Report

Yinghan Ma

***Abstract*—In this project, the $1 gesture recognition algorithm is implemented. The code is implemented in Java and the basic recognition algorithm strictly follows the pseudocode which was given in the official paper of $1 [1]. The program comes with a graphic user interface which will allow users to draw their gestures on a canvas. Users can also add templates as well. The dataset used to test on the algorithm is 'Unistroke gesture logs' from the official website[2], which contains 11 users' (including a pilot user) data about 16 basic unistroke gestures within 3 different speeds. Based on the dataset, $1 has an average accuracy rate of 97.19% with only 1 loaded template and over 99% with more than 2 loaded templates. The average accuracy over 10 users from the dataset is 99.09%.**

***Index Terms*—Gesture recognition, unistrokes, strokes, recognition rates, user interfaces, rapid prototyping.**

I. Introduction

Mobile devices and touch screens become more and more popular nowadays. Thus, the implementation of fast and platform-independent gesture recognition algorithms becomes an important topic to study. $1 is one of the "Dollar family" algorithms that can meet the criteria. It is an algorithm that can recognize user's unistroke input with as few as only 1 template needed to load in before the use. $1 can also be implemented on any platform within any programming languages. No complicated machine learning or cluster algorithm is needed to understand the algorithm.

In this project, a dataset that contains 10 users' 10 different inputs of 16 unistroke gestures within 3 different speeds was used to test the accuracy and the performance of the algorithm. A user-dependent "random-100" test is used to test the algorithm. We tested on the effect of the number of templates on recognition error, the effect of speed on recognition error and the recognition error rate per user.

II. Dataset Details

The dataset used for the "random-100" test is directly downloaded from the official $1 website. The whole dataset contains information from 11 users. The first user is a pilot user, so the test is operated on user 2-11. Each user's input is classified into 3 groups based on the input speeds, which are slow, medium and high. For each speed, the dataset contains 16 different unistroke gestures, which are "arrow", "caret", "check", "circle", "delete_mark", "left_curly_brace", "left_sq_bracket", "pigtail", "question_mark", "rectangle", "right_curly_brace", "right_sq_bracket", "star", "triangle", "v", "x". Each user input the same gesture 10 time so each gesture type has 10 different versions.

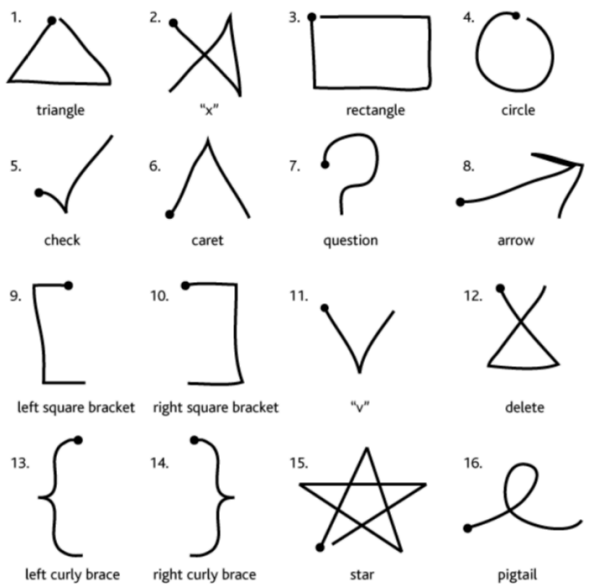

Figure 1 16 unistroke gestures used in the dataset [1]

III. Implementation Details

Java is used to implement $1 algorithm. The whole program contains two part. The first part is the user interface part. The second part is the testing part. Two directories were created in this project.

The first directory is called "$1", which contains all the recognition functions along with the canvas. Inside directory "$1", it contains Canvas.java, Point.java, Process.java, Recognizer.java, Result.java and Template.java. Canvas.java provided the functions for the graphic user interface. Java Swing was used in this project. A class called DrawPad extends the Jcomponent class in order to record the points and perform the recognition. Point.java contains a Point class, which contains the x and y coordinates of a point. A toString() method is also inside the Point class which can print out the information of the point easily. Process.java contains all four preprocess functions (resample, rotate to zero, scale to square and translate to zero). Besides these four main functions, there are also a few helper functions, for example, the pathDistance() function which will return the average path distance between two lists of points. A BoundBox class serves as a data structure to get the size of the bounding box faster and easier. The Result.java contains a Result class which contains the score and recognition

result name. This will make it easier for the recognizer function to return back result with multiple attributes. The constructor of the Recognizer class will take in a boolean variable called test as the parameter. If test is false, then it will pre-load the 16 unistroke gestures to a list of templates. Otherwise, it will keep the template list empty. The recognize() function is overloaded. The first one will take a list of points as parameter, which is the candidate gesture, and it will be preprocessed first and then compare with the templates inside the template list. Then it will return a Result object which contains its score and name. The second one will take two UserInput objects as parameters. This one is used for the "random-100" test. It will also return a Result object.

The second directory contains everything for the "Random-100" test. The subdirectory "xml_logs" contains all the dataset downloaded from the $1 website. Then the XmlParser.java contains the method to read in XML files and get the information. The Driver.java file contains the main function which will conduct the test. The bufferedwritter is used to write the result out to a file. The UserInput class serves as a data structure which will preprocess the point list of a gesture input and it also contains all the other information like name and user ID and speed of the gesture.

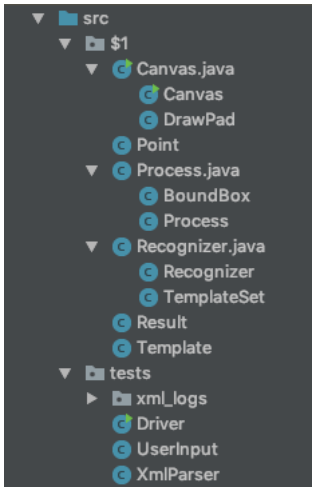

Figure 2 The code structure

IV. Offline Recognition Test Results

A "random-100" test is conducted on the algorithm (pseudo code is in the appendix). There are 10 users' input in total (user 2-11) and each user's input contains each gesture type in 10 different versions in three different speeds. As a result, there are 432,000 outputs in total. For a given subject at a given speed, we first set a value E which indicates the number of templates we choose from each type of gesture. The initial value of E is 1, then we increase 1 until E becomes 9 after we finish each round of test. At each level of E, 100 random-test is conducted by choose E random template(s) and one candidate from each gesture type (16 in total) and add them to a template set and a candidate set. Then we use the gestures inside the candidate set to compare with each gesture inside the template set and get the comparison score. Then we sort the template set based on its recognition score (descending order, also known as N-Best list).

We can then generate three different test results. Figure 3 shows the relationship between number of training samples/templates and the accuracy. As the figure shows, when number of templates is greater than 2, the accuracy can already reach more than 99%.

Figure 4 shows the relationship between the input speed and the recognition accuracy. Based on the result, the accuracy of medium speed is the highest.

Figure 5 demonstrates the per-user accuracy. User 11 has the largest error. However, the average accuracy is 99.09%, which proves that $1 is an algorithm with enough accuracy in most of the cases.

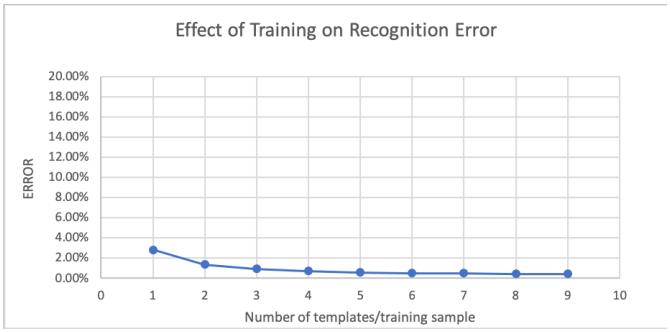

Figure 3 Recognition error rates as a function of templates or training (lower is better).

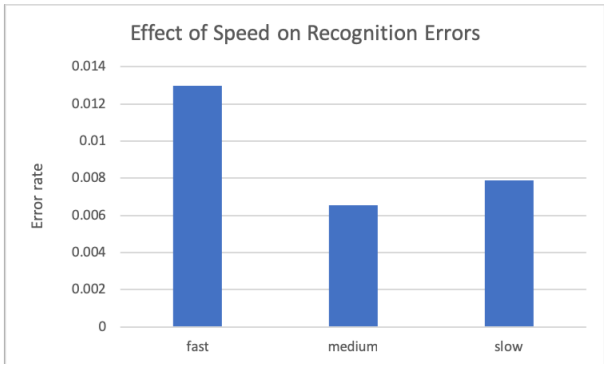

Figure 4 Recognition error rates as a function of articulation speeds (lower is better)

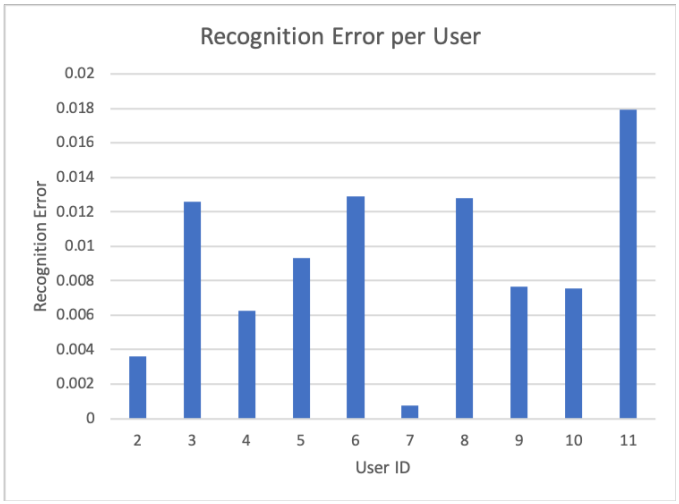

Figure 5 Recognition error rates per user (lower is better)

REFERENCES

1. Wobbrock, J.O., Wilson, A.D. and Li, Y. (2007). Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '07). Newport, Rhode Island (October 7-10, 2007). New York: ACM Press, pp. 159-168.
2. Depts.washington.edu. (2019). $1 Recognizer. [online] Available at: http://depts.washington.edu/madlab/proj/dollar/index.html [Accessed 21 Mar. 2019].

APPENDIX – RANDOM-100 TEST

```
for user u:= 1-> 100:
   for each speed s:=slow, medium, fast:
      for E := 1 -> 9:
         for i:= 1 -> 100:
            for each gesture type g (1 to 16):
               Randomly choose E templates from {u, s, g} and
               add them to a test set;
               Randomly choose one template which is outside
               of the test set and add it to a candidate set;
            for each candidate c in candidate set:
               for each template t in test set:
                  Recognize(c, t);
               end
            end
         end
      end
   end
end
```