Project Report

FIT3036 – Computer Science Project

James White - 22129030

12

Project Report

Contents

Word Count: ~3500 (not including tables)

Introduction

Online handwritten signature verification (online refers to the use of special hardware such as a tablet device to capture the movement of a pen as signature data) provides a very effective, cheap and easy method of verifying a person's identity – and because of this it is an area still heavily studied to this day.

[Gupta 2006] discusses the state of the art in online handwritten signature verification, including multiple techniques of verifying human signatures. These techniques are more broadly referred to as the parametric or functional approach (with some combination the techniques forming a hybrid approach). This project focuses on the parametric approach. In this approach a set of global features is extracted from a signature (the signature data obtained from a tablet device), computed and then compared to those feature values computed from another signature. It is worth noting that this approach will calculate and store global features but does not store or compare the signature itself (this 'feature only' representation is sometimes referred to as compression).

This Java-based online handwritten signature verification (OHSV) system implements a parametric technique of signature verification. The parametric technique allows a large number of global features or values to be computed about any signature; however the exact features used within this system are discussed further within the report. This system aimed to keep error rates such as FAR (False Acceptance Rate), FRR (False Rejection Rate) and EER (Equal Error Rate) as low or as close to 0% as possible, but has achieved ERR of approximately 11%, or FAR = ~8% FRR = ~14% in a best case scenario. The ultimate objective of keeping this error rate close to 0% is to assess the long term, wide scale viability of HSV as a secure digital method of user identification.

Project Plan

Overview

The purpose of this project was to select a HSV technique and implement it within a system – In this case the parametric approach was chosen. This system utilises the selected technique to determine the legitimacy of a given signature (from a provided database of 60+ different people, including genuine and forged versions). This system has been evaluated by its implementation and effectiveness of this approach, and by its compliance with any requirements specified below.

The system takes multiple genuine signatures (1…*, but at least 5-10 preferably), calculates a threshold value (using weighted Euclidean distance) based on the chosen global features and builds a "reference signature". The system then allows test signature comparisons with the reference signature, judging the legitimacy of the test signature.

Functional requirements

Some functional requirements have been either omitted or altered since the project specification – this primarily includes any functionality related to the storage of 'enrolled' data between uses. These requirements were altered due to certain constraints.

- System is able to take a set of signature data containing one or multiple signatures (no minimum) and construct a reference signature. This reference signature consists of certain feature means, their standard deviations, a mean weighted Euclidean distance (WED, described later) and the WED standard deviation.
- System is able to (once a reference signature has been created) take a single test signature data set and compare its features with the reference data set, and display the results (its verification guess of genuine or forgery) of this comparison.
- System has a GUI (graphical user interface) that provides the user with each function the system provides.
- System also supports a (more limited) CLI (command line interface).

Non-Functional Requirements

System qualities include:

- The system is easy to use.
- The system is effective to the error rate set for it in the project specification (<30%), with FAR (False Acceptance Rate) of ~8-9%, and FRR (False Rejection Rate) kept to ~14-15% in good conditions.
- The system is portable (a benefit of using Java) – can be run on any major operating system that has a fully function JRE (Java runtime environment).
- The system is fairly light on resources.
- The system is easily modifiable – new comparison features can be added with minimal effort.

Constraints

There are numerous constraints that have limited and defined the scope of the system – these primarily include:

- Various project deliverable deadlines.
- Software development and HSV knowledge of stakeholders.
- Dataset – provided a limited database of signatures to test with.
- Limited access to computing and other hardware resources (must work on consumer desktop computer).
- Limited workforce with only a single contributor.

Risk Analysis

To avoid risks and mitigate their effects, first they must be identified and a plan must be established. A list of the risks most likely to affect this project and their mitigation strategies are listed in the table below:

Risk	Impact	Probability	Mitigation
Miss Deadline	High	Moderate	• Keep and stick to schedule • Make sure other risks are mitigated
Knowledge insufficient to complete a task	Moderate	Moderate	• Identify topic or area of insufficient knowledge • Research topic until confident in ability to complete the task
Project scope too large	Moderate	Moderate	• Make sure project requirements are understood • Ensure project analysis and design is carried out and viability is confirmed be before implementation begins • Ensure schedule is viable
Need to change software architecture or component	High	Low	• Make sure project requirements are understood • Ensure project analysis and design is carried out and viability is confirmed be before implementation begins
Need to change HSV technique	High	Low	• Make sure project requirements are understood • Ensure project analysis and design is carried out and viability is confirmed be before implementation begins
Misunderstood requirement	Moderate	Low	• Carefully read any documentation regarding project requirements • Make sure all terms, techniques and technologies involved are understood
Data Loss	High	Low	• Important project related material (documentation, source code) is back up in multiple places including cloud services

During the course of the project development several of these risks were encountered. During the course of implementation of the HSV technique, it was found that the knowledge of several aspects of the technique were lacking. This lead to a halt in implementation while further research, analysis and design was enacted. Overall its impact was low, as the mitigation strategy provided an adequate plan to get back on track.

During implementation it was also found that the projects scope was a little too ambitious, suffering from some 'feature-creep' during the initial analysis and design phase. This resulted in certain requirements in the project specification that ultimately had to be dropped in order to complete implementation and testing to a satisfactory level before the deadline.

Resource Requirements

There are numerous resource requirements for this system. They include:

Software:

- Java JDK (to develop)
- Java JRE (to run)
- OS (should work on any system with an adequate JRE, Windows/Mac/Linux)
- Eclipse IDE (to develop)
- Third-party libraries
    - JFreeChart – used to provide graphs for the GUI
    - WindowBuilder Pro – for quick GUI design
    - JUnit – for testing

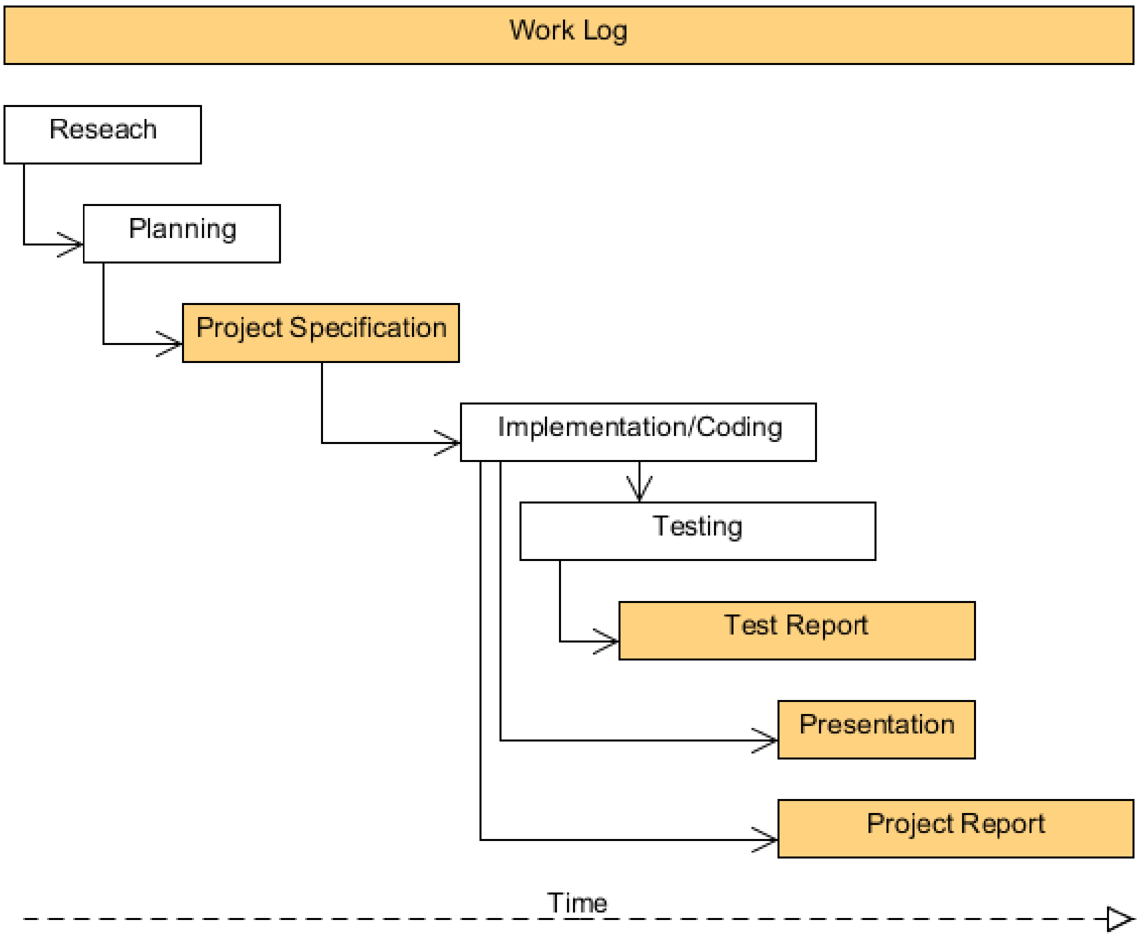
Hardware:

- Computer to run/develop the system
- Tablet to collect signatures (if new signatures not in database are to be tested)

Other:

- HSV literature and resources
- Signature database to test with
- Java API's and documentation
- Third-party library API's and documentation

Project Components



The diagram above shows various high level project components and their relationships over the course of time. The arrows show what components 'lead' to others, while the boxes that are shaded describe components that are also deliverables.

Schedule

The finished schedule did not deviate much from the planned schedule. For more in-depth coverage of the work done at each stage, refer to the work log book.

Date	Stage	Tasks
1 – 23/7-29/7	-	• Start workbook (updated every week)
2 – 30/7-5/8	Analysis/Design	• Studied HSV literature and techniques
3 – 6/8-12/8	Analysis/Design	• Studied HSV literature and techniques • Picked a major approach (Parametric)
4 – 13/8-19/8	Analysis/Design	• Study HSV literature (for features) • Picked parametric features (described later)
5 – 20/8-26/8	Analysis/Design	• Study HSV literature and techniques • Chose Java as software platform due to familiarity and portability. • Decided on most feature requirements • Started designing high-level software structure • Started writing project specification

6 – 27/8-2/9	Analysis/Design	• Refined software design – changed and added planned features • Produced concept/class/use case diagrams • Designed a mock GUI • Finished writing project specification
7 – 3/9-9/9	Implementation/Testing	• Started implementing basic system features o Signature file reading ▪ Everything in a file ▪ Just sigs to build a reference ▪ Just genuine sigs ▪ Just forged sigs o The more basic feature calculations like: ▪ total time taken ▪ number of pen ups ▪ pen up time • Started working on JUnit tests for implemented feature calculations
8 – 10/9-16/9	Implementation/Testing	• Continued implementing features o More feature calculations ▪ Start on velocity and acceleration related features o Feature set comparisons ▪ Means ▪ Standard deviations ▪ Constructing reference signature ▪ Constructing test signature o Implemented CLI • Added more JUnit tests for the new features
9 – 17/9-23/9	Implementation/Testing	• Continued implementing features o More feature calculations ▪ Finished implementing velocity and acceleration features o Feature set comparisons ▪ WED ▪ Finished reference sig construction o Started outline of the GUI • Added more JUnit tests and other manual tests for newly implemented features
10 – 24/9-30/9 (Break)	Implementation/Testing	• Continued implementing features o Focused on finding a good threshold heuristic o Continued working on GUI o Implemented graphs for the GUI (shape and profile of test sig) • Added more testing for implemented features
11 – 1/10-7/10	Implementation/Testing	• Implemented a large test to allow quick feedback on different thresholds • Polished system for demonstration • Polished GUI • Fixed many remaining bugs

		• Began Test Report • Began Final Report • Began working on presentation
12 – 8/10-14/10	Implementation/Testing	• Polished system for demonstration • Finished working on presentation • Decided on threshold heuristic • Continued working on Test Report • Continued working on Final Report
13 – 15/10-19/10	-	• Finished Test Report • Finished Final Report

External Design

Production and Deployment

Dependencies

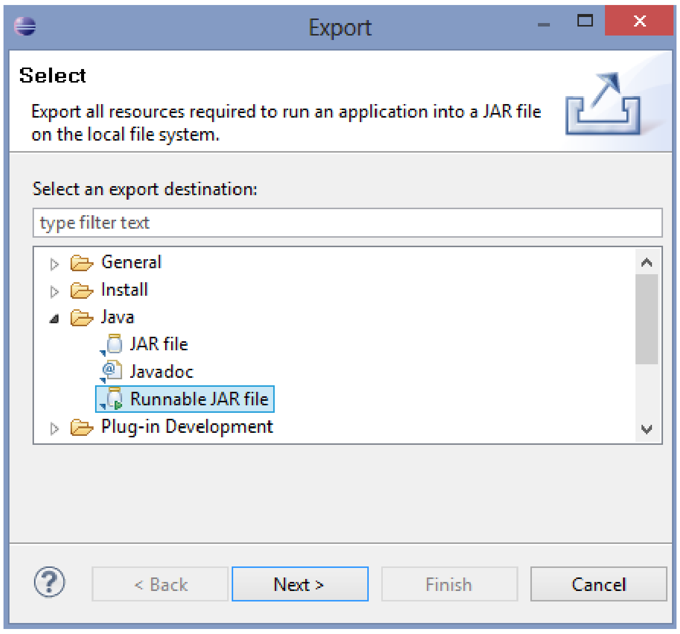
A small number of technologies and libraries were used to produce this project, leading to a few dependencies:

- Java JDK (to implement/compile)
- Java JRE 1.6+ (to run)
- JUnit 4.0+ (for test suites)
- JFreeChart 1.0.14 (for GUI graphs)
- Windowbuilder Pro (March 27th or later)
- Any platform with suitable JRE (Windows, Mac OSX, Linux)

Compilation

This project was implemented with the use of the Eclipse IDE. A precompiled build of the software should be provided along with this report, however, to compile within Eclipse:

File → Export → Java / Runnable JAR File:



Export

Select

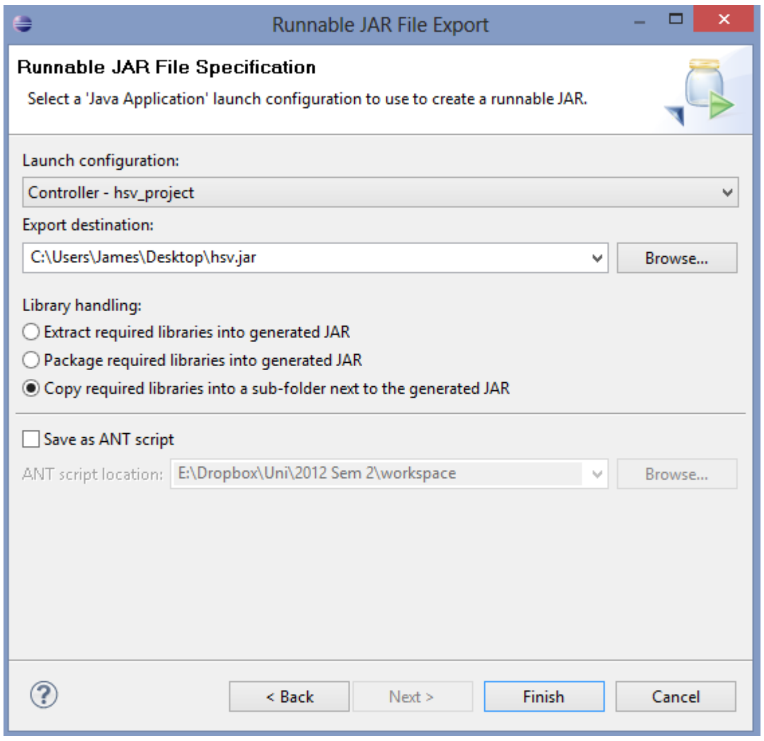
Export all resources required to run an application into a JAR file on the local file system.

Select an export destination:

type filter text

Figure 1 – Exporting Process

For launch configuration, make sure the Controller class is selected:



Runnable JAR File Export

Runnable JAR File Specification

Select a 'Java Application' launch configuration to use to create a runnable JAR.

Launch configuration: Controller - hsv_project

Export destination: C:\Users\James\Desktop\hsv.jar

Library handling:
- ( ) Extract required libraries into generated JAR
- ( ) Package required libraries into generated JAR
- (●) Copy required libraries into a sub-folder next to the generated JAR

[ ] Save as ANT script

ANT script location: E:\Dropbox\Uni\2012 Sem 2\workspace

Figure 2 – Exporting Process (2)

Running

There are two options when running this software – a CLI with a limited feature set, and a GUI.

CLI Usage:

From the terminal/command prompt, navigate to the directory where the *.JAR file is stored and type the following:

```
java –jar hsv.jar refSig.file testSig.file
```

GUI Usage:

To launch the GUI, simply double click the `hsv.jar` file, or run the CLI command without an argument, like so:

```
java –jar hsv.jar
```

Parameters

CLI:

The CLI usage has two parameters;

- **`refSig.file`** **–** the signature file that contains the multiple signatures used to create the reference signature. Usually this file should contain 5-10 signature samples.
- **`testSig.file`** – the signature file that contains a single signature that we want to verify the authenticity of.

GUI:

The GUI also requires very little input to operate, and in fact uses the same parameters as the CLI. To use the GUI;

1. From the picture below, press the button labelled with a 1. This will launch a file browser dialog. Select a signature file (**`refSig.file`**) that contains your reference signatures.
2. Press the button labelled with a 2. This will launch a file browser dialog again. Select a signature file (**`testSig.file`**) that contains your test signature.

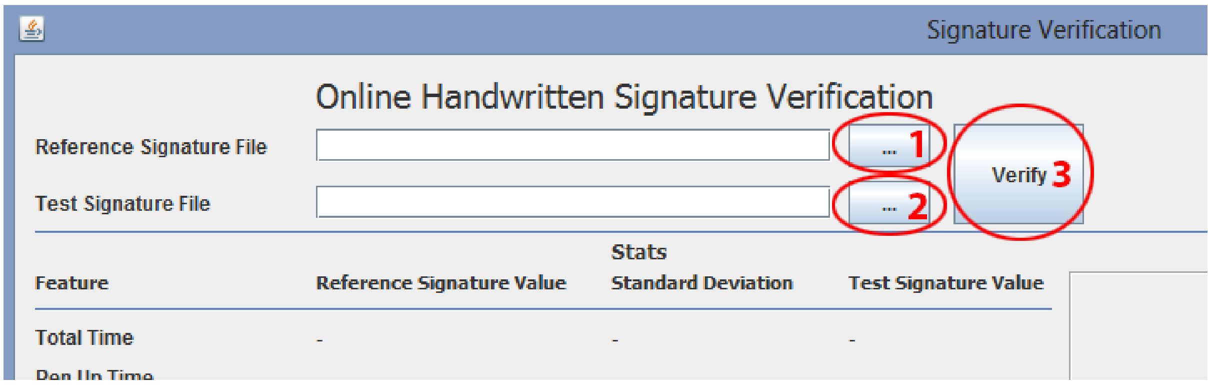
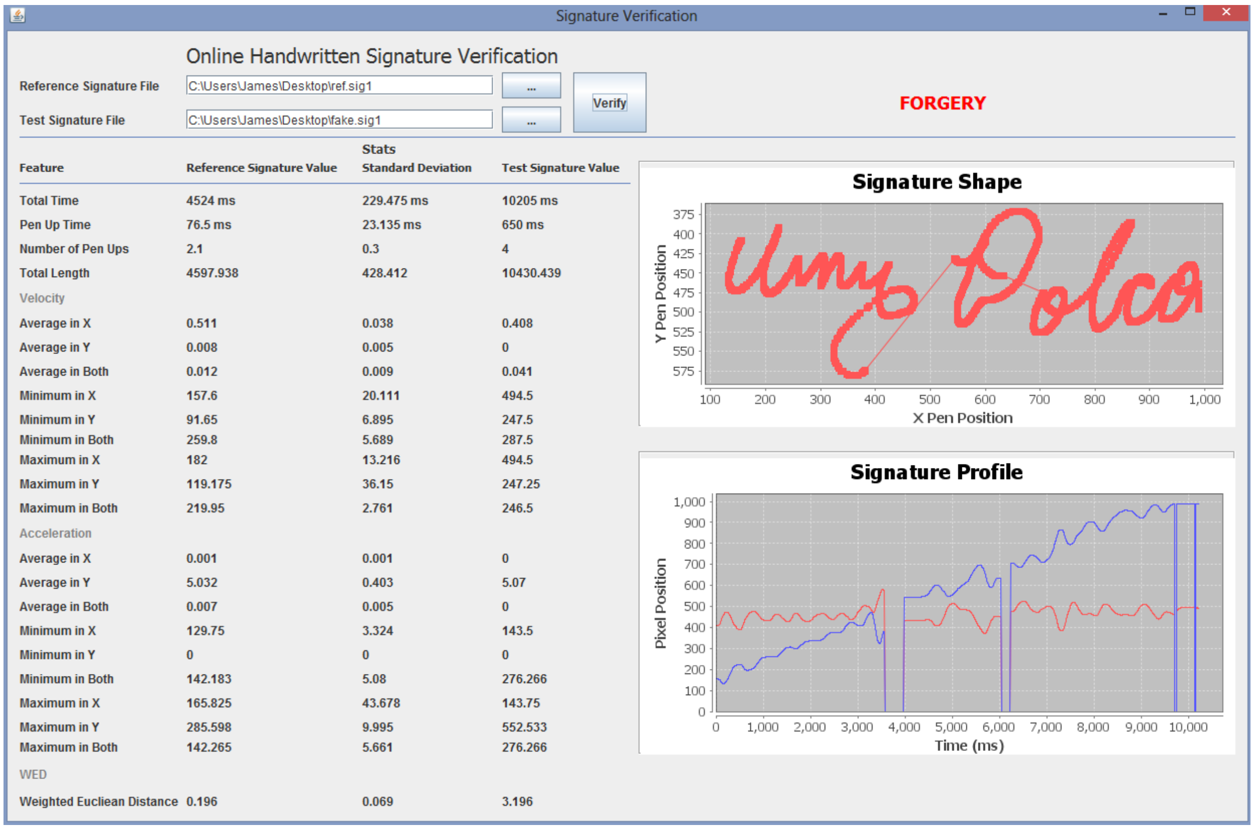


Figure 3 – GUI Parameters

User Interface

The main user interface for the software is the GUI. After launching the GUI and entering your parameters (as described above), click the 'Verify' button labelled with a 3 in figure 3. The software will then process the signatures, attempt to verify the test signature, and the display the results, as shown in figure 4.



Feature	Reference Signature Value	Stats Standard Deviation	Test Signature Value
Total Time	4524 ms	229.475 ms	10205 ms
Pen Up Time	76.5 ms	23.135 ms	650 ms
Number of Pen Ups	2.1	0.3	4
Total Length	4597.938	428.412	10430.439
Velocity			
Average in X	0.511	0.038	0.408
Average in Y	0.008	0.005	0
Average in Both	0.012	0.009	0.041
Minimum in X	157.6	20.111	494.5
Minimum in Y	91.65	6.895	247.5
Minimum in Both	259.8	5.689	287.5
Maximum in X	182	13.216	494.5
Maximum in Y	119.175	36.15	247.25
Maximum in Both	219.95	2.761	246.5
Acceleration			
Average in X	0.001	0.001	0
Average in Y	5.032	0.403	5.07
Average in Both	0.007	0.005	0
Minimum in X	129.75	3.324	143.5
Minimum in Y	0	0	0
Minimum in Both	142.183	5.08	276.266
Maximum in X	165.825	43.678	143.75
Maximum in Y	285.598	9.995	552.533
Maximum in Both	142.265	5.661	276.266
WED			
Weighted Eucliean Distance	0.196	0.069	3.196

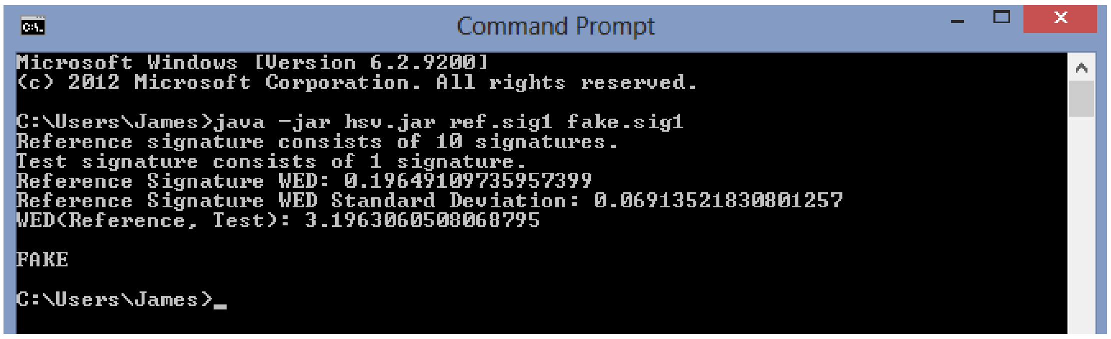
Figure 4 – GUI Usage

At the top right the result of verification is displayed. If the software thinks the test signature is forged, it will display 'FORGERY' in red text. If the software thinks it is genuine, it will display 'GENUINE' in green text.

On the left of the GUI, the software will display the results of feature calculations (only displayed to 3 decimal points – this sometimes results in a '0' being shown), including reference signature means/standard deviations, and test signature values.

On the right are two graphs of the test signature. The above is the shape of the signature, and below is its XY profile. Because these graphs are produced using JFreeChart, each graph is interactive – this means you can zoom in/out, and change axis scaling. Currently there is a bug in which these graphs will not change if you try and verify *another* test signature without first closing and reopening the software.

Using the CLI has been demonstrated previously, and the only UI is the parameter input and text output of the software. Figure 5 demonstrates this usage:



```
Microsoft Windows [Version 6.2.9200]
(c) 2012 Microsoft Corporation. All rights reserved.

C:\Users\James>java -jar hsv.jar ref.sig1 fake.sig1
Reference signature consists of 10 signatures.
Test signature consists of 1 signature.
Reference Signature WED: 0.19649109735957399
Reference Signature WED Standard Deviation: 0.06913521830801257
WED(Reference, Test): 3.1963060508068795

FAKE

C:\Users\James>_
```

Figure 5 – CLI Usage

Performance

Theoretical Performance:

Time and space complexity (Big-O) can be difficult to ascertain in non-trivial code bases. In terms of time complexity the design would be close to O(n). Space complexity is quite large however – as the design has focussed on a simpler, more easily modifiable approach the space complexity has suffered. This is an acceptable trade off however, as the system should never use enough space to cause issues on reasonably modern hardware.

Real World Performance:

These tests were performed on a mid-high end consumer desktop (Intel core i5, 8GB RAM).

Using the GUI, the time taken between clicking the ‘Verify’ button and all results being returned averaged approximately 0.19 seconds. Using the CLI, the time taken between entering the command and receiving a result averaged approximately 0.23 seconds.

This discrepancy in real world execution time is interesting, as the CLI performs less work than the GUI, although it may have to do with small time delays in outputting text to the console.

Memory usage is tough to ascertain, as the software is tied to the Java JRE which itself causes quite an overhead. During testing this JRE hosting the software averaged approximately 40MB of memory usage (but this may vary slightly depending on JRE and platform).

Internal Design

For the main function of the system (comparing a test signature to a reference signature from the GUI) a sequence diagram (Figure 6) has been constructed to demonstrate the overall interaction between software components and the logic flow.

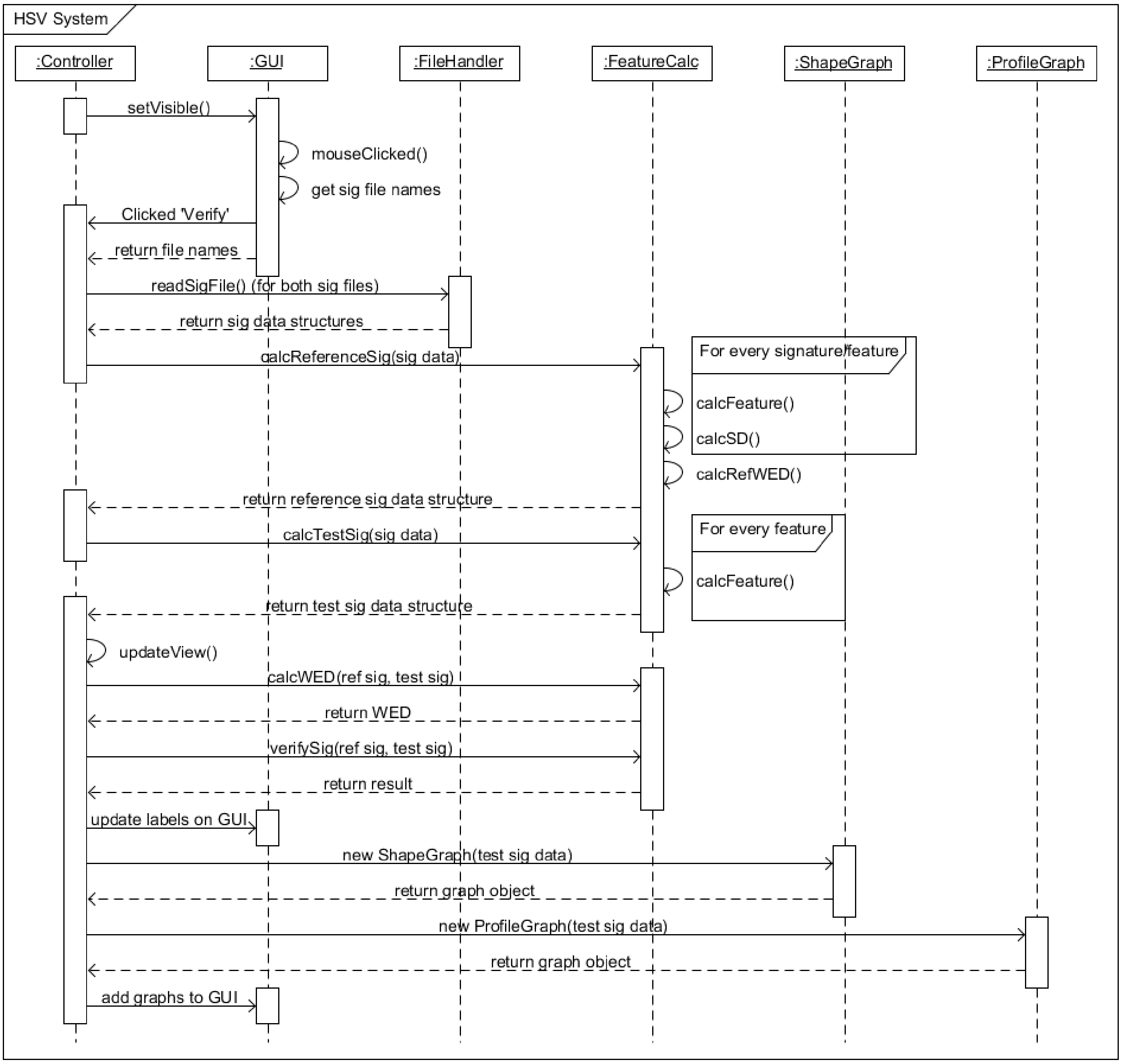


Figure 6 – Sequence Diagram

The Controller class controls the logic flow of the system – it has access to each of the other software components and hands off control to another more specialised component to accomplish a given task, e.g. the controller will request that the FileHandler read a file (by calling readSigFile()) containing signature data obtained from a tablet and return that data to the controller in a more suitable format or data structure. This allows the controller to then in turn pass that data structure to another component for further work (such as the FeatureCalc class).

The system incorporates the use of some third-party libraries and software components. The signature shape and profile graphs are drawn using the third-party library JFreeChart, while the GUI was primarily built with the assistance of WindowBuilder Pro.

Software Architecture

Class Diagram

Figure 7 describes the final class diagram of the system. The system has been split into two main packages or components – the Model and the View. The Model package is responsible for the control and logic flow of the system, as well as the systems state and other model information such as file reading/writing, feature calculation and signature comparison. The View package handles classes relating to the users 'view' of the system (however the role of the CLI is performed by the Controller).

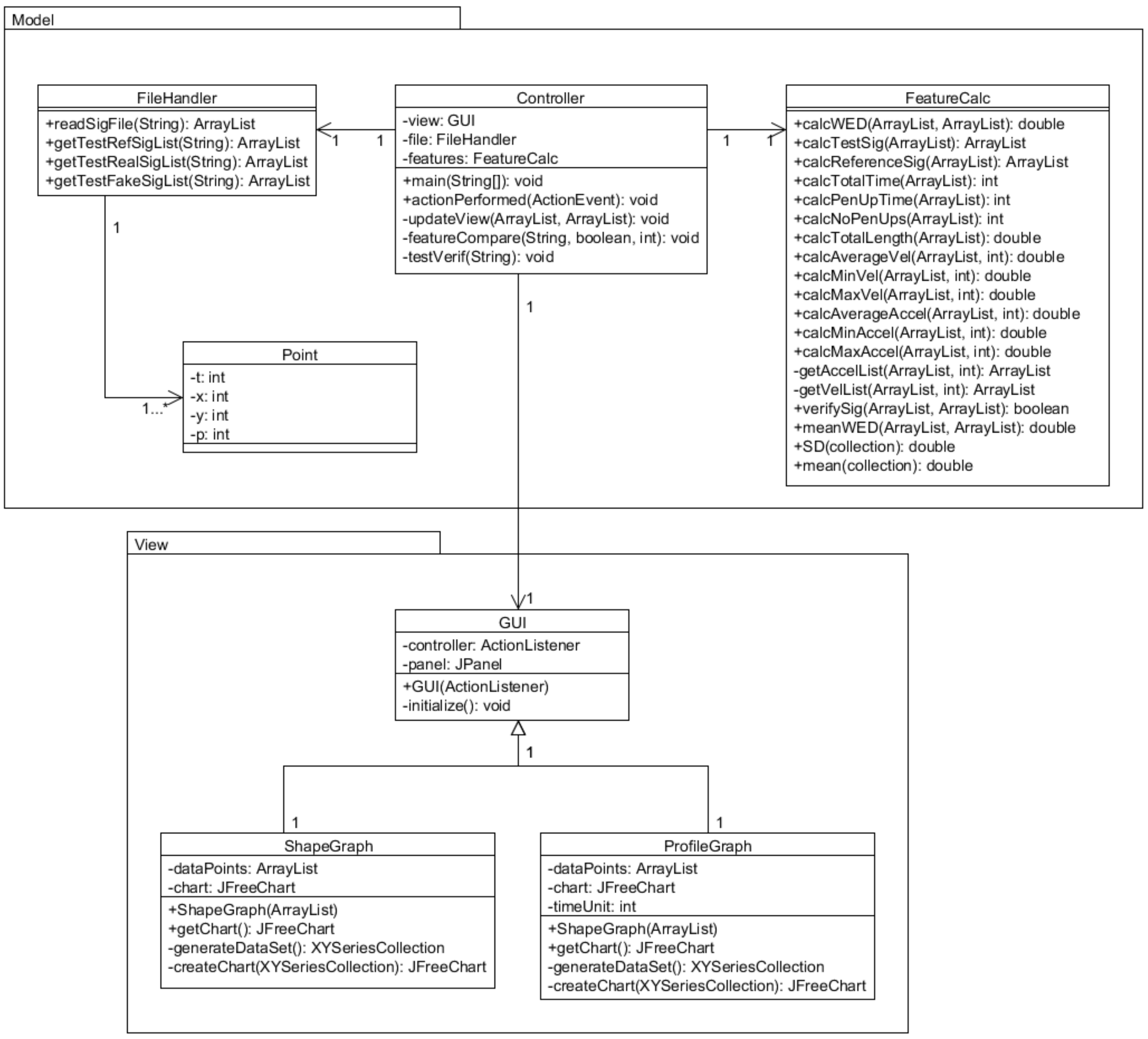

Model
FileHandler
+readSigFile(String): ArrayList
+getTestRefSigList(String): ArrayList
+getTestRealSigList(String): ArrayList
+getTestFakeSigList(String): ArrayList
Controller
-view: GUI
-file: FileHandler
-features: FeatureCalc
+main(String[]): void
+actionPerformed(ActionEvent): void
-updateView(ArrayList, ArrayList): void
-featureCompare(String, boolean, int): void
-testVerif(String): void
FeatureCalc
+calcWED(ArrayList, ArrayList): double
+calcTestSig(ArrayList): ArrayList
+calcReferenceSig(ArrayList): ArrayList
+calcTotalTime(ArrayList): int
+calcPenUpTime(ArrayList): int
+calcNoPenUps(ArrayList): int
+calcTotalLength(ArrayList): double
+calcAverageVel(ArrayList, int): double
+calcMinVel(ArrayList, int): double
+calcMaxVel(ArrayList, int): double
+calcAverageAccel(ArrayList, int): double
+calcMinAccel(ArrayList, int): double
+calcMaxAccel(ArrayList, int): double
-getAccelList(ArrayList, int): ArrayList
-getVelList(ArrayList, int): ArrayList
+verifySig(ArrayList, ArrayList): boolean
+meanWED(ArrayList, ArrayList): double
+SD(collection): double
+mean(collection): double
Point
-t: int
-x: int
-y: int
-p: int
1
1
1
1
1
1...*
1
View
1
GUI
-controller: ActionListener
-panel: JPanel
+GUI(ActionListener)
-initialize(): void
1
1
1
ShapeGraph
-dataPoints: ArrayList
-chart: JFreeChart
+ShapeGraph(ArrayList)
+getChart(): JFreeChart
-generateDataSet(): XYSeriesCollection
-createChart(XYSeriesCollection): JFreeChart
ProfileGraph
-dataPoints: ArrayList
-chart: JFreeChart
-timeUnit: int
+ShapeGraph(ArrayList)
+getChart(): JFreeChart
-generateDataSet(): XYSeriesCollection
-createChart(XYSeriesCollection): JFreeChart

Figure 7 – Class Diagram

File Formats

The signature file format used as an input to the system:

ID	±ID	NumberOfPoints
x_1	y_1	p_1
…		
x_n	y_n	p_n

Sign	Meaning
ID	The unique ID number of the person whose signature this is
±ID	The unique ID number of the person whose signature this is • +ve = genuine signature • -ve = forgery
NumberOfPoints	The total number of data points (lines) in this signature (not including the first line)
x	Pen x-axis position
y	Pen y-axis position
p	Pressure • 0 = pen lifted • 1 = pen down

Parametric Features

This system focuses on the parametric approach to on-line HSV – therefore there are a number of global signature features that the system must compute and compare. The features implemented within the system have been selected as a small subset of the features deemed most effective from previous work, namely [Crane and Ostrem 1983], [Lee et al.1996], [Ketabdar et al. 2005] and [Fierrez-Aguilar, Nanni et al. 2005].

- Total Time to write the signature
- Total pen up time
- Number of pen ups
- Total Line Length
- Average, minimum and maximum velocity in X
- Average, minimum and maximum velocity in Y
- Average, minimum and maximum velocity in both axis
- Average, minimum and maximum acceleration in X
- Average, minimum and maximum acceleration in Y
- Average, minimum and maximum acceleration in both axis

Distance Measurement

In order to compare feature sets between the reference and test signature, a method to compute their distance must be employed. This system employs the Weighted Euclidean Distance (as shown in Fig 8):

R is the reference signature with feature means *($r_1, r_2, \ldots, r_n$)* and standard deviations *($s_1, s_2, \ldots, s_n$)*
T is the test signature with feature set *($t_1, t_2, \ldots, t_n$)*
G(T) provides the WED between test signature *T* and the reference signature *R*.

$$G(T) = \left(\frac{1}{n}\right)\sqrt{\sum_{i=1}^{n}\left(\frac{t_i - r_i}{s_i}\right)^2}$$

Figure 8 – Weighted Euclidean Distance

Threshold Heuristic

The aim of this threshold was to develop a method that would scale well with the natural WED variance between different signatures of a single person. Another goal of this threshold was that it should be different for each person.

In order to accomplish this I first tried to sample the variance between different signatures from a single person. To do this I first calculate the feature means and standard deviations for each signature in the reference signature file. Once this is complete I begin a process of comparing each of these signatures against the rest of the set.

For example, a reference signature consists of 5 genuine signatures. In order to compare them I take a single one of those signatures and pretend it is a test signature, calculating WED by feeding it this 1 test signature and the other 4 signatures as a reference signature. This process is repeated for each signature within the set (in this case 5 times in total).

The standard deviation of these five WED results is then calculated, and the WED results averaged to a mean result. Using this new data the threshold can be described as so:

To be genuine, **WED(Reference, Test) < referenceWED_Mean + n(referenceWED_SD)**

Where *n* is an arbitrary number (of standard deviations away). This threshold tries to emulate a normal distribution. For example, using *n* = 4 should account for 99.994% of possible values given the sampled set.

This is not always accurate though, for a number of possible reasons. In order to find the best value for n for this system, n was continuously changed and the results recorded to produce the graph under Experimental Results (figure 10). We can see from this graph that the best value for *n* = 8 with an a FAR of 8.79% and FRR of 14.33%, but obviously this can be changed depending on the requirements of the system (high/low security or high/low convenience).

Interpretation and Analysis

Ultimately within the information technology industry the primary reason for researching the use of various biometrics (such as online handwritten signature verification) is to discern its suitability for use within fields that require secure forms of identification. As the world moves to a society more heavily reliant on digital technologies, the need to find an appropriate secure identification method only increases. Signatures have become a popular choice of biometric due to its prevalence in analogue mediums, and its status as being non-invasive.

This project aimed to assess this viability on a naïve level by implementing a technique of HSV, with a goal of producing an implementation that would achieve a minimal FAR and FRR – thresholds needed to be suitable in real world use depend on importance of security.

Much research has already been performed within the field, and many of these results have informed the direction of this project. The early work provided by [Osborn 1929] sets the basis by providing many observations about signature variations, also recommending 5-10 signatures be used as training or sample sets. Osborn also noted many signature features, however most feature choices implemented within this project stem from the works described earlier in this document under Parametric Features. Work by [Gupta 2006] has also directed the basic methodology to evaluating the performance of the HSV technique, as well as more accurate ways to sample feature data (such as using central differences to reduce error for velocity and acceleration calculations).

Experimental Results and Interpretation

After the implementation of the overall technique (including features and threshold) was complete, a method was needed to determine the project's success. In order to assess the success and viability of the techniques involved, the metrics FAR and FRR of a dataset are used.

To obtain the best results possible for FAR and FRR, a value for *n* needed to be chosen from the threshold below:

```
WED(Reference, Test) < referenceWED_Mean + n(referenceWED_SD)
```

To quickly find this data a test class was devised to test each signature within the supplied signature database. For each file (usually 15 genuine signatures, 5 forgeries), this process would build a reference signature from the first 10 signatures and then test this against the remaining genuine and forged signatures while recording results. This allowed much quicker result gathering.

As can be seen in the results table (figure 9), *n* = 8 produced the lowest combined error rate of 23.12%. The 'best' value may differ wildly however, based on the purpose of the system – a system where security must be high may put more value in keeping unauthorised users out, requiring a low FAR where something like *n* = 4 maybe be more appropriate. On the other hand, a fairly low security consumer system may prefer a lower FRR to not inconvenience its users, and thus choose a larger value of *n*.

These results do not provide any significant insights as to the long term viability of this HSV approach. Previous work as early as [Sternberg 1975] with FRR = 6.8%-17% and FAR = 3.2%, have achieved more successful results before.

n Value	FAR	FRR	Combined Error Rate
1	0.30%	71.00%	71.30%
2	0.61%	58.33%	58.94%
3	1.52%	46.33%	47.85%
4	2.73%	34.67%	37.39%
5	3.94%	28.33%	32.27%
6	5.45%	22.00%	27.45%
7	7.27%	16.00%	23.27%
8	8.79%	14.33%	23.12%
9	11.82%	11.67%	23.48%
10	13.94%	10.67%	24.61%
11	16.67%	10.00%	26.67%
12	19.09%	9.33%	28.42%
13	20.61%	9.33%	29.94%
14	21.52%	7.67%	29.18%
15	22.42%	6.33%	28.76%
16	23.94%	6.00%	29.94%
17	24.24%	5.67%	29.91%
18	26.67%	5.33%	32.00%
19	28.18%	5.33%	33.52%
20	30.30%	5.00%	35.30%

Figure 1 – Results Table

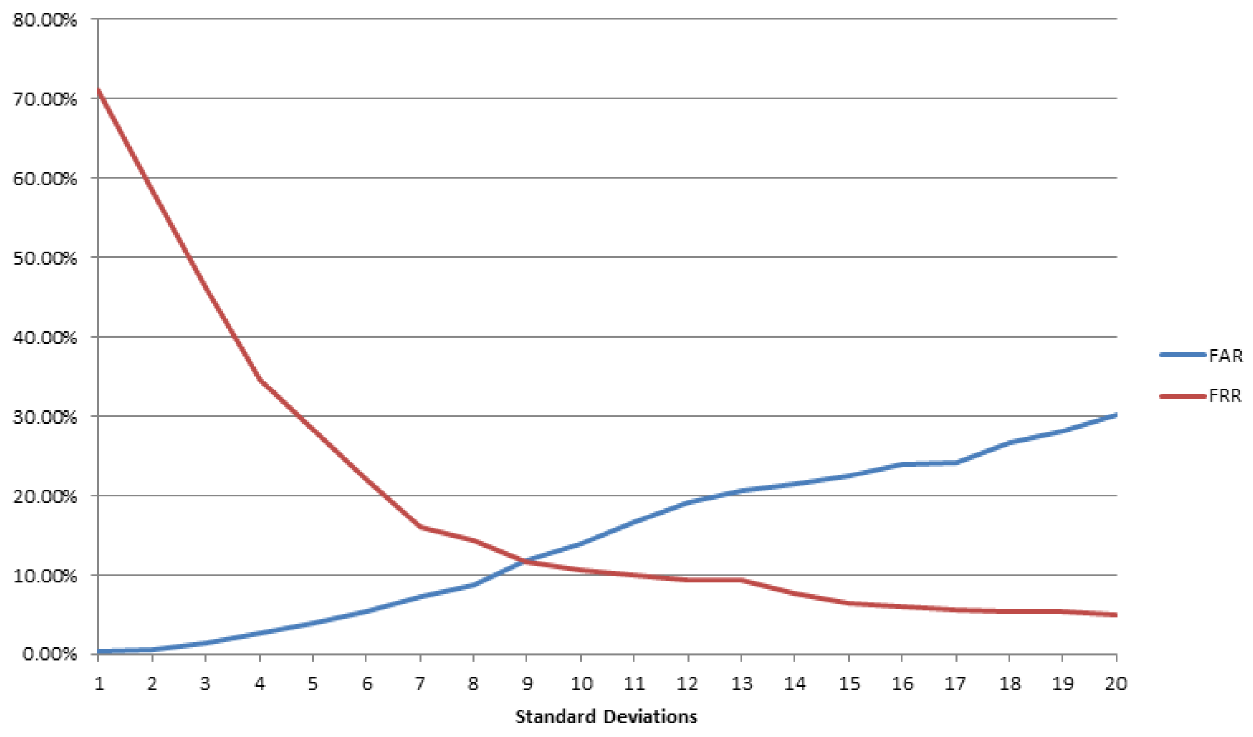


Figure 2 – Graph of Results

Conclusions

At the completion of this project, all of its (admittedly modest) goals and requirements have been met, and in this case the project was successful. But it terms of its impact on previous research and its implication on future research, the project fails to provide any new directions or information to achieve the long term goal of improving error rates within the field of handwritten signature verification. In a commercial setting, error rates similar to those achieved in this project would be unacceptable for almost any practical use.

The error rate achieved by this project is also not indicative of an error rate from use within a general population. The real conditions data may be far less consistent, especially if large amounts of time had elapsed between samples or the device used to capture samples differed (i.e. in screen size or accuracy). This outcome would lead to a higher error rate is real world use.

Experimentation was limited during the projects development. Perhaps the most important limiting factor to the final error rates produced was the poor choice of parametric features used – a small subset of most effective features was chosen from a previous list compiled by [Gupta 2006]; however the process of selection was not based on any sort of testing against a signature database. A larger list of possible features should have been chosen, with a process to determine the *most effective* features when tested against the signature database, while pruning any features of little value. A lack of signature data normalisation may have also affected the projects error rate performance. While the database used did have fairly consistent signature data, features such as those related to acceleration, velocity and line length may have been negatively affected by any signatures that were not consistent in size or shape. In addition to feature choice and source data normalisation, the project would have benefitted from a more scientific approach to choosing a distance measurement. Similarly to the procedure for the features, multiple possible candidates for distance measurement should have been used to determine the one most appropriate for this application.

In the future the limitations mentioned above must be addressed before one could expect to see an improvement in the systems performance that would lead to a useful contribution to previous research.

References

CRANE, H. and OSTREM, J. Automatic Signature Verification using a Three-axis Force-Sensitive Pen. IEEE Trans on Systems, Man, and Cybernetics, SMC-13, 3, (1983).

FIERREZ-AGUILAR, J., NANNI, L., LOPEZ-PENALBA, J., ORTEGA-GARCIA, J. and MALTONI, D. An On-line Signature Verification System Based on Fusion of Local and Global Information, in Proc. 5th IAPR Intl. Conf. on Audio- and Video-based Biometric Person Authentication, AVBPA, Springer LNCS-3546, New York, USA, (2005).

GUPTA, G. The State of the Art in On-line Handwritten Signature Verification. Faculty of Information Technology, Monash University, Building 75, Clayton, Victoria 3800, Australia (2006). www.csse.monash.edu.au/publications/2006/tr-2006-200-full.pdf

KETABDAR, H., RICHIARDI, J., and DRYGAJLO, A. Global Feature Selection for On-line Signature Verification, Proc. 12th Conf of the Int Graphonomics Society, Solerno, Italy, (2005).

LEE, L.L. Neural approaches for human signature verification, 3rd International Conference on Signal Processing, 2, (1996).

MAUCERI, A. Feasibility Studies of Personal Identification by Signature Verification. Report No. SID 65 24 RADC TR 65 33, Space and Information System Division, North American Aviation Co, Anaheim, California, (1965).

OSBORN, A. Questioned Documents. Boyd Printing Co, Albany, NY, 2nd Edition, (1929).

STERNBERG, J. Automated Signature Verification using Handwriting Pressure. 1975 WESCON Technical Papers, No 31/4, Los Angeles, (1975).