FINAL REPORT

Wii Want to Write

A Real-Time Accelerometer based Writing & Gesture Recognition Technology

18-551

Group 5 of Fall 2008

Jeffrey Lai (jlai1@andrew.cmu.edu)
Tian Seng Leong (tleong@andrew.cmu.edu)
Jeffrey Panza (jpanza@andrew.cmu.edu)

Table of Contents

1. Introduction – The Project

The goal of our project was to classify gestures using only x-, y-, and z-axis accelerometers as input. We used a min-max algorithm created from scratch that used dynamic time warping (DTW) to calculate the distance a test input would be from those boundaries.

2. The Problem

The adoption of small mobile devices is becoming more widespread today. One of the major problems with these devices is obtaining inputs. Traditional methods of obtaining inputs such as keyboard have limitations in terms of speed, accuracy (touch screen keyboard), size of keys (too small to press), etc. Besides the keyboard, a pen stylus can be also used to obtain inputs. However, the stylus has its own sets of issues. They range from the inaccuracy of handwriting recognition through image processing (lack of time information), and the stylus is easily lost. Making small intricate hand movements using either the keyboard or stylus may aggravate certain conditions such as Carpal tunnel and arthritis. Today, there are still no popular devices that track human gesture movement for daily communication purposes.

The goal of our project is to make use of a well developed hardware device, the accelerometer, to recognize hand gesture and alphanumeric characters. Currently, accelerometers are present in a multitude of electronic devices such as the Wii controller. The Wii controller is going to be the input device of our system.

3. Motivation

The main motivation when we came up with the idea for Wii gesture recognition was that it was a unique challenge that had not been surmounted previously in an 18-551 project. Instead, we looked to conference papers and IEEE explorer to find resources related to our area of interest. Surprisingly, we had trouble finding much information related to the Wiimote or accelerometer based character recognition. The few papers we did find seemed inadequate, only looking at small subsets of the general problem. Some looked at small sets of dynamic gestures (ones where the user moves his or her arm arm) to recognize, for example differentiating between a circle, square, rotating the wiimote along the length of it, the letter Z, and a tennis serve [3]. These had the obvious fault of expansion. Of course, five gestures will be easily recognized. But there was no way to know if the algorithms suggested would support a larger data set. Others looked at only static gestures, and differentiated between them, determining the orientation of the Wiimote by looking at which direction was the direction of gravity. This isn't very useful except for video games where the user will be sitting motionless. Gestures are generally dynamic and in large quantities. We were motivated to try and develop a more robust, general gesture recognition system that could be user-independent and support larger gesture sets: in our case, the English alphabet, Arabic numerals, and a set of general gestures.

4. Our Initial Solution

We chose to use dynamic time warping, or DTW, as the algorithm for our project, although as the project continued the use of DTW was refined. Dynamic Time Warping is a Dynamic Programming technique to find the distance between two given images [5] (in our case accelerometer signals). We accumulate the distance between the current sample of the test set and the current sample of the training set, and add it to the minimum value of the adjacent spots in the matrix (left, down, and down-left). Thus in the top right corner we will have the shortest

distance from the current training and test image calculated. That is the value we use to determine which character is matched. Whichever training set the test set is closest to is determined to be the recognized character. The code we found goes further calculating the normalized distance between the characters. It accumulates squared distances and at the end divides by the length of the line (the number of steps from start to finish).

At first our goal was to use DTW over the whole training image to find the distance between a training image and a test image, but that had a multitude of issues including time consumption. DTW is an O(n^2) function and the length of a training set (for one axis) is around 45 to 100 samples. So it will take a long time to fill up a DTW matrix with that many samples per side.

We found faster DTW code that gave us under one second per character to be classified which we felt was good enough for real-time. A method to speed up this task is to constrain our search space [6]. The C code we found does this using Sakoe-Chiba band. We will not do calculations outside of that radius (the radius is ours to choose). The code we found goes further calculating the normalized distance between the characters. It accumulates squared distances and at the end divides by the length of the line (the number of steps from start to finish). The code was faster since it constrained the DTW matrix into a parallelogram, using the Sakoe-Chiba method. We implemented this on the DSK and ran some more tests. It showed that the algorithm was fine with the training set being the current users, but when someone else tried to use the system, their numbers would be much lower. After hearing about lots of possible faults in this design, like brute force, nearest neighbor decisions on classification, and the large storage waste in the brute force method, we decided to develop a better algorithm. The suggestions we tried to improve on were user-independence and clustering to conserve on memory space.

5. Clustering Algorithm

The new algorithm we began to implement was based off basic clustering where we would average the DTW of all the training sets for a character onto one of the training sets. This would then be the “center of gravity” of that cluster, so instead of running DTW five times for every test image and averaging the distances, we’d do the DTW once on the average of the five inputs. Below is an example of this original clustering algorithm. It is the cluster for A (x-axis only).

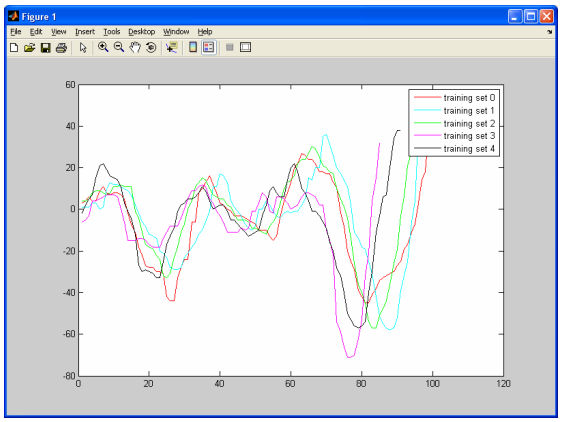

Figure 1
Training Sets 0 to 4 (Training Set 0 is red)

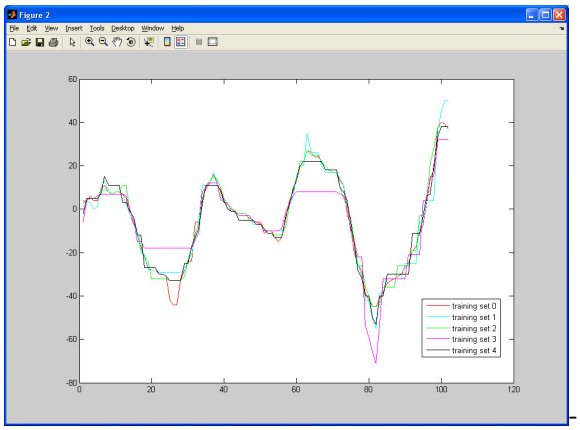

Figure 2
Output after all have DTW done with Training Set 0

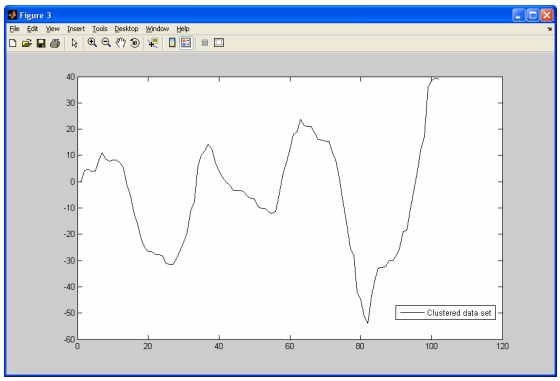

Figure 3
Average of the five DTW outputs

This algorithm actually produced worse results than our original algorithm which makes sense since we are combining lots of information into one representative curve. Basically the goal of clustering is to reduce the amount of memory storage for our training set, which is a useful optimization, especially when the user's data set is large, but this is not the case with our project. The problem is the cost of accuracy which is important to character recognition.

We still wanted to achieve user-independence and clustering. Instead of using the DTW implementations we are used to, we decided to go in a different direction.

6. Final Algorithm (Max-Min Bound)

The algorithm uses the same simple method of using DTW to calculate the distance from the cluster or a single training set, but instead we calculate the distance from the maximum and minimum boundary of the training set.

First we take in the five training sets of one character from the user. The training sets will then be resampled to 50 samples. Then, they will get normalized from -1 to 1 for every axis and training set, individually. Finally, this data will be used to define a global max curve over all five training sets. The same will be done for a global minimum curve.

Once we accept training sets for all gestures, we will then accept a test input. This input will also be resampled to 50 samples and normalized to -1 to 1. This input will then be sent to the DSK for processing.

The input will then be compared to each max and min curve for each character. When the input curve goes over the maximum bounding curve, we will run DTW over all points until the input goes back within bounds. And, when the input curve goes below the minimum bounding curve, we will run DTW over all points until it goes back within bounds. We sum all of these DTW results for one character.

The DTW we use is the original, slow DTW, NOT the fast DTW using constraints. It seems the fast DTW used squared distances that got divided by the length of the shortest distance path length, and this seemed to cause problems with the algorithm dropping the algorithms accuracy by 40% at least, so we moved back to the original DTW algorithm that used Euclidean Distance and had no constraints. It was okay to do this since even though DTW is O(n^2) in complexity to compute, we are doing many small DTWs now, so many short DTWs

will not add up to the runtime of one DTW that runs over the entire signal, which we used to use. This could also be a problem of the constrained DTW. It may have gone out of bounds to easily since the matrix was so small, although we put FLT_MAX as the radius of the constraint (aka run with no constraints) and we still got very inaccurate results, we are fairly confident it has to do with the fact that the distance was not Euclidean.

We sum all of these DTW results for one character. Then, we weight these results to attenuate the y-axis. In our system, the y-axis is the direction outward from the user, the least important direction for natural writing. We don't completely get rid of the y-axis simply because we noted a drop in accuracy when doing this. We believe that the y-axis is important because the user will not always hold the Wiimote with the x- and z-axis in those directions, there could be lots of wrist motion and as a result, all directions are important, but in-XZ-plane data is most important for recognition of a two-dimensional image. In the end, we chose 40% for x- and z-axis weighting and 20% for the y-axis weighting. Finally, we take the top three shortest distances calculated and return those characters to the user.

The idea we use to think about how this works is to compare it to the original clustering algorithm, since it is just a variation on a clustering algorithm. To recap, the original algorithm takes a scattering of points of a known cluster and then averages them to find a center of gravity (note the black dots in Figure 5). When given a test input, the user would then find which cluster center it is nearest.

Our new algorithm will instead make a boundary around our cluster and then decide which boundary it is closest to (note the colored, dotted lines in Figure 5). This is more lenient to training sets that aren't as tight or well-clustered. Basically, this assumes that if the user is good at making his or her training set, he or she will continue to be good at writing that character and will require tests very close to the actual character to be accepted, which a character that the user seems to have trouble writing, it will assume that the user is writing it more often than another character.

The benefits of this algorithm are that it not only reduces the number of training sets to two instead of five or any number of training sets, so memory storage on the DSK is reduced, but that it also has the benefit of user-independence according to our test results. It might be a result of the leniency that the algorithm gives to harder to write characters and gestures.

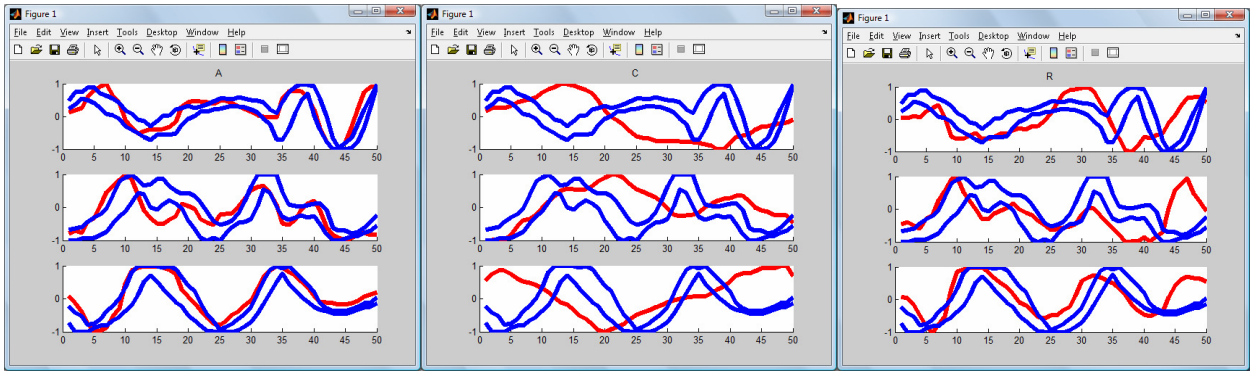

Figure 4 – These graphs show the bounding algorithm at work in Matlab. The blue graphs are the max and min curves for the training set of the letter "A". The red line for each graph is the test input for A, C, and R respectively. To get an idea of the distances calculated in these graphs, the result for A is 0.5934, C is 9.9235, and R is 4.7123.

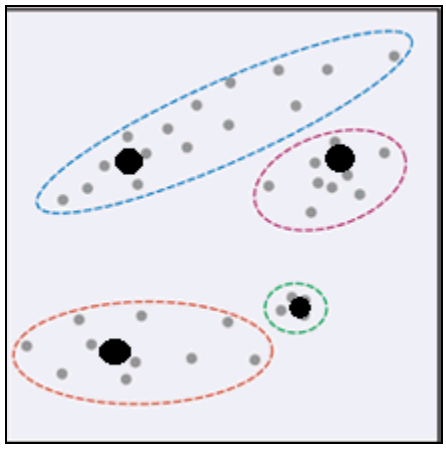

Figure 5
http://www.nature.com/nbt/journal/v23/n12/fig_tab/nbt1205-1499_F1.html

7. Test Results

Recognition Result for Each Character Based on 5 User Inputs

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Medium Speed Jeffrey Lai Set 2
Algorithm : Max-Min Bound Algorithm
Recognition Rate : 94.62%

Figure 6 - Although the user input is Q, it is being recognized as G in this graph. We think the reason for this is that the stroke pattern of alphabet Q is similar to G. Another reason that we can think of is that Jeffrey Lai made some mistake when training his medium speed Q alphabet.

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Fast Speed Jeffrey Lai Set 2
Algorithm : Max-Min Bound Algorithm
Recognition Rate : 87.69%

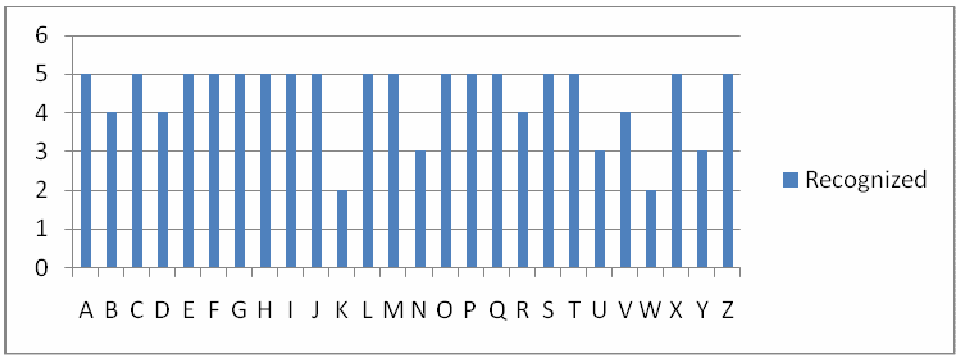

Figure 7 - The actual recognized result for character K is R, which is really similar to K in terms of stroke order. In addition, when the user writes K in a fast manner, the probability of it resembling R is high. The same reasoning goes with N which is recognized as W in this test. In addition, the character W is recognized as N which proves that N & W are often confused with each other. Alphabet Y is also confused with alphabet T for stroke order reasons.

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Slow Speed Jeffrey Lai Set 2
Algorithm : Max-Min Bound Algorithm
Recognition Rate : 90.77%

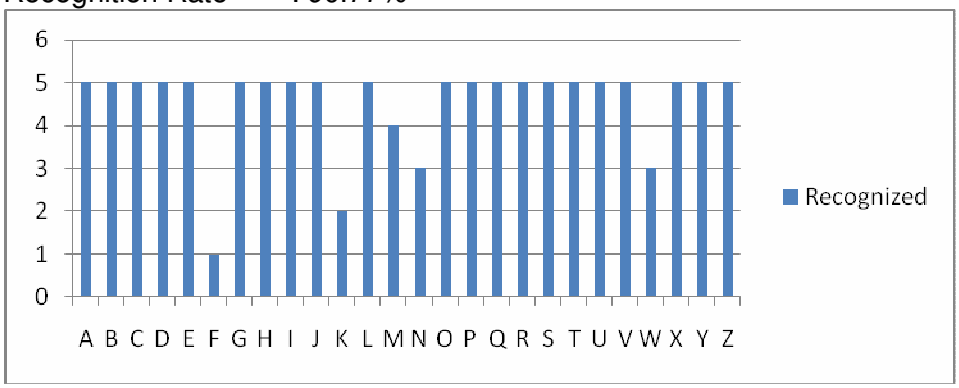

Figure 8 - The most pronounced error here is for alphabet F to be recognized as K. We speculate that F and K look similar after resampling and normalization. Since the user input is a slow version, the acceleration data when F is written is not so pronounced and thus makes it look like a normally written K.

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Medium Speed Jeffrey Lai Set 2
Algorithm : Fast DTW with a Clustered Training Set
Recognition Rate : 78.42%

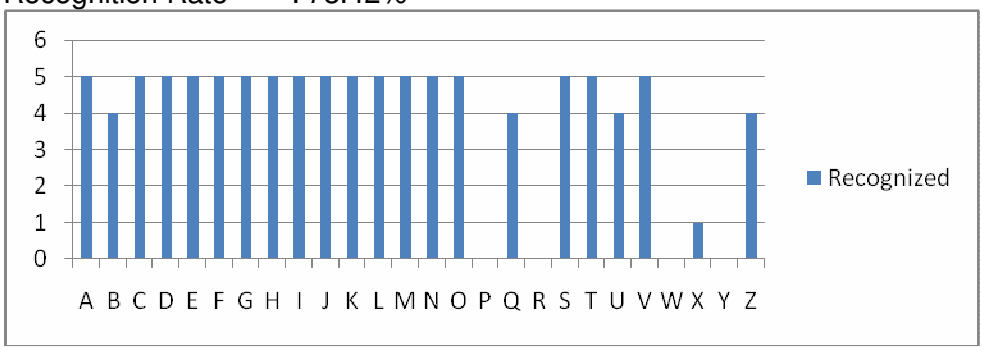

Figure 9 - The most obvious error here is for alphabet P to be recognized as D, alphabet R to be recognized as K, and alphabet W to be recognized as N. This error is more pronounced when the training set is clustered as 4 out of the 5 training set will resemble the 1st training set.

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Medium Speed Jeffrey Lai Set 2
Algorithm : Fast DTW without a Clustered Training Set
Recognition Rate : 73.85%

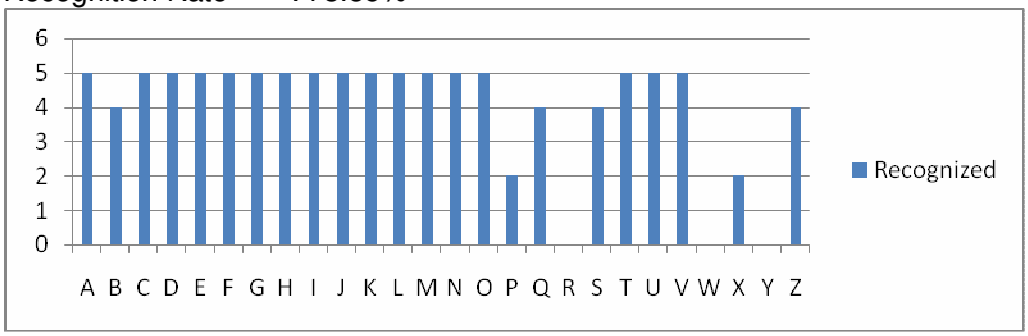

Figure 10 - In this case, alphabet P was recognized as D, R as K, W as N, Y as R and X as D. All of the combinations are quite familiar to us by now except for X as D. We believe that our user input for character X was corrupted in a manner unfamiliar to us.

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Fast Speed Jeffrey Lai Set 2
Algorithm : Fast DTW without a Clustered Training Set
Recognition Rate : 28.46%

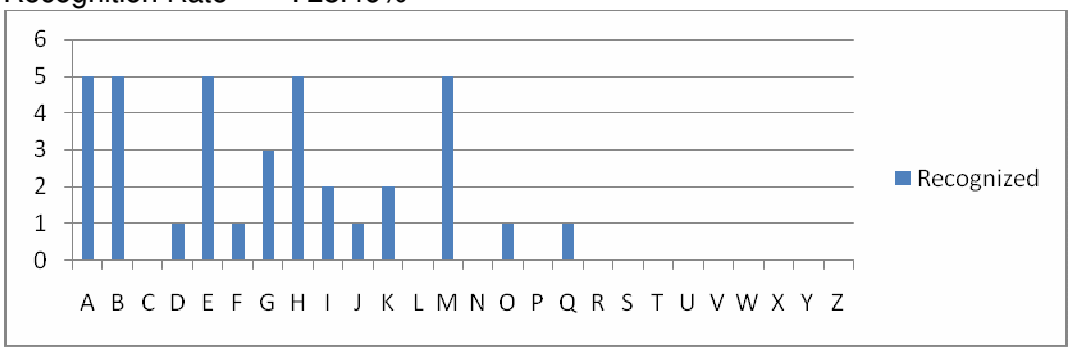

Figure 11 - The recognition rate for this test is so low due the fact that when a user writes really fast, much of the acceleration data is inconsistent and warping them to the training set will try to make the input resembles the erroneous training set.

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Medium Speed David Leong Set 1
Algorithm : Max-Min Bound Algorithm
Recognition Rate : 75.38%

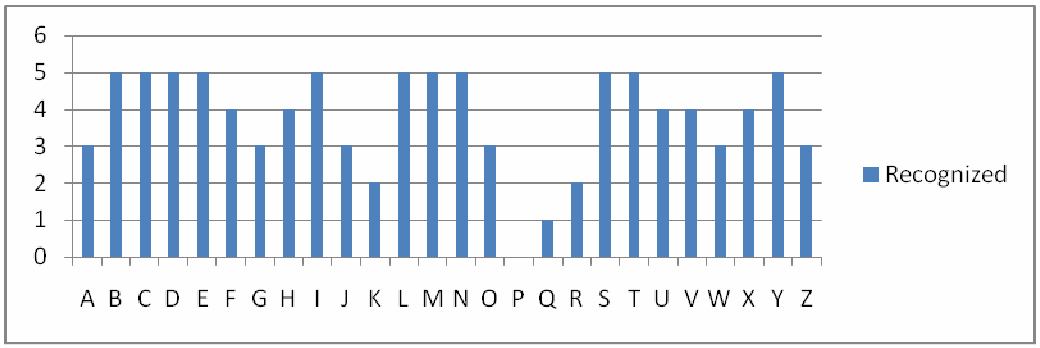

Figure 12 - We believe that the cross user test result yield was good except for P which was recognized as D and Q recognized as G. These two problems have been present due to the similarities of P with D and Q with G. Additionally, the additional variation supplied by having Jeffrey Lai train the training set and Tian Seng Leong testing the training set makes the data vulnerable to wrists (angular) motion which we believe account for some inputs being recognized correctly and some not.

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Medium Speed David Leong Set 1
Algorithm : Fast DTW with a Clustered Training Set
Recognition Rate : 56.15 %

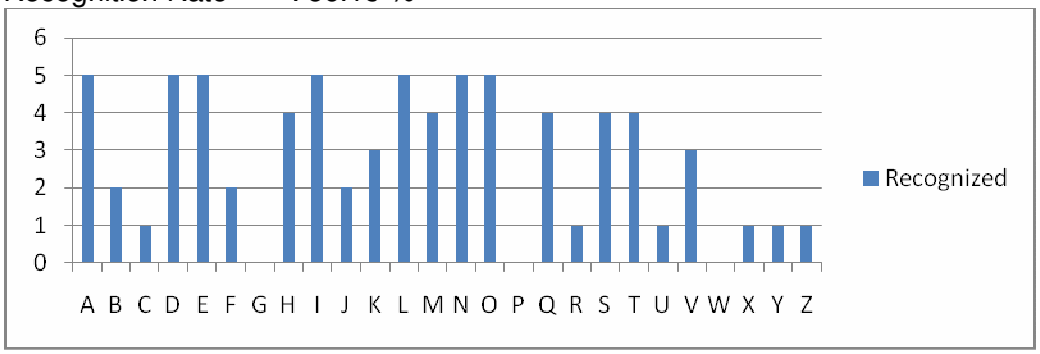

Figure 13

Training Set : Medium Speed Jeffrey Lai Set 1
User Input : Medium Speed David Leong Set 1
Algorithm : Fast DTW without a Clustered Training Set
Recognition Rate : 63.85 %

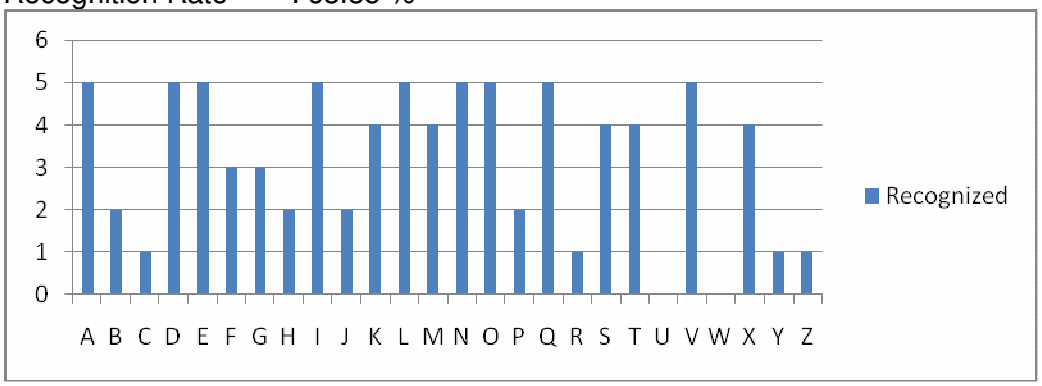

Figure 14 - The recognition rate for this test is much better than for the same user (Jeffrey Lai when he inputs a fast set). As mentioned before, when a user writes fast, there is a lot of

variation in the acceleration data and Fast DTW will try to warp the fast input to the medium speed training set, resulting in loss of data. As usual, there are a lot of differences in recognition results between two users due to wrists motion and writing style in addition to characters being almost similar such as R and K, W and N, U and V and Z and I

8. Training Set Specifications

The training sets for testing the algorithms in Matlab before implementing and testing on the DSK were collected from members of the group.

The training sets are acceleration outputs sampled at 30 Hz from the Wiimote. The driver cannot control the sampling rate. It seems to be sent over Bluetooth already sampled.

We are requiring five training set inputs to get a good idea of how the user writes. We thought any smaller and the user might not have a representative character set, and if it was any larger the user might get bored and make bigger mistakes that may skew the min-max algorithm.

The training set will go through resampling and normalization before being put on the DSK. We decided to resample all user input to 50 samples to simplify C coding and to optimize slightly. We decided on this value specifically because of the following data collected:

Jeffrey Lai
Average length in samples for quickly drawn gestures - 48.4154
Average length in samples for average speed gestures - 99.7615
Average length in samples for slowly drawn gestures - 112.0692
Average length in samples for a varied speed set of gestures - 75.6923

David Leong
Average length in samples for quickly drawn gestures - 43.4000
Average length in samples for average speed gestures - 73.1154
Average length in samples for slowly drawn gestures - 83.5692
Average length in samples for a varied speed set of gestures - 59.600

Jeffrey Panza
Average length in samples for quickly drawn gestures - 36.2231
Average length in samples for average speed gestures - 54.4846
Average length in samples for slowly drawn gestures - 69.1154
Average length in samples for a varied speed set of gestures - 43.2462

We plan to only focus on capital letters, numbers and gestures separately and ignore the lower case character recognition, since we want to concentrate on recognition accuracy and not quantity.

The upper and lower bound are the only data that is going to be stored on the DSK.

(1 max bound + 1 min bound) * 3 axis * sizeof(float) * 50 = 1200 bytes per character/number/gesture

Since there are 26 alphabets + 10 numbers + 16 gestures, we need to have 62400 bytes. The 16MB off-chip memory capacity dwarfs the 62400 bytes required. And this carries the benefit

that it will never increase if we were to increase the training set size. The only way it would increase is if we were to add more characters to our training set entirely, but there is no way to avoid that.

9. Analysis of The Core of Our Code (the DTW)

The DTW function is the core of the program since it gets called hundreds of times to help classify one character. So it would be good to explain the details of this core algorithm.

The function first has to initialize a matrix of the Euclidean distance between any two samples (A, B): "A" being a point along the training set and "B" being a point along the test set given. Since we are filling a 2-dimensional matrix, this portion of the code will take O(n^2) time. To calculate Euclidean distance for each cell in the array we need:

- 2 multiplications
- 3 additions/subtractions

The next function in the DTW is to initialize the first column of our final matrix. We basically are adding the current distance in the Euclidean distance matrix with the value below it in the new matrix. This function is obviously O(n). This has this many operations per loop:

- 3 multiplications
- 2 additions/subtractions

The next function in the DTW is to initialize the first row of our final matrix. We basically are adding the current distance in the Euclidean distance matrix with the value to the left of it in the new matrix. This function is obviously O(n). This has this many operations per loop:

- 2 additions/subtractions

The final function of the DTW is to finish filling the matrix by taking the minimum value of what is below, left, and down-left diagonally from the current value in the new matrix and adding it to the current position in the Euclidean distance matrix. Since this is filling a 2-dimensional matrix, it will take O(n^2) time to complete. The specific operations are too many to list here but there are many conditionals to calculate the minimum value and there are many pointer arithmetic operations and normal arithmetic operations per loop.

We tried to optimize this core function of our code. We checked to see how good our loop was with no optimization, optimization two, and optimization three. All returned 1x parallel in each subloop of the DTW. We then tried to improve this by loop unrolling. We did loop unrolling by two and still no better than 1x parallel in each subloop. This makes sense though. DTW cannot really be optimized since every calculation in the matrix is dependent on three previous values of the matrix. As a result, we stopped trying to optimize our core part of the DSK code.

10. Profiling Results

As aforementioned, the DTW function is of much importance. Therefore, it is imperative that we obtain the results from profiling the DTW function. Based on the profile function method, we managed to obtain the following results after running the Core DTW function (dtwMeh()) 100 times:

Include Average = 633118 cycles

We think the following computations contribute most to the cycle count
Loop 1: O(n^2) → 50^2 = 2500 cycles
Loop 2: O(n) → 50*2 = 100 cycles(We multiply 2 due to 3 mult ops)
Loop 3: O(n) → 50 = 50 cycles
Loop 4: O(n^2) → 2500.5 = 12500 cycles(We multiply 5 due to 5 mult and 14 add ops)
Memory transfer from External to Internal Memory - 7.125 cycles per word
Loop 1: There are 2 external to internal memory transfers and 1 internal to external: 2500 *3
Loop 2: There are 2 external to internal memory transfers and 1 internal to external: 50 *3
Loop 3: There are 2 external to internal memory transfers and 1 internal to external: 50 *3
Loop 4: There are 4 external to internal memory transfers and 1 internal to external: 2500 *5
: There are 8 external to internal memory transfers in the min() function : 2500 *8

Total cycles predicted 287137.5 * 2 = 574275 cycles

Another function that we think is important is the dtwTest() function. The dtwTest() function is the outer shell of the Core DTW function. It goes through all the characters to be recognized and then decides the top three choices based on the distances returned by the Core DTW function. In addition to calling Core DTW, the dtwTest() function calls dtwNew() which is responsible for maximum and minimum bound checking. The following results were obtained when dtwTest() was profiled.
Include Average = 36932275 cycles

11. Task Division between PC & DSK

Task	Machine	Reason
Obtaining User Input	PC	GUI implementation in Java that resides on PC
Storing Training Set	DSK	Training set is small compared to the memory available on the DSK. Data available on the DSK is more easily accessible in terms of speed.
Computing Fast Dynamic Time Warping	DSK	Main algorithm should utilize the DSK for computation
Displaying Recognition Results	PC	The Java GUI which resides on the PC will need to display the recognition results to the user.

12. What Code We Reused

The only code that we were able to reuse was the DTW code in Matlab which many other previous groups have used, the fast DTW code in C with capabilities for converting to Matlab (a .mex file) using Sakoe-Chiba band as a constraint, and finally we inherited the Wiimote Java driver to collect acceleration data via Bluetooth which is Mac compatible only from a friend of the group, Peter Pong. In addition, we used the LEDataStream class [13] to convert a byte stream from big-endian to little-endian and vice versa.

Otherwise, all code was theorized and implemented by our group. This includes the Java GUI, all other C code for each implementation (min-max bound and original DTW), all Java code unrelated to the Wii driver (which includes data transfer to the DSK and preprocessing of data before sending off to the DSK).

13. DSK Packet Structure

Header Structure

Before sending the user input to the DSK for either storage or recognition, a header structure of five integers (20 bytes) is sent to provide some information. The header structure is illustrated and described further below.

flag	mode	char_identifier	sample_no	dataArrLength * sizeof(float)

Field	Description
flag	If flag == 1 -> Training set If flag == 2 -> User input for recognition
mode	If mode == 1 -> English Alphabet Recognition If mode == 2 -> Arabic Numeral Recognition If mode == 3 -> General Gesture Recognition
char_identifier	The character that is going to be sent next. This identifier is based on the ASCII encoding. For example, we will set char_identifier to 65 for 'A'
sample_no	Training set number – This field ranges from 0 to 1 0 == Max boundary curve 1 == Min boundary curve
dataArrLength * sizeof(float)	Size of user input (in bytes) that is going to be sent to the DSK.

Result from DSK

The DSK will return the result to the PC in the form shown below. This data structure will be an array of three bytes. The top three matches is returned to the PC so that the PC could give better suggestions to the user.

char1	char2	char3

Field	Description
char1	Best character match obtained from recognition process
char2	2nd best character match obtained from recognition process
char3	3rd best character match obtained from recognition process

14. DSK Flowgraph

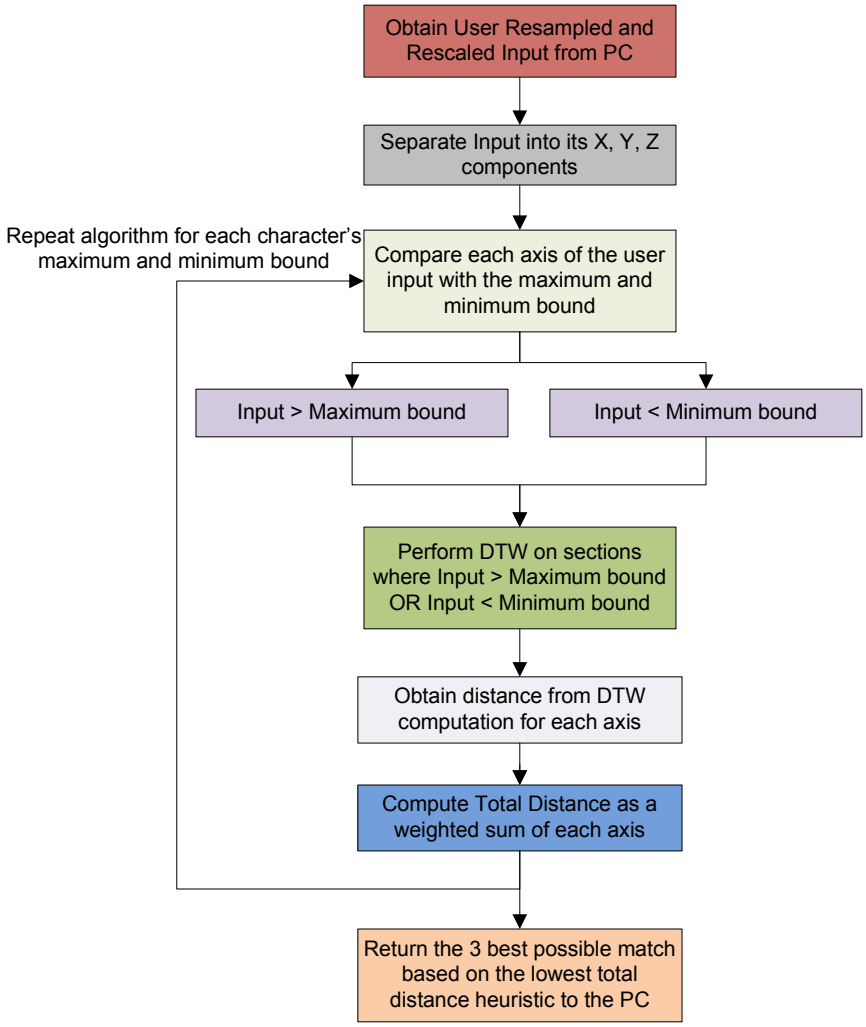

Figure 15

15. Data Flow Graph

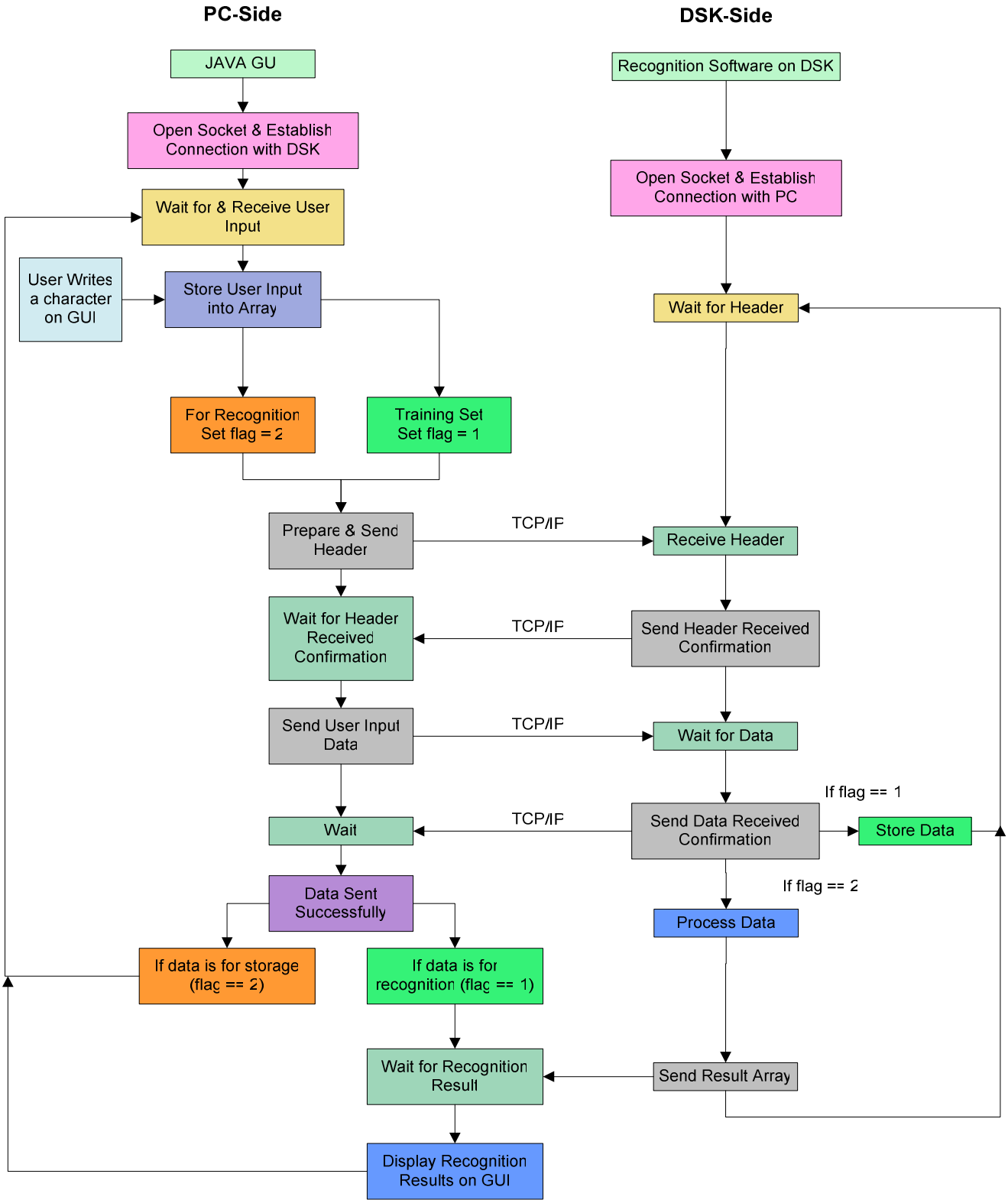

Figure 16

16. DSK Detail and Discussion

Data Transfer Rates

For every user input, the PC will send a 20-byte fixed size header to the DSK for processing. This header will describe the property of the data array that will be sent to the PC after the header has been sent.

The transfer rate between the PC and DSK according to specification is 2.5 MB/s. However, due to the nature of our project requiring a small packet header, we were not able to achieve this optimal transfer rate. We manage to obtain an estimation of the PC to DSK transfer rate by doing the following:

Transferring a 20-byte fixed size header 100 times (Time elapsed: 0.015s)

(100*20) * 1 MB/ 1048576 bytes * 1/0.015s = 0.127156575 MB/s

Transferring a 20-byte fixed size header 1000 times (Time elapsed: 0.172s)

(1000 *20) * 1 MB/ 1048576 bytes * 1/0.172s = 0.1108924 MB/s

Transferring a 20-byte fixed size header 100000 times (Time elapsed 17.04s)

(100000 *20) * 1 MB/ 1048576 bytes * 1/17.04s = 0.111934 MB/s

On the other hand, the specified transfer rate between the DSK and PC is 10MB/s. As expected, we were unable to replicate this transfer rate. Our estimation of the DSK and PC transfer rate based on sending a 3-byte fixed array is as follows:

Transferring a 3-byte fixed size array 100 000 times (Time elapsed: 5.125s)

(100 000 * 3) * 1 MB/ 1048576 bytes * 1/5.125s = 0.055824 MB/s

Transferring a 3-byte fixed size array 1 000 000 times (Time elapsed: 48.826s)

(1 000 000 * 3) * 1 MB/ 1048576 bytes * 1/48.826s = 0.0585963 MB/s

Transferring a 3-byte fixed size array 100000 times (Time elapsed: 486.138s)

(10 000 000 * 3) * 1 MB/ 1048576 bytes * 1/486.138s = 0.0588521 MB/s

Since we are sending a 3-byte array from the DSK to PC, we believe that the DSK did not utilize the buffer for sending data from the DSK to PC. The buffer played an important role in achieving high transfer rates in Lab 2. In addition, we see a slower rate compared to the PC – DSK transfer because of the overhead computation required for each transfer. Sending small amounts of data is not cost effective as a lot of overhead cost had to be incurred before data can be sent. (Setting up connection, making sure that each party receives the correct amount of data, etc). Therefore, if more data is sent at once instead of small bits of data, a higher data

throughput can be achieved. This is the reason our project packed the acceleration data of three axes (X, Y & Z) into an array to be sent to the DSK in one load.

Storage considerations on the DSK

Based on the DSK Specifications, we have 256kB of On-Chip Memory and 16MB of Off-Chip Memory.

Average size of a user input data : 1153 bytes
Number of Characters : 26
Number of Training Sets : 5
Total Storage Needed : 149890 bytes ≈ 150kB

17. Graphic User Interface

The GUI consists of two phases: Training Mode and User Input Mode. The GUI written in JAVA was used to facilitate the above two steps. The following table shows the basic controls for the Wiimote:

Purposes	Buttons
Drawing Characters	Button B (and hold)
Switch From Training Mode → User Input Mode	Button 2
Switch From User Input Mode → Training Mode	Button 1
Switch Between Alphabet Input/ Numerical Input/Gesture Input	Button A
Navigation	Directional pad
Load all Training Sets in the "Training" Folder	+

Figure 17 - http://wiki.linuxmce.org/images/a/ab/Wiimote.jpg

Training Mode

Figure 18

In this mode, users will be prompted to draw five samples for each of the 26 English characters (A-Z), 10 Arabic numerals (0-9) and 16 gestures. The GUI will move on to the next characters automatically when the users finish recording each character. The red dot inside the bold character will slide continuously to indicate the stroke orders that the user is supposed to follow.

The rectangular scope below will display the user input acceleration data in x, y, z directions simultaneously as the user starts drawing. Red represents the x-direction, blue represents the y-direction, and green represents the z-direction.

User Input Mode

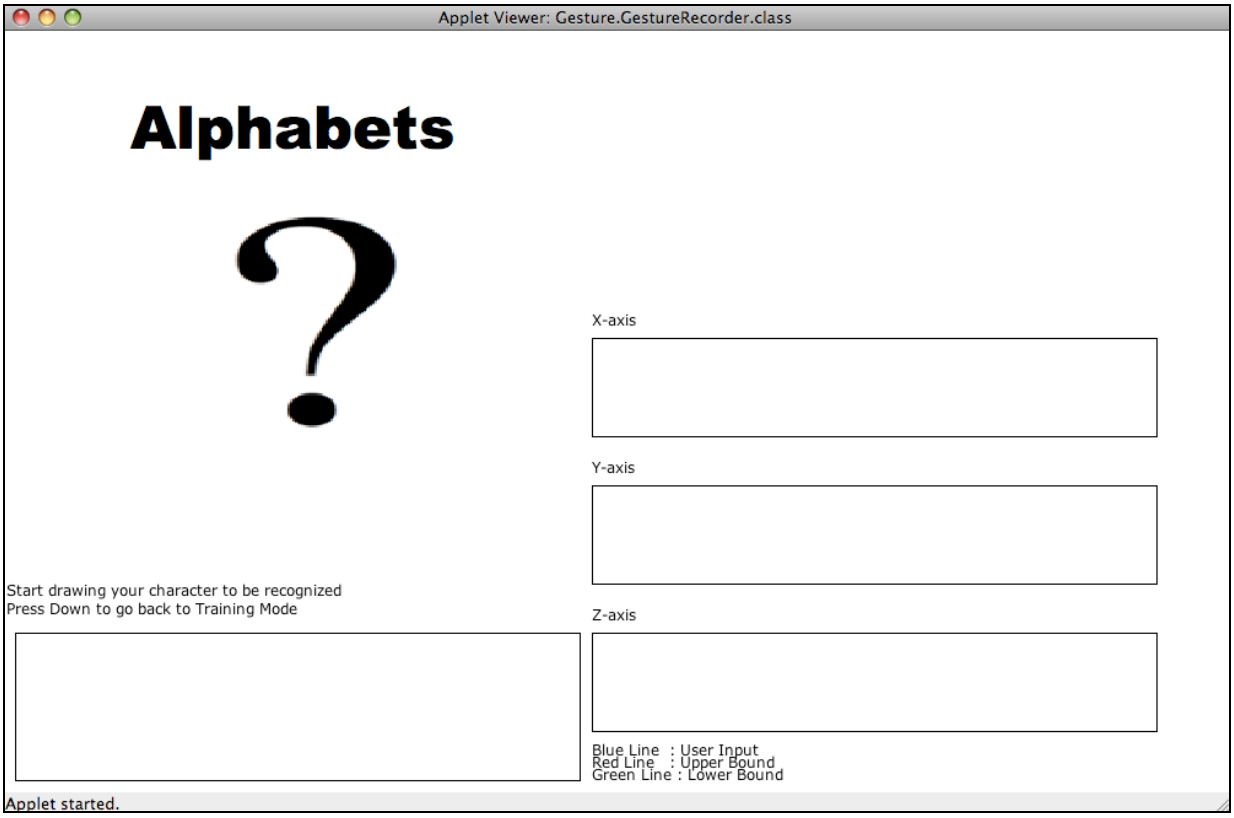

Figure 19

In this mode, users will be prompted to enter gestures that they wish to be recognized. Users can select which dataset (Alphabets, Numbers, Gestures) by pressing Button A. Once the user has drawn the character on the GUI, the three corresponding closest matches will be printed on the GUI, with the closest match in the middle (largest), second closest on the left, and third closest on the right.

Once the closest match has been made, the three scopes on the right will display both the maximum bound and minimum bound in the training set for the recognized character. When the user presses left/ right, the top three closest matched characters will rotate, and the max-min bounds on the three scopes will change accordingly.

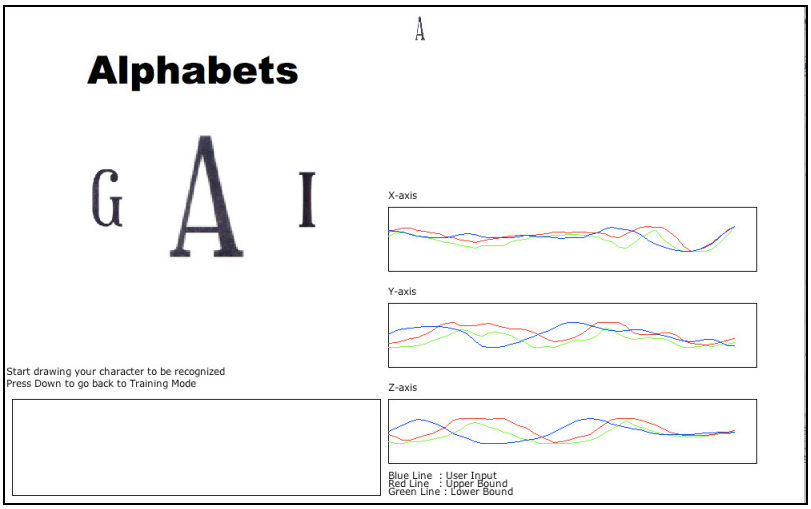

Figure 20

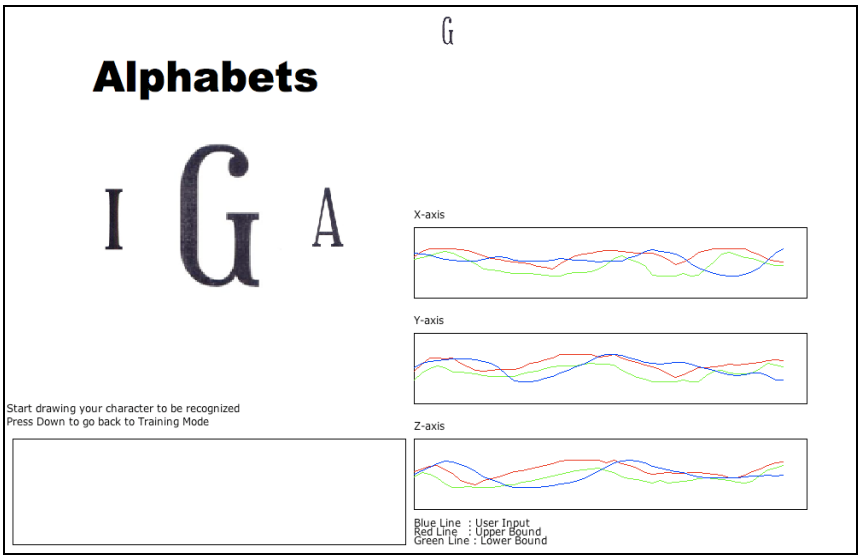

Figure 21

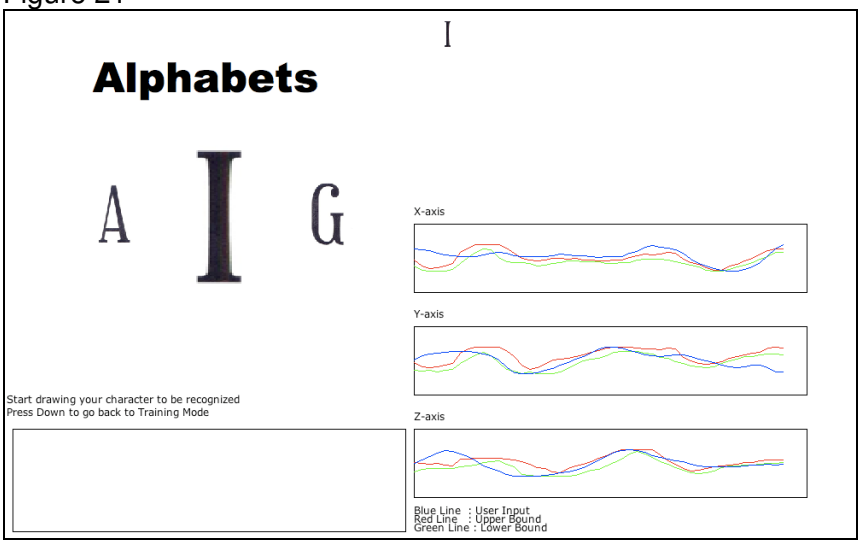

Figure 22

Mini Text Editor

Our GUI also provides a mini text editor for texts, numbers, symbols displaying on the upper right space. This allows users to be able to continuously keep writing characters. To backspace, simply press the “Up” button. Second and third closest characters can also be chosen by pressing the left and right button.

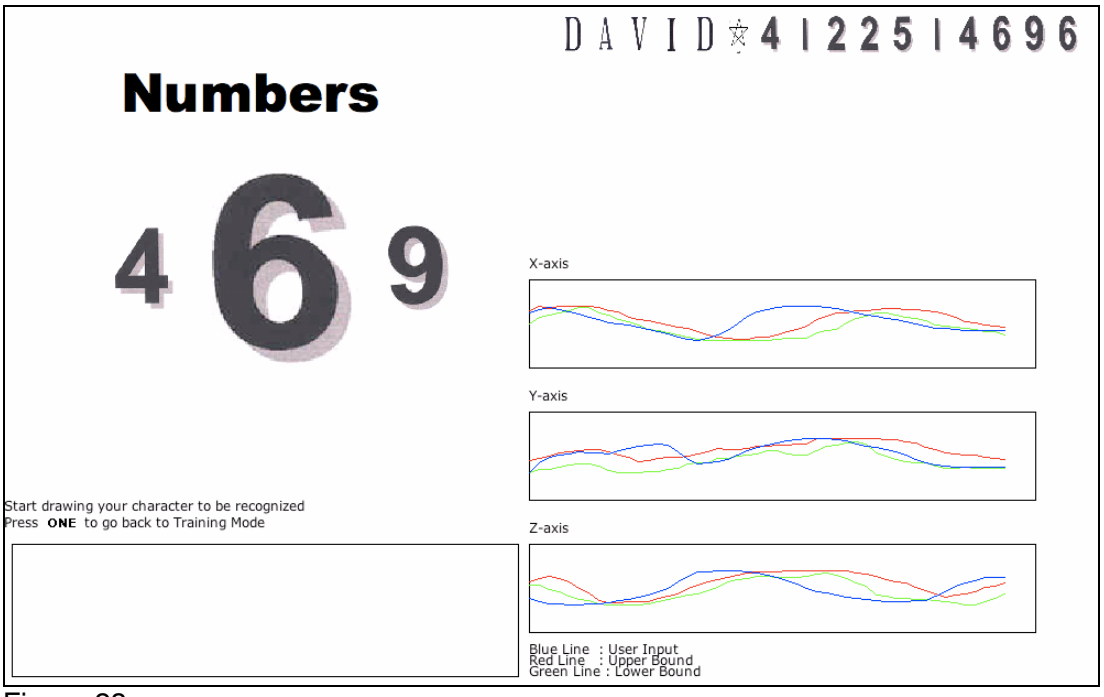

Figure 23

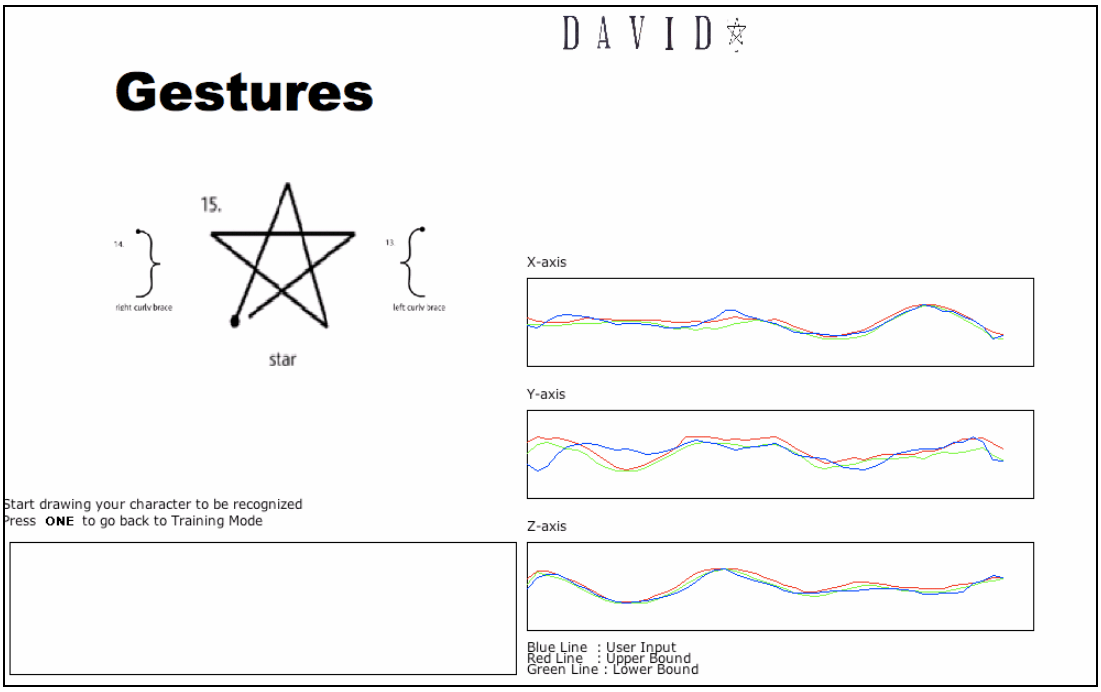

Figure 24

18. Full Semester Schedule

Dates	Goals	Who
10/08/08- 10/17/08	Determined initial DTW algorithm for classification	All
10/11/08	Found driver in JAVA to synchronize wiimote with DSK	David
10/17/08-11/15/08	Tested Algorithm – where we found fast DTW code and stress tested it.	All
10/17/08-11/15/08	Implemented fast DTW algorithm in Matlab	Jeff L, David
11/15/08-12/01/08	Implemented fast DTW algorithm in C	Jeff P
11/02/08-11/16/08	Developed a JAVA GUI for demo	Jeff L
11/15/08-12/01/08	Revamped algorithm – implemented new clustering algorithm in MATLAB	Jeff L
11/20/08- 12/01/08	Implemented final algorithm in JAVA – Resampling and normalization	David
11/20/08-12/01/08	Implemented final algorithm in C – new DTW calculations	Jeff P
11/20/08-12/02/08	Finalized the GUI. Improved for ease of use and appearance.	Jeff L

19. Task Division

There are no clear set boundaries on what each member should or should not do in this project. Almost every aspect of this project was done together. However, there are certain areas where one member contributed more than the others due to familiarity of hardware or software. In other words, each member tried to contribute more in his area of expertise.

Jeffrey Lai did most of the GUI development and algorithm testing in Matlab.

Tian Seng Leong helped with the algorithm testing in Matlab and did the PC Side Processing in Java and DSK – PC communications protocol.

Jeffrey Panza did most of the C implementation & DSK side processing in addition to helping in Matlab with algorithm testing.

20. Hardware Purchases

We spent a total of $53.28 for hardware purchases on this project. The break down of this amount is as follows:

Wii Remote Controller + Shipping	$46.94
Bluetooth 1.2/2.0 USB Adapter + Shipping	$6.88

21. Future Improvements and General Recommendations

A) Gesture Rejection Mechanism:

Our current algorithm does not include any rules for gesture rejection. Our algorithm chooses the lowest DTW output, the gesture with the shortest distance from the max-min bounds and blindly claims it to be the correct character. What if what was drawn was none of the characters in the training set? The output will look random and will probably frustrate the user.

One of our proposed solutions to reject incorrect inputs is to create a DTW threshold for every training sample. The threshold is to ensure that the DTW output is always within two standard deviations of the DTW value for the successful match before the output alphabet for the corresponding lowest DTW score is considered a match. For English characters, all 26 threshold values will be calculated before hand using the five training samples as input to obtain the referencing DTW score.

Below is the gesture rejection mechanism:

lowest_DTW_output suggests 'B' as the output
*If (Lowest_DTW_output < 2*std (DTW_value for input = actual'B'))*
    Accept 'B' as output;
Else
    Reject 'B' as output;

Remarks: two standard deviations contains 95.45% of all possible outputs (it should be all inclusive if we add two standard deviations to the mean

Char	Top Three	Distance #1	Distance #2	Distance #3
X	XDP	0.67118	1.2314	1.8647
X	XDP	0.5496	1.1124	1.4765
X	XDP	0.66642	1.0798	1.645
X	XPD	0.58172	1.2202	1.3385
X	XDP	0.86958	0.96327	1.776
Mean	0.6677			
1.5 STD	0.8546			
2 STD	0.9169			

We thought two standard deviations might be too lenient and will not recognize any inputs as “no decision”, so we tried restricting the limit to 1.5 standard deviations.

For example, in this case we’d expect X to be recognized as correct. If we use 1.5 standard deviations it will be a “no decision”, and if we use 2 standard deviations, we will recognize X. This is just one example of why using 2 standard deviations is a safe policy to prevent false negatives. This algorithm would be added simply to inform the user that the test results would be unusual (seemingly random) since the input was nowhere near any cluster.

Now, we will introduce a new idea to set up a tolerance to which an output can be accepted. As we see in the following there are three DTW result plots. Each of these plots represent the corresponding DTW results obtained by warping test inputs A,B,C to all 26 training sets. The red dots occur at places where the DTW values are the minimum, indicating the best match for that specific test input.

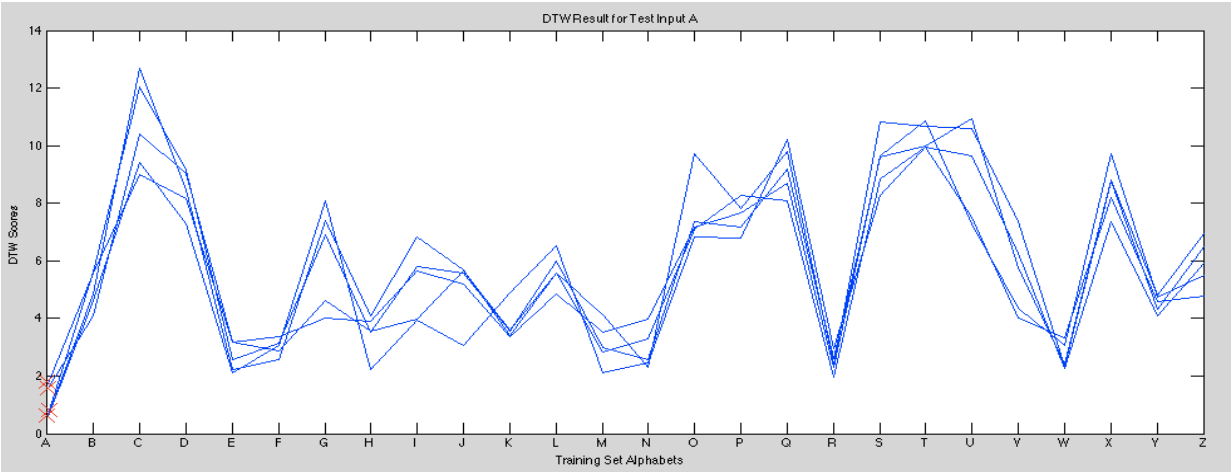

Figure 25 - E.g. Red dots occurring at Training Set A shows that the test input (A) is best matched with training set (A)

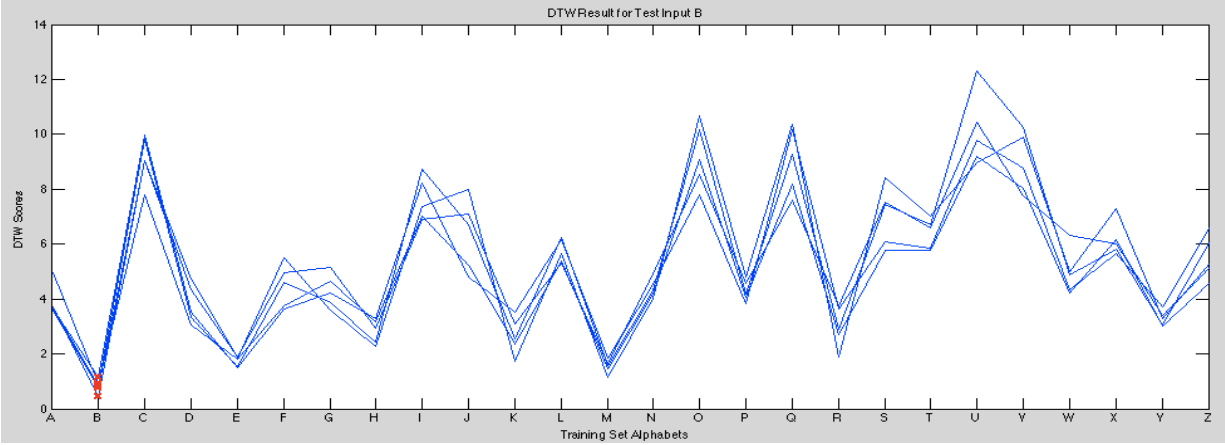

Figure 26

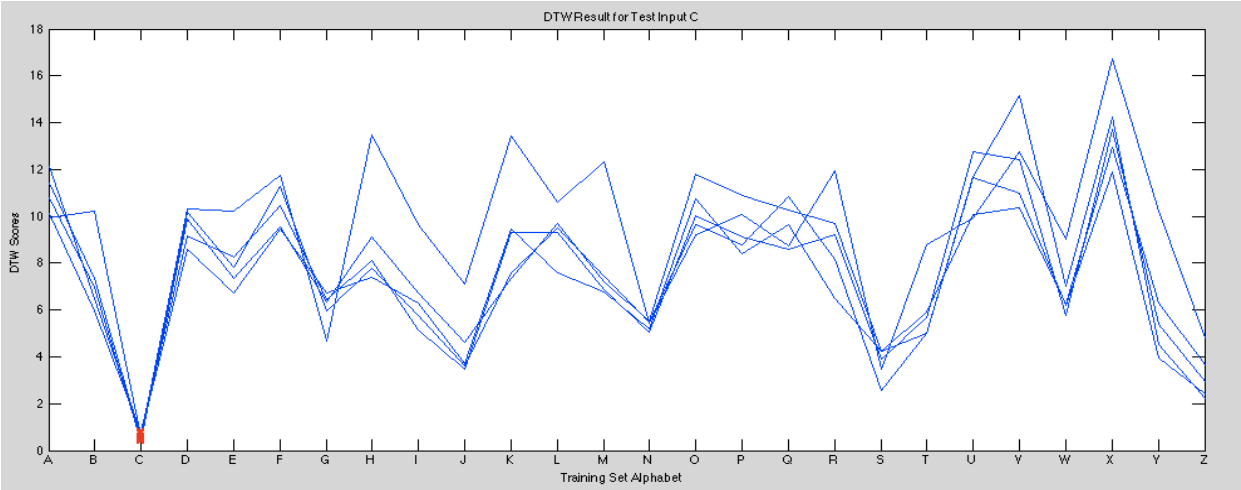

Figure 27

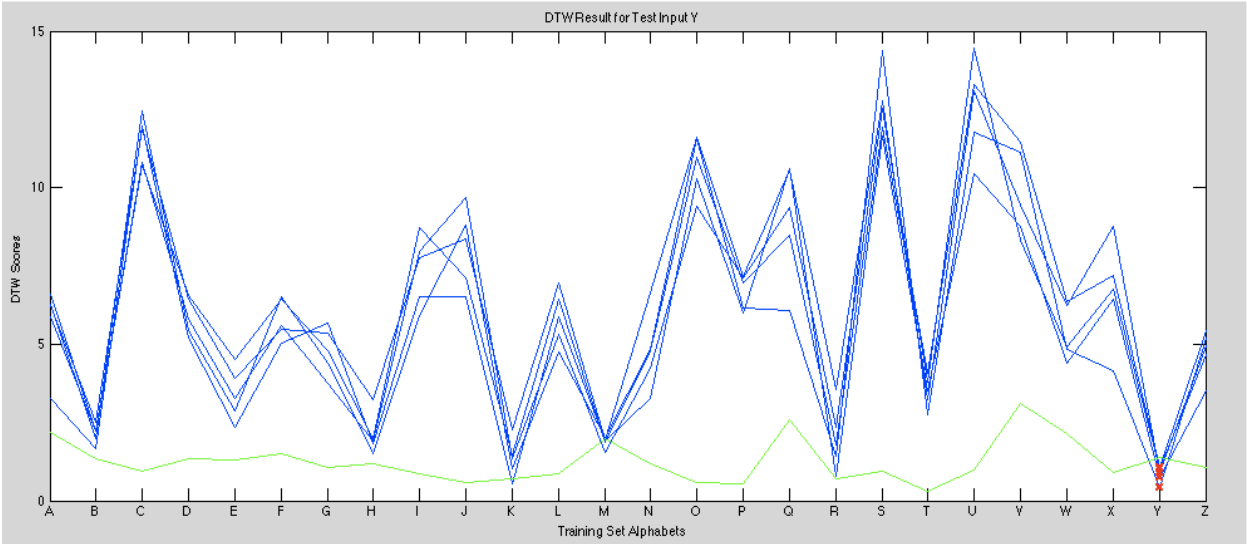

Figure 28 - The process of locating the minimum DTW values is repeated for all test input A-Z until we obtain the plot connecting every minimum red dot (Green line below). We call this line the “Gesture Rejection Threshold”.

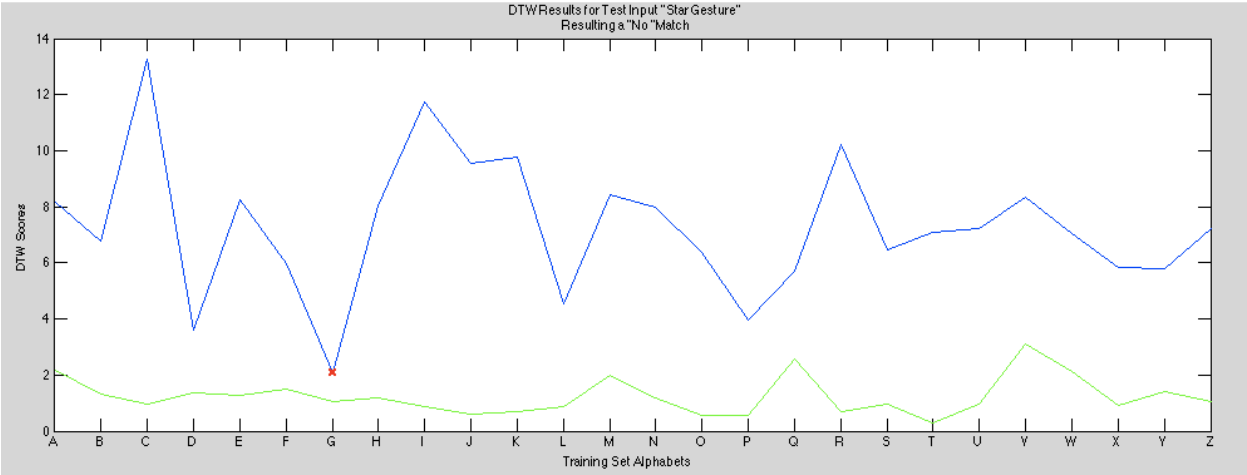

Figure 29 - "Gesture Rejection Threshold" is useful in justifying the calculated gesture outputs. For instance, a "G" might be returned to the user if he/she draws something randomly, say a "Star". But since the lowest DTW values returned at "G" is above the Gesture Rejection Threshold, the output as a result will be rejected.

Since we never implemented this on the DSK, we have no definitive answer for this problem. This is just some initial research for those who have an interest in improving our algorithm.

B) Better Formation of Bounding Curves

Consider the case where a user-inputs a varying training set. For example, the user has four consistent training sets and one really bad training set that makes the boundary for accepted inputs too lenient. Several problems can arise:

1) The cluster that gets trained may be so big it engulfs many other clusters. We noticed this issue when testing on the DSK. If we made an intentionally bad training set for one character, it would almost always return that badly trained character for any test input. This algorithm thus requires a learning curve for the user so they can make their own training set, or as we did in our demo, we made a good training set and imported it upon loading our GUI.
2) Next is the issue of training. We were hoping the user could add more training to it and it could adapt, but if the min-max ever has one bad training input, the user will have to clear all the training sets and start again to make sure there isn't an oversized cluster formed. We want something that will still stay very tight, maybe not so accepting to inputs.

Our idea for a future improvement was, instead of generating a max and min curve explicitly from the five training sets, to generate a max and min bound from the mean by adding or subtracting a standard deviation or two to the curve. We tried implementing this in Matlab and for tests involving both Jeff Lai's training and test data, the accuracy went down by about 2% consistently from the max-min bound method. And, when doing cross tests, where Jeff Lai's training set would be used and Jeff Panza's inaccurate training set was

used, the numbers went up by about 5-10%. This implies that this method could be used to possibly increase a more user-independent system especially if people have lots of trouble learning it. But, if all the users have passed the learning curve with the system, then we get better results using the min-max (probably because the user is good enough to make a good training set and to follow it). Thus we feel that the max-min bound is still the best algorithm which is why in the end we chose it to be our final algorithm. We wish we could have coded both though, since most of the problems with the max-min algorithm seemed to be resolved by this other version.

Figure 30 - http://en.wikipedia.org/wiki/Wiimote

Nintendo recently announced the Wii MotionPlus add-on to the Wii remote controller that accurately traces motions in 3-D space using improved hardware.

"According to Nintendo, the device incorporates a dual-axis "tuning fork", angular rate sensor which can determine rotational motion. The information captured by the angular rate sensor can then be used to distinguish true linear motion from the accelerometer readings."

We believe that the addition of this enhancement will enable us to cancel angular motion coming from a user's wrist and elbow movements. Variations in our data are mostly caused by angular motion which seems to be unique to each writer.

With the gyroscope, there might be also a possibility of doing a 2-D projection from the acceleration data. Future groups can experiment with image processing algorithms to recognize a character if they can successfully project a 2-D image from the acceleration data.

C) Alphabet Similarities

One of the biggest challenges to recognize English alphabets is that some characters are very similar in the way they are drawn. In our experience, drawing 'K' and 'D' are particularly similar to drawing 'R' and 'P' respectively. By observing the DTW values for these troublesome characters, we can see how similar these characters really are.

The two plots below show the corresponding DTW results when K/R/D/P are warped with all A –Z training sets. The lowest valued alphabet on the x-axis indicates the closest match to the test input. (Fig.1) shows the DTW result plot for K and R while (Fig.2) shows D and P's.

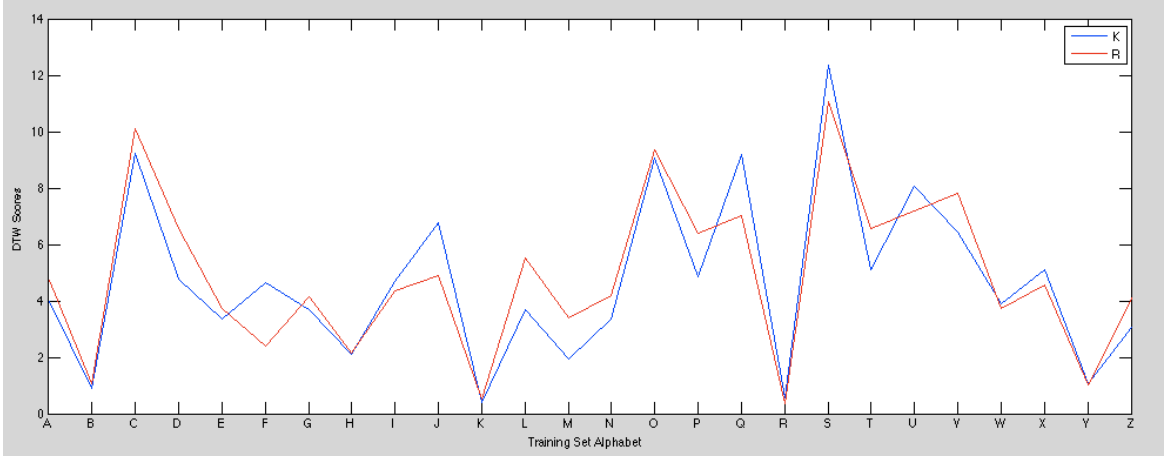

Figure 31

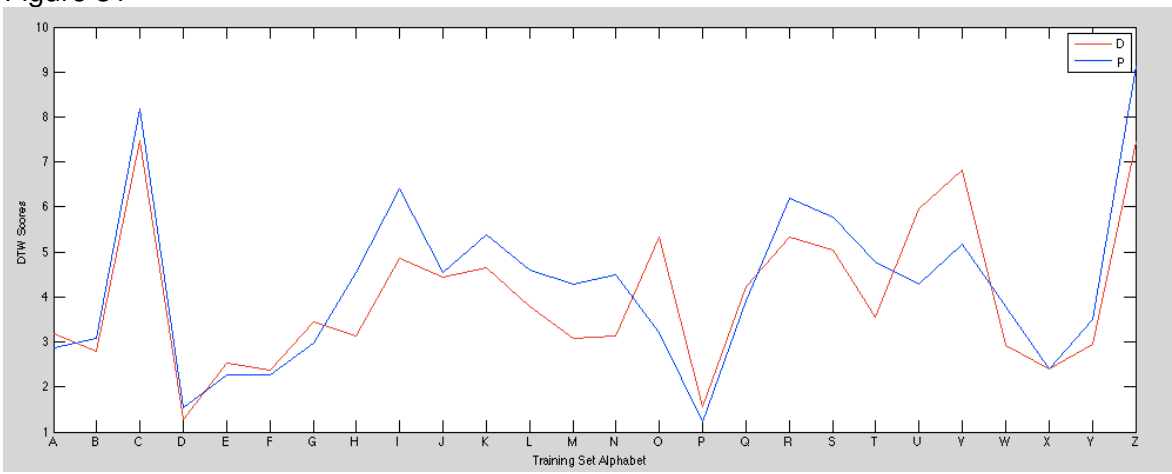

Figure 32

From the two examples above, it is obvious to see that letter K (D) could be recognized as R (P), or the other way around since they both have the lowest peak at the same spot. However, our new algorithm is smart enough to take care of this subtle difference and still be able to produce the right result, except for one highlighted one.

Analytical Result

[Test Input]->[1st match]	[2nd match]	[3rd match]	[corresponding DTW values]
K->KRB	0.41173	0.58181	0.89671
K->KRB	0.5431	0.80175	1.3856
K->KRY	0.38362	0.526	1.2002
K->KRY	0.64066	0.66103	1.57
K->KRB	0.45108	1.0242	1.0246
[Test Input]->[1st match]	[2nd match]	[3rd match]	[corresponding DTW values]
R->RKY	0.37042	0.50804	1.0082
R->RKB	0.62143	0.69834	1.0669
R->RKB	0.39845	0.51589	1.4576
R->KRB	0.55106	0.65464	1.0038
R->RKY	0.39878	0.63668	1.2422

[Test Input]->[1st match]	[2nd match]	[3rd match]	[corresponding DTW values]	
D->DPF	0.57185	1.1314	2.7519	
D->DXP	1.063	1.3071	1.7466	
D->DPX	0.32336	0.69153	1.8722	
D->DPX	0.99424	1.5385	2.3345	
D->DPX	0.41732	1.2048	1.7177	
[Test Input]->[1st match]	[2nd match]	[3rd match]	[corresponding DTW values]	
P->PDE	0.45998	1.0806	2.5271	
P->PDX	0.46929	0.67341	1.2488	
P->PDX	0.28182	0.84716	1.4013	
P->PXD	0.43219	0.67704	0.80983	
P->PDX	0.37115	0.9114	1.7501	

22. References & Comments

[1] Wilson, D and Wilson, A. "Gesture Recognition Using the XWand". http://www.cs.cmu.edu/~dwilson/papers/xwand.pdf
A previous CMU work where an accelerometer controller was developed and tried using LTW, DTW, and HMM for simple dynamic recognition of 8 characters. HMM was discovered to be the best option

[2] Cho, S-J, et al. "*Magic Wand:* A Hand Drawn Gesture Input Device in 3-D Space with Inertial Sensors". 9th International Workshop on Frontiers in Handwriting Recognition. 2004. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1363895&isnumber=29882
Similar to 1, except used Bayesian networks on a set of 12 dynamic gestures.

[3] Schlomer, T., et al. "Gesture Recognition with a Wii Controller". Second International Conference on Tangible and Embedded Interaction. Feb. 2008. Pg 11. http://delivery.acm.org/10.1145/1350000/1347395/p11-schlomer.pdf?key1=1347395&key2=5972436221&coll=GUIDE&dl=&CFID=10254916&CFTOKEN=63031480
Uses a Wiimote as well, implements HMM and has a set of 5 dynamic gestures.

[4] Gabayan, K. and Lansel, S. "Programming-By-Example Gesture Recognition". http://www.stanford.edu/class/cs229/proj2006/GabayanLansel-GestureRecognition.pdf
Uses the same accelerometer built into a Wii. Very vague on details, but claims to have used LTW, DTW, and HMM in testing gesture recognition. Pointed us to GT2K (Georgia Tech Gesture Toolkit).

[5] Kale, K. Dynamic Time Warping. http://www.cnel.ufl.edu/~kkale/dtw.html
DTW - explained the algorithm so we could easily understand, so we could try different ideas with this function.

[6] Salvador, S. and Chan, P. "FastDTW: Toward Accurate Dynamic Time Warping in Linear time and Space". http://www.cs.fit.edu/~pkc/papers/tdm04.pdf

FDTW - explained how the code we found was much, much faster, although in the end it decreased the accuracy of our final implementation.

[7] Ellis, D. Dynamic Time Warp in Matlab. http://labrosa.ee.columbia.edu/matlab/dtw/
Initial MATLAB code that was slow, O(n^2) time.

[8] DeBarr, D. http://www.mathworks.com/matlabcentral/fx_files/12319/1/cdtw_dist.c
FDTW code that we stripped down and added to our DSK code that accepts data from PC. Tested in MATLAB and proved less useful in final version.

[9] Keogh, E and Pazzani, M. "Derivative Dynamic Time Warping".
http://www.cs.rutgers.edu/~mlittman/courses/lightai03/DDTW-2001.pdf
Discusses the possibility of doing dynamic time warping on the derivative instead of the actual data, but this was for position data from drawing a character. It also does a good job explaining DTW in a more complete mathematical way than [5]

[10] Wikipedia – Wiimote page – http://en.wikipedia.org/wiki/Wiimote
Resource for discovering new wiimote attributes that would help for future algorithms

[11] Westyn, T., et al. "Georgia Tech Gesture Toolkit: Supporting Experiments in Gesture Recognition". http://www.cc.gatech.edu/ccg/publications/westeyn_ICMI2003.pdf
A toolkit that implements HMM as a form of gesture recognition. We wanted to implement this in our project, but we wanted to finish what we started concerning DTW.

[12] Lucowicz, P., et al. "Recognizing Workshop Activity Using Body Worn Microphones and Accelerometers". PERVASIVE. 2004. Pg 18.
http://www.springerlink.com/content/bdkmbgv8d57dhbl4/fulltext.pdf
One of the papers referenced in [11] that used accelerometers. Not used, but could be useful for other accelerometer related projects.

[13] Green, R. "LEDataStream". http://mindprod.com/jgloss/ledatastream.html
LEDataStream is a little-endian analog to DataInputStream and OutputInputStream that lets a developer write little-ending Intel format binary, least significant byte first. This class is needed since the DSK is little-endian while Java is big-endian.