HCIRA Project #1 Final Report

Sri Chaitanya Nulu

Abstract—In this project, the $1 Unistroke Recognizer Algorithm was implemented. For testing the algorithm, two different datasets were used: the Unistroke gesture logs dataset provided on the website, and a custom dataset consisting the gestures of five different users was collected. The total average scores computed were 0.985 for the Unistroke gesture logs data set, and 0.921 for the collected custom data set.

1 Introduction

In the current scenario, many people try to use Machine Learning for every single task. There are many ways one can recognize gestures in this way, but for that to happen, we need a lot of data. And collecting such large datasets or creating them is easier said than done. This project implements an algorithm called $1 Recognizer[1, 4]. It is simple to understand, doesn't need a lot of data (just one example per gesture), and gives accurate results.

This algorithm is completely implemented using Python language with the help of some external libraries like Tkinter[5] and PIL[6] to implement the GUI. For testing purposes, the Unistroke gesture logs dataset provided on the website[4], and a custom collected dataset (consisting of the gestures of five different users) were used. These datasets record some details about the gestures such as the number of points, time, points, etc. These datasets are then used to do offline testing. We also perform live testing where a user can draw the gesture, and the algorithm predicts the result immediately.

2 Related Work

This project was implemented by understanding the "$1 Recognizer for User Interface Prototypes" paper. It describes the major steps and provides the pseudo-code for all the functions. The implementation of the program is also provided in JavaScript on their website [4]. The $1 Recognition Algorithm has a set of templates for each supported gesture. And for the given test template, it processes the points and then tries to determine how close it is to the default templates set. This is done by computing the euclidean distance between them and choosing the shape that is closest to the given test template[1,4].

One of the tools called GECKo (GEsture Clustering toolKit) made it easier to understand how users draw the gestures from an input dataset. This tool visualizes the order, direction and shows how the gesture was drawn by providing a playback. It also helps to modify the clusters if necessary. This step was necessary after custom dataset collection because we need to check if all the gestures drawn by the users were drawn properly[2].

To identify the variation of the user-drawn gestures, GHoST (Gesture HeatmapS Toolkit) tool was used. This tool takes gesture datasets within a user or among different users and shows the articulation properties of the gestures via gesture heatmaps. These heatmaps are colored, and the color depends on how the variation is among the gestures drawn by the user(s). We can also see the gestures in the background if needed. This tool helped in getting several insights which are discussed in the other sections[3].

3 Dataset Details

3.1 $1 Unistroke Dataset

The first dataset that was used to perform the offline test was the Unistroke gesture logs dataset provided by the authors on the website [7]. This dataset consists of 16 different gestures from 11 different users. Each gesture is drawn 10 times at 3 different speeds. All these gestures are sorted into folders accordingly. For this project, the medium speed dataset of the last 10 users was used (The first user was already used initially as a template).

All the gesture files are recorded as XML files. They have information about the gesture shape, sample number, speed, unique user ID, time taken to draw the gesture, and all the points along the path of that particular gesture. During the offline recognition, each of these XML files is analyzed and a corresponding row with some set of output columns (discussed in the following sections) is written as output.

3.2 New Dataset Collected

Five different users were asked to draw gestures. They all used a stylus to draw all 16 gestures. Each user drew 10 samples for each gesture. The program's interface lets them choose any gesture in any order. These strokes were recorded in the same format as the Unistroke dataset[7], with some differences. This dataset didn't require users to draw gestures at a different speeds or records date and time.

The XML files saved details about the gesture name and sample number along with the unique user Id and the number of points. Then the points were written inside

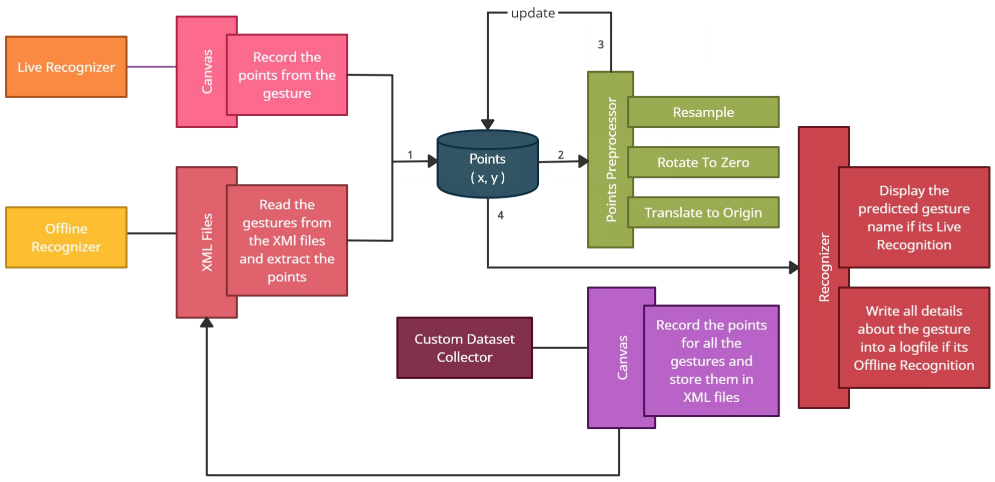

Live Recognizer
Canvas
Record the points from the gesture
Offline Recognizer
XML Files
Read the gestures from the XMl files and extract the points
update
1
2
3
4
Points (x, y)
Points Preprocessor
Resample
Rotate To Zero
Translate to Origin
Recognizer
Display the predicted gesture name if its Live Recognition
Write all details about the gesture into a logfile if its Offline Recognition
Custom Dataset Collector
Canvas
Record the points for all the gestures and store them in XML files

Figure 1: System Architecture Diagram

the file. These files were processed just like how the Unistroke Dataset was processed by the offline recognition module of the program.

4 Implementation Details

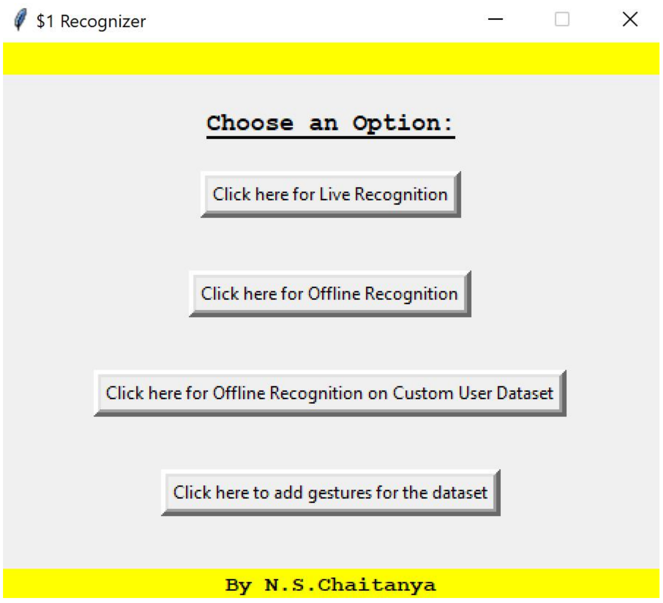
4.1 Language and Code Structure

As mentioned earlier, the complete program is written using Python. There is a canvas implemented in the program, which allows users to draw shapes over it. The program is divided into the following modules[1,4] (also see Figure 1):

- **Points Preprocessor:**
  All the points recorded on the canvas are sent here for preprocessing. There are three steps in this module:
  - First, we resample the points so that there are only 64 equidistant points.
  - Then we rotate the whole shape such that the indicative angle is at 0 degrees.
  - Finally, we scale all the points so that the sizes match and then translate to origin.
- **Gesture Recognizer:**
  It takes the preprocessed points and then compares them with all the templates. It tries to find which template is closest to this set of points based on the euclidean distance.
- **Online Recognizer:**
  The program displays the canvas to the user where they try to draw a gesture. These points are recorded once the user finishes drawing them and sent to preprocessing. Then it finds the best match from the templates and displays the shape name with accuracy and time on to the screen.
- **Offline Recognizer:**
  It scans the specified folders in the path, and reads all the XML files one by one. It predicts the shapes and saves it along with many other details (user id, n-best list, etc.) into a log file (csv format). We repeat the recognition process by increasing the number of training sets for each gesture from 1 to 9. This module iterates for 100 times and the accuracy is then averaged.
- **Custom Dataset Collector:**
  It provides interface for the user to draw all the gestures. Once the points are recorded, they are written into an XML file with details like shape name, sample number, number of points, time taken to draw, and all the points.

4.2 Runnable Components

Figure 2 shows the default interface of the program. There are four different buttons: Live Recognition , Offline Recognition, Offline Recognition on Custom Dataset, and Custom Dataset Collection[1,4].


$1 Recognizer

Choose an Option:

Click here for Live Recognition

Click here for Offline Recognition

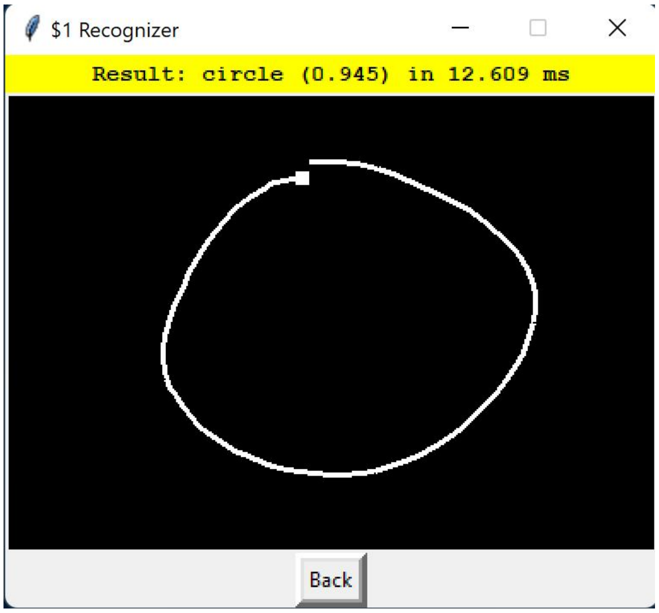
Click here for Offline Recognition on Custom User Dataset

Click here to add gestures for the dataset

By N.S.Chaitanya

Figure 2: Default Program Interface

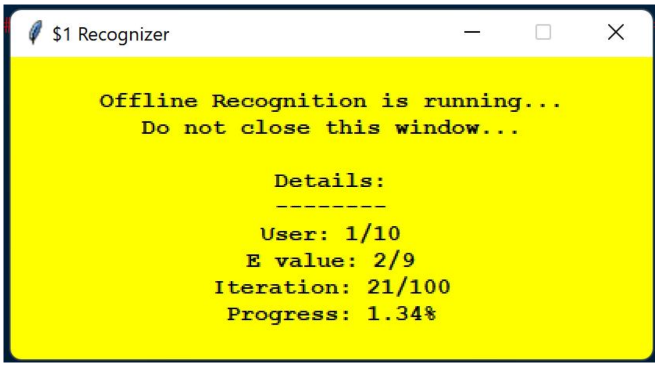
Just like the names suggest, each button does exactly what they mean.



```
Result: circle (0.945) in 12.609 ms
```

Figure 3: Live Recognition

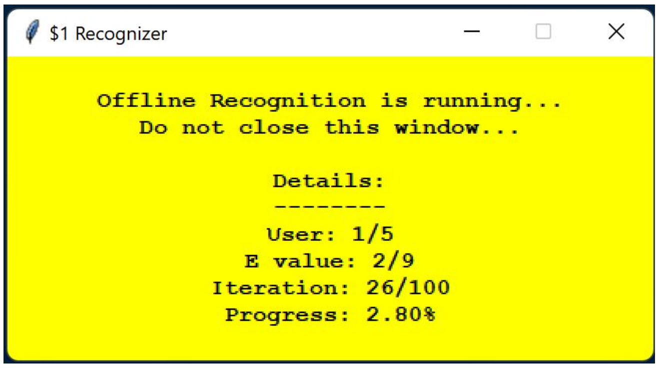
Figure 3 shows what happens during live recognition. The program predicted the gesture drawn as circle along with accuracy and time taken. Figure 4 and Figure 5 demonstrate the offline recognition process for Unistroke and Custom Datasets.



```
Offline Recognition is running...
Do not close this window...

Details:
--------
User: 1/10
E value: 2/9
Iteration: 21/100
Progress: 1.34%
```

Figure 4: Offline Recognition for Unistroke Dataset

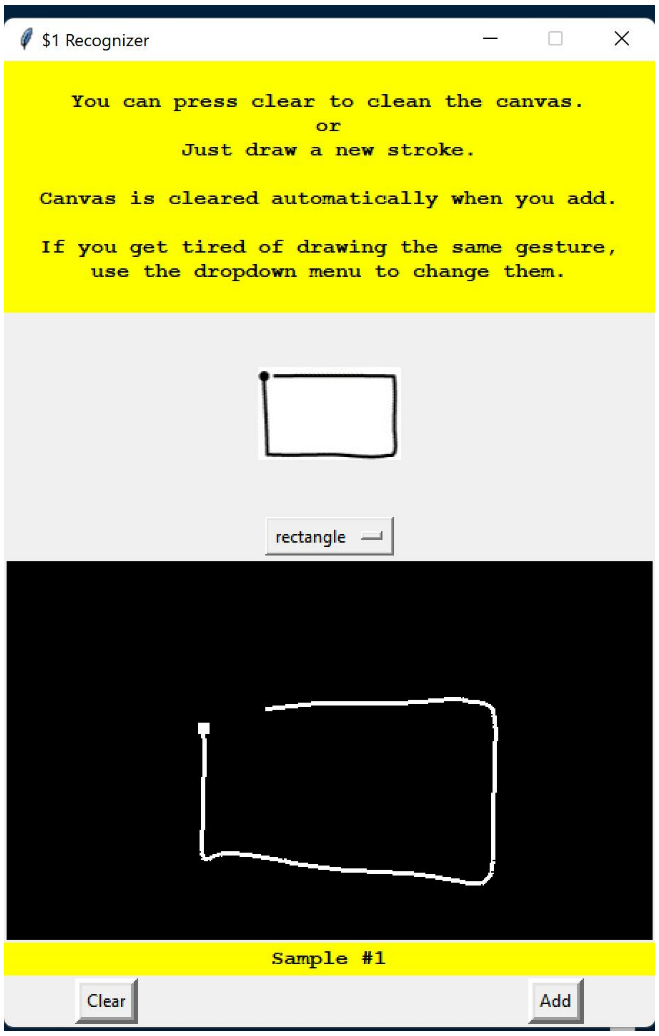


```
Offline Recognition is running...
Do not close this window...

Details:
--------
User: 1/5
E value: 2/9
Iteration: 26/100
Progress: 2.80%
```

Figure 5: Offline Recognition for Custom Dataset

We can also see what happens during the Custom dataset collection process in Figure 6. In this figure, the user is drawing a sample for the rectangle gesture.



```
You can press clear to clean the canvas.
or
Just draw a new stroke.

Canvas is cleared automatically when you add.

If you get tired of drawing the same gesture,
use the dropdown menu to change them.

Sample #1
```

Figure 6: Custom Dataset Collection Process

4.3 Implementation Challenges and Solutions

Some of the challenges while implementing the $1Recognizer algorithm are shown below:

- During the resampling stage, we sometimes get less than 64 points. This problem was resolved by appending the last few points again so that the count becomes 64.
- Debugging Offline Recognition for 100 iterations was difficult, although this issue was solved by testing it for some less number of iterations/ the number of training examples.
- Tools like GhoST[3] and GECKo[2] require the XML files to be in a certain format or else they cannot analyze the files. The name of the gesture shape along with the sample number should be of the format, "shapenameXX", where XX represents a 2 digit number (01 to 10).

4 Offline Recognition Results

For the offline recognition results, the program checks each gesture drawn by the user stored in an XML file, It then runs the offline recognition for all of them and writes some details into a log file. These details include User Id, Iteration Number, Number of training Examples, Total Size of Training Set, Candidate (Current user gesture), Recognition Result, A boolean variable (0 or 1) to check if the recognition is correct or not, Recognition Score, Recognition template that was the best match and the N-best Sorted list containing all templates in decreasing order of the scores.

It was observed that as the number of training examples increased, the accuracy of the algorithm also increased. This can be observed in the Figure 7 shown below. This is because as the number of training examples increase, the algorithm has more templates to compare them with the test gesture.

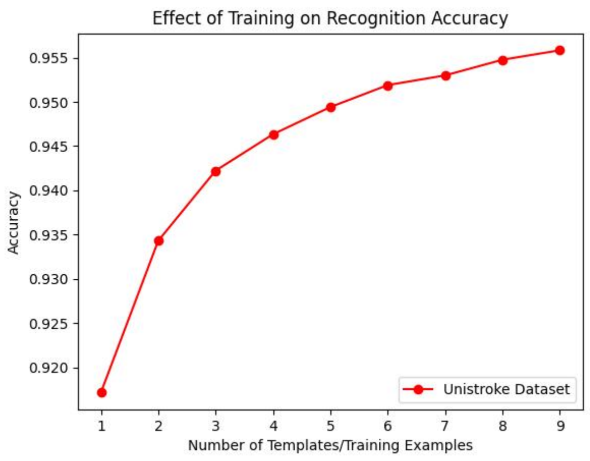


Figure 7: Recognition Accuracy as a function of templates or training for Unistroke Dataset (higher is better)

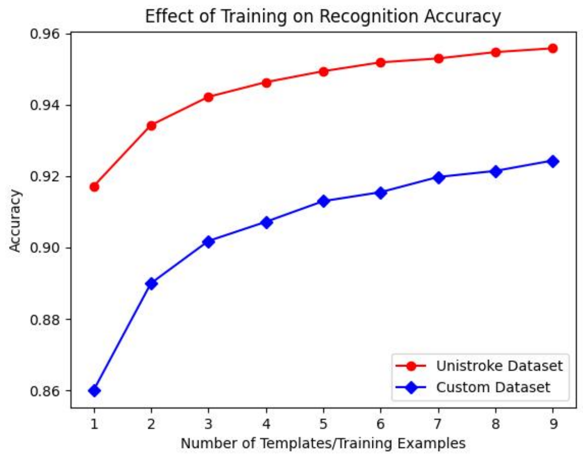


Figure 8: Comparison between Recognition Accuracies as a function of templates or training for Unistroke Dataset and Custom Dataset (higher is better)

As seen in figure 8, even for Custom Dataset, the accuracy increases along with increase in number of training examples. But these accuracies are lower than the accuracies of the Unistroke Dataset[7]. This is because for the Custom Dataset, the users were not asked to maintain a fixed speed like in Unistroke Dataset Collection. Also, factors like fatigue, pressure, disinterest also influence to a certain degree. But still these results are acceptable.

The total average accuracies were 0.985 for Unistroke Dataset and 0.921 for the Custom Dataset.

6 Understanding Data

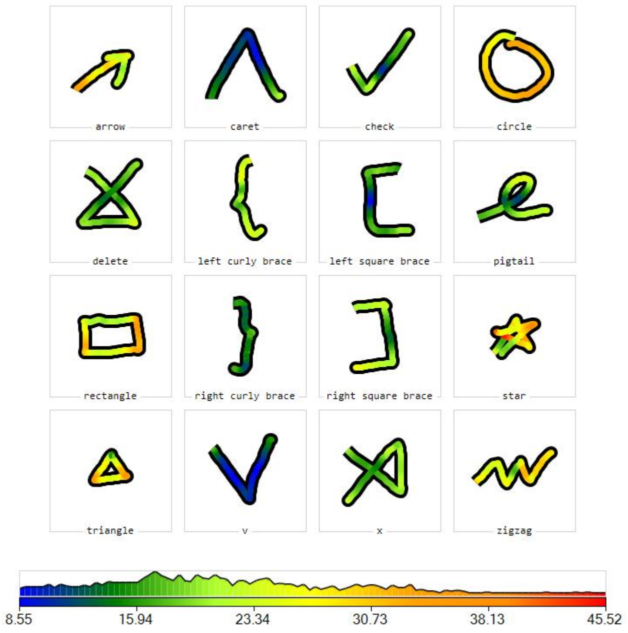


Figure 9: Heatmap generated for the Custom Dataset by the GHoST[3] Application

None of the users had proper prior experience with the stylus so they are fairly new to this type of input method. All users were right-handed. From the heatmap image, the following things are observed:

- **Caret** and **V** have the lowest variability. This is because these gestures aren't that complicated and were quite easy to draw using the stylus. The users only had to change the direction of the stroke once (one corner).
- **Arrow**, **Circle**, **Rectangle**, **Star**, **Triangle,** and **Zig-zag** gestures show a lot of variation.
- For shapes like **circles, rectangles, stars,** and **triangles**, users were trying to make sure that the starting point and the ending point meet. Also, these gestures have many turns that complicated things (especially see circle and star).
- Users were new to **Arrow** and **Zig-zag**. Most of them had trouble drawing during turns. For zig-zag, users were confused about the number of strokes in that gesture and made errors (even though a reference image was provided).

- **Check, Delete, Pigtail** and **X** were some gestures that users found easier despite some of them seeing these for the first time. These gestures are easy to draw using a stylus because of how fast you can draw them with ease.
- The **left curly brace** and the **right square brace** almost have the same set of colors in the heatmap. The same can be said for the **left square brace** and **right curly brace**. But the noticeable difference is that the former has significantly higher variability than the latter. These things happened because all the users were right hand users. If there were any left hand users, an outlier may have been observed.
  - **left curly brace and right square brace :**
    The colors seen are yellow for most areas and light green for some areas.
  - **left square brace and right curly brace**:
    The colors seen are blue for some areas and dark green for most.

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. * https://dl.acm.org/citation.cfm?id=1294238

[2] Lisa Anthony, Radu-Daniel Vatavu, and Jacob O. Wobbrock. 2013. Understanding the consistency of users' pen and finger stroke gesture articulation. In Proceedings of Graphics Interface 2013 (GI '13). Canadian Information Processing Society, CAN, 87–94.

[3] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2014. Gesture Heatmaps: Understanding Gesture Performance with Colorful Visualizations. In <i>Proceedings of the 16th International Conference on Multimodal Interaction</i> (<i>ICMI '14</i>). Association for Computing Machinery, New York, NY, USA, 172–179. DOI:https://doi.org/10.1145/2663204.2663256

[4] Wobbrock, J. O., Wilson, A. D., &amp; Li, Y. (n.d.). $1 recognizer. Retrieved from https://depts.washington.edu/acelab/proj/dollar/index.html

[5] Python. (n.d.). Tkinter - Python interface to TCL/TK¶. tkinter - Python interface to Tcl/Tk - Python 3.10.3 documentation. Retrieved from https://docs.python.org/3/library/tkinter.html

[6] Alex Clark and Contributors. (n.d.). Pillow. Retrieved from https://pillow.readthedocs.io/en/stable/

[7] Wobbrock, J. O., Wilson, A. D., &amp; Li, Y. (n.d.). $1 recognizer. $1 recognizer. (n.d.). Retrieved from http://depts.washington.edu/acelab/proj/dollar/index.html