of the probable shape strokes as illustrated by InkKit's architecture diagram in Figure 9. The strokes are not physically joined together, rather they are grouped into one data structure that allows the basic shape recognition engine to take into account the characteristics of each stroke in the group and recognise that group as a particular basic shape based on the results.

We thought that if joining the strokes in this manner helps basic shape recognition then it may help the divider as well. By grouping the strokes that are most probably part of the same word or shape and using the features in the classification trees to determine the type of the whole group of strokes rather than relying on a single strokes data we could potentially improve the accuracy of the divider. The risk is that shape and text strokes may be mistakenly grouped together resulting in the misclassification of some strokes in the group. Therefore, when implementing the Version I and II trees into dividers three groupings of strokes are used as follows.

- Single strokes

  Where no grouping is used, each stroke represents its own group. This is illustrated in Figure 28a where a rectangle shape is sketched. Each stroke group has a blue bounding box around it. There are four bounding boxes here for each of the four strokes of that make up this rectangle. This is used as an option to allow comparisons to a divider that do not use any stroke grouping and helps determine if grouping does in fact have some effect on a dividers accuracy that can overshadow any risk of misclassification due to incorrect grouping.

- Intersecting strokes

  Strokes are grouped together if they intersect one another. Figure 28b shows such a grouping where a rectangle drawn with four separate strokes is grouped into three groups (shown by the three blue bounding boxes). The top horizontal stroke and the right side vertical stroke are intersecting and therefore form one group. The other two strokes form the remaining groups. The premise is that strokes that intersect each other are most likely to be part of the same word or shape and therefore can form a logical group of strokes for the divider to classify together as one.

- Close strokes

  Strokes are grouped together if they intersect one another or if they "almost" intersect one another, meaning their endpoints are in very close proximity to each other. Figure 28c illustrates this grouping where the four strokes of the rectangle drawn are

grouped into one group (shown by the blue bounding box). Strokes whose endpoints are very close together are likely to be part of the same word or shape; they do not necessarily have to intersect as we have observed that when people sketch they often leave a small distance between strokes of the same component. Therefore strokes that are intersecting and very close by can form a logical group of strokes that can be classified by the divider as one.

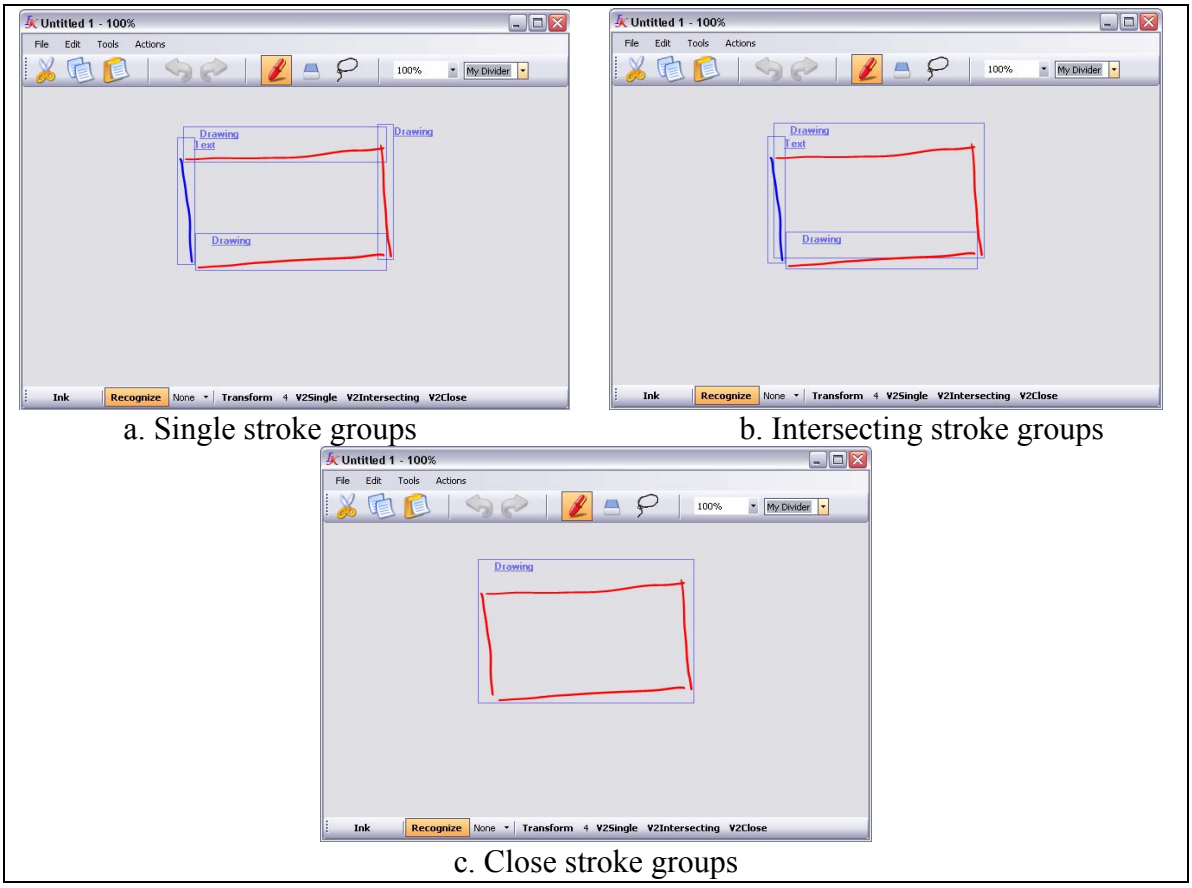


Figure 28. Single, Intersecting and Close Stroke Groupings

For groups with more than one stroke, the calculation of each feature in the tree is modified slightly to take into account the characteristics of each stroke in the group and combine them together to calculate a single value for each feature for this group. Therefore each group of strokes has this single combined value for each feature in the tree that is used to classify the group of strokes as text strokes or shape strokes. A description of how each feature in the Version I and II trees is calculated for a group of

strokes is show in Table 31 below. The effects in terms of the accuracy of each divider using this pre-processing step are shown by the evaluation in Chapter 7.

Name	Calculation	Divider
Time till next stroke	Same calculation except that it is the time from the last stroke in the group to the first stroke in the next group.	Version I & II
Bounding box width	Bounding box width of stroke collection obtained from C# Strokes class	
Distance from last stroke	Same calculation except that it is the distance from the last stroke in previous group to the first stroke in the current group.	
Speed till next stroke	Same calculation except that it is the speed from the last stroke in the group to the first stroke in the next group.	Version I
Distance to next stroke	Same calculation except that it is the distance from the last stroke in the group to the first stroke in the next group.	
Amount of ink inside	∑ amount of ink inside for each stroke using the C# Stroke classes bounding box for all strokes in the group	
Perimeter to area	Area of convex hull / perimeter of convex hull for the convex hull made from all the strokes in the group.	
Total angle	∑ total angle for each stroke in the group.	
Speed from last stroke	Same calculation except that it is the speed from the last stroke in the previous group to the first stroke in the current group.	Version II
Number of polyline cusps	∑ # polyline cusps for each stroke in the group.	
Total length/bounding box diagonal length	∑ Length of each stroke in the group / length of the stroke collections bounding box diagonal	

Table 31. Feature Calculations for a Group of Strokes

6.6. Implementation Experiences

Implementing the dividers is not difficult. As shown by the pseudo-code in Figure 23 the bulk of the classification tree is a simple collection of if/else statements. Also calculating the features of each stroke to be used in the classification tree is straightforward as long as you are able to obtain the ink data. Using the Ink SDK ensures good access to the Ink

and Stroke objects and their corresponding data such as the stroke points and timing information. Although special stroke features such as time and pressure need to be explicitly enabled so that their information is collected for each stroke.

The more difficult challenge is learning how to navigate through a large project such as InkKit. Discovering how large projects work and where additions are best made to the system without affecting the rest of its functionality requires a lot of time and effort.

Overall the implementation is only a small part of this investigation. The most significant part of this research is the features identified which are subsequently used by the divider to improve classification of text and shape strokes and sketch recognition in general.

With a complete implementation of the new dividers, Version I and II, ready, we evaluate them against the existing InkKit divider and the Microsoft divider to prove which has the greatest accuracy in classifying text and shape strokes.

Chapter 7

Evaluation

The goal of this research is to improve sketch recognition, focusing on improving the divider of text and shape strokes to reach this goal. Through the use of a formal statistical technique we have identified two new sets of features, these feature sets are implemented as the Version I and II dividers. To test the accuracy of our two new dividers an evaluation is carried out according to the plan laid out in Section 3.5. The performance of the new dividers is compared with InkKit's existing divider and the Microsoft divider to see which is able to divide text and shape strokes with the lowest misclassification rate. The misclassification rate is a measure of the proportion of strokes that are incorrectly classified. The four dividers are used to divide the original training set of diagrams and a new diagram set. Results are compiled based on the proportion of strokes that are correctly and incorrectly classified as text or shape strokes. The level of accuracy of each divider is then assessed.

The four dividers, Versions I and II, InkKit and Microsoft, are all implemented in parallel within InkKit. In order to gather divider data the recognition process is stopped at the end of division for all dividers, excluding phase II of InkKit's divider as this is using a second iteration for division using the classification of strokes close by to strengthen the classification of the current stroke. This second iteration is future work for our divider and therefore this is not part of the evaluation.

This chapter presents an outline of what is done to evaluate the dividers performance, and proceeds to show the results of the dividers evaluation on the original diagram set and the new diagram set. The results allow us to fully compare and evaluate the performance of each divider so that the most accurate one is identified.

7.1. Outline of the Evaluation

The following is a discussion of the pre-processing steps that are tested on each divider and consequently evaluated, the diagram sets that are used in the evaluation and an overview of the evaluation process.

7.1.1. Pre-processing Strokes by Grouping

A pre-processing step is used before classifying the strokes by each divider as described in Section 6.5. This involves grouping strokes that are likely to belong to the same shape or word and classifying the group of strokes together using a divider. The three grouping alternatives are implemented for all dividers except the Microsoft divider as it would be complex to over-ride its default behaviour and appears that it does its own grouping.

7.1.2. Diagrams

The original training set of diagrams shown in Table 20 collected for analysis is used for the evaluation. These diagrams were chosen to be part of the set because they were considered to be a good representation of the typical shapes and text combinations that occur in diagrams. However after the analysis was complete we reflected on the choice of diagrams used in the training set and it was recognised that some special types of diagrams were missing. This led to the creation of a new diagram set which will be discussed further in Section 7.3.

The addition of a new diagram set separates the evaluation into two parts. First the evaluation of the original diagram set. This is presented in Section 7.2. Then the evaluation of the new diagram set is presented in Section 7.3.

7.1.3. Evaluation Process

Despite the difference in diagram sets the evaluation process is the same for both parts. Each diagram is processed by the Version I and II dividers, InkKit's existing divider and the Microsoft divider, which are all implemented within InkKit, where the dividers classify the diagrams into probable text and shape strokes. The dividers are implemented so that probable text strokes are coloured in blue and probable shape strokes in red. There are also blue bounding boxes around each group of strokes labelled as "text" or "drawing" based on the classification. Figure 29 shows an example of a diagram of a button that is divided by the Version I divider with a single stroke grouping. The feedback given, showing the rectangles strokes coloured in red and the all the text strokes in blue confirm that this diagram is correctly divided into shape and text strokes.

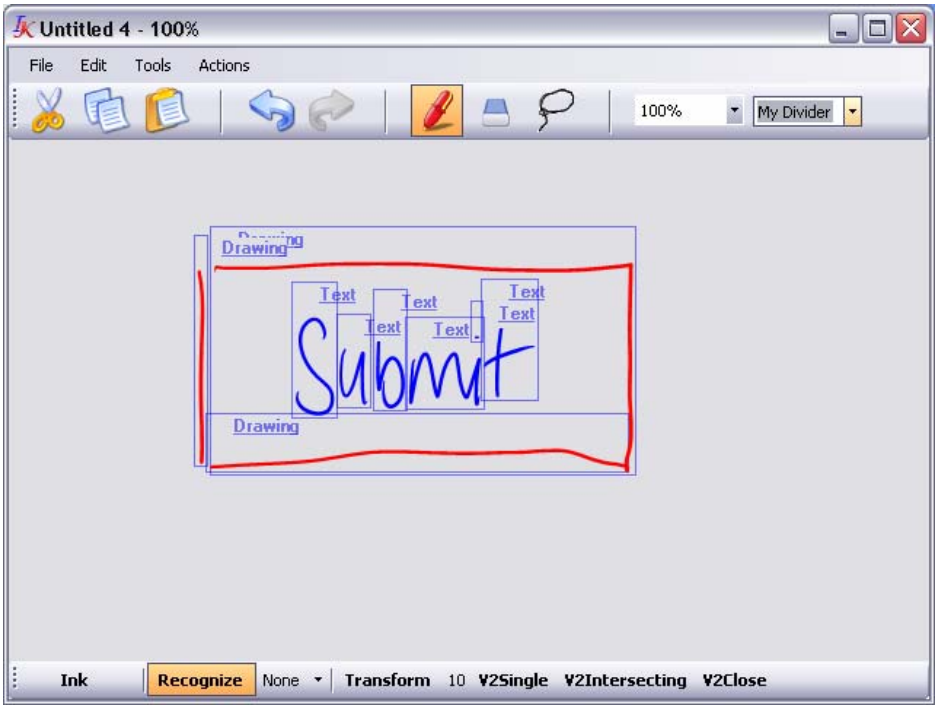


Figure 29. Feedback of Stroke Classification using Stroke Colouring

Table 32 shows the various measures that are observed and recorded during evaluation for each sketch using the four basic methods of division. The information collected is the number of strokes, text and shape strokes, then the number of text and shape strokes that are correctly classified and the number of text and shape strokes that are misclassified.

Each of these is easily counted from the divider results as shown in Figure 29. When all the diagrams are processed by each divider and the information summarised in Table 32 is gathered, the accuracy of each divider is determined using the measures shown in Table 33. These are essentially calculations collating all the individual evaluation information in Table 32 into the percentage of correctly classified and misclassified strokes for each divider.

Evaluation Measure	**Description**
ID	Identification number of the person who sketched the diagram
Diagram Type	Description of what is sketched in the diagram
Division Method	Either Version I, Version II, InkKit or Microsoft divider
# Strokes	Total number of strokes in the sketch
# Text Strokes	Total number of text strokes in the sketch
# Shape Strokes	Total number of shape strokes in the sketch
# Correct Text Strokes	Number of text strokes that are correctly classified by the divider
# Correct Shape Strokes	Number of shape strokes that are correctly classified by the divider
# Misclassified Shape Strokes	Number of shape strokes that are incorrectly classified by the divider
# Misclassified Text Strokes	Number of text strokes that are incorrectly classified by the divider
General Comments	Any observations of parts of the sketch that are correctly/incorrectly classified by the divider

Table 32. Information Recorded during Evaluation of Dividers.

Evaluation Measure	**Description**
Total % Correct Text Strokes	∑# Correct Text Strokes / ∑# Text Strokes * 100
Total % Correct Shape Strokes	∑# Correct Shape Strokes / ∑# Shape Strokes * 100
Total % Misclassified Shape Strokes	∑# Misclassified Shape Strokes / ∑# Shape Strokes * 100
Total % Misclassified Text Strokes	∑# Misclassified Text Strokes / ∑# Text Strokes * 100
Total % Correct Strokes	(∑# Correct Text Strokes + ∑# Correct Shape Strokes) / ∑# Strokes
Total % Misclassified Strokes	(∑# Misclassified Text Strokes + ∑# Misclassified Shape Strokes) / ∑# Strokes

Table 33. Measures of Accuracy Calculated for Evaluation of Dividers.

7.1.4. Analysis

The focus of the analysis of the evaluation results is on which divider is the most accurate at classifying strokes overall. From this we can identify the most effective feature set to divide text and shape strokes. We are also interested in how well each divider handles each diagram type from both the original and new diagram set. If we identify diagrams the dividers do not classify well it infers some features of diagrams that have not been acknowledged. We expect that the new diagram types fair worse than the original diagram set under the Version I and II dividers based on the fact that the original set is used to train the new dividers and they represent different types of diagrams – particularly music which isn't really a diagram.

The evaluation also tells us if the division methods benefit from strokes being grouped together before division. If so we are interested in which grouping produces the best division results so that this pre-processing step can be used in the future for text and shape stroke classification. Analysis of the evaluation results also allows us to learn if individual variation, i.e. the differences in sketching between people, has an effect on the performance of each divider and which divider it has the greatest effect on.

7.2. Performance of Dividers on the Original Diagram Set

The evaluation of the dividers carried out using the original diagram set that is collected for analysis shown in Table 20. Each diagram in the set is divided into probable text and shape strokes by the Version I and II dividers, InkKit's existing divider and the Microsoft divider. The performance of each divider in terms of their accuracy of stroke classification is presented as follows.

7.2.1 Divider Type

The percentage of strokes that are misclassified for each divider is shown in Figure 30 is broken into the percentage of misclassified text and shape strokes and the total percentage of misclassified strokes per divider.

The Microsoft divider has the highest percentage of misclassified shape strokes at 75.7% and the lowest percentage of misclassified text strokes, where no text strokes are incorrectly classified at all. This indicates very strongly that this divider classifies almost all strokes as text. Such a divider is not very useful for diagrams which are composed of text and shape strokes.

The Version I divider is clearly the best divider at classifying shape strokes with the lowest proportion of misclassified shape strokes when compared with the other dividers, at 10.8%. It also has the second lowest proportion of misclassified text strokes only second to the Microsoft divider which as already established classifies the majority of strokes as text which ensures its favourable classification of text strokes. Overall we can see from the total percentage of strokes misclassified that the Version I divider is the most accurate divider as classifying strokes for this diagram set where only 9.4% of strokes are misclassified.

The InkKit divider has a very high misclassification rate for shape strokes at 67.4%, coming in as the second highest errors of all dividers, however in contrast it has a low percentage of misclassified text strokes. This is similar to the pattern of the Microsoft divider. This could be explained by the fact that InkKit uses the Microsoft divider classification results as a part of its divider, see Section 2.2 for more details on InkKit's divider.

The Version II divider has the highest percentage of misclassification for text strokes where 49.8% of text strokes are incorrectly classified. This divider has a much lower rate of misclassification for shape strokes where 19.1% of shape strokes are incorrectly

divided. However the total percentage of misclassified strokes is 41.2% for this divider making it the worst divider at classifying strokes for this diagram set. Possible reasons for the Version II dividers poor performance are that fabricated values are used for missing feature data when a stroke does not have a previous or next stroke as explained in Section 6.3. The values are substituted into the dataset for features indicating a spatial or temporal relationship between strokes e.g. distance/time to the next stroke, and the dataset is subsequently analysed by the tree-based partitioning technique. This means that the resulting tree that is constructed is based in part on these made up feature values. Therefore the divider is not based on true data, so when it is used to classify real strokes the classification is not accurate. For strokes with no previous or next stroke especially, the actual spatial and temporal relationships between strokes are not reflected accurately by the features.

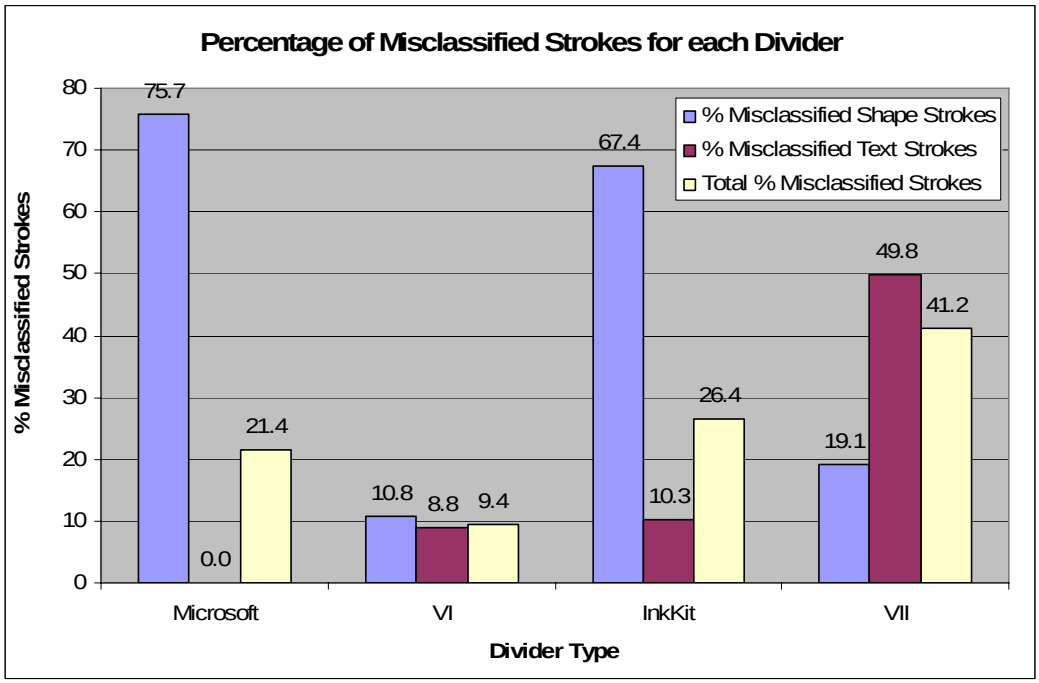


Figure 30. Percentage of Misclassified Strokes for each Divider

7.2.2 Divider and Diagram Type

The performance of each divider when classifying each type of diagram in the original diagram set is shown in Figure 31.

The Microsoft divider varies greatly in its performance between diagram types. It performs best for the Text diagram with 0% of strokes being misclassified. It is worst at dividing the Connector diagram with 86.8% of strokes misclassified. Other diagrams that have very high misclassification rates are the Combo box, Hexagon and the Resistor, all consisting of shapes only. These results confirm that it tends to classify almost all strokes as text.

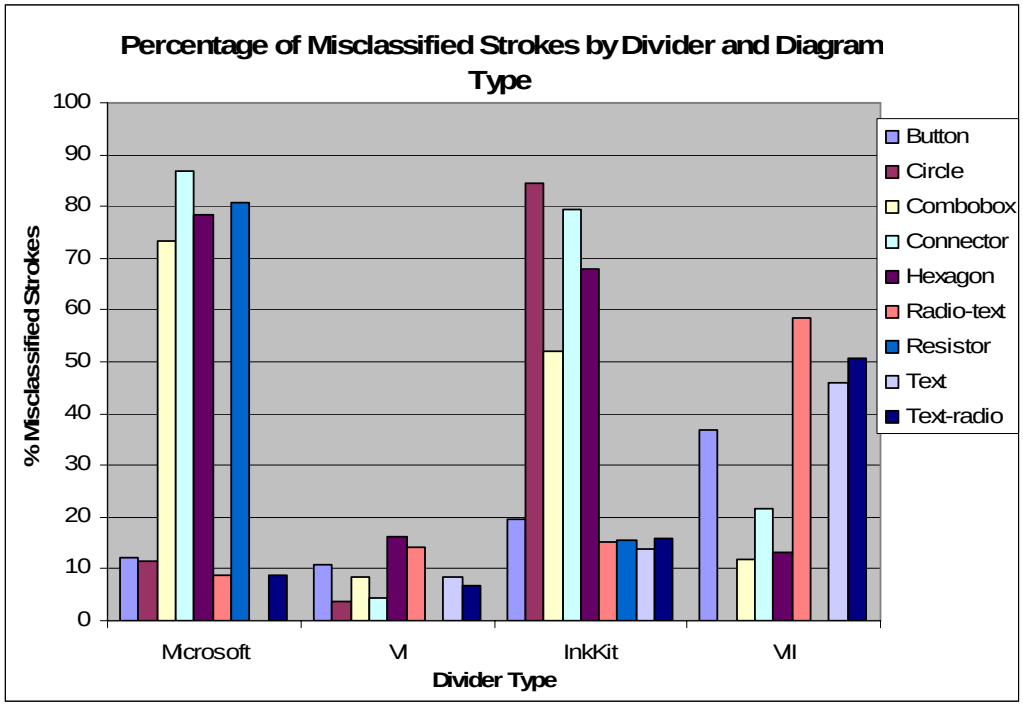


Figure 31. Percentage of Misclassified Strokes by Divider and Diagram Type

The Version I divider has very little variation in performance when classifying each diagram type. It has a 0% misclassification rate for the Resistor diagram and the highest misclassification rate for the Hexagon diagram at 16.1%. The resistor diagram is most commonly drawn with a single stroke whereas the hexagon can range from one to six strokes or more which may make it more difficult to classify. This directs further research into features for classifying hexagons and other polygons to improve their division.

Like the Microsoft divider, InkKit's divider shows a large variation in performance between the different diagram types. The worst misclassification rate is for the Circle diagram at 84.6%, with high misclassification rates for other shape diagrams Connector,

Hexagon and Combobox. The best classification is for the Text diagram at 14.0%. Once again this confirms this dividers similarity to the Microsoft divider in classifying most strokes as text strokes, which could be due to its dependence on the Microsoft divider's results as one of its features.

The Version II divider has a 0% misclassification rate for the Circle and the Resistor diagram. This could be because circles and resistors are commonly drawn with single strokes making them easier to classify. It has the highest percentage of misclassified strokes for the Radio-text diagram at 58.3%. Through observation of this divider when classifying this diagram type, it has the most trouble classifying the radio button as a shape. The radio button is a small circle which can easily be mistaken for text if we simply look at its geometric properties, however the relationship of this shape stroke with the rest of the diagram, i.e. the distance of the shape from the text label should provide a distinguishing feature for this diagram, for example the distance of this stroke to the next stroke should be larger than for the letter 'o' in a word. However as mentioned earlier the Version II divider is not able to accurately classify strokes based on their relationship to other strokes. Some of the values for the features that express a stroke's temporal and spatial relationships are substituted values as the data is missing for strokes with no previous or next stroke.

7.2.3 Divider and Stroke Grouping

Figure 32 illustrates the performance of each divider when using one of the three preprocessing techniques to group strokes, a single stroke grouping, intersecting and a close stroke grouping. The Microsoft divider is not included here as we did not apply any grouping to it.

For the Version I divider there is a negligible difference in the percentage of misclassified strokes when each type of stroke grouping is used ranging from 9.1% for close stroke grouping, to 9.6% for single stroke grouping. InkKit also has very similar values for the three stroke groupings ranging from misclassification rates of 26.0% for intersecting

stroke grouping to 26.8% for close stroke grouping. Therefore stroke grouping is not seen to have any affect for these two dividers.

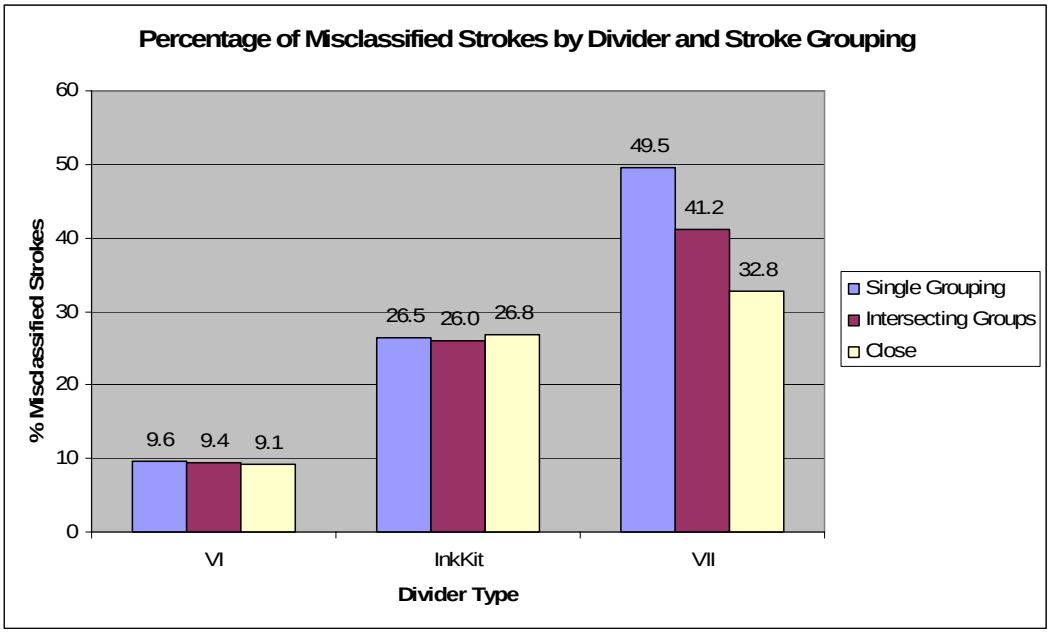


Figure 32. Percentage of Misclassified Strokes by Divider and Stroke Grouping

The Version II divider is the only divider showing a difference in performance when different stroke groupings are used. It has the highest rate of misclassification when a single stroke grouping is used at 49.5% and the lowest at 32.8% strokes misclassified when a close stroke grouping is used. This indicates that a close stroke grouping produces the most accurate results for the Version II divider. It is unclear as to why grouping makes a difference to this divider and not the others.

7.2.4 Divider and Participants

Figure 33 shows the distribution of the percentage of strokes misclassified for each divider over all the participants. The participants are the people who drew the original diagram set that is used for feature analysis. This indicates the effect of individual variation on each dividers result.

The Version I divider shows the smallest distribution of results therefore is least affected by individual variation. It also has the lowest standard deviation which further confirms these results shown in Table 34.

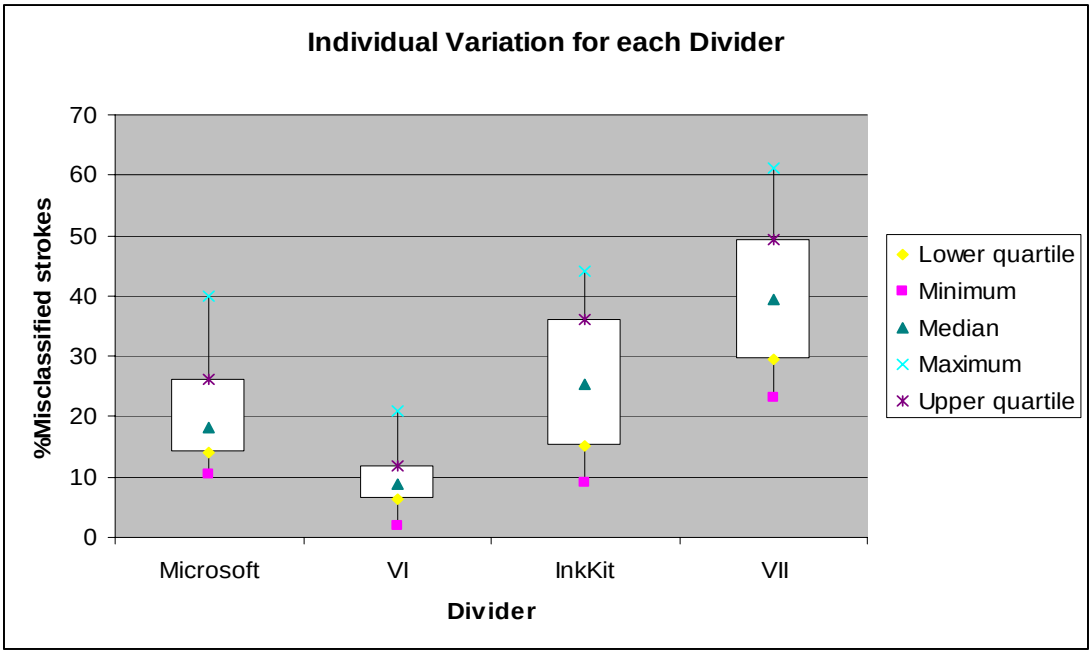


Figure 33 . Individual Variation for each Divider

	% Misclassified Strokes			
	Microsoft	**Version I**	**InkKit**	**Version II**
Mean	21.1	9.7	25.9	40.2
Standard deviation	9.1	5.1	10.7	12.0

Table 34. Mean and Standard Deviation of Stroke Misclassification by Divider over all Participants

The Version II divider and InkKit show similarly large distributions of results showing that they are most affected by individual variation. Their standard deviation from Table 34 shows that the Version II divider has a slightly larger standard deviation. We conclude that the Version II divider is most affected by individual variation.

Also observing the relative position of each box plot shows that half of all participants results when the Version I divider is used have a lower total misclassification rate than when all other dividers are used. This is apparent as the median value for the Version I

divider is lower than the minimum value of every other divider, further confirming the success of the Version I divider.

7.3. Performance of Dividers on the New Diagram Set

The diagram set that is used to train our new dividers, shown in Table 20, were chosen as they are considered to be examples illustrating the common characteristics seen in most diagrams. Upon reflection it is apparent that there are some special types of diagrams or characteristics that are not represented in the original set. This includes arrows, diagrams with deliberate intersections, single letters inside shapes and check boxes. Musical notes are also added as a student had written an add-in for musical notes in InkKit so we are interested to see if the new dividers are able to handle such a complex and specific diagram type. Therefore a new diagram set is created to include these diagrams as shown in Table 35 so that we can evaluate the dividers against these examples to see how well they perform.

In order to evaluate these diagrams using the four dividers new sketches were collected. Nine participants are involved, ranging from 18 to 35 years old, four of them coming from a computer science background. The diagrams were sketched on a Tablet PC using the InkKit application. Most had no previous experience sketching on a Tablet and are given a few minutes to familiarise themselves with using the pen to sketch. Each participant is given a piece of paper with sample drawings of each diagram in the required set and are asked to sketch the complete set in one sitting. When writing any text they are asked to omit any punctuation. Also any further questions they have are answered.

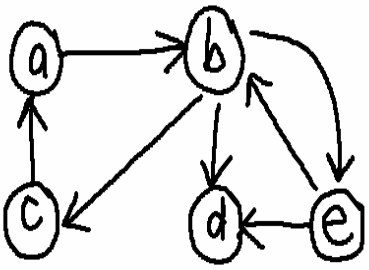
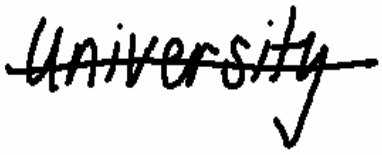
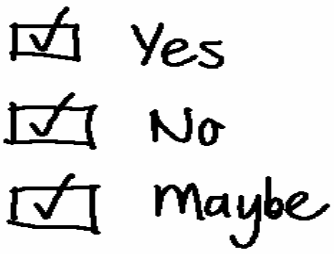
Shape Description	Example Sketch	Rationale for Choice
Directed Graph		This diagram illustrates a combination of shapes and words with the containment of a single letter within a shape that is very common in diagrams. It also gives us an example with a new complex shape, the arrow. Several arrows are included here to cover as many possible directions and orientations.
Cross out: a word crossed out by a line.		This is a new combination of shapes and words that includes deliberate intersections of the line crossing out the word. It is included to represent an example of what may appear in diagrams such as to-do lists.
Check boxes		This gives us another combination of shapes and words where a label is beside a shape. More importantly it is an example including a composite shape with the box and a tick intersecting it, not considered in the original diagram set.
Musical Notes		These complex shapes are included out of curiosity to see how our dividers handle such shapes without any special consideration of their unique features such as the filled in circles attached to lines. There are no expectations as to the performance of the dividers on this highly specialised diagram.

Table 35. New Diagram Set

7.2.5 Divider Type

Each sketch from this new diagram set is then divided into probable text and shape strokes by the InkKit, Microsoft and new dividers. Figure 34 shows the performance of each divider on the new diagram set in terms of the percentage of misclassified shape and text strokes and the total percentage of misclassified strokes for each.

The Microsoft divider misclassifies 93.1% of shape strokes which is the highest of all dividers. In contrast it has the lowest misclassification rate for text strokes at only 1.4%. This is the same pattern shown by the performance of the divider on the original diagram set where there is gross misclassification of shapes strokes due to the divider simply classifying almost all strokes as text regardless of their type.

The Version I divider classifies 42.1% of shape strokes incorrectly, the lowest misclassification rate of shape strokes for all dividers. The Version I dividers percentage of misclassified text strokes is the second highest at 21.4%. Overall this divider maintains its ranking, as the most accurate divider with a total of 34.1% of strokes misclassified. These are still very high rates of misclassification even if comparatively they are not as bad as other dividers. However this is expected as the Version I divider is not trained to classify these diagrams.

InkKit's existing divider has a very high percentage of misclassified shape strokes at 80.8% and the second lowest rate of misclassification for text strokes at 17.2%, once again following the pattern of the Microsoft divider by classifying most strokes as text.

The Version II divider has high misclassification rates for both shape and text strokes at 59.0% and 52.0% respectively. It is the worst of all dividers at classifying text strokes here just as it was in classifying the original diagram set. This is somewhat expected as the Version II divider is not trained to classify the diagrams in this diagram set and had previously performed poorly.

Also observe that all the dividers are much worse at classifying shape strokes than text strokes. This suggests that more shape specific features may need to be identified.

Overall looking at the total percentage of misclassified strokes the Microsoft, InkKit and Version II dividers all have a similar percentage of strokes misclassified ranging from 56.2% to 57.6%. It is clear that the Version I divider has out performed all the others with 34.1% of strokes misclassified for the new diagram set.

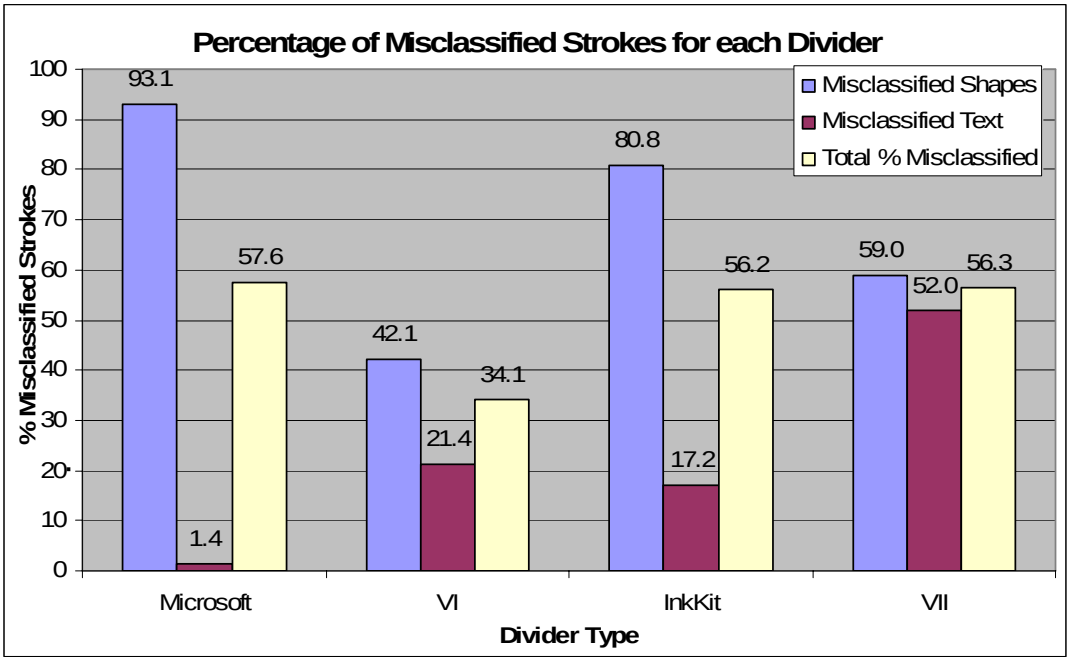


Figure 34. Percentage of Misclassified Strokes for each Divider for the New Diagram Set

7.2.6 Divider and Diagram Type

The classification accuracy of each divider on each diagram type in the new diagram set is shown in Figure 35.

The Microsoft, Version I and InkKit dividers all have the most accurate results for the cross-out diagram at 2.0%, 24.8% and 20.1% of strokes misclassified respectively. This is a very simple diagram mainly composed of text which we know the Microsoft and InkKit dividers always classify well therefore this is an expected result.

The Version II divider has the best results for the directed graph where 43.2% of strokes are misclassified. This could be because it is very good at classifying circles as shown by the evaluation results of the original dataset in Figure 31 where it has a 0% misclassification rate for the circle diagram.

All dividers have the worst stroke misclassification rates for the musical notes where the InkKit divider is the highest at 96.5% of strokes being misclassified and the Version I

divider has the lowest rate at 55.4% misclassified. This result is also expected as the musical notes are a highly specific diagram type that none of the dividers are trained for. This diagram is only tested out of curiosity to see how well the dividers can handle it. The fact that the Version I divider is much better than the other dividers at classifying this diagram is a very positive sign that this divider can be used to classify shape and text strokes in a wide range of diagram domains with comparatively more accuracy than the other dividers, even if it is not trained specifically for the domain.

Also note that all the misclassification rates except for Microsoft's 2% rate for cross-outs are very high. Considering that none of the dividers are trained for these diagrams it is not surprising that the misclassification rates are high.

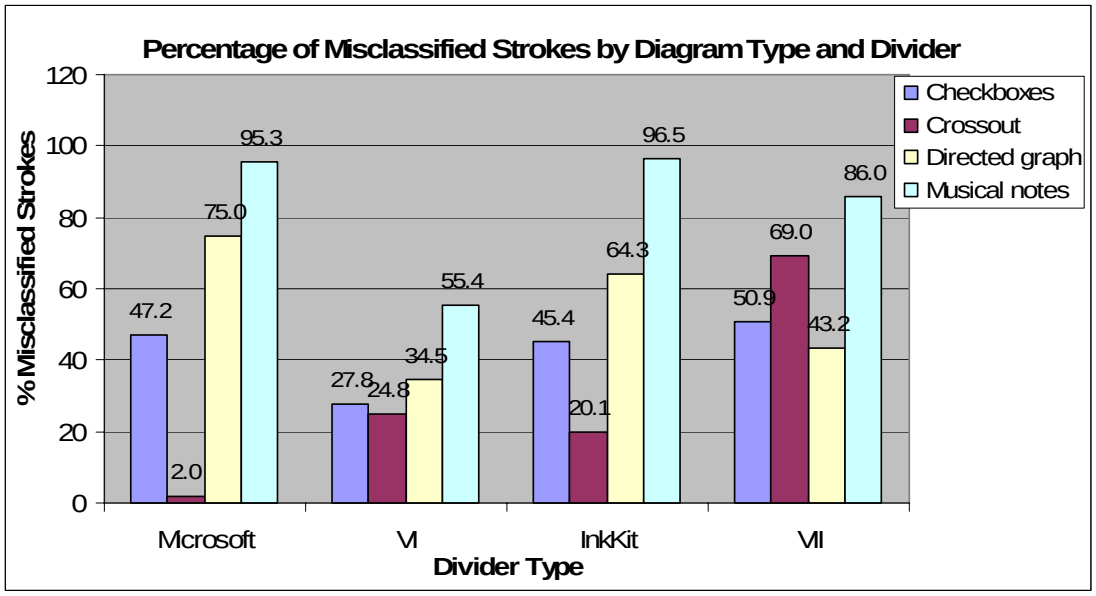


Figure 35. Percentage of Misclassified Strokes by Diagram Type and Divider for the New Diagram Set

7.2.7 Divider and Stroke Grouping

Figure 36 illustrates the performance of each divider when using a single, intersecting or close stroke grouping as a pre-processing technique. The Microsoft divider is not included here.

There does not appear to be a great difference in the performance of each divider when a different stroke grouping is used. For the Version I divider the single stroke grouping gives slightly worse classification results than the other two groupings. InkKit and the Version II divider are more accurate when a single stroke grouping is used. Overall the variation is not different enough to conclude that any stroke grouping in particular should be used as a pre-process to any of these dividers. This is similar to the results of the evaluation on the original diagram set.

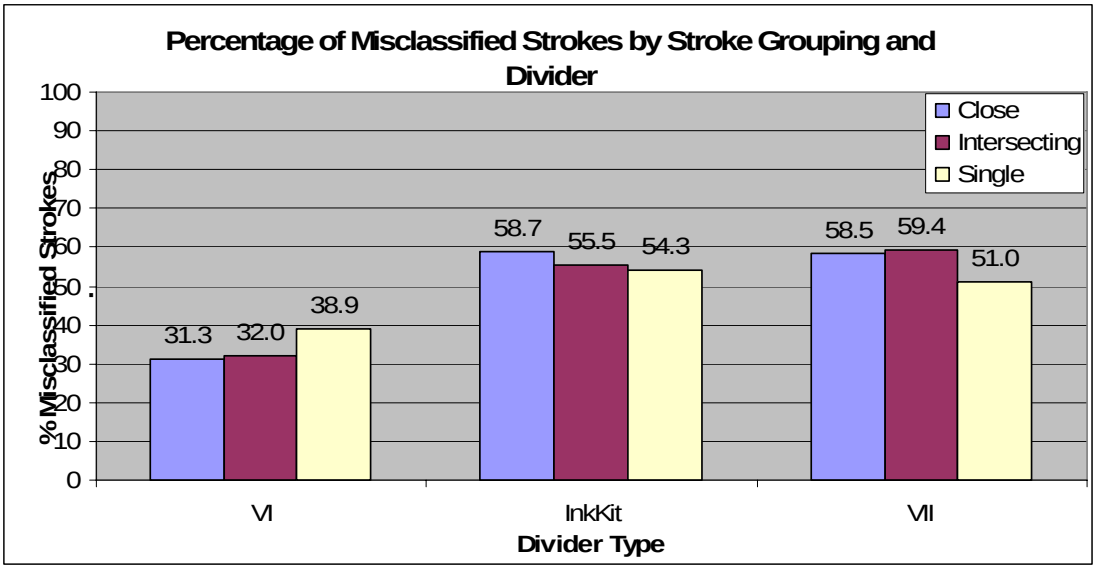


Figure 36. Percentage of Misclassified Strokes by Stroke Grouping and Divider for the New Diagram Set

7.2.8 Divider and Participants

Figure 37 shows the distribution of the percentage of strokes misclassified for each divider over all the participants. The participants are the people who drew the new diagram set that is collected for this evaluation. This indicates the effect of individual variation on each dividers result.

As with the original diagram set the Version I divider shows the smallest distribution of results therefore is least affected by individual variation. It also has the lowest standard deviation which further confirms these results shown in Table 36.

Also following the original diagram set results the Version II divider and InkKit show similarly large distributions of results showing that they are most affected by individual variation. Their standard deviation from Table 36 shows that the Version II divider has a slightly larger standard deviation therefore suggesting that this divider is most affected by individual variation.

Also observing the relative position of each box plot shows that all participants results when the Version I divider is used have a lower total misclassification rate than when all other dividers are used. This is apparent as the maximum value for the Version I divider is lower than the minimum values of all other dividers, further confirming the success of the Version I divider.

	% Misclassified Strokes			
Average	**Microsoft**	**Version I**	**InkKit**	**Version II**
Mean	57.5	34.2	55.7	55.5
Standard deviation	6.0	4.3	6.9	7.1

Table 36. Mean and Standard Deviation of Stroke Misclassification by Divider for each Participant for the New Diagram Set

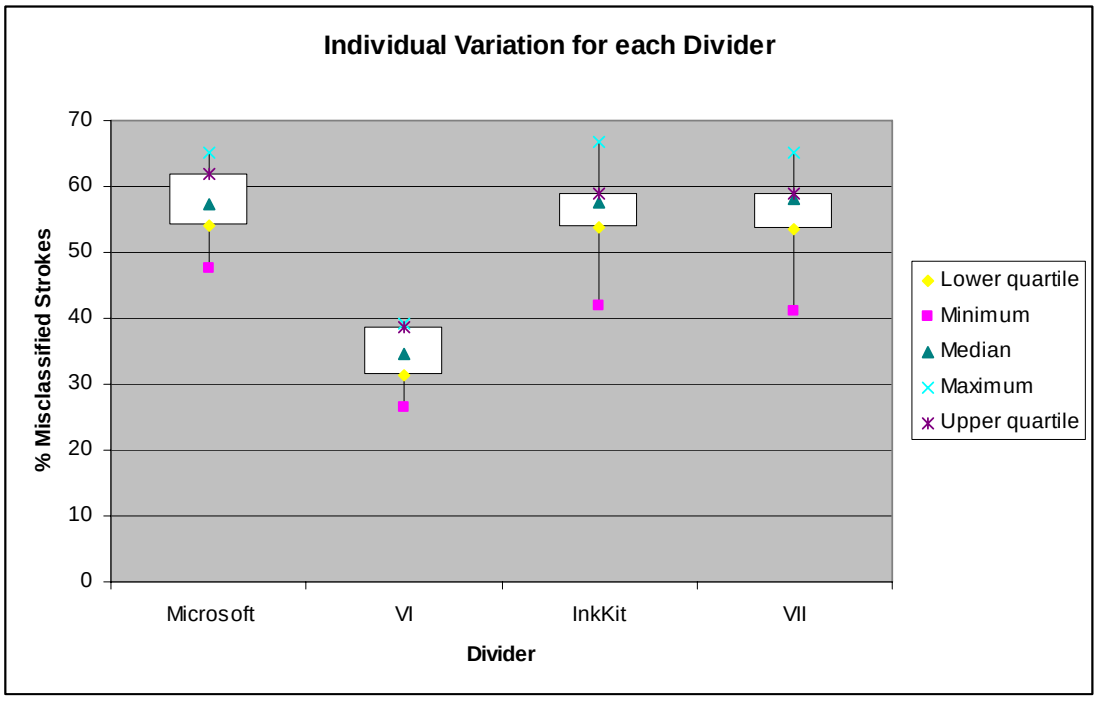


Figure 37. Individual Variation for each Divider for the New Diagram Set

The evaluations of the dividers on both diagram sets show similar results. The Version I divider is consistently the most accurate divider for both the original and new diagram sets. For the original diagram set the Version II divider is clearly the worst divider, but for the new diagram set the Version II, Microsoft and InkKit dividers have similarly high misclassification rates. Also the Version I divider also shows the least amount of variation in its performance classifying different diagram types and the smallest distribution in results indicating it is least affected by individual variation. Furthermore its performance is not affected significantly by stroke grouping. The Version I divider is the best divider for classifying shape and text strokes and we have met our goal of improving the accuracy of the divider to consequently improve sketch recognition.

Chapter 8

Discussion

The main objective of this research is to improve recognition of hand-drawn diagrams so that the benefits of designing on pen and paper are combined with the benefits of a computer. Pen and paper offer an informal environment where we can easily follow a creative process of expressing and refining ideas (Black 1990; Goel 1995). Tablet PC's allow users to sketch directly onto the screen with a stylus, imitating interaction with pen and paper. By recognising these sketches on a Tablet PC we are able to transform them into more formal looking diagrams and take advantage of the computers functionality such as easy editing and distribution. As well as formalising a diagram, recognition can allow sketches to be animated and executed providing an interaction level far above pen and paper (Davis, Agrawala et al. 2003; Thorne, Burke et al. 2004).

Most diagramming sketch tools have ignored the need for text recognition and concentrate solely on identifying shapes. However if we are to model the interaction that occurs when diagramming with pen and paper it is obvious that text recognition must be addressed as well, as writing is an important component of most diagrams. This is achieved with a modal interface such as that in Freeform however mode changes between writing and drawing does little to preserve human's natural interaction with pen and paper (Plimmer 2004). Another approach, which is evident in InkKit (Chung, Mirica et al. 2005; Young 2005; Plimmer, Tang et al. 2006; Freeman and Plimmer 2007), is to divide the strokes into probable text and shape strokes automatically before separate character and basic shape recognition. Young (2005) suggests that developing an accurate text and shape divider will be of great benefit to the overall accuracy of sketch recognition. Therefore we have focused our research on improving the existing methods of dividing text and shape strokes.

The literature review shows that sketch recognition is composed of two elements, *features* and *algorithms*. Features of strokes form the basic measures of the majority of recognition engines and algorithms are used to combine these features in a meaningful way so that strokes can be classified correctly. We concentrate on the first step of recognition by identifying features to distinguish between text and shape strokes successfully as a way to improve the divider.

If recognition is to be accurate, features that reflect the most important distinguishing characteristics of strokes must be identified. Most past research identify features simply based on observation and have no statistical basis of their significance to the recognition domain. We use a more rigorous approach for feature identification applying a formal statistical analysis technique to determine the features that are significant to classifying text and shape strokes. Here we discuss the results of the feature analysis followed by a discussion of the role of features to general sketch recognition and further improvements that can be made.

8.1. Feature Analysis

The first step of this research was to compile a feature set of as many possible stroke features that could contribute to a divider by looking at features used in previous work in sketch recognition and features that are available as a result of new hardware. Additional features were added based on our own observations and hypotheses. Forty six features appear in our features set, they measure aspects of stroke size, time, intersections, curvature, pressure and Tablet OS recognition values. Future analysis may include more features such as pen tilt when hardware becomes available. However we expect this is unlikely to be significant to division considering that pressure was not found to be significant.

The sketches collected of example diagrams were processed into a dataset for formal statistical analysis. The analysis resulted in a Version I and II classification tree that contain features to divide strokes. These dividers are implemented within InkKit

alongside the Microsoft Tablet OS divider and the existing InkKit divider for evaluation purposes.

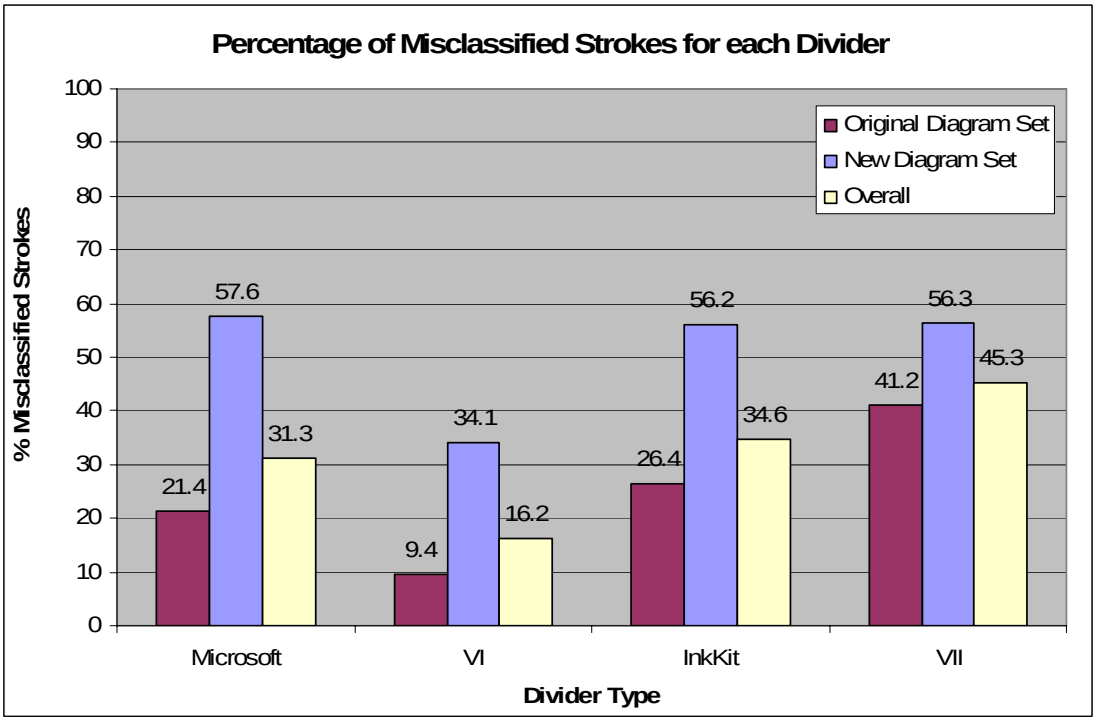


Figure 38. Total Percentage of Misclassified Strokes for each Divider on the Original and New Diagram Sets

The evaluation shows that the Version I divider is undoubtedly the best divider at classifying strokes. It has the lowest percentage overall of misclassified strokes when tested using both the original and new diagram sets at 9.4% and 34.1% of strokes misclassified respectively, with a misclassification rate of 16.2% overall as illustrated by Figure 38. The large difference in misclassification rates for each diagram set highlight the fact that this divider is much better at dividing the diagram types that it is trained for. In fact all dividers are considerably worse at dividing the new diagram set which also indicates the complexity of the diagrams in this set when compared with the original diagram set. However the fact that the Version I divider remains the most accurate divider when tested on the new diagram set shows that the features used by this divider provide an excellent foundation for general use in the division problem regardless of diagram type. Further training with the new diagram set may lead to better recognition rates.

The Version II divider evaluation results showed that it is consistently the worst at dividing text strokes and is considered to be the worst divider overall with a misclassification rate of 45.3% as illustrated by Figure 38. For the original diagram set it misclassifies 41.2% of strokes and for the new diagrams set 56.3% of misclassified strokes also illustrated by Figure 38. The dataset used to train it contains substituted values for missing data for strokes that do not have a previous or next stroke, i.e. the first or last stroke of a sketch. As these values do not originate from the actual strokes they do not represent the true nature of the strokes. It was better to prune the tree rather than warp the data.

The Version II tree was created to try to deal with features affected by missing data when strokes have no previous or next stroke; however it is worse than the Version I tree. An alternative approach that could be used in the future to deal with features that have missing data is to use surrogate features that are generated by the partitioning software (Venables and Ripley 2002). Features are identified that have similar characteristics to each feature in the tree. They are then used to replace the affected feature in the tree for the strokes that have missing data. For example if a stroke is the first stroke in a diagram it has no value for the distance from the last stroke, so a surrogate feature such as the distance to the next stroke could be used to replace that feature and allow the stroke to continue its path down the classification tree.

The Microsoft divider is best at classifying text strokes however this is only at the cost of having the highest misclassification rates for shape strokes, refer to Figure 30 and Figure 34 for these results. The Microsoft divider classifies most strokes as text regardless which explains why it has such low misclassification rates for text strokes and such high rates for shapes. Evaluation of InkKit's existing divider shows similar patterns to the Microsoft divider. This could be due to its use of the Microsoft divider results as one component of its features set.

This evaluation used only the first phase of InkKit's divider. The results indicate that this first phase does little to accurately classify shape and text strokes. Phase two of InkKit's

divider, which is a second iteration of division using the classification of nearest neighbours may be the part that makes the most difference to division reported by Young (2005). It is not included in the evaluation because the new dividers have concentrated on features. Future work will include researching a second iteration of division and evaluating its effectiveness.

An overall ranking of the dividers based on performance results illustrated by Figure 38 is as follows from the best divider to the worst:

1. Version I divider
2. Microsoft divider
3. InkKit divider
4. Version II divider

8.2. The Role of Features to Sketch Recognition

A formal statistical analysis has successfully identified significant features for a divider, implemented as the Version I divider. A rigorous approach such as this has not been used before to identify features, rather previous work has relied heavily on observation. The success of the Version I divider in comparison to the other dividers confirms the benefit of using such an approach. It shows that if the right features are identified, recognition can be improved significantly.

Comparing the Version I and II dividers, the Version II divider performs very poorly. The difference between these dividers is the set of features. Although the same feature analysis approach is used for both of the new dividers, the dataset for the Version II divider contains substituted data entered in for missing values. The fact that a divider based on such data has performed so poorly indicates the importance of using the correct data for analysis. Even though few features are affected by these substituted values, the impact on the divider's accuracy is huge. This suggests that careful feature selection is crucial to the success of a divider and sketch recognition in general as features form the basis of all components of recognition.

The evaluation shows that all the dividers consistently show higher misclassification rates for shapes strokes than text strokes (see Figure 30 and Figure 34 for these results). Future work could involve analysing the significance of additional features that bring out the characteristics of shapes. Features used by CALI (Fonseca and Jorge 2000; Fonseca, Pimentel et al. 2002) described in Table 9 may be useful for this as they concentrate on geometric features of shapes that may distinguish them from text. Some of these features are already part of the feature set; two of these are statistically significant and therefore used by the Version I divider, the bounding box width and perimeter to area of the convex hull. However more of CALI's features may help this aspect of division, they were not all included initially as they were considered very similar to features already part of the set.

The significant features have been identified and prove to be successful at division however this is only the first step of recognition, we now need to identify the best algorithm to use to meaningfully combine these features. The tree-based partitioning technique used to analyse the dataset of sketch features not only identifies feature sets for distinguishing text from shape strokes but also provides a method of utilising and combining those features in a practical way using a binary tree approach to classification.

Although the Version I classification tree works well at combining the significant features together to divide shape and text strokes, greater flexibility is needed in such an algorithm than what is provided by a tree structure. Sketching is imprecise in nature therefore recognition algorithms need to be robust enough to accommodate variability. The binary classification trees here use hard rules to classify observations which are based on a single dataset. There are no provisions made for factors such as individual variation or diagram domain unless these are covered by the training dataset.
The combination of the Version I dividers feature set and algorithms such as Hidden Markov Models (Rabiner 1990), Bayesian networks (Alvarado and Davis 2004; Liao, Wang et al. 2005) and fuzzy logic (Jorge and Fonseca 1999; Fonseca and Jorge 2000; Fonseca and Jorge 2001; Fonseca, Pimentel et al. 2002) needs investigation to determine the best algorithm for a more accurate divider.

Additional improvements can be made to the divider by considering the use of a second phase to the process whereby the classification of the nearest neighbour to the stroke is used to further confirm the current strokes classification. For example strokes are classified by the first phase of division as we have done here with the Version I divider. Then in the second phase if the nearest neighbour to the current stroke is classified as text by the first phase of the divider then the current strokes probability of also being a text stroke is strengthened as they are most probably part of the same word or sentence. InkKit's divider uses this concept as a second phase to the process; see Section 2.2 for more details on the method InkKit uses for this. Using a similar approach for the Version I divider may yield even better results.

The divider is only one part in the sketch recognition process. Further improvement to sketch recognition should be possible if the same rigorous approach of feature analysis is applied to the basic shape and component recognition phases. The identification of significant feature sets for each of these phases together with an evaluation of various algorithmic approaches to combining these features should produce more accurate recognition results as demonstrated by this investigation and therefore improve sketch recognition as a whole.

Nonetheless, a divider is a useful recognition tool for many applications other than diagrams such as document outlines and intelligent to-do lists. For all cases what is crucial is that the divider's performance is accurate and reliable.

Chapter 9

Conclusion and Future Research

This chapter presents a summary of this thesis and review the results in the context of the primary objective of improving sketch recognition through the improvement of the text and shape divider and concludes with suggestions for future work.

9.1. Conclusion

This thesis has investigated how current sketch recognition techniques for diagrams can be improved. It has focused on improving the division process of recognition where text and shape strokes are automatically divided into two groups to allow each to be recognised using separate algorithms.

The review of previous work in sketch recognition, in Chapter 2, found that very few diagramming sketch tools support recognition of shapes and text together. Tools that support both commonly use a modal interface, relying on the user to separate their diagrams into text and shapes while they draw. InkKit is a tool that performs stroke division automatically, allowing the user to freely sketch diagrams as they would with pen and paper without the distraction or interruption of a modal interface. The literature review also highlights that *features* of ink strokes and *algorithms* to combine the feature information serve as the core of recognition processes of most sketch tools. However, heuristics have been used to identify the best features and algorithms for recognition. As a feature set forms the first element of any recognition engine we chose to focus this investigation on identifying the stroke features that display the most distinguishing characteristics of text and shape strokes using a rigorous approach. Our hypothesis was

that identifying these features would provide a strong foundation for a more accurate divider.

The first step of this investigation consists of identifying a feature set of possible stroke features that could contribute to a shape and text divider. These features come from related work and include features available due to the development of new hardware that have not been considered previously and are also from our own thoughts. Forty six features are compiled, each measuring some aspect of stroke size, curvature, time, pressure, intersections and Tablet OS recognition values. Chapter 4 describes these features in detail.

Before the feature set is analysed to discover which of the 46 features are significant to performing text and shape division, data on each feature is collected from example sketches. Diagrams are collected from 26 participants, each sketching a diagram set of nine diagrams. Each diagram in the diagram set is chosen to represent common characteristics of diagrams including basic shapes and text, composite shapes, complex shapes and various combinations and orderings of text and shapes. Each diagram is processed to measure the 46 features identified by the feature set for each stroke in every diagram. These measurements are compiled together into a dataset of stroke feature measurements with 1519 stroke observations in total, 432 of these are shape strokes and 1087 are text strokes.

The dataset is analysed using a tree-based partitioning technique. The measurements of each feature in the dataset are used to find optimal partitions of the strokes into text and shapes. The most significant features are those that provide the most accurate partition of text and shape strokes with the least amount of overlap i.e. the least amount of misclassified strokes. A binary classification tree is constructed using the most significant features identified. The tree for the Version I divider is illustrated in Figure 19. Eight features out of the 46 that are in the feature set are found to be significant to dividing text and shape strokes. These features are measures of size, time and curvature.

The Version I divider is implemented within InkKit. Implementation of the divider involves some pruning of the tree to deal with special cases for some of the features. The special cases are to account for those strokes that do not have a previous or next stroke, i.e. they are the first, last or only stroke in the sketch. This is necessary as half of the features used by this divider require a stroke to have a previous or next stroke. In addition to pruning the trees to solve this problem, an alternative approach was considered where extreme values are substituted into the dataset for the missing feature values. This modified dataset is then re-analysed by the partitioning technique to identify significant features and produce a classification tree for a Version II divider illustrated in Figure 25. Six out of the 46 features in the feature set are found to be significant to division. These features come from the same categories as the Version I divider, size, time and curvature. Features measuring size appear to have the greatest contribution to the new dividers, then time and lastly curvature. Three features common to both Versions are bounding box width, the distance from the last stroke and the time till the next stroke.

Implementation of these dividers also prompted investigation of a pre-processing method of stroke grouping before passing the strokes to the dividers. Here strokes are grouped into single, intersecting or close stroke groupings to determine if pre-processing in this way benefits the accuracy of the dividers. The Version I, Version II and existing InkKit dividers are all implemented with the 3 options of stroke groupings within InkKit for evaluation.

The evaluation shows that overall the Version I divider is the most accurate at dividing the training set of diagrams. To further verify the results a new diagram set is compiled, to account for some diagram characteristics that were not included in the original set such as arrow heads and also to test some unique diagram types such as musical notes. Again the Version I divider performs considerably better than the other three dividers even though it is not trained to divide these diagram types. This suggests its superiority in accurately dividing a wide range of diagram types.

Evaluation results of the Version II divider for the original diagram set shows that this is the worst at classifying strokes. It has the most trouble classifying text strokes where other dividers show more misclassifications of shape strokes. For the new diagram set the Version II divider has very similar misclassification rates to the InkKit and Microsoft dividers overall and is equally as bad at classifying text as it is at shape strokes. The poor performance of this divider could be explained by the fact that it is trained with a dataset that uses made up values for missing data. It is not trained with data that exhibits the true nature of strokes and therefore is unsuccessful at recognising strokes based on real data.

The Microsoft divider consistently demonstrates gross misclassification of shape strokes, being the worst divider overall for shape strokes, and is the best at classifying text strokes. It classifies the majority of strokes as text which explains why it shows such extreme results. InkKit's existing divider shows similar behaviour to Microsoft's divider. Although not to the extreme, it has very high misclassification rates for shape strokes and low misclassification of text strokes. InkKit relies on the Microsoft divider for a component of its feature set which may be the cause of their similarity.

All dividers perform better at classifying text than shape strokes indicating a tendency to classify shapes as text a lot of the time.

When compared with the Version II, InkKit and Microsoft dividers, the Version I divider is far more accurate at dividing diagrams with an overall error rate of 16.2%. Therefore we conclude that the features selected through formal statistical analysis for the Version I divider have improved the accuracy of division of text and shape strokes in a sketched diagram. In doing so, the general recognition of hand drawn diagrams has been improved thus meeting the primary objective of this thesis.

The contributions of this research are:

- The identification of a feature set of distinguishing features to divide shapes and text from the Version I divider. These are listed in Table 37. They measure aspects of stroke size, curvature and time.

- The advantage gained from use of a formal statistical method to identify these features.
- Division of text and shape strokes has improved by way of our new divider, Version I. The Version I divider uses the identified feature set in Table 37 to divide shape and text strokes more accurately than the Microsoft, InkKit and Version II dividers with an error rate of 16.2% as opposed to an average error rate of 37.1% for the other dividers.
- Sketch recognition techniques have therefore improved through the use of a more accurate divider.

Category	Version I Features
Size	Bounding box width Distance from last stroke Distance to next stroke Amount of ink inside Perimeter to area
Time	Time till next stroke Speed till next stroke
Curvature	Total angle

Table 37. Significant Feature Set

9.2. Future Research

There are several directions for further research in the area of sketch recognition in the future described here.

The classification tree approach has worked well at combining the significant features together for more accurate division of strokes into character text and geometric shapes. However the use of trees does not provide the level of flexibility in a divider that is essential for sketch recognition. Hand drawn diagrams are more often than not imprecise representations of the true figures that are intended by the user. For example lines are not perfectly straight or strokes that are meant to be intersecting to form a corner of a shape do not always meet completely. Therefore sketch recognition must allow for the rough nature of sketches by providing robust algorithms, rather than hard rules such as those used by a classification tree. Future work for the divider could involve experimenting

with other techniques to combine the identified feature set such as using Hidden Markov Models (Rabiner 1990), Bayesian networks (Alvarado and Davis 2004; Liao, Wang et al. 2005) and fuzzy logic (Jorge and Fonseca 1999; Fonseca and Jorge 2000; Fonseca and Jorge 2001; Fonseca, Pimentel et al. 2002) for a more robust divider.

Further development of the features employed by the divider can be investigated. Additional features may be tested such as pen tilt when hardware becomes available, and analysis of more of CALI's features which concentrate on geometric shape characteristics may improve shape stroke classification as all the dividers show lower recognition rates for shapes than for text. They were not included initially as they were considered very similar to features already part of the set. The divider can also be trained with new diagram examples such as those that are in the new diagram set in Table 35. An alternative approach to handling the missing data, when strokes have no previous or next stroke information, may be investigated. Surrogate features generated by the partitioning software (Venables and Ripley 2002) can be used in place of features that can not be calculated due to this missing data as discussed in Chapter 8. As an extension to the Version I divider, features using the classification of close strokes could be incorporated as a second iteration of the classification process, much like phase 2 of the InkKit divider to take advantage of the concept that the classification of surrounding strokes can further strengthen the current strokes classification.

There is a question as to whether a general feature set to divide all diagram types ranging from simple organisation charts to musical notes will ever be able to reach the levels of accuracy that is required, although the Version I divider shows very promising results on a large range of diagrams shown by the evaluation results using the divider on the new diagram set. Would it be better to use domain knowledge to guide division? Could we employ algorithms to learn which features are the most relevant for a particular domain? Should users be able to tailor the divider to their own needs by selecting some of the features, for example a physically impaired user for which movement is very slow and therefore features involving time must be tuned to allow for this. Also how well will a general feature set be able to apply to different sized screens that can range from hand-

held devices such as PDA's to large screen displays such as electronic whiteboards considering that size is an important component of this feature set. These are all questions to be answered by future work.

There is a great deal of improvement that can be made in areas of sketch recognition other than the divider such as basic shape and domain specific component recognition. There is a need for a definitive set of features to be identified for each of the various stages of recognition and also to identify the most effective algorithmic approach to using these features. The choice of features is critical to the success of the recognition, yet heuristics have the basis of selection. While various algorithms have been tried, there have been no studies identified scientifically comparing different algorithms. Future projects can systematically examine each stage of sketch recognition, scientifically analysing the features employed and comparing different algorithmic approaches to using these features. The expected result is to demonstrate the effectiveness of the identified optimal recognition techniques by allowing more accurate recognition of sketched diagrams.

It is essential that diagramming sketch tools be able to provide the most accurate and reliable recognition possible with minimal constraints and flexibility to match human interaction with pen and paper. Only then will these tools be able to provide their potential benefit to users.

Appendix A:

Code used for Feature

Measurement

```
.....inclusion of various packages....code omitted..

public class FeatureCalculations
{
  public static void CollectData(Strokes strokes, Guid theTimeGuid)
  {
    ...initialisation of list variables.......code omitted...

    //timestamp in milliseconds for each stroke
    List<string> timeStamp = CollectTimeData(strokes, theTimeGuid);

    //initialising microsoft text recognition
    RecognizerContext theRecognizerContext =new RecognizerContext();
    Microsoft.Ink.RecognitionResult theRecognitionResult;
    Microsoft.Ink.RecognitionStatus theRecognitionStatus;

    //various counters
    int distCounter = 1;
    int timeCounter = 0;

    foreach (Stroke stroke in strokes)
    {
      #region Pressure calculations
      //Deal with pressure measurements first
      List<int> tempPressure = new List<int>();
      bool fFound = false;
      for (int i = 0; i < stroke.PacketDescription.Length; i++)
      {
        if(stroke.PacketDescription[i]==PacketProperty.NormalPressure)
        {
          fFound = true;
          break;
        }
      }
      if (fFound)
      tempPressure.AddRange(stroke.GetPacketValuesByProperty(PacketPrope
      rty.NormalPressure));

      //working out avg, min, max Pressure for the stroke
      double avgPressure;
      int minPressure, maxPressure;
      if (tempPressure.Count > 0)
      {
```

```
    avgPressure = 0.0;
    minPressure = tempPressure[0];
    maxPressure = tempPressure[0];
    foreach (int i in tempPressure)
    {
      avgPressure += i;
      if (i > maxPressure) maxPressure = i;
      if (i < minPressure) minPressure = i;
    }
    avgPressure = avgPressure / tempPressure.Count;
    // pressure values for each point in the stroke
    pressureValues.AddRange(tempPressure);
    // maximum, minimum and average pressure for the stroke
    pressureMax.Add(maxPressure.ToString());
    pressureMin.Add(minPressure.ToString());
    pressureAvg.Add(avgPressure.ToString());
     int pressureCount = 0;
     bool minFound= false;
     //work out pressure variation
     for (int i = 1; i < tempPressure.Count; i++)
     {
        if ((tempPressure[i] - tempPressure[i - 1]) < 0)
        {
            minFound = true;
        }
        else if((tempPressure[i]-tempPressure[i-1])>0&&minFound){
            pressureCount++;
            minFound = false;
        }
     }
     pressureVariation.Add(pressureCount); // # of pressure minima
}
 #endregion

 #region Extract x and y coordinates of the stroke
 List<int> tempX = new List<int>();
 List<int> tempY = new List<int>();
 //x,y coordinates of each stroke point
 tempX.AddRange(stroke.GetPacketValuesByProperty(PacketProperty.X))
 tempY.AddRange(stroke.GetPacketValuesByProperty(PacketProperty.Y))
 #endregion

 #region Time measures
 List<int> tempTicks = new List<int>();

 tempTicks.AddRange(stroke.GetPacketValuesByProperty(PacketProperty
 .TimerTick));
 List<string> tempRealTime = new List<string>();

 //calulate real time values
 foreach (int i in tempTicks)
 {
   Doubletemp=(Convert.ToDouble((string)(timeStamp[timeCounter])))+
   (double)(i)*0.0001;
   tempRealTime.Add(temp.ToString());
 }
   timeCounter++;
```

```
    //calculating time from last stroke and time till next stroke
    if (timeCounter == 1)    //no previous stroke, this is the first
        timeFrom.Add("na"); //time from last stroke
    else if (timeCounter == strokes.Count)
    {
                double temp =
(Convert.ToDouble((string)(tempRealTime[0]))) -
(Convert.ToDouble((string)(realTime[realTime.Count - 1])));
        timeTo.Add(temp.ToString());
        timeFrom.Add(temp.ToString());
        timeTo.Add("na");
    }
    else
    {
                double temp =
(Convert.ToDouble((string)(tempRealTime[0]))) -
(Convert.ToDouble((string)(realTime[realTime.Count - 1])));
        timeTo.Add(temp.ToString());
        timeFrom.Add(temp.ToString());
    }

    if (strokes.Count == 1)
        timeTo.Add("na");
    //total time
    double t = (double)(tempTicks[tempTicks.Count - 1]) * 0.0001;
    totalTime.Add(t.ToString());    //total duration of the stroke

    #endregion

    //length of stroke
    lengths.Add(MyUtils.GetStrokeLength(stroke).ToString());
    //bounding box width and height
    bboxHeights.Add(stroke.GetBoundingBox().Height);
    bboxWidths.Add(stroke.GetBoundingBox().Width);

    // location of self intersections, bezier cusps, polyline cusps and
    strokes intersecting
    selfIntersections.AddRange(stroke.SelfIntersections);
    bezierCusps.AddRange(stroke.BezierCusps);
    polylineCusps.AddRange(stroke.PolylineCusps);
    intersections.AddRange(stroke.FindIntersections(strokes));

    // # of self intersections, bezier cusps, polyline cusps and
    strokes intersecting
    selfInterCount.Add(stroke.SelfIntersections.Length / 2);
    bezierCount.Add(stroke.BezierCusps.Length);
    polylineCount.Add(stroke.PolylineCusps.Length);
     interCount.Add(stroke.FindIntersections(strokes).Length -
(stroke.SelfIntersections.Length/2));

    int endPoint = 0;
    if (ShapeBoundsFactory.IsShapeEndsClose(stroke))
      endPoint++;

      //# endpoint self intersections and non endpoint selfintersections
      endPointSelf.Add(endPoint);
      midPointSelf.Add(selfInterCount[selfInterCount.Count-1]-endPoint);
```

```
        //raw data of stroke timestamps, timer ticks for each stroke point
        realTime.AddRange(tempRealTime);
        timerTicks.AddRange(tempTicks);

        //raw data of x,y coordinates of stroke
        xCoordinates.AddRange(tempX);
        yCoordinates.AddRange(tempY);

        #region Distance calculations
        List<string> tempDist = new List<string>();
        double maxDistance = 0;

        //and calculate distance
        for (int i = distCounter; i < xCoordinates.Count; i++)
        {
          double temp = Math.Sqrt((double)(xCoordinates[i]-xCoordinates[i-
1]) * (double)(xCoordinates[i] - xCoordinates[i - 1]) +
(double)(yCoordinates[i] - yCoordinates[i - 1]) *
(double)(yCoordinates[i] - yCoordinates[i - 1]));
          tempDist.Add(temp.ToString());
          //find the maximum distance
          if (temp > maxDistance && tempDist.Count > 1)
            maxDistance = temp;
        }
        distCounter = xCoordinates.Count;

        //calculate distance from last stroke and to next stroke
        if (timeCounter == 1)    //no previous stroke, this is the first
          distFrom.Add("na"); //distance from last stroke
        else if (timeCounter == strokes.Count)
        {
           distTo.Add(tempDist[0]);
           distFrom.Add(tempDist[0]);
           distTo.Add("na");
        }
        else
        {
           distTo.Add(tempDist[0]);
           distFrom.Add(tempDist[0]);
        }
        if (strokes.Count == 1)
          distTo.Add("na");
        distance.AddRange(tempDist);
        #endregion

        #region Stroke speed calculations
        List<string> tempSpeed = new List<string>();
        //does not calculate speed between strokes
        if (timeCounter == 1)
        {   //for the first stroke, special case
         for (int i = 0; i < tempDist.Count; i++)
         {
                    double time1 =
(Convert.ToDouble((string)(tempRealTime[i + 1])) -
Convert.ToDouble((string)(tempRealTime[i])));
          if (time1 != 0)
```

```
          {
            double s = Convert.ToDouble((string)(tempDist[i])) / time1;
            tempSpeed.Add(s.ToString());
          }
        }
      }
      else
      {  //for all following strokes
        for (int i = 1; i < tempDist.Count; i++)
        {
          double time1 = (Convert.ToDouble((string)(tempRealTime[i])) -
Convert.ToDouble((string)(tempRealTime[i - 1])));
          if (time1 != 0)
          {
              double s=Convert.ToDouble((string)(tempDist[i]))/time1;
              tempSpeed.Add(s.ToString());
          }
        }
      }

      double avgSpeed = 0.0;
      double minSpeed = 0.0;
      double maxSpeed = 0.0;

      if (tempSpeed.Count > 0)
      {
        avgSpeed = 0.0;
        minSpeed = Convert.ToDouble((string)(tempSpeed[0]));
        maxSpeed = Convert.ToDouble((string)(tempSpeed[0]));
        foreach (string i in tempSpeed)
        {
          double sI = Convert.ToDouble(i);
          avgSpeed += sI;
          if (sI > maxSpeed) maxSpeed = sI;
          if (sI < minSpeed) minSpeed = sI;
        }
        avgSpeed = avgSpeed / tempSpeed.Count;

        int speedCount = 0;
        bool minFound = false;
        List<double> movingAvg = new List<double>();
        for (int i = 2; i < tempDist.Count-2; i++)
        {
          movingAvg.Add((Convert.ToDouble((string)(tempDist[i-2]))+
Convert.ToDouble((string)(tempDist[i - 1])) +
Convert.ToDouble((string)(tempDist[i])) +
Convert.ToDouble((string)(tempDist[i + 1])) +
 Convert.ToDouble((string)(tempDist[i + 2])))/5.0);
        }
         //work out speed variation based on distance
         for (int i = 1; i < movingAvg.Count; i++)
         {
           double temp1 = movingAvg[i];
           double temp2 = movingAvg[i - 1];
           if ((temp1 -temp2) < 0 && (temp1/maxDistance) < 0.20)
           {
               minFound = true;
```

```
                }
                else if ((temp1 - temp2) > (0.05*maxDistance) && minFound)
                {
                    speedCount++;
                    minFound = false;
                }
            }
            speedVariation.Add(speedCount);
        }

        speed.AddRange(tempSpeed);  //raw speed data for each stroke point
        speedMax.Add(maxSpeed.ToString());  //maximum stroke speed
        speedMin.Add(minSpeed.ToString());  //minimum stroke speed
        speedAvg.Add(avgSpeed.ToString());  //average stroke speed

        #endregion

        #region OS text probability
        theRecognizerContext.Strokes=stroke.Ink.CreateStrokes(new int[]{
stroke.Id });
        theRecognitionResult=theRecognizerContext.Recognize(out
theRecognitionStatus);
        OSprob.Add(theRecognitionResult.TopConfidence.ToString());
        OSalternative.Add(theRecognitionResult.TopString);
        #endregion

        #region Amount of ink inside the bounding box
        int numIntersections = 0;
        int numPoints = stroke.GetPoints().Length;
        System.Drawing.Rectangle rect1 = stroke.GetBoundingBox();
        rect1.Inflate(rect1.Width / 5, rect1.Height / 5);

        StrokeIntersection[] intersections1=
stroke.GetRectangleIntersections(rect1);
        foreach (StrokeIntersection intersection in intersections1)
        {
          int beginIndex = (intersection.BeginIndex != -1 ?
(int)intersection.BeginIndex : 0);
          int endIndex = (intersection.EndIndex != -1 ?
(int)intersection.EndIndex : numPoints - 1);
          numIntersections += (endIndex - beginIndex + 1);
        }
        inkInside.Add(numIntersections);
        #endregion

        #region Feature 1 & 2: Cos & Sin of Initial Angle

        System.Drawing.Point[] points = stroke.GetPoints();
        System.Drawing.Point point1, point2, point3;

        point1 = points[0];
        point2 = (points.Length > 1 ? points[2] : points[0]);

        double tempDouble = MyUtils.ComputeDistance(point1, point2);

        if (tempDouble == 0)
        {
```

```
            feature1cos.Add("0");
            feature2sin.Add("0");
        }
        else
        {
            feature1cos.Add(((point2.X-point1.X)/tempDouble).ToString());
            feature2sin.Add(((point2.Y-point1.Y)/tempDouble).ToString());
        }
        #endregion

        #region Feature 3: Length of the Gesture

        // Calculate the maximum and minimum points.
        double totalGestureLength = 0;
        double minX, maxX, minY, maxY;

        point1 = points[0];

        minX = point1.X;
        maxX = point1.X;
        minY = point1.Y;
        maxY = point1.Y;
        for (int i = 1; i < points.Length; i++)
        {
          if (i < points.Length - 1)
          {
            point2 = points[i + 1];
            totalGestureLength+=MyUtils.ComputeDistance(point1, point2);
            point1 = point2;
          }
          // Max / Min Points
          if (points[i].X < minX)
          {
            minX = points[i].X;
          }
          else if (points[i].X > maxX)
          {
            maxX = points[i].X;
          }
          if (points[i].Y < minY)
          {
            minY = points[i].Y;
          }
          else if (points[i].Y > maxY)
          {
            maxY = points[i].Y;
          }
        }

    double boundingBoxSize=(MyUtils.ComputeDistance(minX,minY,maxX,maxY));
        feature3bboxLength.Add(boundingBoxSize.ToString());
        #endregion

        #region Feature 4: Angle of Bounding Box
        if ((maxX - minX) == 0)
        {
          feature4bboxAngle.Add("0");
```

```
      }
      else
      {
 feature4bboxAngle.Add((Math.Atan((maxY-minY)/(maxX-minX))).ToString());
      }
      #endregion

 #region Features 5,6,7: Distance, Cos and Sin between 1st & last points
      point1 = points[0];
      point2 = points[points.Length - 1];
      tempDouble = MyUtils.ComputeDistance(point1, point2);
      if (boundingBoxSize == 0)
      {
          feature5dist.Add("0");
      }
      else
      {
        feature5dist.Add((tempDouble / totalGestureLength).ToString());;
      }
      if (tempDouble == 0)
      {
        feature6cosFL.Add("0.707106");
        feature7sinFL.Add("0.707106");
      }
      else
      {
        feature6cosFL.Add(((point2.X-point1.X)/tempDouble).ToString());
        feature7sinFL.Add(((point2.Y-point1.Y)/tempDouble).ToString());
      }
      #endregion

      #region Feature 8: Total Gesture Length
feature8totalLength.Add((totalGestureLength/boundingBoxSize).ToString())
      #endregion

#region Features 9,10,11: Total Angle, Sum of Absolute & squared angles
      double angleSum, angleAbsSum, angleSqrSum;
      angleSum = 0;
      angleAbsSum = 0;
      angleSqrSum = 0;

      point1 = points[0];
      if (points.Length > 0)
      {
        point2 = points[1];
      }

      for (int i = 1; i < points.Length - 1; i++)
      {
         point3 = points[i + 1];

         tempDouble = (
             (
                 (
                 (point3.X - point2.X) * (point2.Y - point1.Y)
                 )
                 - (
```

```
                    (point2.X - point1.X) * (point3.Y - point2.Y)
                    ))
                    /
                (double)(
                (
                (point3.X - point2.X) * (point2.X - point1.X)
                )
                    + (
                    (point3.Y - point2.Y) * (point2.Y - point1.Y)
                    ))
                );
            tempDouble = Math.Atan(tempDouble);
            if (Double.IsNaN(tempDouble))
            {
                tempDouble = 0;
            }

            angleSum += tempDouble;
            angleAbsSum += Math.Abs(tempDouble);
            angleSqrSum += Math.Pow(tempDouble, 2);

            point1 = point2;
            point2 = point3;
        }

        feature9totalAngle.Add(angleSum.ToString());
        feature10absAngle.Add(angleAbsSum.ToString());
        feature11sqrAngle.Add(angleSqrSum.ToString());

        #endregion

        #region Features 12: Ratio of Perimeter to Area of Convex Hull
        MyConvexHulls convexHull = new MyConvexHulls(points);
        feature12perimeter.Add((convexHull.GetConvexArea()/
convexHull.GetConvexPerimeter()).ToString());
        #endregion

#region Features 13: Ratio of Area of Convex Hull to Area of Enclosing
Rectangle
        MyConvexHulls.RotatableRectangle rotRect =
convexHull.GetEnclosingRectangle();
        feature13area.Add((convexHull.GetConvexArea() /
rotRect.Area).ToString());
        #endregion

        #region Feature 1: Ratio of width to Height
        System.Drawing.Rectangle boundingBox = stroke.GetBoundingBox();
        feature1aRatioWH.Add(((double)boundingBox.Width /
boundingBox.Height).ToString()); ;
        #endregion

        #region Feature 1b:Ratio of width to height of enclosing rectangle

        if (rotRect.Rectangle.Width > rotRect.Rectangle.Height)
        {
         feature1bRatioEn.Add(((double)rotRect.Rectangle.Width /
rotRect.Rectangle.Height).ToString());
```

```
      }
      else
      {
        feature1bRatioEn.Add(((double)rotRect.Rectangle.Height /
rotRect.Rectangle.Width).ToString());
      }
      #endregion
  }

  #region Speed between strokes
  //work out speed between strokes -speed from previous and speed to
next stroke
  for (int i = 0; i < distFrom.Count; i++)
  {
      if ((timeFrom[i]).Equals("na") || (distFrom[i]).Equals("na"))
          speedFrom.Add("na");
      else
      {
        double xx = Convert.ToDouble((string)(distFrom[i])) /
Convert.ToDouble((string)(timeFrom[i]));
        speedFrom.Add(xx.ToString());
      }

      if ((timeTo[i]).Equals("na") || (distTo[i]).Equals("na"))
        speedTo.Add("na");
      else{
        double xx = Convert.ToDouble((string)(distTo[i])) /
Convert.ToDouble((string)(timeTo[i]));
        speedTo.Add(xx.ToString());
      }
  }
  #endregion

  #region # of Other Intersections excludig self intersections
  for (int i = 0; i < strokes.Count; i++) {
    otherInter.Add(Math.Abs(interCount[i] - selfInterCount[i]));
  }
  #endregion

  #region Number of strokes that intersect this stroke (including and
excluding self intersections)
  for (int i = 0; i < strokes.Count; i++ )
  {
     int self = 0;
     int other = 0;
     for (int j=0; j<strokes.Count; j++)
     {
                 if (i==j &&
(strokes[i].FindIntersections(strokes[j].Ink.CreateStrokes(new int[] {
strokes[j].Id }))).Length > 0)
                   self++;
                 else if (i != j &&
(strokes[i].FindIntersections(strokes[j].Ink.CreateStrokes(new int[] {
strokes[j].Id }))).Length > 0)
                   other++;
     }
     numStrokesOther.Add(other);
```

```
      numStrokesTotal.Add(self + other);
   }
   #endregion
     ////then write all data to spreadsheet.....code omitted.....
   }

     //to get the timestamp for each stroke
    private static List<string> CollectTimeData(Strokes strokes, Guid
theTimeGuid){

       List<string> time = new List<string>();
       foreach (Stroke theStroke in strokes)
       {
          // Test for the timestamp property on this stroke.
          if(theStroke.ExtendedProperties.DoesPropertyExist(theTimeGuid))
          {
             // Get the raw data out of this stroke's extended
             // properties list, using the previously defined
             // Guid as a key to the required extended property.
            long theLong =
(long)theStroke.ExtendedProperties[theTimeGuid].Data;
            // Then turn it (as a FileTime) into a time string.

           time.Add(DateTime.FromFileTime(theLong).TimeOfDay.TotalMillis
           econds.ToString());
           }
       }
       return time;
    }
}
```

Appendix B:

R Code used for Feature Analysis

Decision Tree: Version I

```
## Analysis Using The Original Data
## I.e. With "uncorrected" data values.
##
## Analysis uses the "rpart" package.

require(rpart)
[1] TRUE

## Graphics Setup

pdf(file = "Analysis1-fig%d.pdf", width = 5, height = 4,
+     onefile = FALSE, paper = "special", pointsize = 9)

## Save the original data set as csv using OpenOffice.
## Convert all "na" strings to "NA".
## Convert a single "Infinity" to "Inf".

data = read.csv("data.csv")

## Analysis omits variables:
##
##    Person, Stroke, Stroke.ID, Stroke.before, Stroke.After,
##    OS.alternative.
##
## The last of these was omitted because it is a factor with
## too many levels.

z = rpart(Type ~ ., data = data[-c(1, 3:6, 34)])

## Examine misclassification rates for pruning.

plotcp(z)

## Prune, using the 1-SE rule.
## The .015 magic number comes from the plot above.
```

```
zz = prune(z, cp = 0.015)
printcp(zz)
Classification tree:
rpart(formula = Type ~ ., data = data[-c(1, 3:6, 34)])

Variables actually used in tree construction:
[1] amount.of.ink.inside      bbox.width
[3] Distance.from.last.stroke Distance.to.next.stroke
[5] perimeter.to.area         Speed.to.next.stroke
[7] time.till.next.stroke     total.angle

Root node error: 432/1519 = 0.2844

n= 1519

        CP nsplit rel error  xerror     xstd
1 0.490741      0   1.00000 1.00000 0.040700
2 0.083333      1   0.50926 0.54167 0.032568
3 0.069444      2   0.42593 0.48843 0.031202
4 0.032407      3   0.35648 0.40741 0.028876
5 0.024306      4   0.32407 0.41204 0.029018
6 0.018519      7   0.25000 0.37269 0.027772
7 0.015000      8   0.23148 0.33565 0.026510

## Final Tree

print(zz)
n= 1519

node), split, n, loss, yval, (yprob)
      * denotes terminal node

  1) root 1519 432 text (0.28439763 0.71560237)
    2) bbox.width>=1847.5 304  46 shape (0.84868421 0.15131579)
      4) total.angle< 10.09914 272  15 shape (0.94485294 0.05514706) *
      5) total.angle>=10.09914 32   1 text (0.03125000 0.96875000) *
    3) bbox.width< 1847.5 1215 174 text (0.14320988 0.85679012)
      6) Distance.from.last.stroke>=2553.684 86  25 shape (0.70930233 0.29069767)
       12) time.till.next.stroke>=660.8878 64   7 shape (0.89062500 0.10937500) *
       13) time.till.next.stroke< 660.8878 22   4 text (0.18181818 0.81818182) *
      7) Distance.from.last.stroke< 2553.684 1129 113 text (0.10008857 0.89991143)
       14) Distance.to.next.stroke>=1646.576 140  55 text (0.39285714 0.60714286)
         28) Speed.to.next.stroke< 3.18416 81  30 shape (0.62962963 0.37037037)
           56) amount.of.ink.inside>=51.5 42   5 shape (0.88095238 0.11904762) *
           57) amount.of.ink.inside< 51.5 39  14 text (0.35897436 0.64102564)
            114) perimeter.to.area< 25.31871 14   3 shape (0.78571429 0.21428571) *
```

```
          115) perimeter.to.area>=25.31871 25   3 text (0.12000000 0.88000000) *
        29) Speed.to.next.stroke>=3.18416 59   4 text (0.06779661 0.93220339) *
      15) Distance.to.next.stroke< 1646.576 989  58 text (0.05864510 0.94135490) *
plot(zz)
text(zz, xpd = T, cex = 0.5)
```

Decision Tree: Version II

```
## Analysis Using The Corrected Data
## I.e. With filled-in data values.
##
## Analysis uses the "rpart" package.

require(rpart)
[1] TRUE

## Graphics Setup

pdf(file = "Analysis2-fig%d.pdf", width = 5, height = 4,
+     onefile = FALSE, paper = "special", pointsize = 9)

## Save the original data set as csv using OpenOffice.
## Convert all "na" strings to "NA".
## Convert a single "Infinity" to "Inf".

data = read.csv("ndata.csv")

## Analysis omits variables:
##
##    Person, Stroke, Stroke.ID, Stroke.before, Stroke.After,
##    OS.alternative.
##
## The last of these was omitted because it is a factor with
## too many levels.

z = rpart(Type ~ ., data = data[-c(1, 3:6, 34)])

## Examine misclassification rates for pruning.

plotcp(z)

## Prune, using the 1-SE rule.
## The .016 magic number comes from the plot above.

zz = prune(z, cp = 0.016)
```

```
printcp(zz)

Classification tree:
rpart(formula = Type ~ ., data = data[-c(1, 3:6, 34)])

Variables actually used in tree construction:
[1] bbox.width              Distance.from.last.stroke
[3] Speed.from.last.stroke  time.till.next.stroke
[5] X.polyline.cusps

Root node error: 432/1519 = 0.2844

n= 1519

        CP nsplit rel error  xerror     xstd
1 0.550926      0   1.00000 1.00000 0.040700
2 0.155093      1   0.44907 0.46065 0.030440
3 0.037037      2   0.29398 0.31019 0.025587
4 0.021991      3   0.25694 0.30093 0.025238
5 0.020833      5   0.21296 0.28009 0.024428
6 0.016000      6   0.19213 0.25463 0.023382

## Final Tree

print(zz)
n= 1519

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 1519 432 text (0.28439763 0.71560237)
   2) Distance.from.last.stroke>=1842.821 458 110 shape (0.75982533 0.24017467)
     4) time.till.next.stroke>=645.9118 359  27 shape (0.92479109 0.07520891) *
     5) time.till.next.stroke< 645.9118 99  16 text (0.16161616 0.83838384)
      10) bbox.width>=1861 15   3 shape (0.80000000 0.20000000) *
      11) bbox.width< 1861 84   4 text (0.04761905 0.95238095) *
   3) Distance.from.last.stroke< 1842.821 1061  84 text (0.07917059 0.92082941)
     6) Speed.from.last.stroke< 0.1815911 32   8 shape (0.75000000 0.25000000) *
     7) Speed.from.last.stroke>=0.1815911 1029  60 text (0.05830904 0.94169096)
      14) bbox.width>=1680 75  31 text (0.41333333 0.58666667)
        28) X.polyline.cusps< 4.5 39  10 shape (0.74358974 0.25641026) *
        29) X.polyline.cusps>=4.5 36   2 text (0.05555556 0.94444444) *
      15) bbox.width< 1680 954  29 text (0.03039832 0.96960168) *
plot(zz)
text(zz, xpd = T, cex = 0.5)
```

References

Alvarado, C. and R. Davis (2004). SketchREAD: a multi-domain sketch recognition engine. Proceedings of the 17th annual ACM symposium on User interface software and technology, Santa Fe, NM, USA, ACM Press.

Anonymous. (2007). "MSDN Stroke.BezierCusps Property." from http://msdn2.microsoft.com/en-us/library/microsoft.ink.stroke.beziercusps.aspx.

Anonymous. (2007). "MSDN Stroke.PolylineCusps Property." from http://msdn2.microsoft.com/en-us/library/microsoft.ink.stroke.polylinecusps.aspx.

Bailey, B. P. and J. A. Konstan (2003). Are Informal Tools Better? Comparing DEMAIS, Pencil and Paper, and Authorware for Early Multimedia Design. CHI 2003, Ft Lauderdale, ACM.

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

Breiman, L., J. H. Friedman, et al. (1984). Classification and Regression Trees. New York, Chapman & Hall / CRC Press.

Calhoun, C., T. F. Stahovich, et al. (2002). Recognising Multi-Stroke Symbols. AAAI Spring Symposium on Sketch Understanding.

Chen, Q., J. Grundy, et al. (2003). An E-whiteboard application to support early design-stage sketching of UML diagrams. Human Centric Computer Languages and Environments, Auckland, NZ, IEEE.

Chesnut, C. (2002). "Ink in Internet Explorer, Pen Stroke Info, errata." 2006, from http://www.mperfect.net/tabletInk/.

Chung, R., P. Mirica, et al. (2005). InkKit: A Generic Design Tool for the Tablet PC. CHINZ 05, Auckland, ACM.

Coyette, A., S. Faulkner, et al. (2004). SketchiXML: towards a multi-agent design tool for sketching user interfaces based on USIXML. Proceedings of the 3rd annual conference on Task models and diagrams, Prague, Czech Republic, ACM Press.

Damm, C. H., K. M. Hansen, et al. (2000). Tool support for cooperative object-oriented design: Gesture based modelling on and electronic whiteboard. Chi 2000, ACM.

Davis, J., M. Agrawala, et al. (2003). A Sketching Interface for Articulated Figure Animation. Eurographics/SIGGRAPH Symposium on Computer Animation, San Diego.

Do, E. Y. L. and M. Gross (2001). "Thinking with Diagrams in Architectural Design." Artificial Intelligence Review(15): 135-149.

Fonseca, M. J. and J. A. Jorge (2000). Using Fuzzy Logic to Recognize Geometric Shapes Interactively. Proceedings of the 9th International Conference on Fuzzy Systems (FUZZ-IEEE).

Fonseca, M. J. and J. A. Jorge (2001). Experimental Evaluation of an on-line Scribble Recognizer. Pattern Recognition Letters.

Fonseca, M. J., C. e. Pimentel, et al. (2002). CALI: An Online Scribble Recogniser for Calligraphic Interfaces. AAAI Spring Symposium on Sketch Understanding, IEEE.

Freeman, I. and B. Plimmer (2007). Connector Semantics for Sketched Diagram Recognition. AUIC, Ballarat, Australia, ACM.

Goel, V. (1995). Sketches of thought. Cambridge, Massachusetts, The MIT Press.

Hammond, T. and R. Davis (2002). Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. 2002 AAAI Spring Symposium on Sketch Understanding.

Hammond, T. and R. Davis (2003). LADDER: A Language to Describe Drawing, Display, and Editing in Sketch Recognition. IJCAI.

Hong, J. I. and J. A. Landay (2000). SATIN: a toolkit for informal ink-based applications. Proceedings of the 13th annual ACM symposium on User interface software and technology, San Diego, California, United States, ACM Press.

Hse, H. and A. R. Newton (2004). Recognition and Beautification of Multi-Stroke Symbols in Digital Ink. AAAI Fall Symposium Series, Washington D.C.

Hse, H. and A. R. Newton (2004). Sketched Symbol Recognition using Zernike Moments. International Conference on Pattern Recognition, Cambridge, UK.

Hse, H., M. Shilman, et al. (2004). Robust Sketched Symbol Fragmentation using Templates. International Conference on Intelligent User Interfaces, Funchal, Portugal.

Hutton, G., M. Cripps, et al. (1997). A Strategy for On-line Interpretation of Sketched Engineering Drawings. Proceedings of the 4th International Conference on Document Analysis and Recognition, IEEE Computer Society.

Jorge, J. A. and M. J. Fonseca (1999). A Simple Approach to Recognise Geometric Shapes Interactively. In Proceedings of the Third Int. Workshop on Graphics Recognition (GREC'99), Jaipur, India.

Landay, J. and B. Myers (1995). Interactive sketching for the early stages of user interface design. Chi '95 Mosaic of Creativity, ACM.

Landay, J. and B. Myers (1996). Sketching storyboards to illustrate interface behaviors. CHI '96, Vancouver, BC Canada, ACM.

Lank, E. H. (2003). A Retargetable Framework for Interactive Diagram Recognition. Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 1, IEEE Computer Society.

LaViola, J. J. and R. C. Zeleznik (2004). MathPad2: a system for the creation and exploration of mathematical sketches, ACM Trans. Graph.

Li, J., X. Zhang, et al. (2005). Sketch recognition with continuous feedback based on incremental intention extraction. Proceedings of the 10th international conference on Intelligent user interfaces, San Diego, California, USA, ACM Press.

Li, Y., K. Hinckley, et al. (2005). Experimental analysis of mode switching techniques in pen-based user interfaces. Proceedings of the SIGCHI conference on Human factors in computing systems, Portland, Oregon, USA, ACM Press.

Liao, S.-Z., X.-J. Wang, et al. (2005). An incremental Bayesian approch to sketch recognition Proceedings of 2005 International Conference on Machine Learning and Cybernetics.

Lin, J., M. W. Newman, et al. (2000). Denim: Finding a tighter fit between tools and practice for web design. Chi 2000, ACM.

Nakai, M., T. Sudo, et al. (2002). Pen Pressure Features for Writer-Independent On-Line Handwriting Recognition Based on Substroke HMM. Proceedings of the 16 th International Conference on Pattern Recognition (ICPR'02) Volume 3 - Volume 3, IEEE Computer Society.

Plimmer, B. (2004). Using Shared Displays to Support Group Designs; A Study of the Use of Informal User Interface Designs when Learning to Program. Computer Science, University of Waikato. **PhD**.

Plimmer, B., G. Tang, et al. (2006). Sketch Tool Usability: Allowing the user to disengage. HCI London, ACM.

Plimmer, B. E. and M. Apperley (2003). Evaluating a Sketch Environment for Novice Programmers. SIGCHI, Ft Lauderdale, ACM.

Plimmer, B. E. and M. Apperley (2003). Software for Students to Sketch Interface Designs. Interact, Zurich.

Qin, S. (2005). Intelligent Classification of Sketch Strokes. EUROCON, Serbia & Montenegro, Belgrade, IEEE.

Rabiner, L. R. (1990). A tutorial on hidden Markov models and selected applications in speech recognition. Readings in speech recognition, Morgan Kaufmann Publishers Inc.**:** 267-296.

Ramos, G. and R. Balakrishnan (2003). Fluid interaction techniques for the control and annotation of digital video. Proceedings of the 16th annual ACM symposium on User interface software and technology, Vancouver, Canada, ACM Press.

Ramos, G., M. Boulos, et al. (2004). Pressure widgets. Proceedings of the SIGCHI conference on Human factors in computing systems, Vienna, Austria, ACM Press.

Rubine, D. H. (1991). The automatic recognition of gestures. Computer Science, Carnegie Mellon University. **PhD**.

Rubine, D. H. (1991). Specifying gestures by example. Proceedings of Siggraph '91, ACM.

Sezgin, T. M. (2001). Free-Hand Stroke Approximation for Intelligent Sketching Systems, MIT**:** 2.

Sezgin, T. M., T. Stahovich, et al. (2001). Sketch based interfaces: early processing for sketch understanding. Proceedings of the 2001 workshop on Perceptive user interfaces, Orlando, Florida, ACM Press.

Thorne, M., D. Burke, et al. (2004). "Motion doodles: an interface for sketching character motion." ACM Trans. Graph. **23**(3): 424-431.

Venables, W. N. and B. D. Ripley (2002). Modern Applied Statistics with S. New York, Springer.

Young, M. (2005). InkKit: The Back End of the Generic Design Transformation Tool University of Auckland.

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

Software:

R Development Core Team (2006). R: A Language and Environment for Statistical Computing. Vienna. Austria, R Foundation for Statistical Computing.