# INTEGRATING MULTIPLE SKETCH RECOGNITION METHODS TO IMPROVE ACCURACY AND SPEED

A Thesis

by

SIDDHARTHA KARTHIK COPESETTY

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

| | |
|---|---|
| Chair of Committee, | Tracy Hammond |
| Committee Members, | Yoonsuck Choe |
| | Daniel Goldberg |
| Head of Department, | Dilma Di Silva |

August 2016

Major Subject: Computer Science

# ABSTRACT

Sketch recognition is the computer understanding of hand drawn diagrams. Recognizing sketches instantaneously is necessary to build beautiful interfaces with real time feedback. There are various techniques to quickly recognize sketches into ten or twenty classes. However for much larger datasets of sketches from a large number of classes, these existing techniques can take an extended period of time to accurately classify an incoming sketch and require significant computational overhead. Thus, to make classification of large datasets feasible, we propose using multiple stages of recognition.

In the initial stage, gesture-based feature values are calculated and the trained model is used to classify the incoming sketch. Sketches with an accuracy less than a threshold value, go through a second stage of of geometric recognition techniques. In the second geometric stage, the sketch is segmented, and sent to shape-specific recognizers. The sketches are matched against predefined shape descriptions, and confidence values are calculated. The system outputs a list of classes that the sketch could be classified as, along with the accuracy, and precision for each sketch. This process both significantly reduces the time taken to classify such huge datasets of sketches, and increases both the accuracy and precision of the recognition.

Dedicated to my mother, father, brother, grandmother, grandfather and aunt.

ACKNOWLEDGEMENTS

I would like to thank Dr. Tracy Hammond, my academic advisor for her infinite levels of energy and positivity. Her enthusiasm, support and encouragement are the sole reasons this work could be possible.

I would like to thank my committee members, Dr. Yoonsuck Choe and Dr. Daniel Goldberg.

I would also like to thank my mentor Vijay Rajanna and fellow SRL members Paul Taele, Ayden Kim, Purnendu Kaul and Josh Cherian.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

# 1. INTRODUCTION

A sketch is a rapidly executed rough, freehand drawing that is usually intended to be a basis for the final work. Sketch recognition is essential [124] in transforming hand drawn sketches and hand written text into meaningful figures and machine readable text respectively. It helps in creating interfaces through which users can communicate in a more natural way, better express themselves [54, 123, 10], learn how to draw, and even get better at drawing. Furthermore, these interfaces stimulate both halves of the brain [104]; improving users' general academic achievement and problem-solving skills [100], honing their analytical skills [26], and improving peripheral skills such as writing, critical thinking, and brainstorming when integrating sketching in their thought processes. Sketching also assists in boosting self-confidence from successful artistic pursuits [107], and improves three-dimensional spatial recognition skills [108].

To achieve these benefits, users must receive real time feedback from the system or interface that they are interacting with. Present systems can reliably classify sketches into ten or twenty classes in real time. However, these ten or twenty classes do not consist of everything that the user could possibly draw, and classification of incoming sketches into one of a much larger set of classes can take a significantly longer period of time. This thesis describes a system that comprises of multiple recognizers, and can classify the incoming sketch into one of 491 classes at a fraction of the time current systems take with high accuracy and precision. This increase in the speed of recognition can aid in the creation of interfaces and systems with real time feedback.

Real time feedback is vital for creating engaging interfaces. Chen et al. [13] identified immediate feedback as one of the most important factors to having a

seamless experience for web activities. Keeping users engaged in the experience through immediate feedback enables them to perform tasks and attain mastery of skills faster.

Artificial intelligence has given machines and systems the ability to not only learn from given data but also the capability of discovering new facts about the data [75]. This thesis descibes how artificial intelligence applied on the sketch data can classify the sketches automatically without human intervention. Gestural features are extracted and used to perform classification to improve classification of geometric algorithms. This thesis identifies a subset of gestural features that are critical to the accurate recognition of a shape, and can increase speed and accuracy of the overall domain.

The major contributions of this thesis are the accurate and real-time classification of an incoming sketch, and the identification of how gestural features of the sketch can be combined with geometric algorithms to identify a sketch. The combined system of both gestural and geometric methods can perform better than any existing systems.

# 2. RELATED WORK

With the development and emergence of newer computing technologies, many researchers, and software developers have focused on applying these technologies in fields related to sketching. This is true with advancements in hardware devices that can support digital sketching such as pen-based devices, mobile, tablet, and touch-screen computers. Before looking into the details of the thesis, this section describes the related and relevant prior work to understand how this system is different from existing systems.

## 2.1 Sketch recognition

Recognition of hand drawn sketches is called sketch recognition. The literature contains a great extent of research in the field of sketch recognition [37]. Sketch recognition algorithms can be classified primarily into three categories: gesture-based recognition, geometry-based recognition, and vision-based recognition.

Gesture-based sketch recognition [103, 132, 69, 22, 14, 15] uses the inherent properties of the sketch to identify shapes. In these recognition algorithms either the system learns the user's style of drawing or the user has to learn the system's style. One dollar [132] is a simple recognition algorithm that uses template matching to identify sketches. Rubine used features including initial angle, sharpness, speed, and total angle traversed to recognize shapes. These are some of the most popular features cited popularly in sketch recognition research and can even be used to predict the shape before it is completed [72]. Sezgin [105] and Staovich [109] took advantage of gesture-based features like speed and curvature to distinguish different shapes.

Geometric algorithms [89, 35] recognize shapes by using geometric constraints that hold for a shape. They allow users to draw shapes naturally [89, 46]. Geometric

recognizers usually do a bottom-up approach where basic shapes such as lines, arcs, and circles are recognized first [51] and strokes are segmented into their components at corners [135, 139, 136, 138]. A higher level recognizer is built on top of this low-level recognizer, which uses geometric constraints [58] to check if the primitive shapes when put together form a more complex shape [42, 38, 44, 41, 36, 40, 39, 35]. Ladder [46] is a language to describe how sketched diagrams in a domain are drawn, displayed, and how recognition algorithms can be written for that domain.

Vision-based algorithms [32, 76, 96] use concepts from computer vision similar to those used on images after preprocessing of the sketches. The screen coordinates are used by Kara and Stahovich [62] to apply template matching algorithms used in their recognizer.

All existing algorithms, or their combination [17] enable us to identify different shapes. Topological information about sketches have also been used for recognition. This approach works well with sketches having multiple spatial components such as chemical bonds or complex electrical circuits [81]. This system uses gesture-based and geometry-based recognition in the initial stage. In the second stage, the system uses all three recognition techniques.

## 2.2 Existing systems

There are many systems [131, 86, 95] in the field of sketching. Mechanix [128, 91, 5, 4, 126, 129, 74, 29, 80, 127, 64], which is a sketch-based tutoring system for engineering students learning statics, allows students to enter free body diagrams, specifically trusses, into the system, which then automatically checks the student's answer against the solution entered by instructor. Maestoso [117] is another system, which uses a sketch-based interface in the field of music for novice learners to learn music. Other representative disciplines where sketch-based systems have been

incorporated include East Asian Languages [116, 118, 120, 121], Math [59, 99], electrical engineering [23], coding [45, 48], and the military [53, 50, 19, 47, 21]. Several applications can recognize hand-drawn facial features; iCanDraw [24, 52, 73], and EyeSeeYou [20] detect, recognized, and give feedback on handdrawn faces and eyes. The Drawing Assistant [56], PortraitSketch [140], and Painting with Bob [8] are the extensions of iCanDraw and EyeSeeYou, which implement a wider set of figures.

All these systems do well for a limited number of classes. Hammond [49] created an application that allows military commanders to hand-draw hundreds of course of action (COA) symbols directly on a digitized map. By utilizing several artificial intelligence techniques, they achieved 89.9% accuracy for 485 symbols when considering the top three interpretations.

# 3. BIG SKETCH DATA

This chapter gives an insight into the data we have at hand. We also look into the actual process of sketching, how a class is defined, what are the different types of classes, data collection, the amount of data per class, how the data is stored, and fast retrieval of the data. This is helpful in understanding the nature of the problem at hand, what the existing systems have to handle, and how my system handles it with ease, and performs better than the existing systems.

## 3.1 What are sketches?

Sketching is an integral part in the life of designers and engineers [124]. Sketches are not only used to document the ideas conceived in one's mind but also help in acting as a stimuli for generating more ideas. Sketching allows users to brainstorm and quickly outline ideas before settling on a final design. Buxton [10] provides in-depth knowledge of the characteristics of a good sketch. He defines sketches as having the following qualities:

- *"Quick - A sketch is quick to make, or at least gives that impression.*
- *Timely - A sketch can be provided when needed.*
- *Inexpensive - A sketch is cheap. Cost must not inhibit the ability to explore a concept, especially early in the design process.*
- *Disposable - If you can't afford to throw it away when done, it is probably not a sketch.*
- *Plentiful - Sketches tend not to exist in isolation.*

- *Clear Vocabulary - The style in which a sketch is rendered follows certain conventions that distinguish it from other types of renderings.*
- *Distinct Gesture - There is a fluidity to sketches that give them a sense of openness and freedom.*
- *Minimal Detail - Include only what is required to render the intended purpose or concept.*
- *Appropriate Degree of Refinement - By its resolution or style, a sketch should not suggest a level of refinement beyond that of a project being depicted.*
- *Suggest and Explore Rather than Confirm - Sketches do not 'tell', they 'suggest'.*
- *Ambiguity - Sketches are intentionally ambiguous, and much of their value derives from their being able to be interpreted in different ways"*

## 3.2 What is a class?

A class is defined as anything a user could possibly draw. For example: a point, a line, a circle, a rectangle, a polyline, a crown, a tree, a combination of an ellipse, a diamond, and a hyperbola, and so on. Figure 3.1 is an example of a course of action diagram that comprises of many different classes. It belongs to the friendly_company_present field_artillery_howitzer_rocket_multi_rocket_truck class.

We will see more on course of action diagrams in the next section. We have a total of 491 different classes and the system can classify an incoming sketch into any of these 491 classes. To put that into perspective, existing techniques [89, 103, 90, 80] classify instances into ten or twenty classes. This system analyzed data of a total of 10,000 sketches.

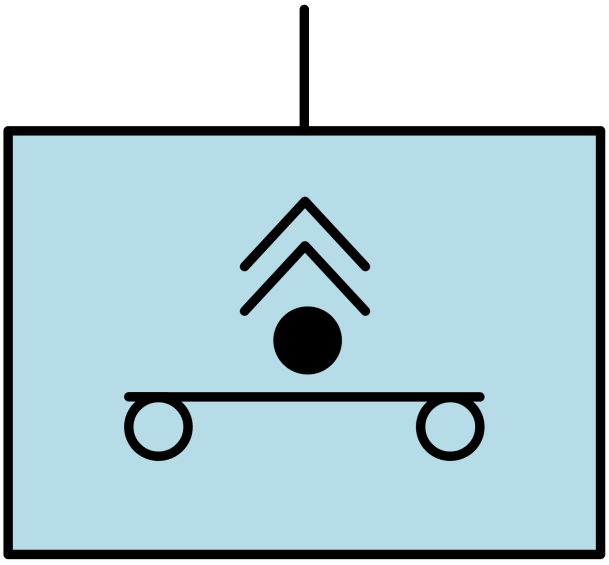

Figure 3.1: Course of action diagram

## 3.3 Data collection

Using a C# application, data was collected from both students and experts, and stored as XML files, using the SketchML format from MIT. The interface used to capture what the user is drawing on screen can be seen in Figure 3.2. The user can view any of the previous sketches by clicking on the "Load" button on the bottom of the screen. This is a valuable tool for users to see what an XML actually contains. Also, to help users while drawing there are a couple of buttons provided on screen - Clear and Undo. The "Undo" button can be used to erase the latest unintended stroke on the screen. This helps greatly when the user is drawing a sketch and makes a mistake after thirty strokes. The "Clear" button erases everything on the screen. The "Save" button, as expected, saves the sketch in XML format.

The sketches collected are handdrawn and thus not perfect. Figure 3.3 shows a sketch that is drawn naturally, starting with a jerk, and having lines that extend

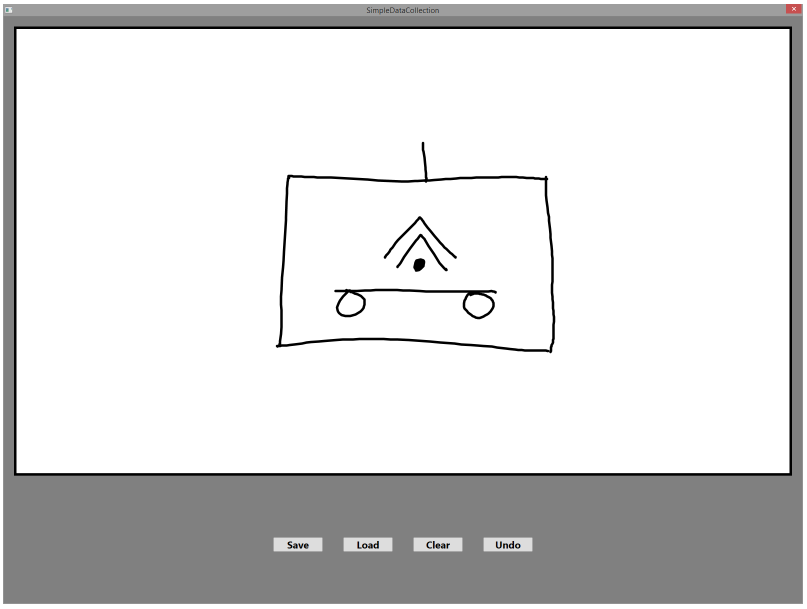


Figure 3.2: Tool used for data collection

beyond the corners. To maintain consistency, all of the data was collected using Wacom Cintiq tablets. Figure 3.4 shows the format in which the sketches are stored.

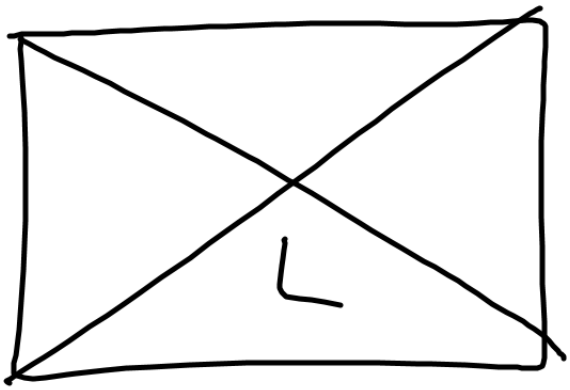

Figure 3.3: Example of naturally drawn sketch

```
<?xml version="1.0" encoding="UTF-8"?>
<sketch id="128d8d46-6c81-79ca-1bc1-36b1e9f33db5" type="SRL">
    <point id="719a288e-3d85-c237-7184-f1d585b68792" pressure="0.0"
        tilt_x="0.0" tilt_y="0.0" time="1463008500001" x="547.0" y="273.0"/>
    <point id="83213d35-f018-d1cb-2591-4842c4a7aa58" pressure="0.0"
        tilt_x="0.0" tilt_y="0.0" time="1463008500018" x="546.0" y="276.0"/>
    <point id="951045a5-bd96-7815-3ed5-5394bac3120e" pressure="0.0"
        tilt_x="0.0" tilt_y="0.0" time="1463008500051" x="545.0" y="288.0"/>
    <point id="b78e298a-d77e-0193-9f36-4c195f1cd2a2" pressure="0.0"
        tilt_x="0.0" tilt_y="0.0" time="1463008597524" x="540.0" y="610.0"/>
    <point id="c024462a-fe63-88a1-3bbb-49ee4e9a6682" pressure="0.0"
        tilt_x="0.0" tilt_y="0.0" time="1463008597530" x="539.0" y="610.0"/>
    <point id="91384c16-4a0b-56f9-6434-3968162c803f" pressure="0.0"
        tilt_x="0.0" tilt_y="0.0" time="1463008597549" x="538.0" y="609.0"/>

    <stroke id="78217b35-f52a-bc9b-b6e8-5a8c6dd7a1aa" visible="true">
        <arg type="point">719a288e-3d85-c237-7184-f1d585b68792</arg>
        <arg type="point">83213d35-f018-d1cb-2591-4842c4a7aa58</arg>
        <arg type="point">951045a5-bd96-7815-3ed5-5394bac3120e</arg>
    </stroke>

    <stroke id="5262bdee-9f79-8602-81b1-30ba2c9d542b" visible="true">
        <arg type="point">b78e298a-d77e-0193-9f36-4c195f1cd2a2</arg>
        <arg type="point">c024462a-fe63-88a1-3bbb-49ee4e9a6682</arg>
        <arg type="point">91384c16-4a0b-56f9-6434-3968162c803f</arg>
    </stroke>
</sketch>
```

Figure 3.4: XML sketch format

The x-coordinates, y-coordinates, the time-stamp, and the point ID is saved for each point. Each individual stroke has a stroke ID and includes the list of point IDs that they comprise of. 125 feature values are calculated for each of the sketches. Each line describes a single sketch and consists of the sketch ID, 125 feature values, and a label associated with it. This led to 7,620,000 cells of data.

To accommodate this much data we used a MySQL database with each tuple consisting of the sketch ID, the feature values, and the label. This allowed for the retrieval of feature values of any sketch with its primary ID. Using a MySQL database aids in the visualization of the data, as seen in Figure 3.5, and facilitates fast retrieval of feature values.

Figure 3.6 refers to a collection of different sketches collected through user studies.

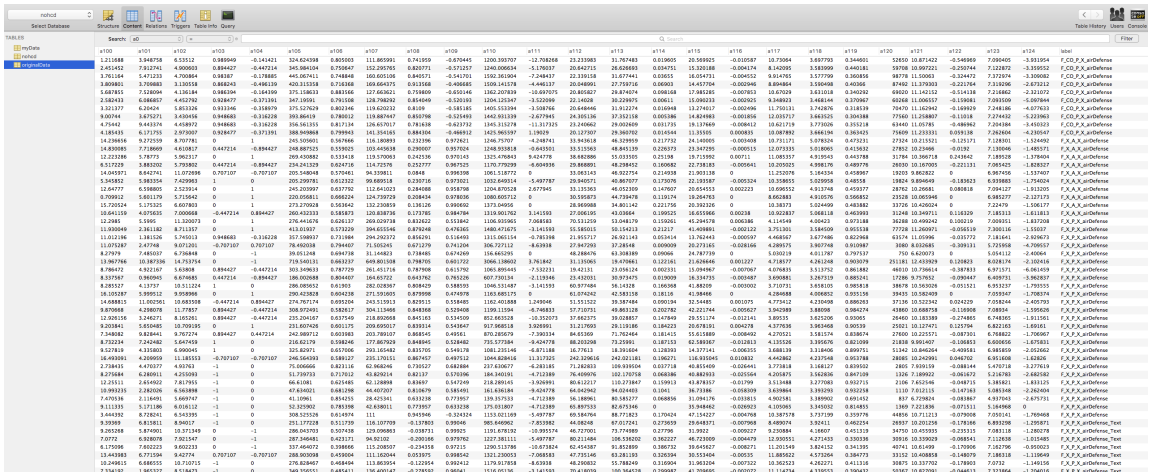

Figure 3.5: MySQL database with feature values

Figures 3.7,3.8,3.9,3.10,3.11 refer to the collection of sketches the system can recognize.

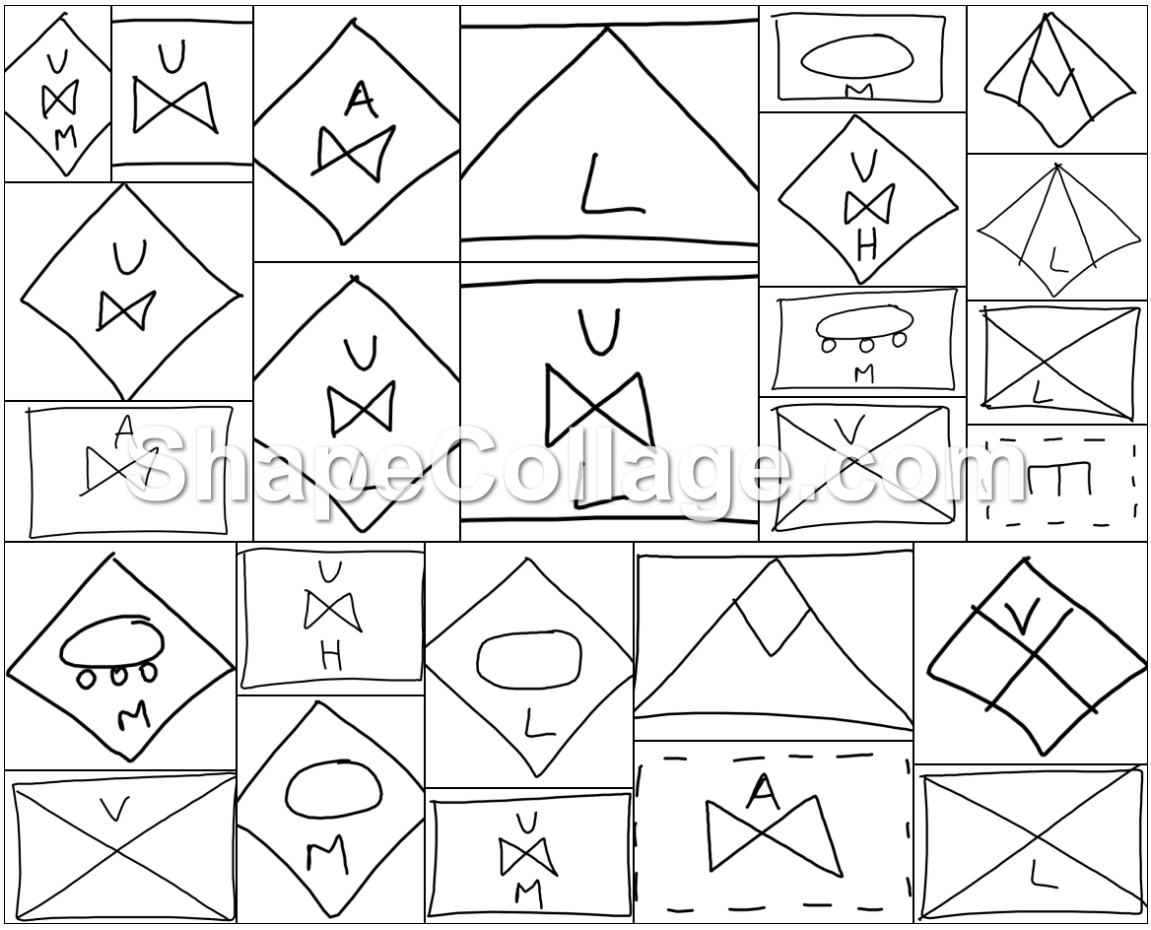


Figure 3.6: Collection of user drawn sketches

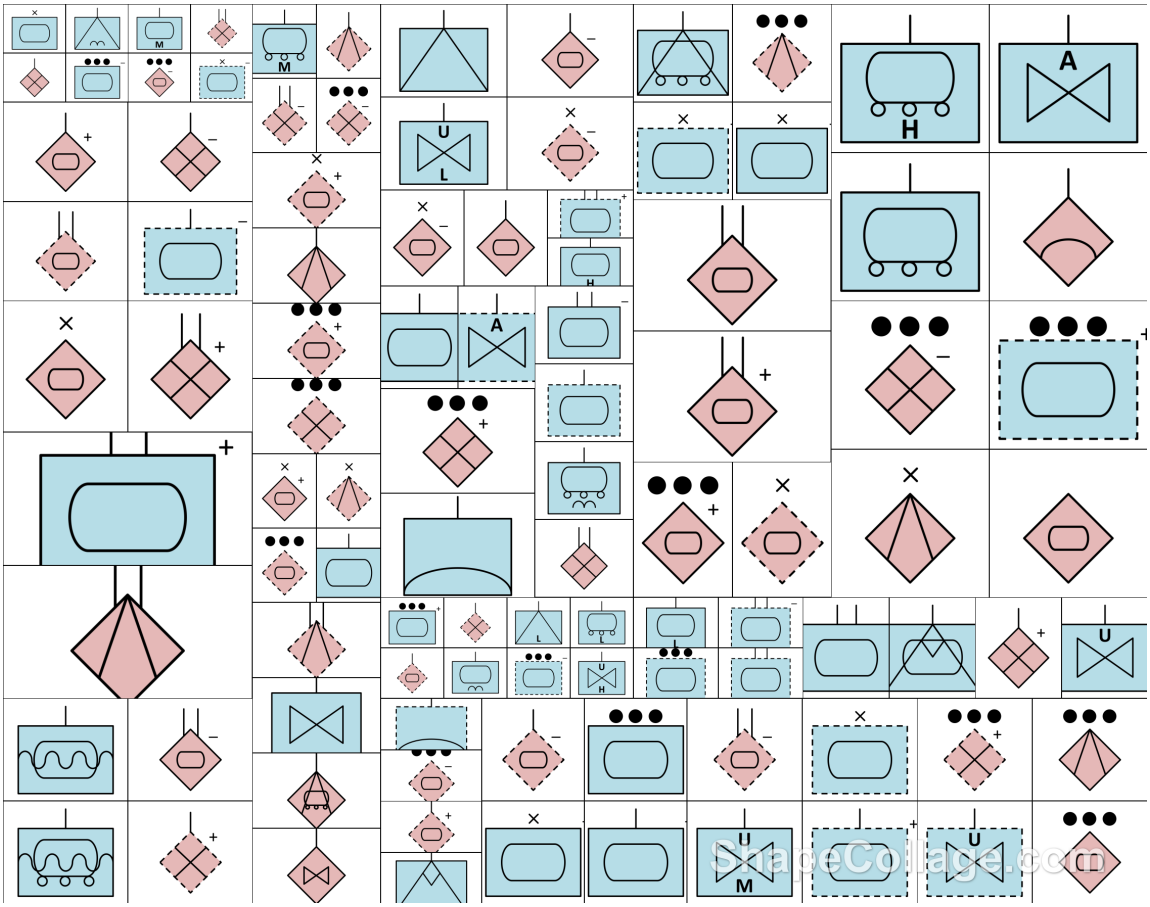


Figure 3.7: Collection of sketch classes 1

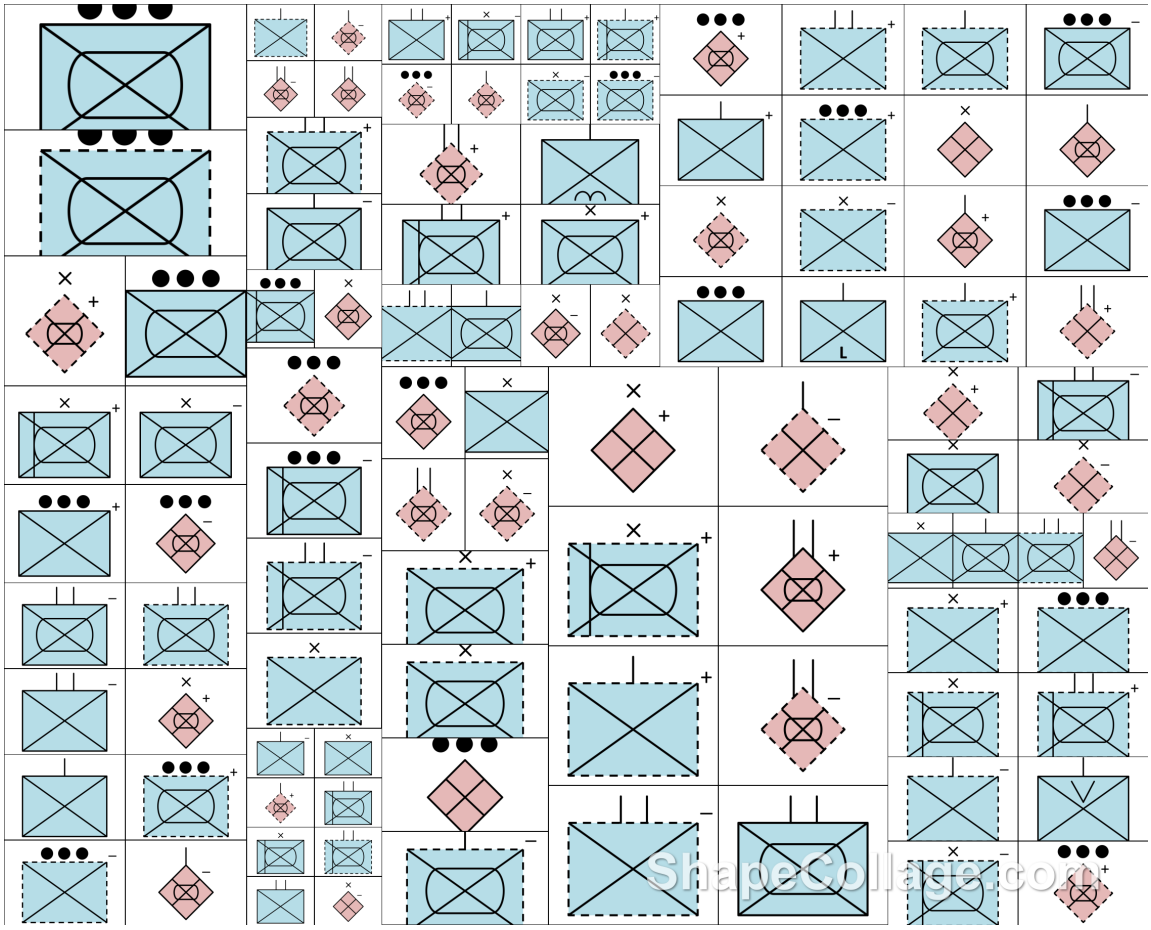


Figure 3.8: Collection of sketch classes 2

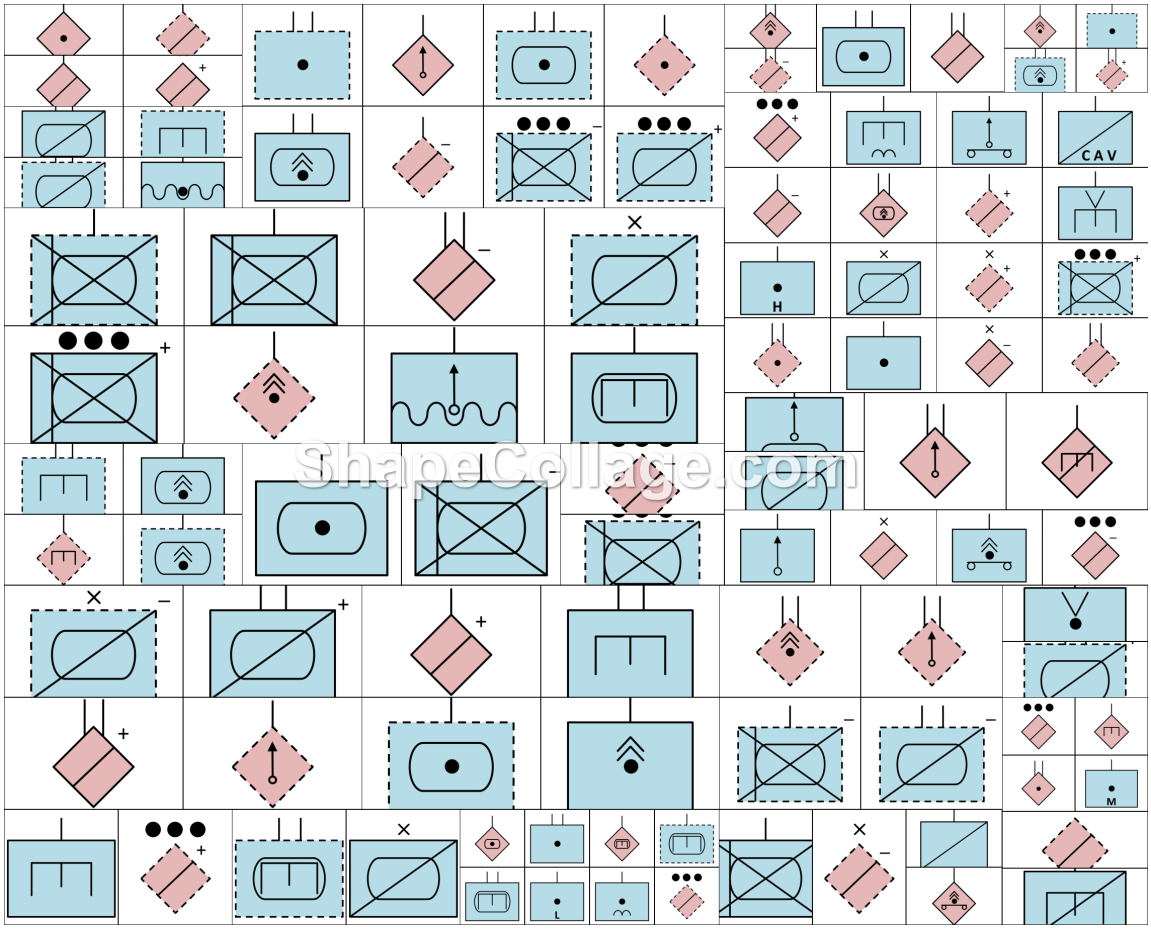


Figure 3.9: Collection of sketch classes 3

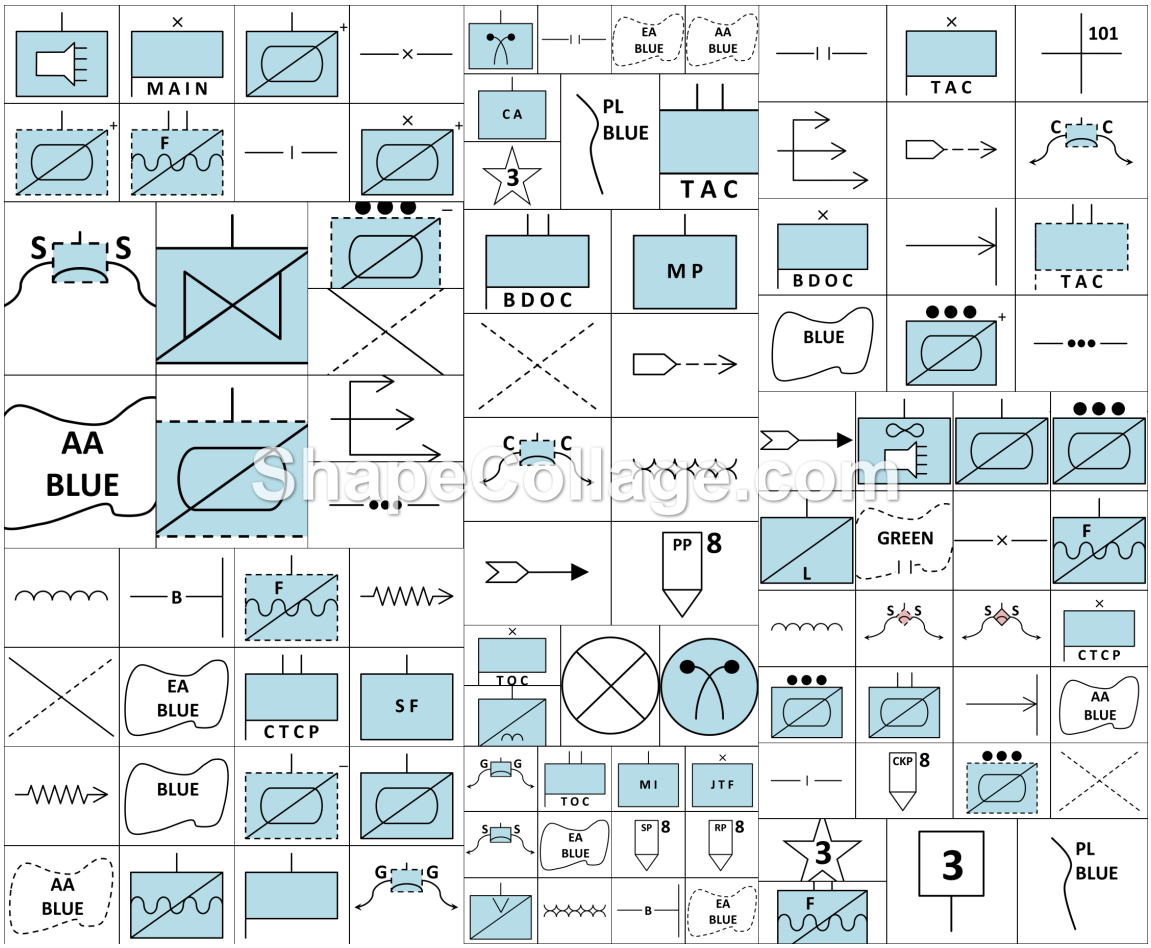


Figure 3.10: Collection of sketch classes 4

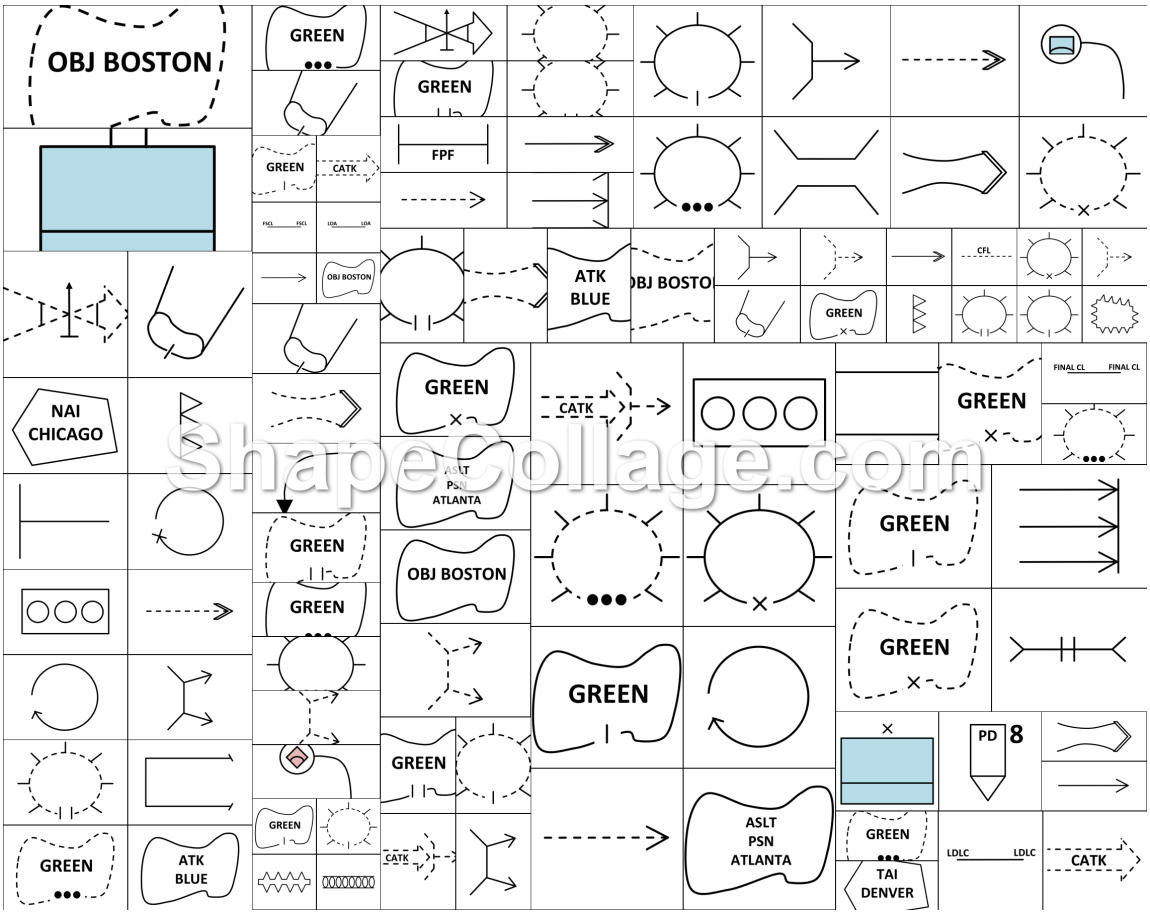


Figure 3.11: Collection of sketch classes 5

# 4. EXISTING SYSTEM

To achieve recognition, the system relies on state-of-the-art techniques of sketch recognition. The research in this field can be categorized into three sub-fields: geometric recognition, gesture-based recognition, and vision-based recognition. the system uses all three recognition techniques to identify the shapes of the incoming sketches.

## 4.1 Course of action diagrams

One real-world domain that has thousands of symbols is military course of action (COA) diagrams. Military COA diagrams are used to depict battle scenarios and include thousands of unique symbols, complete with additional textual, and designator modifiers. Military commanders use COA diagrams to plan field operations, where the symbols represent troops, supplies, obstacles, and movement patterns. Currently, commanders draw COA diagrams by hand on a map for planning purposes, and then these diagrams are entered into a computer through typical WIMP interfaces for the purposes of simulation and communicating plans. Previous attempts [7] to apply sketch recognition to COA diagrams have resulted in systems that can only recognize a handful of the over 20,000 total COA shapes, require users to draw each shape with a specific gesture, and do not allow users to draw freely (Pittman, et al. [16]).

## 4.2 Architecture

The interface to help users draw or view sketches is developed in C#. The sketches are stored in XML format. All the recognition and evaluation algorithms are written in Java.

## 4.3 Sketch representation

Modern pen-based interfaces record input as a collection of points within a two-dimensional plane and the current time of input. Points are generated as the pen or stylus moves over the input device. Each of these points is recorded as an x-y coordinate pair along with a time-stamp. This time-stamp is the current epoch time, which is the total number of milliseconds that have elapsed since 00:00:00 Coordinated Universal Time (UTC) on Thursday, 1 January 1970. Collections of time-ordered points between a pen-down event (a pen touching the sketching interface) and a pen-up event (the pen lifting off of the interface) are called strokes. A sketch is comprised of one or more strokes and is defined as a shape when it satisfies geometric constraints. Figure 3.4 shows the format in which the sketches are stored.

The x-coordinates, y-coordinates, the time-stamp, and the point ID is saved for each point. Each individual stroke has a stroke ID and includes the list of point IDs that comprise the stroke.

## 4.4 Recognition system

This section talks about the different stages of recognition. As a new sketch is drawn on the interface, it is saved as an XML object in the format seen in Chapter 3. Figure 4.1 shows the shape description, written in XML format. It is needed for the system to classify a sketch. The various steps in the recognition process are as follows[49]:

- Phase and boundary line recognition
- Early handwriting and sub-unit recognition (HRS)
- Corner recognition

- Primitive recognition
- HRS with primitive context
- Mid-level shape recognition
- HRS with mid-level shape context
- High-level shape recognition
- Arrow recognition
- Multiple shape disambiguation
- SIDCs

The following sections explain the recognition steps in more detail.

### *4.4.1 Grouping*

The previously existing system [49] consists of a number of different recognizers, including both recognizers that perform well on shapes, and recognizers designed to interpret decision graphics and text. Basic geometric shapes like rectangles, lines, and ellipses are recognized using PaleoSketch [89]. Echelon modifiers, text, and decision graphics are recognized with the help of a handwriting recognizer (HWR). A grouping algorithm ensures that the best recognizer receives the corresponding strokes. During the grouping process, the first step is to find the largest stroke. This is usually the boundary of the unit symbol. Depending on whether they are present inside or outside of the bounding box of the largest stroke, the remaining strokes are grouped into two sets as seen in Figure 4.2. Interior strokes are typically type modifiers, decision graphics or text, and should be processed by the primitive recognizer. To this end, these strokes are parsed by the HWR initially and marked

```
<?xml version="1.0" encoding="UTF-8"?>
<shapeDefinition name="023_F_X_A_X_armor" description="Armor">
    <isAList>
        <isA>Shape</isA>
    </isAList>
    <attributeList>
        <!-- Source: 2525B document, p. 146, row 4 -->
        <attribute key="ATTR_SIDC" value="SFGAUCA---*****" />
    </attributeList>
    <componentList>
        <!-- the frame [rectangle] -->
        <component name="rectangle" type="Dash_Rectangle" />
        <!-- the armor -->
        <component name="ellipse" type="Ellipse" />
    </componentList>
    <constraintList>
        <!-- CONTAINING RELATIONSHIPS -->
        <!-- rectangle contains ellipse -->
        <constraint name="Contains" thresholdMultiplier="1.0">
            <param component="rectangle" />
            <param component="ellipse" />
        </constraint>

        <!-- ellipse contains rectangle's center -->
        <constraint name="Contains" thresholdMultiplier="1.0">
            <param component="ellipse" />
            <param component="rectangle" referencePoint="Center" />
        </constraint>
    </constraintList>
    <aliasList>
        <alias name="PT.1" component="rectangle" referencePoint="Center" />
    </aliasList>
</shapeDefinition>
```

Figure 4.1: Shape description

as decision graphics or text if the HWR's confidence is high enough. Otherwise, the interior strokes are processed by PaleoSketch for recognition as primitive shapes. The HWR more often than not recognizes the echelon modifiers that occur outside the bounding box of the largest stroke.

### *4.4.2 Segmentation*

The process of breaking down strokes into primitive building blocks like lines and arcs is called stroke segmentation. The system splits every stroke into a series of lines. These series of lines (polyline representations of the strokes) are then sent to PaleoSketch [89]. In order to perform polyline segmentation with high accuracy, the system uses a combination of multiple recognizers [134, 135, 1, 65, 25].


Figure 4.2: Example results of the grouping algorithm

### *4.4.3 Primitive recognition*

Primitive recognizers are used to accurately recognize strokes as basic building block shapes. Using high-level shape grammars like LADDER [43], these primitive shapes are combined to form more complex shapes. An extended version of PaleoSketch algorithm [89] is used for primitive recognition. Figure 4.3 shows the fourteen different primitive shapes that PaleoSketch can recognize.

### *4.4.4 Dashed primitives*

The Course of Action domain contains a number of anticipated shapes that are drawn with dashed boundaries and have an arbitrary number of lines. Figure 4.4 represents an anticipated symbol. The first step to recognizing dashed boundaries is to find occurrences of dashed lines. The system iterates through all the shapes on the screen as they were drawn while maintaining a stack of dots and lines that could make a possible dashed line candidate. If it finds significant slope changes or large gaps between strokes, it might mean that a dashed shape is not present. In that case, the entire stack of dots and lines is sent to a function that returns a dashed line if it has more than one shape. Once all the dashed lines have been found, they are

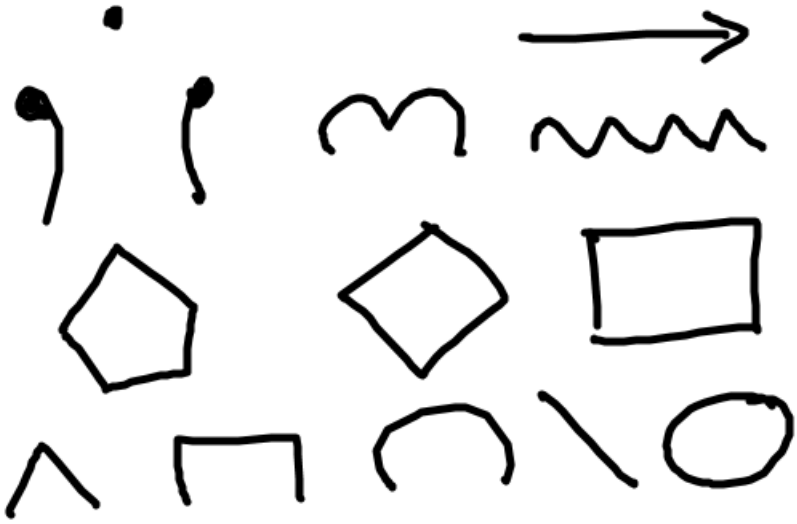

Figure 4.3: The 14 primitive shapes recognized by PaleoSketch

separated into one of four different bins based on their orientation, namely lines with positive slope, lines with negative slope, lines within fifteen degrees of vertical slope, an lines within fifteen degrees of horizontal slope. After sorting them into bins, lines in the positive and negative slope bins are processed in search of diamonds, while lines in the horizontal and vertical bins are processed in search of rectangles. By considering polylines as arcs with dashes, the system is able to recognize dashed ellipses as well. The system generates an intermediate stroke using the endpoints and midpoints of the strokes that are stacked up as candidate dashes for an ellipse. If the intermediate stroke passes the ellipse test, the system groups the strokes together as a dashed shape.

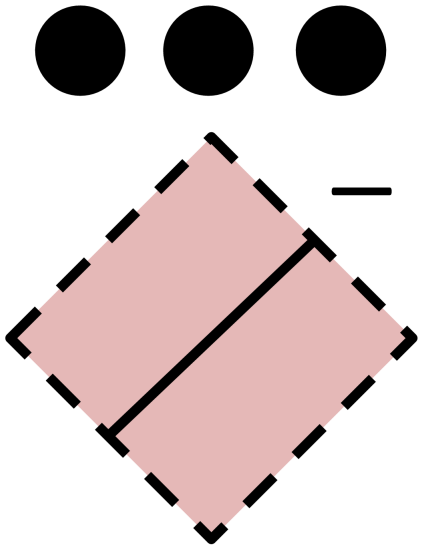

Figure 4.4: Example of an anticipated symbol

### *4.4.5 Decision point recognition*

The decision points in Course of Action diagrams are generally drawn as ten-line stars. The previously existing system uses a similar approach to that of dashed shapes to search for stars. Along with checking the spatial distances between consecutive strokes, it also checks that the angles between lines alternate between acute and obtuse angles. If the shape in question follows this trend, it is recognized as star.

### *4.4.6 Handwriting recognition*

The handwriting recognizer recognizes inner text and echelon text using two pre-built multi-layer perceptron [101] models from Weka [34] respectively. Figure 4.5 refers to the echelon text comprising of the symbols I, II, X, ***, +, and -. The inner text can be digits 0-9, uppercase alphabets or some special decision graphics symbols. The system groups the strokes into logical characters and words using stroke spatial separation and overlap data. It also implicitly assumes that text is

written in a left to right fashion. After forming these stroke groupings, the character, and word possibilities are computed in a brute-force manner. Due to the assumed constraints that no word has more than seven characters and each character is made of at most five strokes, this approach is still relatively quick. A multi-layer perceptron model is used to compute a confidence value for each possible character or word and the handwriting with highest confidence value is used for symbol recognition in the future.

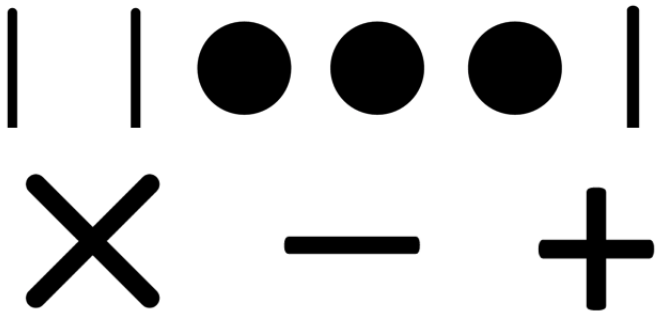

Figure 4.5: List of echelon modifiers

### *4.4.7 CALVIN*

Complex high-level shapes are built by combining primitive shapes (recognized using PaleoSketch [87]) or other high-level shapes in various combinations using a heuristic-driven geometric recognizer based on LADDER [42] called CALVIN. Shapes are combined using geometric conditions as seen in Figure 4.1. CALVIN specifies that the endpoints of two lines are coincident or that one shape contains another shape. If we look at Figure 4.6, we see on the left, a sketch of a reconnaissance cavalry armored unit. The shape description for this shape says that there is a line with positive slope

with the endpoints coincident with the corners of the rectangle. CALVIN has been hand-tuned using heuristics to increase the accuracy and precision in recognizing course of action diagrams in addition to the basic recognition capabilities found in LADDER. For grouping purposes, it re-analyses the set of shapes. The context of recognition is very helpful in re-grouping strokes more accurately even though the strokes have been grouped for recognition in the earlier stages. For example, we can say that certain strokes in certain positions relative to a rectangle should be grouped together if the primitive recognizer has detected a rectangle with a high degree of confidence. After this, CALVIN is able to put together all the individual components of the course of action shape. On the right, we have a reconnaissance cavalry armored unit with a platoon symbol modifier and reinforced strength modifier. Once CALVIN recognizes the dashed rectangle, contextual rules are used to determine the line, ellipse, platoon, and the plus. In this example, the military symbol is identified using an alphanumeric string on the bottom left. Without contextual rules, correctly identifying all these components simultaneously becomes an NP-complete problem.

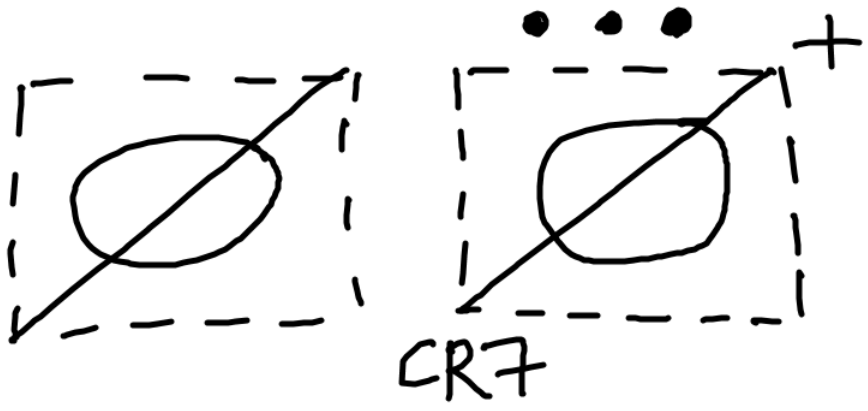


Figure 4.6: Reconnaissance cavalry armored unit on the left

### *4.4.8 Phase and boundary lines*

Figure 4.7 shows that the phase lines and boundary lines define the location steps of boundaries and military operations that confine military units. They can be arbitrary outlines and are recognized first by determining if there is a sufficiently long stroke present. In case such a long stroke is found, all the remaining strokes are sent to the handwriting recognizer under the assumption that they could be text. The handwriting recognizer looks for type identifiers PL or BL and unique identifiers BLUE or ALPHA for distinguishing between phase lines and boundary lines.

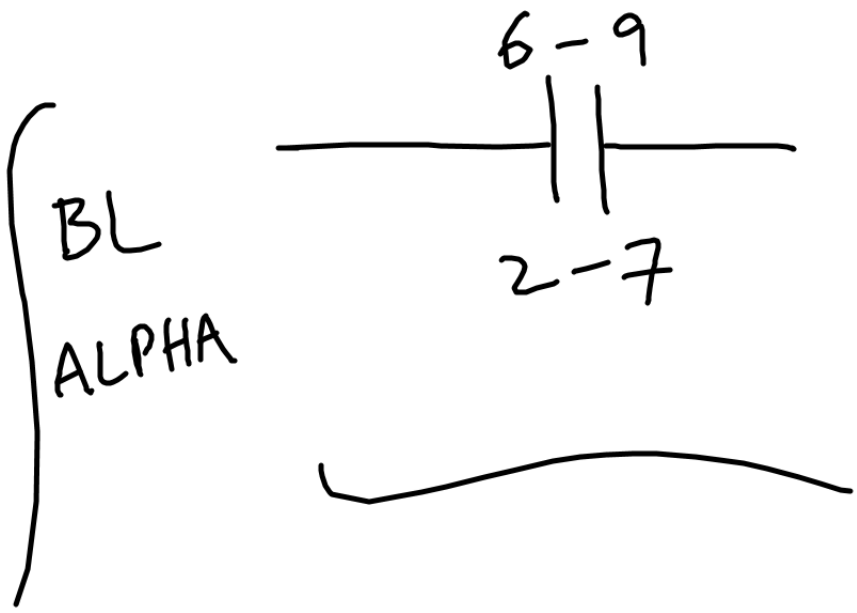


Figure 4.7: Examples of phase and boundary lines

### *4.4.9 Areas*

Large, closed strokes that indicate zones of interest such as landing zones (LZ) are called Areas. If the largest stroke in the sketch is closed, the system thinks that

it could possibly be an area. If it is closed, the previously existing system searches for the type identifier, echelons either on the top or bottom, and identifier text inside of the enclosure.

### *4.4.10 Decision graphics*

Type modifiers with a combination of filled rectangles, un-filled rectangles, and triangles are known as decision graphics. The hand writing recognizer is used to recognize these symbols as the primitive recognizer cannot handle filled-in shapes. These shapes are present in the collection of modifiers definitions, which occur inside the largest shapesś bounding box, as the grouper determines. If it detects instances of decision graphic modifiers, the hand writing recognizer analyzes all shape definitions of the type DecisionGraphic to find the correct symbol.

### *4.4.11 Obstacles*

Figure 4.8 represents Course of Action diagrams that comprise of patterns. That is, the same individual shape repeated a specific number of times. Such shapes are recognized looking at the micro-level. Symbols with similar patterns are recognized using a set of features computed for each symbol and compared against a set of template patterns.

### *4.4.12 Arrows*

Arrows are allowed to have arbitrary paths as they indicate movement of friendly or hostile forces. The system recognizes arrows by looking at the type of arrow head used, the number of strokes drawn, and the number of line segments the strokes can be broken into. Figure 4.9 shows the list of different types of arrows recognized.

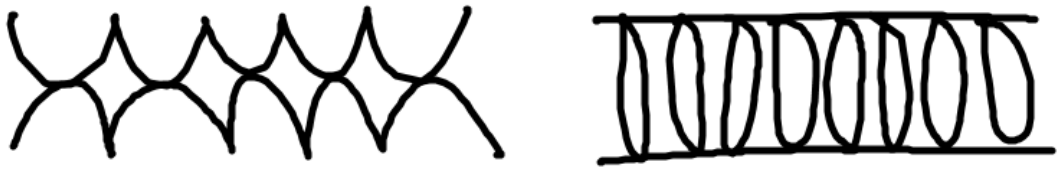

Figure 4.8: Example of obstacles drawn with dynamic patterns

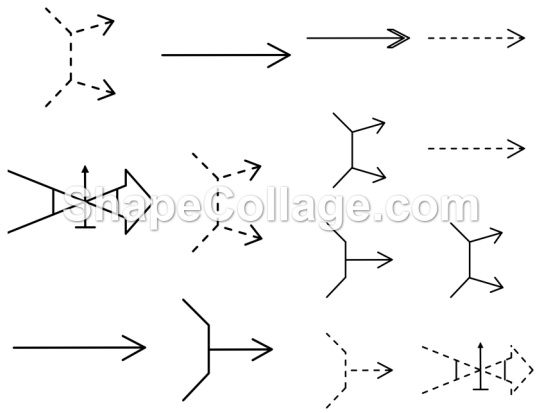


Figure 4.9: List of different types of arrows

# 5. FEATURE SELECTION

Sketch recognition is the application and development of artificial intelligence and machine learning techniques in order to recognize hand-drawn sketches [47, 93, 94]. Fundamentally, there are three types of sketch recognition approaches [37]: (a) gesture-based, which takes into consideration of how a sketch is drawn [14, 15, 27, 71, 106, 17, 50]. (b) geometry-based, which initially recognizes sketches at a lower level and then from a higher level using geometric definitions and constraints [35, 118, 52], and (c) vision-based, which looks at pixel data on the screen [96, 97].

Primitive shapes like lines, arcs, and circles are recognized by segmenting the incoming stroke in geometric-based recognizers. Then, these primitive shapes are combined together and recognized by matching against geometric definitions [42]. A closest match is picked by comparing the input sketch with different datasets in template-matching techniques. Initially, the sketches are resampled, rotated, and scaled. The best optimal angle is found to calculate the score. A Naive Bayesian classifier is used to calculate the Euclidean distance. Examples of representative research in this area include the $-family algorithms (e.g., $1, $N) [2, 133], elastic structure matching [11], and vision-based recognition by Kara [60]. For the the template-matching and vision-based features, this system utilizes the Valentine version of the Hausdorff shape recognizer [126]. This approach requires numerous examples for each shape for use in shape comparisons.

A statistical measure like mean or standard deviation of each shape is used in a statistical feature-based classifier. Many sketch features are taken from Rubine [102], Long [70], and Paulson [83, 84].

The following sections look into different types of recognition to describe the

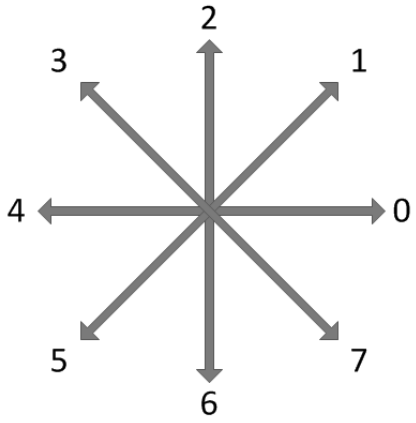


Figure 5.1: Directions in Freeman's chain code [31]

features used in this system.

## 5.1 Gesture-based recognition

Gesture-based features rely on how a stroke is drawn rather than what is visually shown on a display screen.

### *5.1.1 Freeman's chain code*

The direction of each line is used as a feature in Freeman's chain code. It is widely used for representing shapes [31]. Chan used direction of each line in an elastic structural matching algorithm [11, 12]. The direction from a point to the next point is indicated using values from zero to seven.

Figure 5.1 shows how Freeman's chain code works. Using the directions, Chan, et al. [11] described five types of primitives:

- line
- up (curve going counter clockwise)
- down (curve going clockwise)

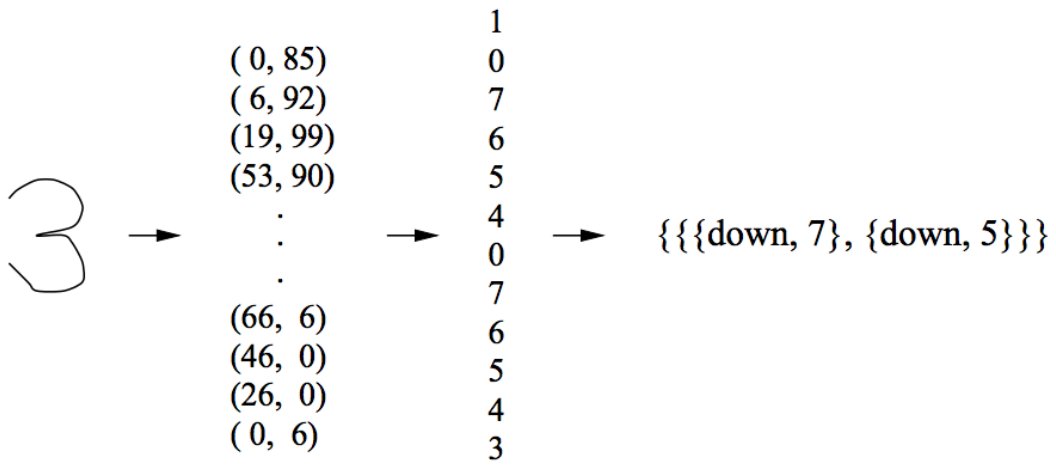


Figure 5.2: An example of directions in Freeman's chain code [11]

- loop (curve joining itself at some point)
- dot ( a very short segment)

Figure 5.2 represents the example of the digit "3", and the letter has two down primitives. Different variances of same shape are not taken into consideration in this approach.

### *5.1.2 Rubine and long*

Classifiers like Bayesian networks are used to retrieve sketch features in statistical feature-based approaches. One of the early works in gesture-based recognition systems is by Rubine [102]. He introduced GRANDMA, a gesture recognizer automated in a novel direct manipulation architecture. Rubine introduced thirteen features, which could accurately classify simple gestures by using features like sine or cosine of angles between different points. Long [70] extended this work by removing two features and adding eleven features. (Table 5.1).

| | |
|---|---|
| 1. Cosine of initial angle | 2. Sine of initial angle |
| 3. Size of bounding box | 4. Angle of bounding box |
| 5. Distance between first and last points | 6. Cosine of angle between first and last points |
| 7. Sine of angle between first and last points | 8. Total length |
| 9. Total angle | 10. Total absolute angle |
| 11. Sharpness | 12. Aspect [abs(45° - #4)] |
| 13. Curviness | 14. Total angle traversed / total length |
| 15. Density metric 1 [#8 / #5] | 16. Density metric 2 [#8 / #3] |
| 17. Non-subjective "openness" [#5 / #3] | 18. Area of bounding box |
| 19. Log(area) | 20. Total angle / total absolute angle |
| 21. Log(total length) | 22. Log(aspect) |

Table 5.1: Long et al. [70] features

.

### *5.1.3 Dollar recognizers*

Multiple gesture-based template matching algorithms were developed by Wabbrock, et al. [3, 78, 130, 133]. $1 recognizer [133] works for single stroke sketches and $N recognizer [2] works for multi-stroke sketches. The preprocessing on incoming sketches include: (1) resampling, (2) rotation, (3) scaling and translating, and (4) calculating best score [133]. Figure 5.3 shows the sixteen different shapes $1 algorithm can recognize.

Figure 5.4 shows an example of how $N works. It considers all combinations of strokes, translating multi-strokes into single stroke shapes (Figure 5.5).

## 5.2 Geometry-based recognition

Strokes are segmented using corner-finding algorithms [138] and basic primitives are recognized using geometry-based recognition techniques. These individual primitives are recognized as a whole from a higher level.

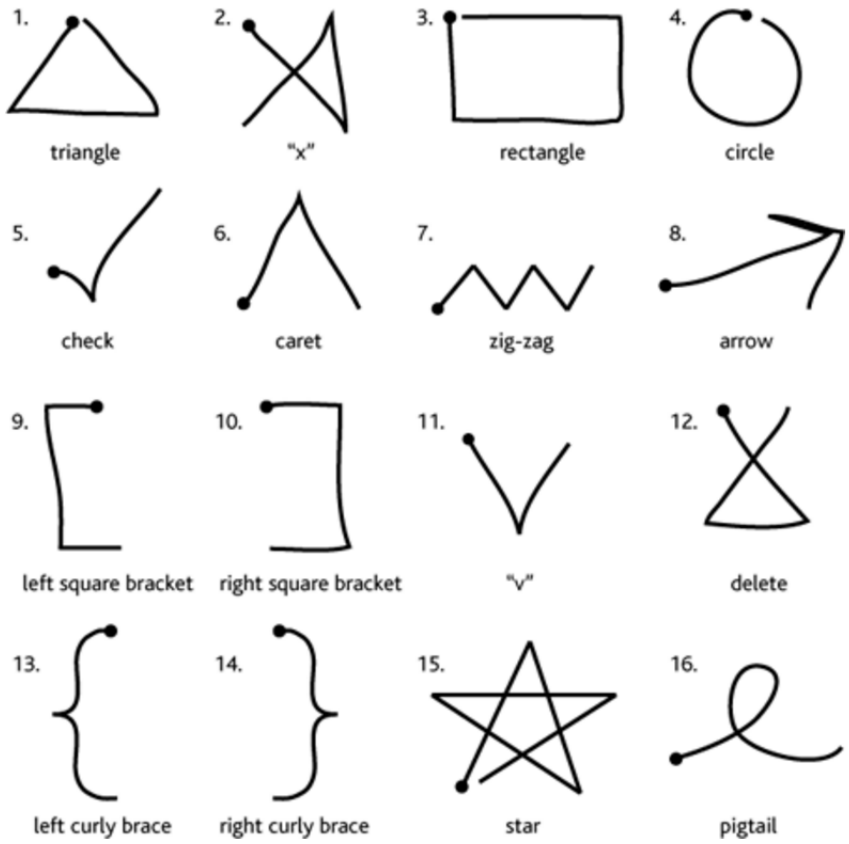


Figure 5.3: Single stroke shapes recognized by $1 recognizer

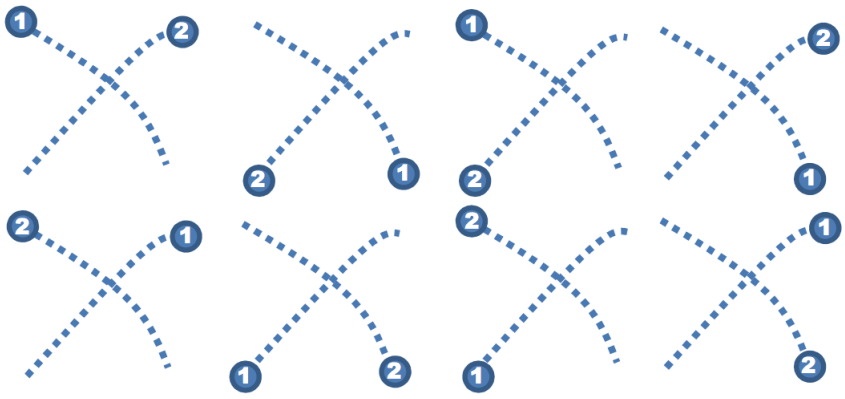

Figure 5.4: The eight different ways an X can be drawn [2]

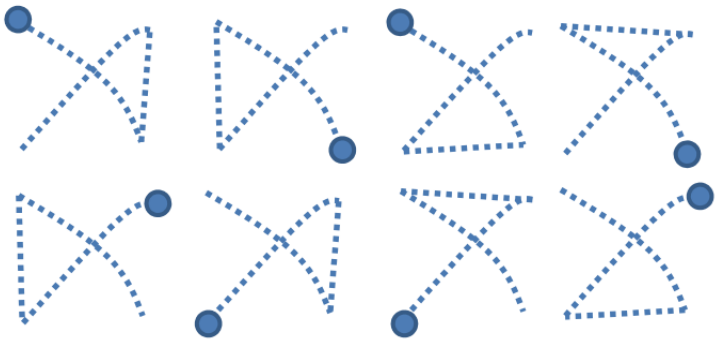

Figure 5.5: Based on the two-strokes in Figure 5.4, there are eight permutations [2]

### *5.2.1 LADDER sketching language*

A sketching Language to Describe, Display, and Editing in Sketch Recognition (LADDER) [36, 45, 38, 39, 40, 41, 42, 44, 51], implemented by Hammond describes how sketches from different domains are drawn, displayed, and edited. In LADDER, sketches are broken down into substrokes using corner-finding algorithms. These substrokes and their combinations are recognized using geometric mathematical definitions and perceptual rules [82, 83, 84]. Simple sketch-based applications [18, 21] use these primitives on their own.

Different applications that use LADDER are Mechanix [4, 5, 28, 33, 63, 68, 79, 126, 129], which recognizes shapes like trusses, and works by Taele [110, 111, 122, 119, 120, 112, 113, 114, 121, 115], for recognizing East Asian character sets.

Figure 5.6 shows how a Chinese character is recognized. The definitions and constraints describing it consists of a horizontal line and a vertical line are shown.

## 5.3 Vision-based features

Vision-based sketch recognition is introduced by Kara [60] and Valentine [28, 126]. These techniques use sketch ink data by converting the sketch into a $40 \times 40$ grid of pixels as seen in Figure 5.7.

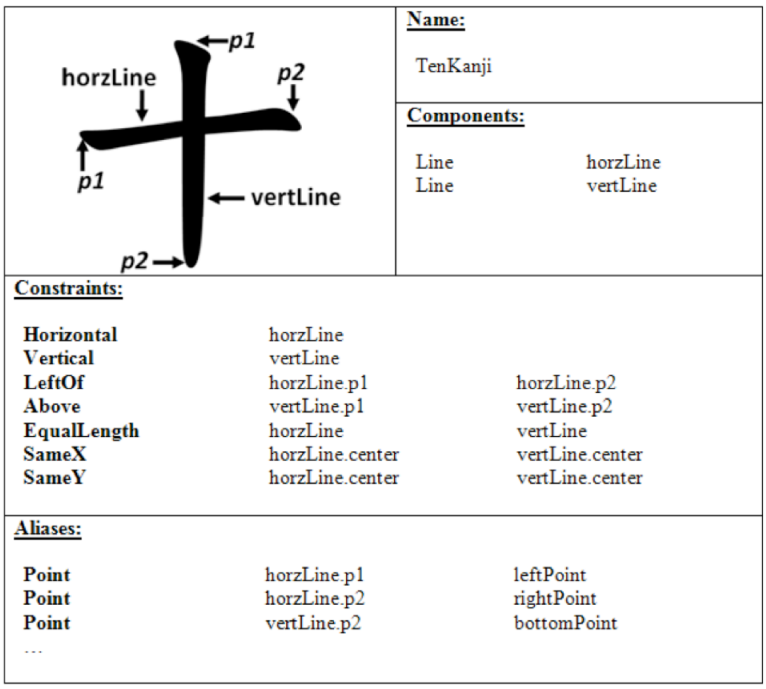


Figure 5.6: A shape description for an East Asian character

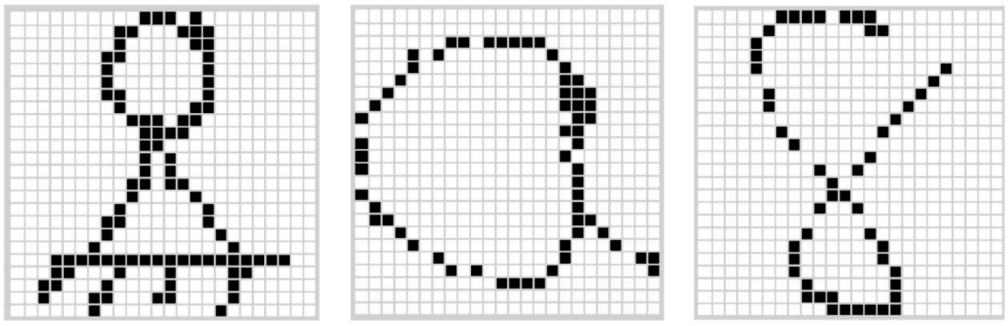

Figure 5.7: The pixelated versions of a pivot, an alphabet, and a digit [60]

An incoming sketch is compared to a previously classified shape using template-based techniques in both Valentine and Kara recognizers. Both techniques translate an incoming sketch into fixed-size pixel boxes: $40 \times 40$ in Valentine's recognizer [125] and $48 \times 48$ in Kara's recognizer [60]. In my system, both the template and a shape are resampled into forty equidistant points and distance between closest points in calculated. From these shortest distances, I calculated similarity metrics. The three metrics are ratio of points with shortest distances less than four pixels over the total number of points (i.e., the Tanimoto coefficient) [28, 60, 61, 126, 137], the maximum of distances (Hausdorff distance), and the average of distances (modified Hausdorff distance). These three are normalized and averaged to get the final similarity confidence value. If the confidence value is above a threshold of 0.65, they are classified as similar [28, 60, 125]. This system employs only the Tanimoto coefficient as it gave the best accuracy. The similarity between two images is calculated using Tanimoto coefficient by:

$$T(A, B) = \frac{n_{ab}}{n_a + n_b - n_{ab}} \tag{5.1}$$

where $n_a$ is the total number of black pixels in A, $n_b$ is the total number of black pixel in B, and $n_{ab}$ is the number of overlapping black pixels in A and B. T(A,B) describes the number of matching points in A and B, and the result is between 0.0 (minimum similarity) to 1.0 (highest similarity). The problem of this equation is that if images contain mostly black pixels, the T(A,B) value can be vanished.

To solve the problem, the following equation is used:

$$T^c(A, B) = \frac{n_{oo}}{n_a + n_b - 2\ n_{ab} + n_{oo}} \tag{5.2}$$

where $n_{oo}$ is the number of matching white pixels.

The two equations can be combined to form the Tanimoto similarity coefficient:

$$T_{sr}(A, B) = \alpha T(A, B) + (1 - \alpha)T^c(A, B) \tag{5.3}$$

$\alpha$ is a weighting factor between 0.0 and 1.0.

## 5.4 Statistical feature-based recognition

In this section, we look into the different statistical feature-based recognition techniques.

### *5.4.1 Paulson*

Paulson et al. [84] used the thirteen Rubine features and thirty-three new geometrical features, and introduced a hybrid approach. He examined 1800 sketches and derived an optimal set of fifteen features (fourteen geometric and one gesture-based).

Table 5.2 shows the list of all features with $\star$ indicating the set of optimal features selected through optimal feature selection techniques.

These studies [70, 83, 84, 102] focused on how sketches were drawn to retrieve the feature sets. We use a feature set of 133 features [30, 55, 70, 84] to improve recognition.

## 5.5 Extended uses of sketch recognition methods

Sketch recognition methods have been used to recognize what objects people were interacting with, based on the shape of their hands [85, 88], three-dimensional body movements [6, 92, 98], and musical instruments [77]. Taking inspiration from corner-finding techniques, Li recognized sketches based on sound [67].

Tables 5.3, 5.4, 5.5, 5.6, and 5.7 comprise of all the 133 features used in this research.

| | |
|---|---|
| 1.⋆ Endpoint by stroke length (100%) | 2.⋆ NDDE (90%) |
| 3.⋆ DCR (90%) | 4. Slope of the direction graph (20%) |
| 5. Maximum curvature (40%) | 6. Average curvature (30%) |
| 7. # of corners (30%) | 8. Line least squares error (0%) |
| 9. Line feature area error (40%) | 10. Arc fit: radius estimate (0%) |
| 11. Arc feature area error (20%) | 12.⋆ Curve least squares error (90%) |
| 13.⋆ Polyline fit: # of sub-strokes (70%) | 14.⋆ Polyline fit: percent of sub-strokes pass line test (50%) |
| 15.⋆ Polyline feature area error (80%) | 16. Polyline least squares error (30%) |
| 17. Ellipse fit: major axis length estimate (20%) | 18. Ellipse fit: minor axis length estimate (30%) |
| 19. Ellipse feature area error (10%) | 20. Circle fit: radius estimate (30%) |
| 21.⋆ Circle fit: major axis to minor axis ratio (80%) | 22. Circle feature area error (0%) |
| 23.⋆ Spiral fit: avg. radius/bounding box radius ratio (60%) | 24.⋆ Spiral fit: center closeness error (70%) |
| 25. Spiral fit: max distance between consecutive centers (20%) | 26. Spiral fit: average radius estimate (10%) |
| 27. Spiral fit: radius test passed (1.0 or 0.0) (40%) | 28.⋆ Complex fit: # of sub-fits (60%) |
| 29.⋆ Complex fit: # of non-polyline primitives (50%) | 30.⋆ Complex fit: percent of sub-fits that are lines (90%) |
| 31.⋆ Complex score / rank (50%) | 32. Cosine of the starting angle (30%) |
| 33. Sine of the starting angle (10%) | 34. Length of bounding box diagonal (20%) |
| 35. Angle of the bounding box diagonal (40%) | 36. Distance between endpoints (10%) |
| 37. Cosine of angle between endpoints (0%) | 38. Sine of angle between endpoints (10%) |
| 39. Total stroke length (20%) | 40.⋆ Total rotation (100%) |
| 41. Absolute rotation (10%) | 42. Rotation squared (10%) |
| 43. Maximum speed (20%) | 44. Total time (30%) |

Table 5.2: Paulson [84] feature set

| Number | Feature |
|---|---|
| a0 | Average distance between closest point to each corner of the bounding box [84] |
| a1 | Average curvature of the stroke [84] |
| a2 | The error of the best fit line of the direction graph [84] |
| a3 | Direction change ratio [84] |
| a4 | Get the distance (normalized by bounding box size) between the furthest corner and the stroke [84] |
| a5 | The endpoint to stroke length ratio of the stroke [84] |
| a6 | The angle of the major axis relative to center [84] |
| a7 | The error of the best fit line of the direction graph [84] |
| a8 | Max distance between closest point and each corner [84] |
| a9 | The error of the best fit line of the direction graph [84] |
| a10 | The maximum curvature to average curvature value [84] |
| a11 | Minimum distance between closest point and each corner [84] |
| a12 | The normalized distance between direction extremes [84] |
| a13 | Computes the sum of the squared values of the angles at each mouse point [30] |
| a14 | The number of revolutions (based on direction graph) that the stroke makes [84] |
| a15 | Percentage of Direction Window Passed [84] |
| a16 | Slope of the direction graph [84] |
| a17 | Standard deviation between closest point and each corner [84] |
| a18 | Length of the stroke [84] |
| a19 | The error of the line fit [84] |
| a20 | The least squares error of the fit / stroke length [84] |
| a21 | Get the error of the arc fit [84] |
| a22 | The estimated radius of the arc [84] |
| a23 | Get the arc to area ratio [84] |
| a24 | The angle between the endpoint [84] |
| a25 | Curve Error [84] |
| a26 | Get the error of the poliline fit [84] |
| a27 | Number of strokes [84] |
| a28 | Get the percentage of substrokes that passed a line test [84] |

Table 5.3: Complete feature list 1

| Number | Feature |
|---|---|
| a29 | Get the error of the elipse fit [84] |
| a30 | Get the error of the circle fit [84] |
| a31 | Get the major axis to minor axis length ratio [84] |
| a32 | Get the error of the spiral fit [84] |
| a33 | Get the percentage of the radius test that passed [84] |
| a34 | Get the average radius to bounding box radius ratio [84] |
| a35 | Get the max distance from a point to the center divided by the average radius [84] |
| a36 | Distance between two points that meet at the head [84] |
| a37 | Size difference between last two strokes (the head) [84] |
| a38 | Number of intersections between the ends of the head and the shaft [84] |
| a39 | The error of the rectangle fit [84] |
| a40 | Get the major axis to bounding box diagonal length ratio [84] |
| a41 | Perimeter (of bounding box) to stroke length ratio [84] |
| a42 | Number of segmented strokes [84] |
| a43 | Stroke length to perimeter (of bounding box) ratio [84] |
| a44 | Get the width to height ratio of the square [84] |
| a45 | Get the error of the diamond fit [84] |
| a46 | Get the perimeter (of bounding box) to stroke length ratio [84] |
| a47 | Get the major axis to bounding box diagonal length ratio [84] |
| a48 | Get the width to height ratio of the square fit used for the diamond fit [84] |
| a49 | Stroke density [84] |
| a50 | Height to width ratio of bounding box [84] |
| a51 | Wave segment size [84] |
| a52 | Get the percentage of the slope test that passed [84] |
| a53 | Get the ratio between the smallest and largest segment of the wave segmentation [84] |
| a54 | Get the ratio between the smallest segment and the sum [84] |
| a55 | Get the angle between the middle segments [84] |

Table 5.4: Complete feature list 2

| Number | Feature |
|---|---|
| a56 | Get the percentage of the horizontal alignment test that passed [84] |
| a57 | Get average slope of first and last segment [84] |
| a58 | Get percentage of slope test that passed [84] |
| a59 | Ratio between largest and smallest segment [84] |
| a60 | Density of sub dot [84] |
| a61 | Number of revolutions of sub dot [84] |
| a62 | Convex hull area / bounding box area [30] |
| a63 | Convex hull area / enclosing rectangle area [30] |
| a64 | Largest quadrilateral area / convex hull area [30] |
| a65 | Largest quadrilateral area / enclosing rectangle area [30] |
| a66 | Largest triangle area / bounding box area [30] |
| a67 | Largest triangle area / convex hull area [30] |
| a68 | Largest triangle area / enclosing rectangle area [30] |
| a69 | Largest triangle area / largest quadrilateral area [30] |
| a70 | Absolute value of bounding box's Y difference / bounding box's X difference [30] |
| a71 | Enclosing rectangle's distance ratio [30] |
| a72 | Absolute value of bounding box's X difference / x value movement in sketch [30] |
| a73 | Number of points inside the triangle [30] |
| a74 | Convex hull area$^2$ / convex hull area [30] |
| a75 | Convex hull perimeter / stroke length [30] |
| a76 | Convex hull perimeter / bounding box perimeter [30] |
| a77 | Convex hull perimeter / enclosing rectangle perimeter [30] |
| a78 | Largest quadrilateral perimeter / convex hull perimeter [30] |
| a79 | Largest quadrilateral perimeter / enclosing rectangle perimeter [30] |
| a80 | Largest triangle perimeter / bounding box perimeter [30] |
| a81 | Largest triangle perimeter / convex hull perimeter [30] |
| a82 | Largest triangle perimeter / enclosing rectangle perimeter [30] |
| a83 | Largest triangle perimeter / quadrilateral perimeter [30] |
| a84 | Stroke Length / convex hull perimeter [30] |

Table 5.5: Complete feature list 3

| Number | Feature |
|---|---|
| a85 | Difference of bounding boxes' largest Y value and smallest Y value / y value movement in sketch [30] |
| a86 | Zernike1 [55] |
| a87 | Zernike2 [55] |
| a88 | Zernike3 [55] |
| a89 | Zernike4 [55] |
| a90 | Zernike5 [55] |
| a91 | Zernike6 [55] |
| a92 | Zernike7 [55] |
| a93 | Zernike8 [55] |
| a94 | Zernike9 [55] |
| a95 | Zernike10 [55] |
| a96 | Zernike11 [55] |
| a97 | Zernike12 [55] |
| a98 | Zernike13 [55] |
| a99 | Zernike14 [55] |
| a100 | Zernike15 [55] |
| a101 | Zernike16 [55] |
| a102 | Zernike17 [55] |
| a103 | Zernike18 [55] |
| a104 | Zernike19 [55] |
| a105 | Zernike20 [55] |
| a106 | Zernike21 [55] |
| a107 | Zernike22 [55] |
| a108 | Zernike23 [55] |
| a109 | Calculates the cosine of the initial angle [70] |
| a110 | Calculates the sine of the initial angle [70] |
| a111 | Get the length of the diagonal of the bounding box [70] |
| a112 | Get the angle of the diagonal of the bounding box [70] |

Table 5.6: Complete feature list 4

| Number | Feature |
|---|---|
| a113 | Calculates the distance between the first and last point [70] |
| a114 | Calculates the cosine between the first and last point [70] |
| a115 | Calculates the sine between the first and last point [70] |
| a116 | Computes the total gesture length [70] |
| a117 | Sum of angle changes between points [70] |
| a118 | Computes the sum of the absolute value of the angles at each mouse point [70] |
| a119 | Computes the sum of the squared values of the angles at each mouse point [70] |
| a120 | Calculates the gesture aspect, which is abs (45 degrees - angle of bounding box) [70] |
| a121 | The sum of gesture intersegment angles whose absolute value is less than 19 degrees [70] |
| a122 | Returns the total angle traversed / total length of gesture stroke [70] |
| a123 | A density metric for the gesture stroke that uses the stroke's length and distance between the first and last point [70] |
| a124 | A density metric for the gesture stroke that uses the stroke's length and bounding box size [70] |
| a125 | How "open" or spaced out is a gesture [70] |
| a126 | Get the area of the bounding box [70] |
| a127 | The log of the bounding box area [70] |
| a128 | Returns the total angle divided by the total absolute angle [70] |
| a129 | Log of the total length [70] |
| a130 | Log of the aspect [70] |
| a131 | Count of corners |
| a132 | Stroke length / bounding box area |

Table 5.7: Complete feature list 5

# 6. THE IMPROVED SYSTEM

This section will explain the process of designing and building the improved system. The process includes:

1. Preprocessing
2. Sketch feature extraction
3. Evaluation
4. Rank features
5. Feature selection
6. Classification using Weka

## 6.1 Preprocessing

During the user study, we collected electronic ink data. The data collected comprises of x- and y-coordinates, and timing information. Using the information, we first generated basic feature sets such as direction value changes, stroke length over time, and total stroke length. Algorithm 1 explains this procedure.

We calculated the 133 features [30, 55, 70, 84, 136] by combining the strokes in the sketch into one stroke using a combined-stroke algorithm 2.

## 6.2 Sketch feature extraction

The goal is to determine if sketch features and classifiers can distinguish between the large number of classes. There is a lack of research work into classifying sketches at this scale. As a result, this approach differentiates between many classes using the 133 sketch features introduced by Cali [30], Hse [55], Long [70], Paulson [84],

**Algorithm 1** Calculate the stroke direction value over time, stroke length over time, and total stroke length

**Input:** (1) *sketch*: the whole strokes on the screen and (2) *numberOfPoints*: the number of points in *sketch*,
**Output:** (1) *directionArray*: a stroke direction value over time, (2) *lengthArray*: a stroke length over time, and (3) *totalLength*: a total stroke length
**for** $i = 0$ ; $i <$ *numberOfPoints*-1 ; $i$++ **do**
  *directionArray* [$i$] = atan (angle whose tangent) values of X and Y values between $i$+1th point and $i$th point
  *lengthArray* [$i$] = sqrt (square root) values of X and Y values between $i$+1th point and $i$th point
  *totalLength* += *lengthArray* [$i$]
**end for**

**Algorithm 2** Combine strokes into one stroke

**Input:** *strokes* in sketch
**Output:** *newStroke*: a combined stroke
**if** size of *strokes* == 1 **then**
  return *strokes*.get(0)
**end if**
*currentStroke* = *strokes*.get(0)
*dist*1 = distance(*currentStroke*.getFirstPoint, *strokes*.get(1).getFirstPoint)
*dist*2 = distance(*currentStroke*.getFirstPoint, *strokes*.get(1).getLastPoint)
*dist*3 = distance(*currentStroke*.getLastPoint, *strokes*.get(1).getFirstPoint)
*dist*4 = distance(*currentStroke*.getLastPoint, *strokes*.get(1).getLastPoint)
*min* = minimum values between *dist*1, *dist*2, *dist*3, and *dist*4
**if** *min* == *dist*1 or *dist*2 **then**
  *newStroke* = add *currentStroke* to the *newStroke* in reverse order
**else**
  *newStroke* = add *currentStroke* to the *newStroke*
**end if**
**for** $i = 1$ ; $i <$ *strokes*.size() ; $i$++ **do**
  *currentStroke* = *strokes*.get($i$)
  *dist*1 = distance(*currentStroke*.getFirstPoint, *newStroke*.getLastPoint)
  *dist*2 = distance(*currentStroke*.getLastPoint, *newStroke*.getLastPoint)
  **if** *dist*1 > *dist*2 **then**
    *newStroke* = add *currentStroke* to the *newStroke* in reverse order
  **else**
    *newStroke* = add *currentStroke* to the *newStroke*
  **end if**
**end for**

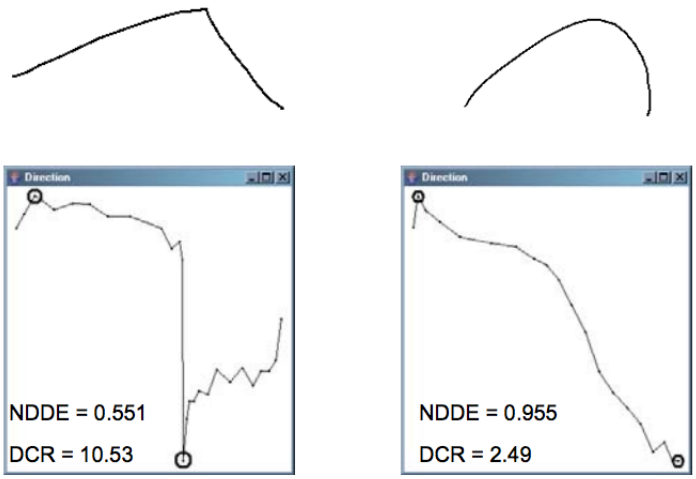


Figure 6.1: Using DCR to differentiate between polylines and arcs [83]

Rubine [102], and Wolin [136]. To show how these features work, We would like to highlight a couple of features, namely, *Direction Change Ratio* (DCR), and *Polyline Test*.

### *6.2.1 Direction change ratio for curves*

*Direction change ratio* (DCR) is used to differentiate between polylines and arcs. It looks for sudden changes in direction [84], and is calculated by dividing the maximum change in direction by average change in direction as seen in this DCR-calculating algorithm 3. Figure 6.1 shows that the polyline on the left has a sudden change in direction and hence resulted in high DCR value, whereas, the arc on the right changes direction smoothly and thus resulted in low DCR value.

$$DCR = \frac{\text{maximum change in direction}}{\text{average change in direction}} \tag{6.1}$$

**Algorithm 3** Calculate the *Direction Change Ratio* (DCR) (max direction change divided by average direction change)

**Input:** *directionArray* from Algorithm 1
**Output:** *DCR*
*maxDirectionChange* = 0
**for** *i* = 0 ; *i* < *directionArray.size*()-1 ; *i*++ **do**
  *tempDirectionChange* = math.abs(*directionArray* [*i*+1] - *directionArray* [*i*])
  **if** *tempDirectionChange* > *maxDirectionChange* **then**
    *maxDirectionChange* = *tempDirectionChange*
  **end if**
  *sumDirectionChange* += *tempDirectionChange*
**end for**
*averageDirectionChange* = *sumDirectionChange* / *directionArray.size*()
*DCR* = *maxDirectionChange* / *averageDirectionChange*

### *6.2.2 Polyline test for corners*

Sketches are broken down into substrokes using corner-finding algorithms [136]. The average least-square error of each substroke is measured, and the number of lines that pass the test [82, 83] are calculated, and is known as the polyline test. The process is explained in polyline-calculating algorithm 4.

$$PolylineTest = \frac{number\ of \text{ substrokes that passed line test}}{number\ of \text{ substrokes}} \tag{6.2}$$

## 6.3 Evaluation

### *6.3.1 Highly correlated data*

To reduce the dimensionality of the dataset, we used a wrapper approach [66]. This technique is used to remove features that are correlated with one another or are redundant. We used the R programming language [57] and Weka [34] for data analysis. Using R, we removed features with more than 0.75 correlation. Figure 6.2 shows

**Algorithm 4** Calculate the *Polyline Test* (the percentage of substrokes that passed a line test)

**Input:** segmented *strokes* from *sketch* using corner-detection algorithm in [136]
**Output:** *Polyline Test*
$numLinePassed = 0$
**for** $i = 0$ ; $i <$ *strokes*.size() ; $i$++ **do**
  $currentStroke =$ *strokes*.get($i$)
  $linePassed =$ decide if the *currentStroke* passed line test using algorithm from [82, 83]
  **if** $linePassed =$ True **then**
    $numLinePassed = numLinePassed + 1$
  **end if**
**end for**
$PolylineTest = numLinePassed$ / *strokes*.size

the resulting correlated matrix. The closer the correlation between two variables is to 1, the more related their behavior, and the more redundant one is with respect to the other. We filtered out the redundant features examining the correlation matrix.

### *6.3.2 Principal component analysis*

Correlation matrix analysis is done using Principal Component Analysis (PCA). Feature extraction approaches transform data in high-dimensional space to a space of fewer dimensions. PCA, an important linear technique for reducing dimensionality, performs a linear mapping of the data to a lower dimensional space in such a way that the variance of the data in the low-dimensional representation is maximized. In other words, PCA analysis builds a set of features by selecting those axes, which maximize data variance.

PCA is a multivariate technique that summarizes systematic patterns of variation in the data. From a data analysis standpoint, PCA is used for studying one table of observations and variables with the idea of transforming the observed variables into a set of new variables, the principal components, which are uncorrelated and explain

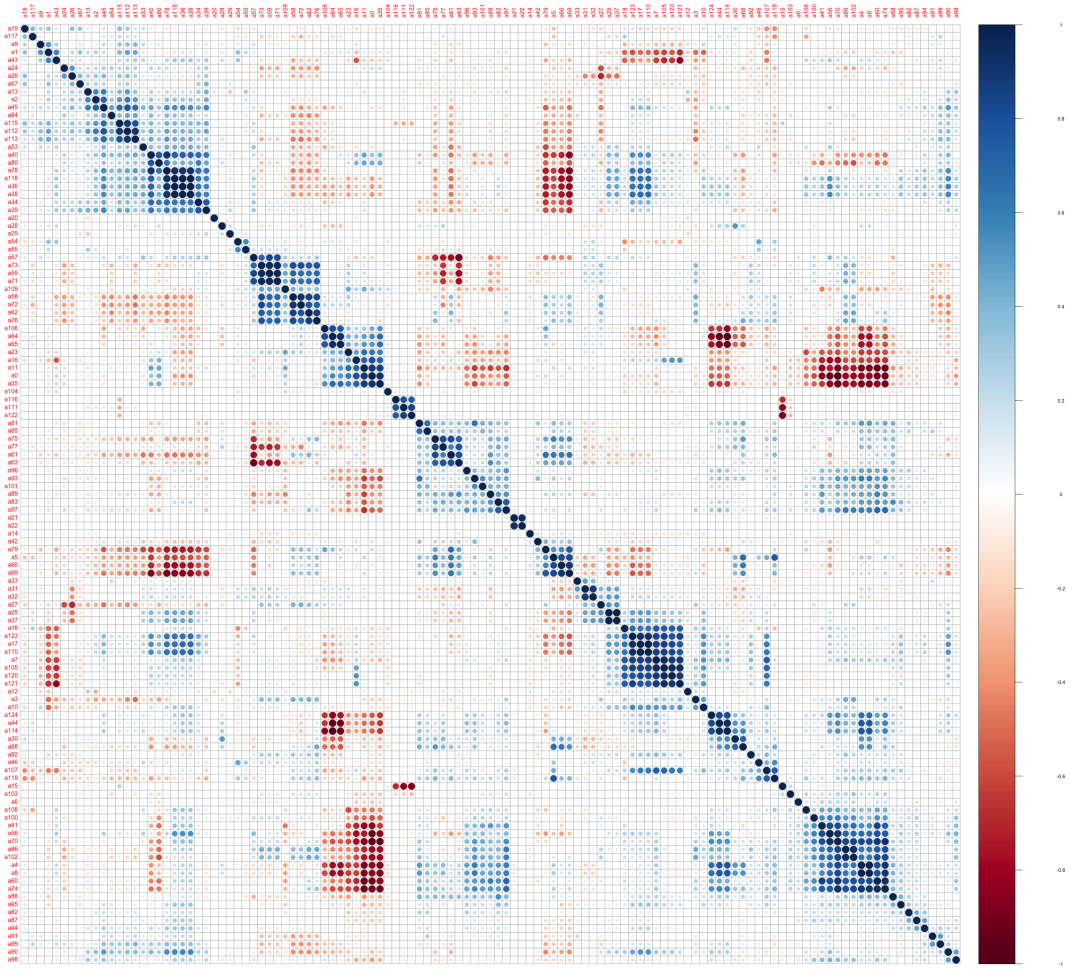

Figure 6.2: Highly correlate matrix for 125 features

the variation of the data. Therefore, PCA can be used to bring down a "complex" dataset to a lower dimensionality, in order to reveal the structures or the dominant types of variations in both the observations and the variables.

To verify if these features can be used to classify 491 classes, we performed PCA. Figure 6.3 shows the PCA plot. Each color represents a class and the grouping of similar colors together with not much overlapping shows that these seventy-two features can be used to differentiate between the 491 classes.

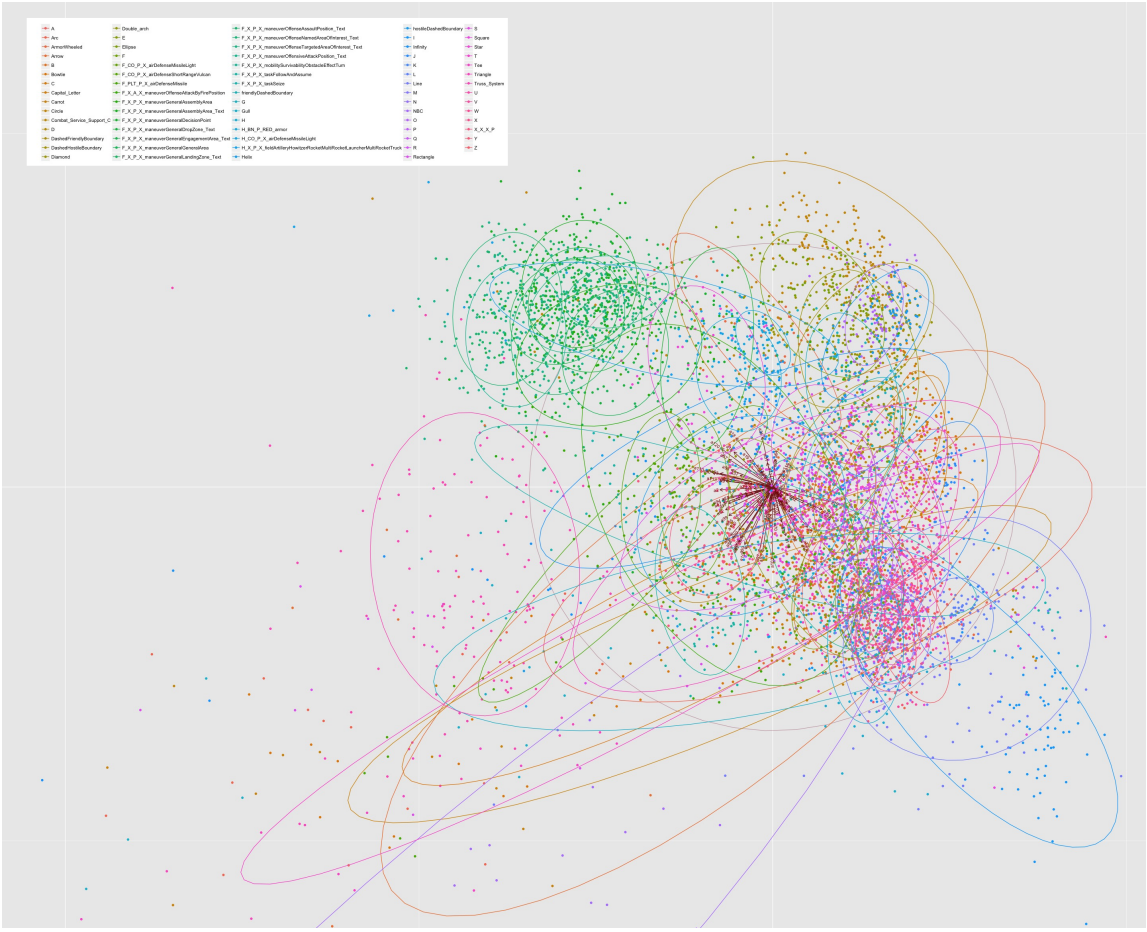

Figure 6.3: Principal component analysis plot

## 6.4 Rank features by importance

The importance of features can be estimated from data by building a model. Some methods like decision trees have a built-in mechanism to report on variable importance. For other algorithms, the importance can be estimated using a Receiver Operating Characteristic (ROC) curve analysis conducted for each attribute. We constructed a Learning Vector Quantization (LVQ) model on the data and estimated the variable importance, which is printed and plotted. Figure 6.4 shows the features and their importance for the arc class. The features are ranked based on their overall importance, across all 491 classes.

## 6.5 Feature elimination

Selecting the right features from the data can mean the difference between below-average performance with long training times and high performance with short training times. We used the caret R package to report the relevance and importance of attributes in the data, and select the most important features. Automatic feature selection methods can be used to build many models with different subsets of a dataset and identify those attributes that are required to build an accurate model. We used the Recursive Feature Elimination (RFE) method provided by the caret R package for feature selection. A Random Forest algorithm is used on each iteration to evaluate the model with 10-fold cross validation. This way, we explored all possible subsets of the attributes. The results show that sixty-nine attributes are selected, although in Figure 6.5, showing the accuracy of the different attribute subset sizes, we can see that only eighteen attributes give nearly comparable results.

Figure 6.6 shows the list of features: accuracy, kappa, accuracy standard deviation, and the kappa standard deviation. At the bottom, you can see the ranking of features. This figure shows that with these sixty-nine features, we can achieve an

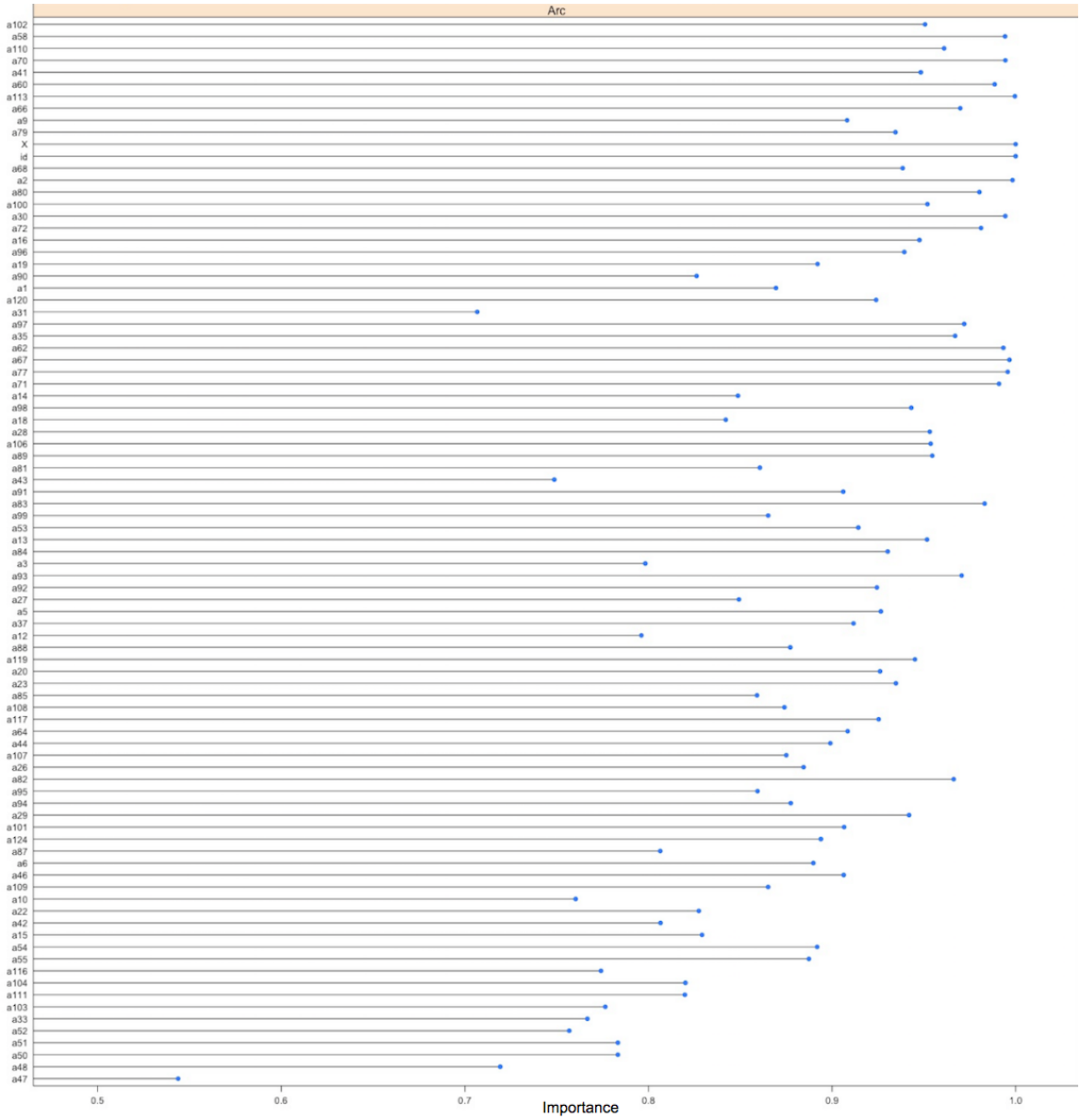


Figure 6.4: Rank of features by importance

accuracy of 85.15% on the entire dataset.

## 6.6 Classification using weka

We divided the data into training and test datasets with ten instances for each class as seen in Figure 6.7. We created models using the training dataset and per-

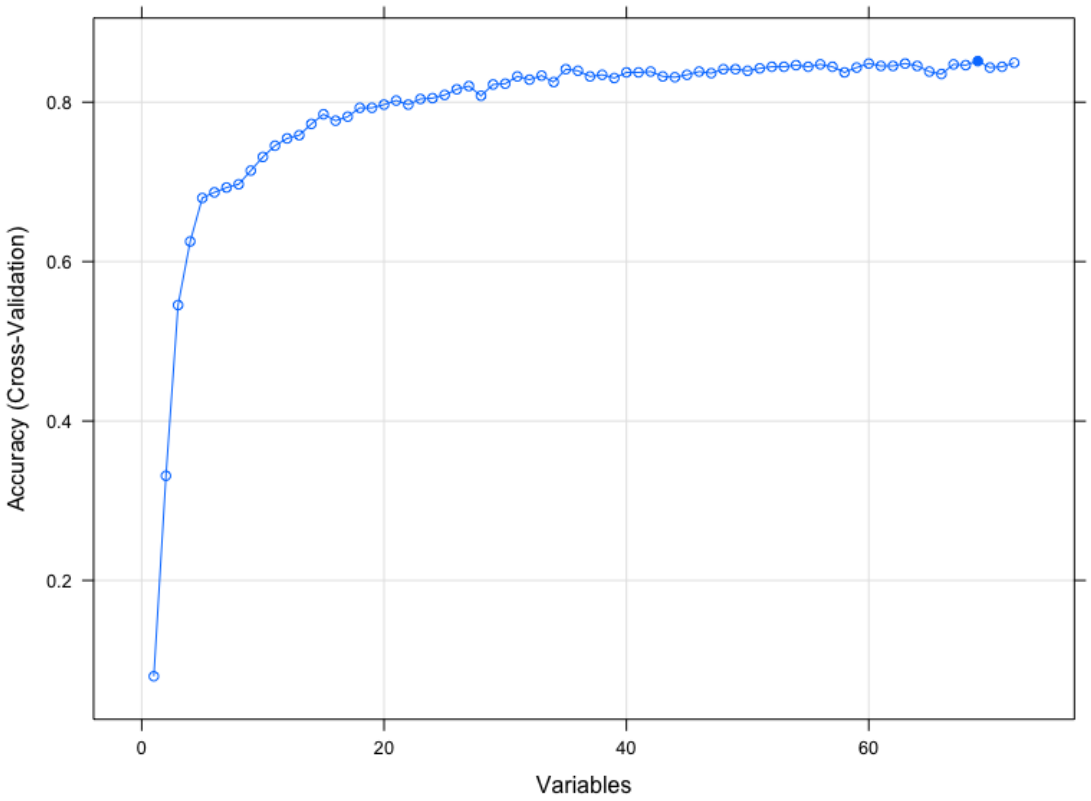


Figure 6.5: Feature selection plot

fomed 10-fold cross-validation using seven classifiers: Random Forest, Bagging, Naive Bayesian, Multi Class, Bayesian networks, J48, and Random Tree in Weka [34]. Table 6.1 shows the list of classifiers used and the accuracies achieved. The Random Forest classifier performed better than other classifiers with ten-fold cross-validation.

We created the model and integrated it into the system. Therefore, the first step is to use this model, in order to classify the incoming test data. We tested the system using the test dataset. The tests resulted in an accuracy of 98% for user specific methods. The sketches with confidence value less than 0.65 are sent to the next

```
Recursive feature selection

Outer resampling method: Cross-Validated (10 fold)

Resampling performance over subset size:

 Variables Accuracy   Kappa AccuracySD KappaSD Selected
         1   0.0798 0.07041    0.02153 0.02175
         2   0.3313 0.32449    0.03713 0.03751
         3   0.5455 0.54082    0.04096 0.04138
         4   0.6253 0.62143    0.04578 0.04625
         5   0.6798 0.67653    0.04763 0.04811
         6   0.6869 0.68367    0.04441 0.04487
         7   0.6929 0.68980    0.05874 0.05934
         8   0.6970 0.69388    0.04994 0.05045
         9   0.7141 0.71122    0.05018 0.05069
        10   0.7313 0.72857    0.04522 0.04568
        11   0.7455 0.74286    0.03620 0.03657
        12   0.7545 0.75204    0.04739 0.04787
        13   0.7586 0.75612    0.04081 0.04123
        14   0.7727 0.77041    0.04345 0.04389
        15   0.7848 0.78265    0.03266 0.03299
        16   0.7768 0.77449    0.03348 0.03383
        17   0.7818 0.77959    0.03130 0.03162
        18   0.7929 0.79082    0.03475 0.03510
        19   0.7929 0.79082    0.03442 0.03477
        20   0.7970 0.79490    0.03763 0.03801
        21   0.8020 0.80000    0.03755 0.03794
        22   0.7970 0.79490    0.03028 0.03059
        23   0.8040 0.80204    0.03570 0.03606
        24   0.8051 0.80306    0.04715 0.04763
        25   0.8091 0.80714    0.04478 0.04524
        26   0.8162 0.81429    0.03589 0.03625
        27   0.8202 0.81837    0.03360 0.03395
        28   0.8081 0.80612    0.02896 0.02926
        29   0.8222 0.82041    0.03664 0.03701
        30   0.8232 0.82143    0.03817 0.03856
        31   0.8323 0.83061    0.04622 0.04669
        32   0.8283 0.82653    0.04232 0.04275
        33   0.8333 0.83163    0.04103 0.04145
        34   0.8253 0.82347    0.03402 0.03437
        35   0.8414 0.83980    0.04365 0.04410
        36   0.8394 0.83776    0.03449 0.03484
        37   0.8323 0.83061    0.04421 0.04466
        38   0.8343 0.83265    0.03785 0.03824
        39   0.8303 0.82857    0.04333 0.04377
        40   0.8374 0.83571    0.04271 0.04315
        41   0.8374 0.83571    0.04136 0.04178
        42   0.8384 0.83673    0.03749 0.03788
        43   0.8323 0.83061    0.03570 0.03606
        44   0.8313 0.82959    0.02937 0.02967
        45   0.8343 0.83265    0.03725 0.03763
        46   0.8384 0.83673    0.03749 0.03788
        47   0.8364 0.83469    0.03222 0.03255
        48   0.8414 0.83980    0.03751 0.03789
        49   0.8414 0.83980    0.03533 0.03569
        50   0.8394 0.83776    0.04081 0.04123
        51   0.8424 0.84082    0.03130 0.03162
        52   0.8444 0.84286    0.03725 0.03763
        53   0.8444 0.84286    0.02825 0.02854
        54   0.8465 0.84490    0.03004 0.03035
        55   0.8444 0.84286    0.03473 0.03509
        56   0.8475 0.84592    0.03102 0.03134
        57   0.8444 0.84286    0.03166 0.03198
        58   0.8374 0.83571    0.03028 0.03059
        59   0.8434 0.84184    0.03571 0.03608
        60   0.8485 0.84694    0.02129 0.02151
        61   0.8455 0.84388    0.02566 0.02593
        62   0.8455 0.84388    0.02383 0.02408
        63   0.8485 0.84694    0.03467 0.03502
        64   0.8455 0.84388    0.02383 0.02408
        65   0.8384 0.83673    0.02777 0.02805
        66   0.8354 0.83367    0.02132 0.02154
        67   0.8475 0.84592    0.03314 0.03348
        68   0.8465 0.84490    0.02889 0.02918
        69   0.8515 0.85000    0.02132 0.02154        *
        70   0.8434 0.84184    0.02662 0.02689
        71   0.8444 0.84286    0.02981 0.03012
        72   0.8495 0.84796    0.02835 0.02864

The top 5 variables (out of 69):
   a109, a66, a117, a79, a35

> predictors(results)
 [1] "a109" "a66"  "a117" "a79"  "a35"  "a119" "a41"  "a71"  "a80"  "a16"  "a108" "a68"  "a23"  "a19"  "a34"  "a106" "a107" "a46"  "a18"  "a90"
[21] "a44"  "a30"  "a120" "a22"  "a84"  "a115" "a98"  "a124" "a67"  "a3"   "a2"   "a1"   "a24"  "a26"  "a53"  "a37"  "a31"  "a54"  "a89"  "a27"
[41] "a77"  "a100" "a55"  "a92"  "a10"  "a93"  "a91"  "a83"  "a81"  "a13"  "a33"  "a99"  "a116" "a6"   "a85"  "a15"  "a82"  "a103" "a87"  "a88"
[61] "a95"  "a94"  "a42"  "a28"  "a101" "a96"  "a12"  "a29"  "a9"
```

Figure 6.6: Feature selection results

stage of recognition, the previously existing system discussed in Chapter 4. Overall, the system achieved an accuracy of 86% on user independent data. Figure 6.8 shows

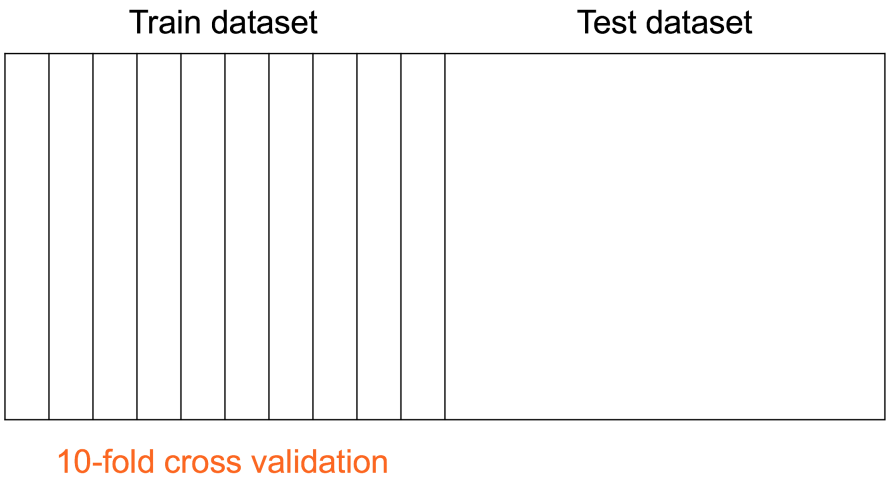


Figure 6.7: Division into test and train datasets

the confusion matrix prior to the addition and Figure 6.9 shows the confusion matrix after running the improvement.

| Classifier (Accuracy) |
|---|
| Random Forest (82.5253%) |
| Bagging (72.9293%) |
| Naive Bayesian (72.4242%) |
| Multi Class (71.4141%) |
| Bayesian networks (70.7071%) |
| J48 (62.2222%) |
| Random Tree (49.1919%) |

Table 6.1: Best classifiers and their accuracies

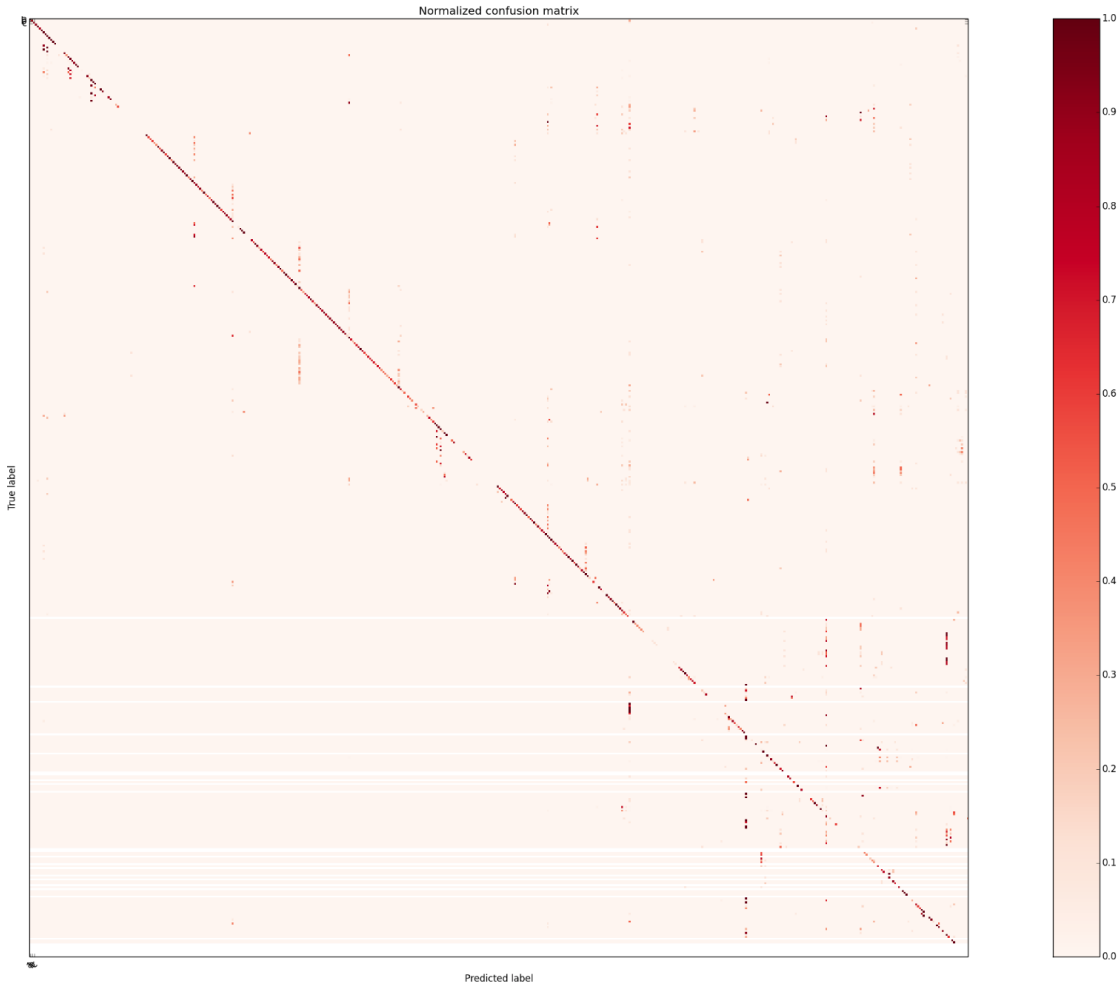


Figure 6.8: Confusion matrix prior to the improvement, achieving an accuracy of 57%

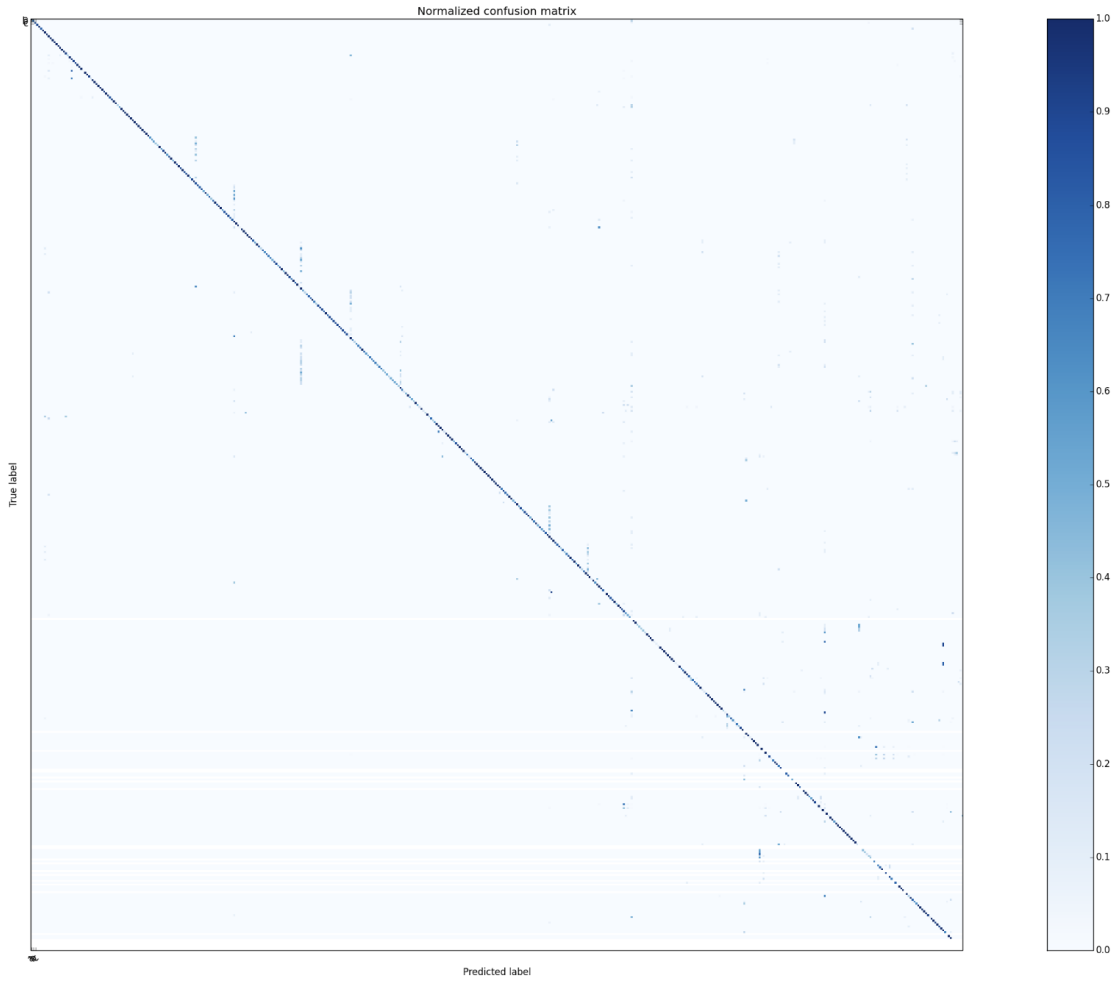


Figure 6.9: Confusion matrix after using the improvement, achieving an accuracy of 86%

# 7. FUTURE WORK

We are interested in implementing a similar technique to increase the number of classes. The entire list of approximately 20,000 Course of Action classes can be recognized if we collect data from different users from conducting further user studies. It is important to approach a wider variety of users, ranging from users with no experience in sketching to domain experts in the field. As we collect new data and re-train our model, we hope the proposed system will perform better in terms of recognition accuracy.

This work analyzed three different aspects of a sketch: the x-coordinates, the y-coordinates, and the time. Other attributes of a sketch can help contribute to better recognition of a sketch, such as a pressure-sensitive device to collect pressure and line weight-related features. These values can be used to derive some more features which can better the accuracy of our system. User studies that collect pressure and other features such as entropy [9] could be tested to determine areas of potential benefits for recognition.

# 8. CONCLUSION

This thesis describes the improvement of a system to now classify sketches into 491 different classes with an accuracy of 86%. There is a lack of research work into classifying sketches at this scale. Of the over 10,000 instances of data collected, we performed extensive data analysis to determine the features necessary for classification of sketches, and then ranked the features based on their importance to differentiate between these 491 classes.

The improved system expends negligible amount of time to classify incoming sketches on existing computing technology. This is comparatively much faster when compared to an older system [53] which takes two or three hours to classify sketches with a top three accuracy of 89%.

Lastly, with the number of possible sketch classes is very large such as course of action diagrams numbering around 20,000 different classes, the improved system can be extended to any number of classes that extends beyond the maximum recognizable classes from existing systems. As new data comes in, the model performs better over time in terms of accuracy without significant amount of performance in terms of time.

# REFERENCES


[1] Christine Alvarado, Tevfik Metin Sezgin, Dana Scott, Tracy Hammond, Zardosht Kasheff, Michael Oltmans, and Randall Davis. A framework for multi-domain sketch recognition. In *MIT Lab Abstract. Artificial Intelligence Laboratory*, Cambridge, MA, 9 2001. MIT. 3 pages.

[2] L Anthony and JO Wobbrock. A lightweight multistroke recognizer for user interface prototypes. In *Graphics Interface 2010*, pages 245–252, 2010.

[3] L Anthony and JO Wobbrock. $ n-protractor: A fast and accurate multistroke recognizer. In *Proceedings of Graphics Interface 2012*, pages 117–120. Canadian Information Processing Society, 2012.

[4] Olufunmilola Atilola, Martin Field, Erin McTigue, Tracy Hammond, and Julie Linsey. Mechanix: A sketch recognition truss tutoring system. In *ASME 2011 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages 645–654, Washington, DC, USA, 2011. American Society of Mechanical Engineers.

[5] Olufunmilola Atilola, Stephanie Valentine, Hong-Hoe Kim, David Turner, Erin McTigue, Tracy Hammond, and Julie Linsey. Mechanix: A natural sketch interface tool for teaching truss analysis and free-body diagrams. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 28:169–192, 5 2014.

[6] Joey Bartley, Jonathon Forsyth, Prachi Pendse, Da Xin, Garrett Brown, Paul Hagseth, Ashish Agrawal, Daniel W. Goldberg, and Tracy Hammond. World of workout: A contextual mobile rpg to encourage long term fitness. In *Pro-*

*ceedings of the Second ACM SIGSPATIAL International Workshop on the Use of GIS in Public Health*, HealthGIS '13, pages 60–67, New York, NY, USA, 2013. ACM.

[7] Rémi Bastide, David Navarre, Philippe Palanque, Amélie Schyn, and Pierre Dragicevic. A model-based approach for real-time embedded multimodal systems in military aircrafts. In *Proceedings of the 6th international conference on Multimodal interfaces*, pages 243–250. ACM, 2004.

[8] Luca Benedetti, Holger Winnemöller, Massimiliano Corsini, and Roberto Scopigno. Painting with bob: Assisted creativity for novices. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 419–428, New York, NY, USA, 2014. ACM.

[9] Akshay Bhat and Tracy Hammond. Using entropy to distinguish shape versus text in hand-drawn diagrams. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence*, IJCAI'09, pages 1395–1400, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

[10] Bill Buxton. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[11] KF Chan. Elastic structural matching for recognizing on-line handwritten alphanumeric characters. In *Technical Report HKUST-CS98-07*, pages 1–29, 1998.

[12] KF Chan and DY Yeung. Elastic structural matching for online handwritten alphanumeric character recognition. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 2, pages 1508–1511. IEEE, 1998.

[13] Hsiang Chen, R.T. Wigand, and M.S. Nilan. Optimal experience of web activities. *Computers in Human Behavior*, 15(5):585 – 608, 1999.

[14] Heeyoul Choi and Tracy Hammond. Sketch recognition based on manifold learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1786–1787, Chicago, Illinois, USA, 2008. AAAI Press.

[15] Heeyoul Choi, Brandon Paulson, and Tracy Hammond. Gesture recognition based on manifold learning. In *Proceedings of the 2008 Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, SSPR & SPR '08, pages 247–256, Berlin, Heidelberg, 2008. Springer-Verlag.

[16] PR Cohen, Liang Chen, Josh Clow, Michael Johnston, David McGee, Jay Pittman, and Ira Smith. Quickset: A multimodal interface for distributed interactive simulation. In *Proceedings of the UIST*, volume 96, pages 217–24. Citeseer, 1996.

[17] Paul Corey and Tracy Hammond. Gladder: Combining gesture and geometric sketch recognition. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1788–1789, Chicago, Illinois, 2008. AAAI Press.

[18] D Cummings, S Fymat, and T Hammond. Sketch-based interface for interaction with unmanned air vehicles. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, pages 1511–1516. ACM, 2012.

[19] Danielle Cummings, Stephane Fymat, and Tracy Hammond. Sketch-based interface for interaction with unmanned air vehicles. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, pages 1511–1516, New York, NY, USA, 2012. ACM.

[20] Danielle Cummings, Francisco Vides, and Tracy Hammond. I don't believe my eyes!: Geometric sketch recognition for a computer art tutorial. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, SBIM '12, pages 97–106, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association.

[21] D. Cummmings, S. Fymat, and T. Hammond. Reddog: A smart sketch interface for autonomous aerial systems. In *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, SBIM '12, pages 21–28, Aire-la-Ville, Switzerland, Switzerland, 2012. Eurographics Association.

[22] Katie Dahmen and Tracy Hammond. Distinguishing between sketched scribble look alikes. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1790–1791, Chicago, Illinois, USA, 2008. AAAI Press.

[23] Ruwanee de Silva, David Tyler Bischel, WeeSan Lee, Eric J. Peterson, Robert C. Calfee, and Thomas F. Stahovich. Kirchhoff's pen: A pen-based circuit analysis tutor. In *Proceedings of the 4th Eurographics Workshop on Sketch-based Interfaces and Modeling*, SBIM '07, pages 75–82, New York, NY, USA, 2007. ACM.

[24] Daniel Dixon, Manoj Prasad, and Tracy Hammond. icandraw: Using sketch recognition and corrective feedback to assist a user in drawing human faces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI 10, pages 897–906, New York, New York, USA, 2010. ACM.

[25] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Car-*

*tographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[26] Koos Eissen and Roselien Steur. *Sketching: Drawing Techniques for Product Designers*. BIS Publishers, Amsterdam, The Netherlands, 12 edition, 2009.

[27] BD Eoff and T Hammond. Who dotted that'i'?: Context free user differentiation through pressure and tilt pen data. In *Proceedings of Graphics Interface 2009*, pages 149–156. Canadian Information Processing Society, 2009.

[28] M Field, S Valentine, J Linsey, and T Hammond. Sketch recognition algorithms for comparing complex and unpredictable shapes. In *Proceedings of the Twenty-Second international Joint Conference on Artificial Intelligence (IJCAI)*, volume 3, pages 2436–2441. AAAI Press, 2011.

[29] Martin Field, Stephanie Valentine, Julie Linsey, and Tracy Hammond. Sketch recognition algorithms for comparing complex and unpredictable shapes. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Three*, IJCAI'11, pages 2436–2441. AAAI Press, 2011.

[30] M. J. Fonseca, C Pimentel, and J. A. Jorge. Cali: An online scribble recognizer for calligraphic interfaces. In *Sketch Understanding, Papers from the 2002 AAAI Spring Symposium*, pages 51–58, 2002.

[31] H Freeman. Computer processing of line-drawing images. *ACM Comput. Surv.*, 6(1):57–97, March 1974.

[32] Leslie Gennari, Levent Burak Kara, Thomas F Stahovich, and Kenji Shimada. Combining geometry and domain knowledge to interpret hand-drawn diagrams. *Computers & Graphics*, 29(4):547–562, 2005.

[33] M Green, B Caldwell, M Helms, J Linsey, and T Hammond. Using natural sketch recognition software to provide instant feedback on statics homework (truss free body diagrams): Assessment of a classroom pilot. In *2015 ASEE Annual Conference and Exposition*, pages 26.1671.1–26.1671.12. ASEE, 2015.

[34] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: An update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[35] T Hammond and R Davis. Creating the perception-based ladder sketch recognition language. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, DIS '10, pages 141–150, New York, NY, USA, 2010. ACM.

[36] Tracy Hammond. Automatically generating sketch interfaces from shape descriptions. In *Proceedings of the 4th Annual MIT Student Oxygen Workshop*, page 4. MIT, 2004.

[37] Tracy Hammond. *Sketch Recognition: Algorithms and Applications*. Cambridge University Press, 2017. draft from March 1, 2016, publication forthcoming.

[38] Tracy Hammond and Randall Davis. Ladder: A language to describe drawing, display, and editing in sketch recognition. In *Proceedings of the International Joint Conference on Aritificial Intelligence (IJCAI)*, pages 461–467, 2003.

[39] Tracy Hammond and Randall Davis. Automatically transforming symbolic shape descriptions for use in sketch recognition. In *Proceedings of the 19th National Conference on Artifical Intelligence*, AAAI'04, pages 450–456. AAAI Press, 2004.

[40] Tracy Hammond and Randall Davis. Shady: A shape description debugger for use in sketch recognition. In *AAAI Fall Symposium on Making Pen-Based*

*Interaction Intelligent and Natural*, Arlington, VA, October 2004. AAAI. 7 pages.

[41] Tracy Hammond and Randall Davis. Testing shape descriptions by automatically translating them for use in sketch recognition. In *MIT Lab Abstract*, page 2. MIT, 2004.

[42] Tracy Hammond and Randall Davis. Ladder, a sketching language for user interface developers. In *Computers & Graphics*, volume 29:4, pages 518–532. Elsevier, 2005.

[43] Tracy Hammond and Randall Davis. Ladder, a sketching language for user interface developers. *Computers & Graphics*, 29(4):518–532, 2005.

[44] Tracy Hammond and Randall Davis. Interactive learning of structural shape descriptions from automatically generated near-miss examples. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*, IUI '06, pages 210–217, New York, NY, USA, 2006. ACM.

[45] Tracy Hammond and Randall Davis. Tahuti: A geometrical sketch recognition system for uml class diagrams. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[46] Tracy Hammond and Randall Davis. Ladder, a sketching language for user interface developers. In *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[47] Tracy Hammond, Brian Eoff, Brandon Paulson, Aaron Wolin, Katie Dahmen, Joshua Johnston, and Pankaj Rajan. Free-sketch recognition: Putting the chi in sketching. In *CHI '08 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '08, pages 3027–3032, New York, NY, USA, 2008. ACM.

[48] Tracy Hammond, Krzysztof Gajos, Randall Davis, and Howard E Shrobe. An agent-based system for capturing and indexing software design meetings. In *In Proceedings of the International Workshop on Agents in Design (WAID02*, volume 2, pages 203–218, 2002.

[49] Tracy Hammond, Drew Logsdon, Brandon Paulson, Joshua Johnston, Joshua Peschel, Aaron Wolin, and Paul Taele. A sketch recognition system for recognizing free-hand course of action diagrams. In *Innovative Applications of Artificial Intelligence*, page 17811786, July 11-15 2010.

[50] Tracy Hammond, Drew Logsdon, Joshua Peschel, Joshua Johnston, Paul Taele, Aaron Wolin, and Brandon Paulson. A sketch recognition interface that recognizes hundreds of shapes in course-of-action diagrams. In *CHI '10 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '10, pages 4213–4218, New York, NY, USA, 2010. ACM.

[51] Tracy Hammond and Brandon Paulson. Recognizing sketched multistroke primitives. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 1(1):1–34, 2011.

[52] Tracy Hammond, Manoj Prasad, and Daniel Dixon. Art 101: Learning to draw through sketch recognition. In *Proceedings of the 10th International Conference on Smart Graphics*, SG'10, pages 277–280, Berlin, Heidelberg, 2010. Springer-Verlag.

[53] Tracy Anne Hammond, Drew Logsdon, Brandon Paulson, Joshua Johnston, Joshua Peschel, Aaron Wolin, and Paul Taele. A sketch recognition system for recognizing free-hand course of action diagrams. In *Twenty-Second IAAI Conference*, pages 1781–1786, July 11-15 2010.

[54] Jay D. Helsel. *Reading Engineering Drawings Through Conceptual Sketching*. Glencoe/Mcgraw-Hill, Columbus, Ohio, USA, 1 edition, 1979.

[55] H Hse and R Newton. Sketched symbol recognition using zernike moments. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1 - Volume 01*, ICPR '04, pages 367–370, Washington, DC, USA, 2004. IEEE Computer Society.

[56] Emmanuel Iarussi, Adrien Bousseau, and Theophanis Tsandilas. The drawing assistant: Automated drawing guidance and feedback from photographs. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 183–192, New York, New York, USA, 2013. ACM.

[57] Ross Ihaka and Robert Gentleman. R: A language for data analysis and graphics. *Journal of Computational and Graphical Statistics*, 5(3):299–314, 1996.

[58] J. Johnston and T. Hammond. Computing confidence values for geometric constraints for use in sketch recognition. In *Proceedings of the Seventh Sketch-Based Interfaces and Modeling Symposium*, SBIM '10, pages 71–78, Aire-la-Ville, Switzerland, Switzerland, 2010. Eurographics Association.

[59] Jr. Joseph J. LaViola and Robert C. Zeleznik. Mathpad2: A system for the creation and exploration of mathematical sketches. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH 2004, pages 432–440, New York, New York, USA, 2004. ACM.

[60] LB Kara and TF Stahovich. An image-based, trainable symbol recognizer for hand-drawn sketches. In *Computers and Graphics*, pages 501–517, 2004.

[61] LB Kara and TF Stahovich. An image-based trainable symbol recognizer for sketch-based interfaces. In *AAAI Fall Symposium*, pages 99–105, 2004.

[62] Levent Burak Kara and Thomas F Stahovich. An image-based, trainable symbol recognizer for hand-drawn sketches. *Computers & Graphics*, 29(4):501–517, 2005.

[63] K Kebodeaux, M Field, and T Hammond. Defining precise measurements with sketched annotations. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, pages 79–86. ACM, 2011.

[64] Kourtney Kebodeaux, Martin Field, and Tracy Hammond. Defining precise measurements with sketched annotations. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 79–86, New York, NY, USA, 2011. ACM.

[65] Dae Hyun Kim and Myoung-Jun Kim. A curvature estimation for pen input segmentation in sketch-based modeling. *Computer-Aided Design*, 38(3):238–248, 2006.

[66] Ron Kohavi and George H John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1):273–324, 1997.

[67] W Li and T Hammond. Recognizing text through sound alone. In *Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 1481–1486. AAAI, 2011.

[68] W Li and T Hammond. Using scribble gestures to enhance editing behaviors of sketch recognition systems. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, pages 2213–2218. ACM, 2012.

[69] Wenzhe Li and Tracy Hammond. Using scribble gestures to enhance editing behaviors of sketch recognition systems. In *CHI '12 Extended Abstracts on*

*Human Factors in Computing Systems*, CHI EA '12, pages 2213–2218, New York, NY, USA, 2012. ACM.

[70] A. C. Long, J. A. Landay, L. A. Rowe, and J Michiels. Visual similarity of pen gestures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, pages 360–367, New York, NY, USA, 2000. ACM.

[71] G Lucchese, M Field, J Ho, R Gutierrez-Osuna, and T Hammond. Gesturecommander: Continuous touch-based gesture prediction. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, pages 1925–1930. ACM, 2012.

[72] George Lucchese, Martin Field, Jimmy Ho, Ricardo Gutierrez-Osuna, and Tracy Hammond. Gesturecommander: Continuous touch-based gesture prediction. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, pages 1925–1930, New York, NY, USA, 2012. ACM.

[73] Nic Lupfer, Martin Field, Andruid Kerne, and Tracy Hammond. sketchy: Morphing user sketches for artistic assistance. In *2011 Intelligent User Interfaces Workshop on Sketch Recognition*, page 4, Palo Alto, CA, February 13-16 2011. ACM.

[74] Michael Helms Julie S. Linsey Matthew G. Green, Benjamin W. Caldwell and Tracy Anne Hammond. Using natural sketch recognition software to provide instant feedback on statics homework (truss free body diagrams): Assessment of a classroom pilot. In *2015 ASEE Annual Conference and Exposition*, Seattle, Washington, June 2015. ASEE Conferences. https://peer.asee.org/25007.

[75] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media,

2013.

[76] Erik G Miller, Nicholas E Matsakis, and Paul A Viola. Learning from one example through shared densities on transforms. In *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, volume 1, pages 464–471. IEEE, 2000.

[77] Jace Miller and Tracy Hammond. Wiiolin: A virtual instrument using the wii remote. In *NIME*, pages 497–500, 2010.

[78] ME Mott, R Vatavu, SK Kane, and JO Wobbrock. Smart touch: Improving touch accuracy for people with motor impairments with template matching. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '16)*, 2016.

[79] T Nelligan, S Polsley, J Ray, M Helms, J Linsey, and T Hammond. Mechanix: A sketch-based educational interface. In *Proceedings of the 2015 ACM International Conference on Intelligent User Interfaces*, pages 53–56. ACM, 2015.

[80] Trevor Nelligan, Seth Polsley, Jaideep Ray, Michael Helms, Julie Linsey, and Tracy Hammond. Mechanix: A sketch-based educational interface. In *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion*, IUI Companion '15, pages 53–56, New York, NY, USA, 2015. ACM.

[81] Tom Y. Ouyang and Randall Davis. Chemink: A natural real-time recognition system for chemical drawings. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, IUI '11, pages 267–276, New York, NY, USA, 2011. ACM.

[82] B Paulson and T Hammond. A system for recognizing and beautifying low-level sketch shapes using ndde and dcr. In *ACM Symposium on User Interface Software and Technology (UIST)*, page 2. ACM, 2007.

[83] B Paulson and T Hammond. Paleosketch: Accurate primitive sketch recognition and beautification. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI '08, pages 1–10, New York, NY, USA, 2008. ACM.

[84] B Paulson, P Rajan, P Davalos, R Osuna, and T Hammond. What!?! no rubine features?: Using geometric-based features to produce normalized confidence values for sketch recognition. In *HCC Workshop: Sketch Tools for Diagramming (VL/HCC)*, pages 56–63, 2008.

[85] Brandon Paulson, Danielle Cummings, and Tracy Hammond. Object interaction detection using hand posture cues in an office setting. *Int. J. Hum.-Comput. Stud.*, 69(1-2):19–29, January 2011.

[86] Brandon Paulson, Brian Eoff, Aaron Wolin, Joshua Johnston, and Tracy Hammond. Sketch-based educational games: Drawing kids away from traditional interfaces. In *Proceedings of the 7th International Conference on Interaction Design and Children*, IDC '08, pages 133–136, New York, NY, USA, 2008. ACM.

[87] Brandon Paulson and Tracy Hammond. Marqs: Retrieving sketches learned from a single example using a dual-classifier. *Journal on Multimodal User Interfaces*, 2(1):311, 2008.

[88] Brandon Paulson and Tracy Hammond. Office activity recognition using hand posture cues. In *Proceedings of the 22Nd British HCI Group Annual Conference*

*on People and Computers: Culture, Creativity, Interaction - Volume 2*, BCS-HCI '08, pages 75–78, Swinton, UK, UK, 2008. British Computer Society.

[89] Brandon Paulson and Tracy Hammond. Paleosketch: Accurate primitive sketch recognition and beautification. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI '08, pages 1–10, New York, NY, USA, 2008. ACM.

[90] Brandon Paulson, Pankaj Rajan, Pedro Davalos, Ricardo Gutierrez-Osuna, and Tracy Hammond. What!?! no rubine features?: Using geometric-based features to produce normalized confidence values for sketch recognition. In *HCC Workshop: Sketch Tools for Diagramming (VL/HCC)*, page 5763, Herrsching am Ammersee, Germany, 9 2008. VL/HCC.

[91] Joshua M Peschel and Tracy Anne Hammond. Strat: A sketched-truss recognition and analysis tool. In *2008 International Workshop on Visual Languages and Computing (VLC) at the 14th International Conference on distributed Multimedia Systems (DMS)*, page 282287, Boston, MA, 9 2008. Knowledge Systems Institute.

[92] Joshua M. Peschel, Brandon Paulson, and Tracy Hammond. A surfaceless pen-based interface. In *Proceedings of the Seventh ACM Conference on Creativity and Cognition*, pages 433–434, New York, NY, USA, 2009. ACM.

[93] Beryl Plimmer and Tracy Hammond. Getting started with sketch tools. In *Proceedings of the 5th International Conference on Diagrammatic Representation and Inference*, Diagrams '08, pages 9–12, Berlin, Heidelberg, 2008. Springer-Verlag.

[94] Beryl Plimmer and Tracy Hammond. Workshop on sketch tools for diagramming. In *Proceedings of the 2008 IEEE Symposium on Visual Languages and*

*Human-Centric Computing*, VLHCC '08, pages 4–, Washington, DC, USA, 2008. IEEE Computer Society.

[95] Manoj Prasad and Tracy Hammond. Observational study on teaching artifacts created using tablet pc. In *CHI '12 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '12, pages 301–316, New York, NY, USA, 2012. ACM.

[96] Pankaj Rajan and T. Hammond. From paper to machine: Extracting strokes from images for use in sketch recognition. In *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM'08, pages 41–48, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[97] Pankaj Rajan, Paul Taele, and Tracy Hammond. Evaluation of paper-pen based sketching interface. In *Proceedings of the 16th International Conference on Distributed Multimedia Systems (DMS)*, pages 321–326, 2010.

[98] Vijay Rajanna, Patrick Vo, Jerry Barth, Matthew Mjelde, Trevor Grey, Cassandra Oduola, and Tracy Hammond. Kinohaptics: An automated, wearable, haptic assisted, physio-therapeutic system for post-surgery rehabilitation and self-care. *Journal of Medical Systems*, 40(3):1–12, 2015.

[99] Dwayne Raymond, Jeffrey Liew, and Tracy A Hammond. A vision for education: Transforming how formal systems are taught within mass lectures by using pen technology to create a personalized learning environment. In Tracy Hammond, Stephanie Valentine, Aaron Adler, and Mark Payton, editors, *The Impact of Pen and Touch Technology on Education*, pages 355–363. Springer International Publishing Switzerland, 2015.

[100] Dan Roam. *The Back of the Napkin: Solving Problems and Selling Ideas with Pictures*. Portfolio Publishing, London, England, United Kingdom, 1 edition,

2013.

[101] Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, 1958.

[102] D Rubine. Specifying gestures by example. In *Proceeding of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH 91, pages 329–337, 1991.

[103] Dean Rubine. Specifying gestures by example. In *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '91, pages 329–337, New York, NY, USA, 1991. ACM.

[104] Irene Schiferl. Both sides now: Visualizing and drawing with the right and left hemispheres of the brain. *Studies in Art Education*, 50(1):67–82, 2008.

[105] Tevfik Metin Sezgin, Thomas Stahovich, and Randall Davis. Sketch based interfaces: Early processing for sketch understanding. In *ACM SIGGRAPH 2006 Courses*, SIGGRAPH '06, New York, NY, USA, 2006. ACM.

[106] N Shahzad, B Paulson, and T Hammond. Urdu qaeda: Recognition system for isolated urdu characters. In *Proceedings of the IUI Workshop on Sketch Recognition, Sanibel Island, Florida*, 2009.

[107] K. Sjölén and A. MacDonald. *Learning Curves: An Inspiring Guide to Improve Your Design Sketch Skills*. KEEOS Design Books, 2011.

[108] Sheryl Sorby. Educational research in developing 3d spatial skills for engineering students. *International Journal of Science Education*, 31(3):459–480, Feb 2009.

[109] Thomas F Stahovich. Segmentation of pen strokes using pen speed. In *AAAI Fall Symposium Series*, pages 21–24, 2004.

[110] P Taele, L Barreto, and T Hammond. Maestoso: An intelligent educational sketching tool for learning music theory. In *The Twenty-Seventh Annual Conference on Innovative Applications of Artificial Intelligence at AAAI (IAAI 2015)*, pages 3999–4005. AAAI, 2015.

[111] P Taele and T Hammond. Chinese characters as sketch diagrams using a geometric-based approach. In *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing Workshop on Sketch Tools for Diagramming*, pages 74–82, 2008.

[112] P Taele and T Hammond. Initial approaches for extending sketch recognition to beyond-surface environments. In *CHI'12 Extended Abstracts on Human Factors in Computing Systems*, pages 2039–2044. ACM, 2012.

[113] P Taele and T Hammond. Adapting surface sketch recognition techniques for surfaceless sketches. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI)*, pages 3243–3244. AAAI Press, 2013.

[114] P Taele and T Hammond. Developing sketch recognition and interaction techniques for intelligent surfaceless sketching user interfaces. In *Proceedings of the Companion Publication of the 19th International Conference on Intelligent User Interfaces (IUI) Doctoral Consortium*, pages 53–55. ACM, 2014.

[115] P Taele, J Peschel, and T Hammond. A sketch interactive approach to computer-assisted biology instruction. In *Proceedings of the Workshop on Sketch Recognition at the 14th International Conference of Intelligent User Interfaces Posters (IUI)*. ACM, 2009.

[116] Paul Taele, Laura Barreto, and Tracy Hammond. Hashigo: A next-generation sketch interactive system for japanese kanji. In *Proceedings of the Twenty-First*

*Innovative Applications of Artificial Intelligence Conference*, IAAI '15, pages 153–158, Palo Alto, California, USA, 2009. AAAI.

[117] Paul Taele, Laura Barreto, and Tracy Hammond. Maestoso: An intelligent educational sketching tool for learning music theory. In *Proceedings of the Twenty-Seventh Innovative Applications of Artificial Intelligence Conference*, IAAI '15, pages 3999–4005, Palo Alto, California, USA, 2015. AAAI.

[118] Paul Taele and Tracy Hammond. Using a geometric-based sketch recognition approach to sketch chinese radicals. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1832–1833. AAAI Press, 2008.

[119] Paul Taele and Tracy Hammond. Hashigo: A next-generation sketch interactive system for japanese kanji. In *Proceedings of the Twenty-First Innovative Applications of Artificial Intelligence Conference (IAAI)*, page 153158, Pasadena, CA, 7 2009. AAAI.

[120] Paul Taele and Tracy Hammond. Lamps: A sketch recognition-based teaching tool for mandarin phonetic symbols i. *J. Vis. Lang. Comput.*, 21(2):109–120, April 2010.

[121] Paul Taele and Tracy Hammond. Enhancing instruction of written east asian languages with sketch recognition-based intelligent language workbook interfaces. In Tracy Hammond, Stephanie Valentine, Aaron Adler, and Mark Payton, editors, *The Impact of Pen and Touch Technology on Education*, pages 119–126. Springer Publishing Company, Incorporated, 1st edition, 2015.

[122] Paul Taele and Tracy Anne Hammond. A geometric-based sketch recognition approach for handwritten mandarin phonetic symbols i. In *2008 International Workshop on Visual Languages and Computing (VLC) at the 14th Interna-*

*tional Conference on distributed Multimedia Systems (DMS)*, Boston, MA, 9 2008. Knowledge Systems Instistute. 6 pages.

[123] Barbara Tversky. Visualizing thought. *Topics in Cognitive Science*, 3(3):499–535, 2011.

[124] David G Ullman, Stephen Wood, and David Craig. The importance of drawing in the mechanical design process. *Computers & graphics*, 14(2):263–274, 1990.

[125] S Valentine, F Vides, G Lucchese, D Turner, H Kim, W Li, J Linsey, and T Hammond. Mechanix: A sketch-based tutoring system for statics courses. In *Proceedings of the Twenty-Fourth Innovative Applications of Artificial Intelligence Conference (IAAI)*, pages 2253–2260. AAAI, 2012.

[126] Stephanie Valentine, Martin Field, Anne Smith, and Tracy Hammond. A shape comparison technique for use in sketch-based tutoring systems. In *2011 Intelligent User Interfaces Workshop on Sketch Recognition*, page 4. IUI, 2011.

[127] Stephanie Valentine, Raniero Lara-Garduno, Julie Linsey, and Tracy Hammond. Mechanix: A sketch-based tutoring system that automatically corrects hand-sketched statics homework. In *The Impact of Pen and Touch Technology on Education*, pages 91–103. Springer, 2015.

[128] Stephanie Valentine, Francisco Vides, George Lucchese, David Turner, Hong hoe Kim, Wenzhe Li, Julie Linsey, and Tracy Hammond. Mechanix: A sketch-based tutoring system for statics courses. In *Proceedings of the Twenty-Fourth Innovative Applications of Artificial Intelligence Conference*, IAAI '12, pages 2253–2260, Palo Alto, California, USA, 2012. AAAI.

[129] Stephanie Valentine, Francisco Vides, George Lucchese, David Turner, Hong-Hoe Kim, Wenzhe Li, Julie Linsey, and Tracy Hammond. Mechanix: A sketch-

based tutoring and grading system for free-body diagrams. *AI Magazine*, 34(1):55–66, 2013.

[130] R Vatavu, L Anthony, and J Wobbrock. Gestures as point clouds: A $ p recognizer for user interface prototypes. In *Proceedings of the 14th ACM international conference on Multimodal interaction*, pages 273–280. ACM, 2012.

[131] Francisco Vides, Paul Taele, Hong-Hoe Kim, Jimmy Ho, and Tracy Hammond. Intelligent feedback for kids using sketch recognition. In *ACM SIGCHI 2012 Conference on Human Factors in Computing Systems Workshop on Educational Interfaces, Software, and Technology*, Austin, TX, USA, 2012. ACM.

[132] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 159–168, New York, NY, USA, 2007. ACM.

[133] JO Wobbrock, AD Wilson, and Y Li. Gestures without libraries, toolkits, or training: A $1 recognizer for user interface prototypes. In *Proceeding of the 20th annual ACM symposium on User Interface Software and Technology*, UIST-07, pages 159–168, 2007.

[134] A. Wolin, B. Eoff, and T. Hammond. Shortstraw: A simple and effective corner finder for polylines. In *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM'08, pages 33–40, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[135] A. Wolin, B. Paulson, and T. Hammond. Sort, merge, repeat: An algorithm for effectively finding corners in hand-sketched strokes. In *Proceedings of the 6th Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '09, pages 93–99, New York, NY, USA, 2009. ACM.

[136] Aaron Wolin, Brian Eoff, and Tracy Hammond. Shortstraw: A simple and effective corner finder for polylines. In *Proceedings of the Fifth Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBIM'08, pages 33–40, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[137] Aaron Wolin, Brian Eoff, and Tracy Hammond. Search your mobile sketch: Improving the ratio of interaction to information on mobile devices. In *Proceedings of the Workshop on Sketch Recognition at the 14th International Conference of Intelligent User Interfaces (IUI)*, page 4. ACM, 2009.

[138] Aaron Wolin, Martin Field, and Tracy Hammond. Combining corners from multiple segmenters. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling*, SBIM '11, pages 117–124, New York, NY, USA, 2011. ACM.

[139] Aaron Wolin, Brandon Paulson, and Tracy Hammond. Eliminating false positives during corner finding by merging similar segments. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1836–1837. AAAI Press, 2008.

[140] Jun Xie, Aaron Hertzmann, Wilmot Li, and Holger Winnemöller. Portraitsketch: Face sketching assistance for novices. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 407–417, New York, New York, USA, 2014. ACM.