HCIRA Project #1 Final Report

Yudi Zhang and Jordan Sabo

**Abstract**

In the past, there was no satisfactory solution for developers who wanted to have a quick and lightweight gesture recognizer for their product or prototype. However, in 2007 Jacob Wobbrock, Andrew Wilson, and Yang Li designed a lightweight and simple recognizer to solve this problem. This recognizer, called $1, can be implemented in about 100 lines of code and yields fast and accurate recognition results for single stroke gestures. In this project, we sought to implement the $1 algorithm. To test our algorithm, we used both the data collected and used in the original $1 study and collected our own new data to be processed. From the unistroke dataset in the original $1 study, our implementation had a 99.23% accuracy. From the data we collected, our implementation had a 94.5% accuracy.

—————————— ◆ ——————————

# 1 Introduction

In this project, the $1 Unistroke Recognizer, first created by Jacob Wobbrock, Andrew Wilson, and Yang Li, was implemented. The goal of this algorithm is to be able to classify a single stroke gesture as one of a predefined list of gestures. Our implementation was in the Javascript language, and visual components were created using the intersection of Javascript and HTML. Two different datasets were used in our exploration of this algorithm: the unistroke gesture logs provided by the original study, and our own data collected from willing participants. Our implementation supports both offline and online recognition. That is, it can both iterate through a large set of data and automatically produce a log file with results, and it can have a user draw a gesture live and display a recognition result.

# 2 Related Work

Various works have been completed in the past regarding gesture recognition. In this project, the recognition algorithms are tailored to a simpler template-matching approach, rather than more expensive and complex methods such as neural networks or HMMs. Thus, this section will focus on past studies in this particular domain.

First and foremost, it is critical to discuss the original 1$ recognizer itself. Wobbrock et al. (2007) designed this algorithm with the goal of creating a lightweight gesture recognizer for one stroke that a user can easily integrate into their prototype or product. This recognizer works by taking in template gestures before any recognition is performed. They sought to address the current lack of such resources for developers and note the current complexity and expensiveness of other methods currently used. Then the same operations (such as scaling and rotation) are performed on the templates and the gesture to be recognized (the candidate gesture). The template that has the smallest summed Euclidean distance from its processed points to the candidate’s is declared as the result. In order to test the accuracy of their algorithm, they conducted a study using 4800 gestures collected from 10 subjects. They found that the $1 recognizer had an overall accuracy of 99.02%, outperforming some other more complex models. Some important limitations to the $1 recognizer that are important to address are that it in its processes it does not maintain rotation, size (and by extension, aspect ratio), or the location of the gesture. Additionally, the speed at which the gesture is drawn influences the accuracy of the result; medium speeds outperformed fast and slow speeds. Our implementation closely followed this work of Wobbrock et al. Thus, our implementation is subject to the same limitations but should have the same high accuracy.

In 2010, Yang Li published an extension of the $1 recognizer called Protractor. This algorithm is very similar to 1$, but it aims to address some of the limitations of $1. With a few key differences in the gesture processing steps, Protractor removes the effect of gestures drawing speed on recognition. It also allows the gesture to be drawn in any location on the screen and removes the rotation and aspect-ration invariance. Addiontally, in computing the final recognition results, a closed form method is used rather than the original iterative method. These changes resulted in a very similar accuracy to the $1 recognizer, but Protractor had much faster execution times in comparison. Our project did not account for these limitations, so the work of the $1 Protractor shows some beneficial ways in which our model could be improved.

Also in 2010, $1 was adapted by Lisa Anthony and Jacob Wobbrock to recognize gestures with multiple strokes: the $N recognizer. The $N recognizer essentially turns a multistroke template gesture into a unistroke gesture by generating all possible ways to connect the multiple strokes with a straight line and storing those permutations. Candidate gestures are then connected in the order that they are drawn and compared to all the permutations of templates, and whatever template the closest matching permutation falls under is the recognition result. To test the accuracy of this model, forty middle and high school students provided gestures for 20 different algebraic symbols. With 15 training examples, the $N recognizer had a 96.6% accuracy on the algebraic symbols. However, when tested on the $1 unistroke gesture logs, $N performed worse than $1, as expected. Therefore, when it comes to recognizing unistroke gestures, the $1 algorithm is still more efficient, despite the increased versatility of the $N recognizer.

Similar to how $1 has a Protractor extension, the $N Protractor was developed by Lisa Anthony and James Wobbrock in 2012. Like the $1 Protractor before, instead of using the iterative method to find the recognition results, $N Protractor uses a closed-form template matching method. This introduces significant speed improvements, with an over 91% increase in recognition speed from the original $N recognizer.

In the process of creating the most accurate gesture recognition algorithms, it is important to consider the human side of the issue. That is, it is important to look at how different gestures are articulated by users in order to understand how to make algorithms that can best identify such gestures. A study titled *Understanding the Consistency of Users’ Pen and Finger Stroke Gesture Articulation* conducted by Lisa Anthony, Radu-Daniel Vatavu, and Jacob Wobbrok sought to do just that. In this study, the Gesture Clustering toolkit (GECKo) was used to help analyze gesture data. It was concluded that within a single user, gesture strokes tend to be very consistent, much more so than when looking at gestures across multiple users. Additionally, gestures that are less complex tend to be more consistent. These trends provide important context for our $1 implementation. When reviewing results from our model, certain occurrences, such as lower accuracies for more complex gestures or one user’s gestures being recognized with less accuracy, can be accounted for by these patterns, and we can establish that there was no fault with the $1 algorithm.

Another important facet of gesture recognition is understanding gesture trends that can result in recognition errors. *Gesture Heatmaps: Understanding Gesture Performance with Colorful Visualizations* by Radu-Daniel Vatavu, Lisa Anthony, and Jacob Wobbrock seeks to do just that. In this study, a tool called Gesture HeatmapS Toolkit (GHoST) was used and released. This tool takes in a large amount of user data from drawing gestures and creates a heatmap showing variation in the drawings. This tool was leveraged to draw some interesting conclusions. Firstly, gestures with the same articulation direction are more likely to be mistaken for each other. Additionally, a user who perceives a gesture to be more difficult than others will take more time drawing it. This poses an issue because, as stated before, a slow speed tends to have lower accuracy than a medium speed. We used this GHoST tool to analyze data in this project and to draw our own conclusions regarding our data.

# 3 Dataset Details

In our testing of our $1 recognizer implementation, we used data from various sources, which will be explained in more detail later. Regardless of the source, the data was used in the same format. For each user that drew a set of gestures, the location of the x and y locations of the stored points in their stroke was written to an xml file. Associated with each set of points is the gesture name and the number of points in that articulation.

## 3.1 $1 Unistroke Dataset

The first dataset our $1 recognizer was tested on was the $1 Unistroke Dataset collected by the original study. They had ten people draw out three sets of entries for each of the 16 included gesture types. The first set was to be drawn at a slow pace, the second set to be drawn at a medium pace, and the third set to be drawn at a fast pace. From this data, we used the medium sets in our recognition testing.

## 3.2 New Dataset Collected

We also collected our own data to use in our $1 recognizer implementation. To do so, we first designed an interface that would prompt a user to draw each of the included gestures ten times. The order of the gestures was varied to decrease the chance that a user would get lazy, so to speak, on the later iterations of their drawings. The stroke data was then collected and stored in a .xml file. Ten users were recruited to generate this data, and each person signed an informed consent form prior to completing the data collection. At the end of this process, we had data for 1,600 strokes: 160 strokes per 10 users, and 10 strokes per each of 16 gesture types.

# 4 Implementation Details

In order to get input from user and recognize gesture data collected from users, we have mainly implemented three parts: data collection page, recognizer, and an online recognition page.

## 4.1 Language and Code Structure

The entire project is written in JavaScript and HTML language. JavaScript and HTML were selected as our choice of programming language because of JavaScript’s compatibility with HTML and HTML’s simple process of implementing an UI.

Each component of the program is in a single HTML file. The code of recognizer is divided in functions based on key features of the recognition process and helper functions related to those key features. Once recognition process is initiated, major functions of the program will be called sequentially and helper function will be called inside of the major functions.

The HTML Canvas demo includes a declaration of HTML Canvas, mouse event listeners, a function storing stroke points in an array, and a function for erasing the canvas.

The recognizer contains following major functions:

Resampling: resamples points of the initial graph to a more standardized form with certain number of points.

Rotate To Zero: rotates resampled graph to zero degree.

Scale To Square: rescales data points to fit into a square.

Translate To Origin: change graph’s position according to the centroid.

Centroid: Helper function that calculates the centroid of the graph.

Path Distance: helper function that calculates the entire distance between the candidate and template graph using Pythagorean theorem.

图示

描述已自动生成Distance At Angle: calculates path distance between rotated candidate graph and template graph.

Fig 1. Architecture Diagram for project 1

## 4.2 Runnable Components

图形用户界面, 文本, 应用程序, Word

描述已自动生成Initial HTML Canvas:

图形用户界面, 文本, 应用程序, Word

描述已自动生成Live Recognition Demo:

图形用户界面, 应用程序, Word

描述已自动生成Data Collection Page:

## 4.3 Implementation Challenges and Solutions

During the implementation of the recognizer, debugging process was challenging because of the sheer number of variables to keep track of.

For the offline reorganization, data process is also challenging. We originally decided our data collection page would be a responsive HTML design and soon abandoned the idea. In addition to that, Generation of .XML file is also challenging until we decided to simply generate pre-arranged strings into the .XML file.

During the analysis process, we also had to fix minor imperfections in all of 960 files that will lead to error on GHoST program.

# 5 Offline Recognition Results

Offline recognition program is based on the recognizer mechanism described in section 4.1. On top of the recognizer, iterations were set up to repeatedly test local .XML files and generate a .CSV file as the recognition log.

图表, 折线图

描述已自动生成With the increased number of templates used for the recognition process, it is obvious that the error rage has seen a continuous change until using 8th and 9th template for the recognition.

Fig 2. Recognition Error Rate vs. Number of templates used.

# 箭头 中度可信度描述已自动生成6 Understanding Data

Fig 3. Heatmap generated by GHoST program.

From this heat map, it is clear that the shape error is consistent across most graph types except for zigzag.

**References**

[include at minimum a reference to the original papers read for this project, along with any other resources you used to help you in your project (and make sure they are cited in the appropriate places in the paper!)…]

1. etc.

**Appendices**

[any additional materials you feel would be helpful or important in understanding or evaluating your project; these may go beyond 5 pages…]