Andrew Ladlow

Gesture Interpreter

B.Sc. (Hons) Computer Science

14th March 2016

I certify that the material contained in this dissertation is my own work and does not contain unreferenced or unacknowledged material. I also warrant that the above statement applies to the implementation of the project and all associated documentation. Regarding the electronically submitted version of this submitted work, I consent to this being stored electronically and copied for assessment purposes, including the Department's use of plagiarism detection systems in order to check the integrity of assessed work. I agree to my dissertation being placed in the public domain, with my name explicitly included as the author of the work.

Date:

Signed:

All project files are available at:

http://www.lancaster.ac.uk/ug/ladlow/

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

The project's aim is to develop a gesture interpretation application, designed to recognise gestures used in the British Sign Language (BSL). The proposed application should accomplish this through the use of a motion tracking sensor device, the Leap Motion.

Human-computer interaction has, for many years, been limited to keyboards and mouse devices. With the more recent development of smartphones and tablets we have seen numerous technological advancements utilising touch-screen based devices instead.

Although these techniques are generally well implemented, it could be argued that they're inherently unnatural when compared with real world, physical interaction. The problem lies in the fact that all of these methods require the use of some form of hardware peripheral in order to facilitate human-computer communication.

The project's proposed application aims to utilise gesture recognition in order to provide a more natural form of communication. Gesture recognition involves the use of a middleware (software bridge or interface) which captures and processes images in order to convert them from raw frame data input into an output understood by a computer. In the case of the Leap Motion, this middleware receives input via an associated hardware controller. The distinction between this device and a typical keyboard or mouse is that the former will automatically capture a user's input, whereas the latter requires explicit user input i.e. pressing a key or moving a mouse.

The area of gesture recognition is particularly relevant for deaf or otherwise hearing-impaired users who are reliant on gesture based languages such as the BSL. The task of learning a sign language may seem particularly daunting due to its differences compared with typical spoken languages. Children are said to grasp new languages faster than adults - yet those who are born deaf to hearing parents are likely to be delayed in their exposure to a sign language, during a time period where faster exposure is critical, due to parents' lack of familiarity with them. A system designed to aid users in the understanding of a sign language would therefore be beneficial in cases such as this.

The low cost of the Leap Motion device makes it viable for purchase by individual users for self-study with the proposed application, or for possible involvement with existing sign language courses in order to aid in streamlining the overall learning process. Education courses which rely on a sign language tutor could be augmented through the use of the application, with the tutor storing accepted gestures in the application before distributing it to students. In this way, students would gain independence in their learning and the ability to learn at their own pace.

## 1.2 Project Aims

The aims of the project application as a whole are summarised as follows:

- Record sign language gestures performed by a user by storing image data from a Leap Motion device
- Recognise the gestures representing British Sign Language alphabet characters and distinguish between them
- Given an unknown gesture by a user as input, output an identified matching gesture with an associated similarity score

## 1.3 Report Overview

The structure of the report is as follows:

Chapter 1 discusses the problem domain and the proposed solution, including its aims.

Chapter 2 analyses relevant studies and related work, as well as any similar applications.

Chapter 3 identifies the user requirements for the application, and subsequently details appropriate architectural designs and an initial design for the application user interface.

Chapter 4 covers the implementation of the application, providing an overview of the application layout and structure.

Chapter 5 describes the operation of the application with the aid of screenshots.

Chapter 6 explains the application test procedure and analyses the results of these tests, followed by an overall evaluation of the system.

Chapter 7 concludes the report, summarising how well the application met its original aims, any identified limitations and scope for possible future work. The report ends with an overall conclusion the project.

# Chapter 2

# Background

## 2.1 Leap Motion

The Leap Motion is a palm sized USB device which tracks hands and fingers using optical sensors and infrared lights. The device was first released by Leap Motion, Inc. in July 2013[3]. According to Weichert, et al [20], the Leap Motion is capable of recognising movement with an accuracy of 0.7mm. Comparable devices in a similar price range, such as the Microsoft Kinect, fared much worse in these tests, achieving an average accuracy of 1.5cm. Considering the reasonable success of the Kinect, the Leap Motion provides further innovation potential in the domain of gesture recognition.

A major software update, firmware version 2, was released for the device in May 2014 [6]. The update claimed improved tracking performance and an enhanced API feature set through the inclusion of additional tracking capabilities for each individual bone in the hand, allowing for a level of tracking precision that was previously impossible with the device. The ability to track individual bones is a great benefit for sign language recognition purposes, and could be considered a necessity in order to track the difference between especially similar gestures.

An image of the device is shown in figure 2.1.

**Figure 2.1:** Leap Motion device

One of the earliest studies relating to the use of the Leap Motion controller in order to detect sign language gestures was undertaken by Potter, et al [15]. The authors noted that although the controller showed potential, further development of the API was required. This was mainly due to inaccurate hand detection in certain scenarios such as pinching fingers together, interlocking hands or holding one hand above the other – all of which created anomalous data due to the obscuration of finger position information. It's important to clarify that this study was undertaken using the device's initial release firmware (V1) as opposed to the aforementioned updated firmware (V2). As such, it's likely that the capabilities of the device will have since improved.

## 2.2 British Sign Language in Education

BSL courses such as those offered by the nationally accredited Signature [16] award a number of qualifications ranging from 'Level 1' to 'Level 6', with each level requiring an incrementing range of known vocabulary required in order to qualify. Each level's accompanying qualification is composed of a number of modules, where each of which focus on a particular subtopic e.g. 'BSL conversational skills' or 'Understand varied British Sign Language in a range of work and social situations'. All of these modules include guided contact time as well as additional work intended for private study. There is also an accredited online learning resource available for an additional cost.

The problem with this current system is that users new to the language are incredibly reliant on their tutors for guidance – online resources can help but only to a certain degree. Users may often need reassurance or additional guidance to ensure they are performing gestures correctly or to clarify gestures they're unfamiliar with. The instant feedback provided by the proposed application would be a great benefit in this regard. The inclusion of such an application could prove helpful for both students undertaking these BSL courses, as well as the centres offering them, as tutors would be able to distribute their time more effectively.

## 2.3 Gesture Recognition

Probably the most important aspect of the system is its gesture recognition – as we're dealing with data in real time, a suitable algorithm should calculate match results on the fly without causing delay or otherwise affecting the user when the application is running. Most approaches apply some form of machine learning on the Leap Motion frame data, or a specific subset of it. Generally it's difficult to say with certainty that one algorithm is better than any other due to the variance of testing conditions, input data, sign language variant, and so on. Some feasible examples are shown in subsections 2.3.1 through 2.3.5. Following this, a conclusion of my findings is covered in subsection 2.3.6.

### 2.3.1 Dynamic Time Warping

The use of dynamic time warping (DTW) was explored by Vikram, et al [18]. The appeal of DTW is that it isn't reliant on the time taken or speed of each input in order to accurately compare them – this is particularly useful with gestures which are often performed at varying speeds. The authors demonstrate how DTW could be applied to 2D handwriting gestures and suggested that it could be extended for use with 3D data, i.e. gestures. They conclude that the DTW approach used is suitable for real-time comparison but it remains to be seen whether that is still the case when dealing with the increased complexity of gestures compared with just handwriting.

### 2.3.2 K-Nearest Neighbour & Support Vector Machines

K-Nearest Neighbour (KNN) and Support Vector Machines (SVM) were proposed by Chuan, et al [2] as recognition algorithms for the American Sign Language using the Leap Motion. Tests were carried out using the 26 letters of the alphabet - results showed a recognition accuracy rate of 72.78% and 79.83% for the two methods, respectively. The authors mention

some possible reasons for the low accuracy with both algorithms; compared with the BSL alphabet, the ASL is signed using only one hand – as a result some letter representations are very close to one another which led to misclassifications of the Leap Motion data. The use of BSL with these methods could show improved results as all of its gestures require two hands to perform and are therefore more varied.

### 2.3.3 $P Point-Cloud Recognizer

The $P recognizer ($P), designed by Vatavu, et al [17], is another example of a gesture recognition algorithm. As described in the paper, the $P is "a 2-D gesture recognizer designed for rapid prototyping of gesture-based user interfaces". The $P aims to overcome the complex task of matching user gestures by instead treating them as groups or "clouds" of points and evaluating each one in turn. Even the simplest of gestures could be created in many different ways depending on the properties of its strokes e.g. start and end points, order or time, and direction. The use of a point cloud helps remove any ambiguity from the gesture which simplifies comparison and recognition. According to the paper, the algorithm requires only 70 lines of code to function and delivered over 99% accuracy in user-dependant testing.

### 2.3.4 Hidden Markov Models

Markov models, in particular Hidden Markov Models (HMM), are generally known for their use in pattern matching algorithms for speech recognition or typing prediction. A Markov model is a network of states with each state being connected to another with a specific weight or probability. In a Markov model based system the future state of the system is only dependant on its current state and the probability of the states it's linked to. A HMM differs in that its state is partially obscured. An example of this could be found in a speech recognition system where we are able to observe a waveform of speech but the actual spoken words is hidden. This can be compared to a gesture recognition system where we are given the movement data of a gesture but the actual intended gesture is hidden.

The use of a HMM was proposed by Chen [1] to support 2D and 3D motion recognition, achieving recognition rates of 91.9% in user-dependant testing and 96.9% in user- independent testing.

### 2.3.5 Artificial Neural Networks

The use of artificial neural networks (ANN) was previously proposed by Mohandes, et al [11], in particular a Multilayer Perceptron neural network (MLP) for use with the Arabic Sign Language (ArSL). The proposed system resulted in a classification accuracy of over 99%. An artificial neural network is a type of machine learning algorithm which bears similarity to the human brain in that it is composed of a series of simple processing units, neurons. These neurons are interconnected and each of these connections have a determined weight. The network learns from experience when provided with test data, calculating specific outputs for given inputs.

It's noted that the testing produced some erroneous results, similar to those found in the KNN/SVM with ASL testing described earlier. In this case this was due to fingers being

occluded by the palm of the hand or by other fingers during recognition, as opposed to gestures just being too similar to one another to discern between. The authors suggest the use of a second Leap Motion device positioned to the side of the user. Combined with the Leap Motion in front of the user this should theoretically resolve the observed issues, though further work has yet to be carried out.

### 2.3.5 Research Summary

The above research shows that a number of studies have been carried out relating to sign language recognition with the Leap Motion and similar devices, using a wide variety of gesture recognition algorithms, though with varying levels of success.

As the project's aims are similar to many of these studies, there's not much value in simply repeating them as we can already view the results for a number of sign languages. Similarly, creating an entirely new machine learning algorithm is beyond the scope of the project, more so given that there is already a wide range of publically available algorithms. Instead, it's worth considering identified approaches which have yet to be fully explored.

In particular, the $P recognizer (as discussed in subsection 2.3.3) has yet to be incorporated into a BSL recognition application – therefore it would be beneficial to record the capabilities and performance of this algorithm in greater detail. The $P is based on a Euclidean distance scoring system, similar to that used by Vikram, et al [18] with dynamic time warping, which mentioned possible future work in the 3D domain. The characteristics of the $P suggest it should be plausible to modify it in order to support 3D gestures, though it remains to be seen how effective it will be in practice.

## 2.4 Similar Applications

### 2.4.1 Leap Trainer

'Leap Trainer', created by O'Leary [14], is a browser based gesture learning and recognition framework for the Leap Motion. Developed in JavaScript, Leap Trainer allows users to create and store gestures then replay them at will. The software also allows gesture data to be exported for use in other applications. Leap Trainer implements gesture recognition through the use of neural network-based, cross-correlation, and geometric template matching algorithms.

Unfortunately, from initial testing it appears the software struggles to discern between intricate gestures such as those included in the BSL, frequently generating false positive results. This is likely due to a low level of accuracy in the comparison algorithm. This could be improved by comparing gestures more thoroughly, though it is unknown how severe an effect this would have on the application's performance.

### 2.4.2 UNI

A commercial application, UNI, is currently in development and is scheduled to be released in summer 2016 by MotionSavvy [12]. UNI bares similarities to the proposed application in that it utilises the Leap Motion in order to translate gestures into spoken text. An included 'crowd

sign' library would allow users to add and share gestures with other users via a cloud based dictionary system.

The software is closed source and based on a subscription model of $20 / month with an initial up front cost. This severely limits further adaptation or extension by like-minded developers, instead users are solely reliant on MotionSavvy to support the application in the future. Unfortunately, with no demo or other proof of concept available, it's unknown how well the UNI will achieve the goals described on the website.

### 2.4.3 Similar Applications Summary

Although a fair amount of research has been carried out in this area, the number of applications which are completed or significantly developed are scarce in comparison. The novelty of the Leap Motion is likely a leading factor in this, as developers are mostly unfamiliar with the device or are unwilling to commit resources to projects whilst the device is under continued development.

The Leap Trainer functioned well for gestures which were widely varied, though struggled with more complex gestures with less variance, which are more the case when dealing with sign languages. The UNI claims to be specifically designed for use with sign languages, but at this point has no tangible demo or examples of use.

The proposed application will therefore aim to address both of these shortcomings by providing a gesture recognition interface which will support the use of more intricate gestures (to the best of the Leap Motion's capabilities). Additionally, the application should be readily available for use without subscription models or other such limitations, in order to promote extensibility.

# Chapter 3

# Design

## 3.1 User Requirements

When considering the overall design of an application there are several sections which must be considered. The first of which is the clear definition of user requirements. In software development, a requirement is a "property that a system must contain or exhibit in order for it to satisfy a user". Before any implementation occurs it's crucial to ensure these requirements are clarified.

Requirements are grouped into two categories; functional and non-functional. The former describes the features of a system (what it does) whereas the latter describes how the system behaves (performance, reliability etc.). These requirements can be more easily identified via the creation of use case diagrams (shown in figure 3.1) and use case tables (shown in figures 3.2, 3.3, and 3.4). Diagrams aim to visualize the relationships between users of a system and possible use cases, as well as between use cases themselves. Use case tables specify the function of each use case, but don't consider their implementation.

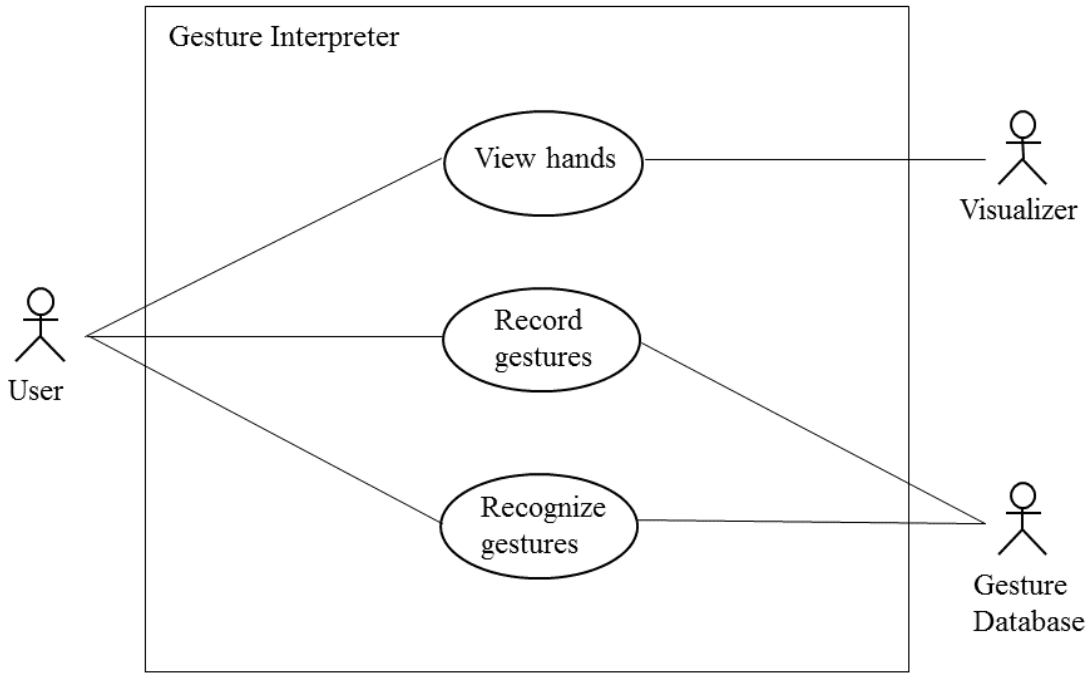



**Figure 3.1:** Use case diagram

| Use case name: | ***View Hands*** |
|---|---|
| Scope: | Gesture Interpreter |
| Primary actor: | *User* |
| Secondary actors: | *Visualizer* |
| Summary: | User wants to display their hands within the application |
| Preconditions: | Leap Motion device is connected and enabled |
| Main success scenario: | 1. User moves hand within the capture radius of the Leap Motion<br>2. Leap Motion recognizes new frame of image data<br>3. Visualizer is informed of the new image data<br>4. Visualizer adjusts hand representation positions according to this data<br>5. New hand positions are redrawn on the screen |
| Alternatives: | N/A |
| Exceptions: | 2a. User moves hands outside of the Leap Motion's capture radius. Hand data is removed from the application window. |
| Post conditions: | User observes graphical representations of hand positions |

**Figure 3.2:** Scenario for the 'View Hands' use case

| Use case name: | ***Record Gestures*** |
|---|---|
| Scope: | Gesture Interpreter |
| Primary actor: | *User* |
| Secondary actors: | *Gesture Database* |
| Summary: | User wants to store the representation of a gesture |
| Preconditions: | Leap Motion device is connected and enabled |
| Main success scenario: | 1. User moves hands within the Leap Motion's capture radius<br>2. Leap Motion recognizes new frame data<br>3. User moves hands above a velocity threshold<br>4. Gesture recording is initiated<br>5. User's hand velocity drops below threshold<br>6. Gesture recording is stopped<br>7. Gesture is saved to database |
| Alternatives: | 3a. User holds hands still for a period of time<br>5a. User moves hands above a velocity threshold (provided 3a was met) |
| Exceptions: | 6a. User moves hands outside of the Leap Motion's capture radius. Gesture recording is stopped and current data is discarded |
| Post conditions: | User successfully stores their gesture in the application |

**Figure 3.3:** Scenario for the 'Record Gestures' use case

| | |
|---|---|
| Use case name: | ***Recognize Gestures*** |
| Scope: | Gesture Interpreter |
| Primary actor: | *User* |
| Secondary actors: | *Gesture Database* |
| Summary: | User wants to perform a gesture and observe the system's interpretation |
| Preconditions: | Leap Motion is connected and enabled |
| Main success scenario: | 1. User moves hands within Leap Motion's capture radius<br>2. Leap Motion sees new valid frame data<br>3. User moves hands above a velocity threshold<br>4. Gesture recognition is initiated<br>5. User's hand velocity drops below threshold<br>6. Gesture recognition is stopped<br>7. Provided gesture is compared with known gestures in database<br>8. Closest gesture match is shown |
| Alternatives: | 3a. User holds hands still for a period of time<br>5a. User moves hands above a velocity threshold (provided 3a was met) |
| Exceptions: | 6a. User moves hands outside of the Leap Motion's capture radius. Gesture recognition is stopped and current data is discarded |
| Post conditions: | User observes the system's interpretation of their gesture with an associated similarity score |

**Figure 3.4:** Scenario for the 'Recognize Gestures' use case

From the use case analysis above, a number of functional and non-functional user requirements were established, as shown in tables 3.1 and 3.2, respectively.

**Table 3.1:** Functional user requirements

| **Requirement ID** | **Requirement Specification** |
|---|---|
| R1 | The system shall display a real time interpretation of the user's hands during operation |
| R2 | A user shall be able to record and store their own data for a given gesture |
| R3 | The system shall recognize a gesture provided by a user |
| R4 | The system shall present user feedback, a normalized score, based on the similarity between a given gesture and stored gestures |

**Table 3.2:** Non-functional user requirements

| **Requirement ID** | **Requirement Specification** |
|---|---|
| R5 | The system's real time display of a user's hands shall be updated with a latency of 5ms or less |
| R6 | The recognition of gestures shall take no longer than 200ms to complete |

| | |
|---|---|
| R7 | The system shall recognize gestures with an accuracy of at least 80% |
| R8 | The system shall implement a simplistic user interface which doesn't rely on mouse or keyboard input and can be understood by a user within 5 minutes of use. |
| R9 | The system shall have a reliability of 100%. (Should never crash or otherwise exhibit failure) |

Regarding R6, the time value was chosen based on a statistical analysis of the average human reaction time from the human benchmark website [5]. The site currently holds records for 26,404,960 reaction time tests, with an average reaction time of 271ms. If we take into consideration the additional overhead of other factors such as the Java VM and the screen refresh rate then the time taken should be lower to compensate. Therefore it should be reasonable to assume that users won't notice a delay in the recognition of gestures provided that it takes less than 200ms to complete.

## 3.2 User Interface Design

As the application will contain a user interface, it's beneficial to draft initial design plans to get a general idea of what the application's appearance will be and how a user should be able to interact with it. Looking back at the user requirements, R8 states that the system should be easy to use without relying on a keyboard and mouse – in other words, the user should be able to fully interact with the application through only the Leap Motion.

The Leap Motion documentation [8] provides a number of guidelines which describe how an application interface should be designed in order for it to be suitable for Leap Motion interaction. The page suggests that buttons on an application should be "large, well organized and include a clear highlight/depressed state". With this in mind, an initial menu design is shown in figure 3.5.

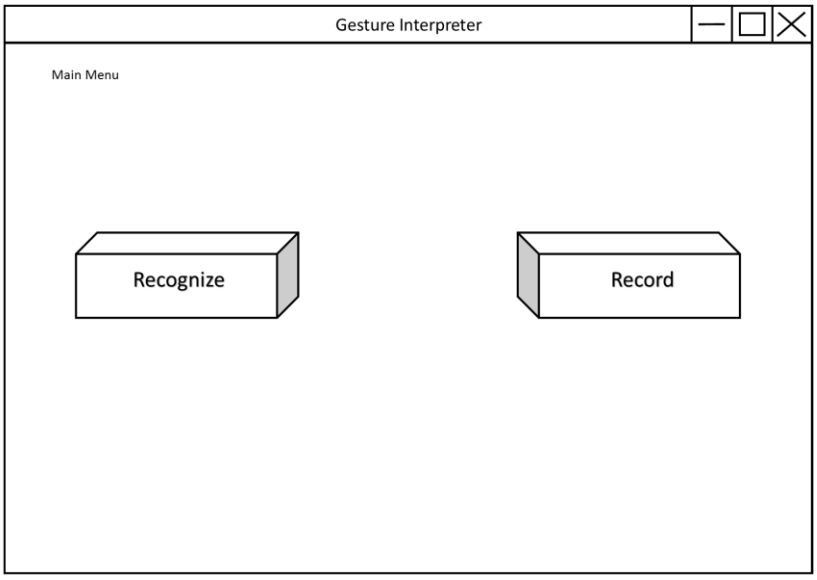



**Figure 3.5:** Initial user interface design for application menu

The primary goal of the interface is simplicity. With a mouse it makes sense to have multiple smaller buttons or sections in an application menu. With the Leap Motion's much lower level of precision this approach isn't practical. Interactable buttons should be deliberately enlarged to ensure they're easy to press on the first try.

Additionally, buttons should provide some level of feedback when pressed, for example, by changing colours or size. For the buttons in this application, as a user moves their finger a button closer to the screen (whilst hovering over a button), the depth of the button should decrease as if it were actually being pressed. This aids in informing the user when their actions have an effect on the application's state where it might otherwise be unclear.

Other aspects of the proposed application should follow this simplicity based design. The application has a clear purpose of recognizing sign language gestures, it should therefore not include any unnecessary features or additions which may confuse the user. Features can easily be added at a later date if required - removing features is likely to be more difficult, however.

## 3.3 System Architecture

Considering the user requirements specified in the previous section, the application will consist of three main subsystems:

- **Leap Motion subsystem**: Updates application with new hand position data as the physical Leap Motion device captures it
- **Visualization subsystem**: Displays a visual interpretation of the current Leap Motion data, providing the user with feedback of their actions
- **Gesture subsystem**: Handles storage and recognition of performed gestures

A high level overview of each of these subsystems is shown in figure 3.6. The diagram highlights only which subsystems should be present within the application and how they should be associated with one another, rather than how they should be implemented – at this stage in the design process this information isn't relevant.

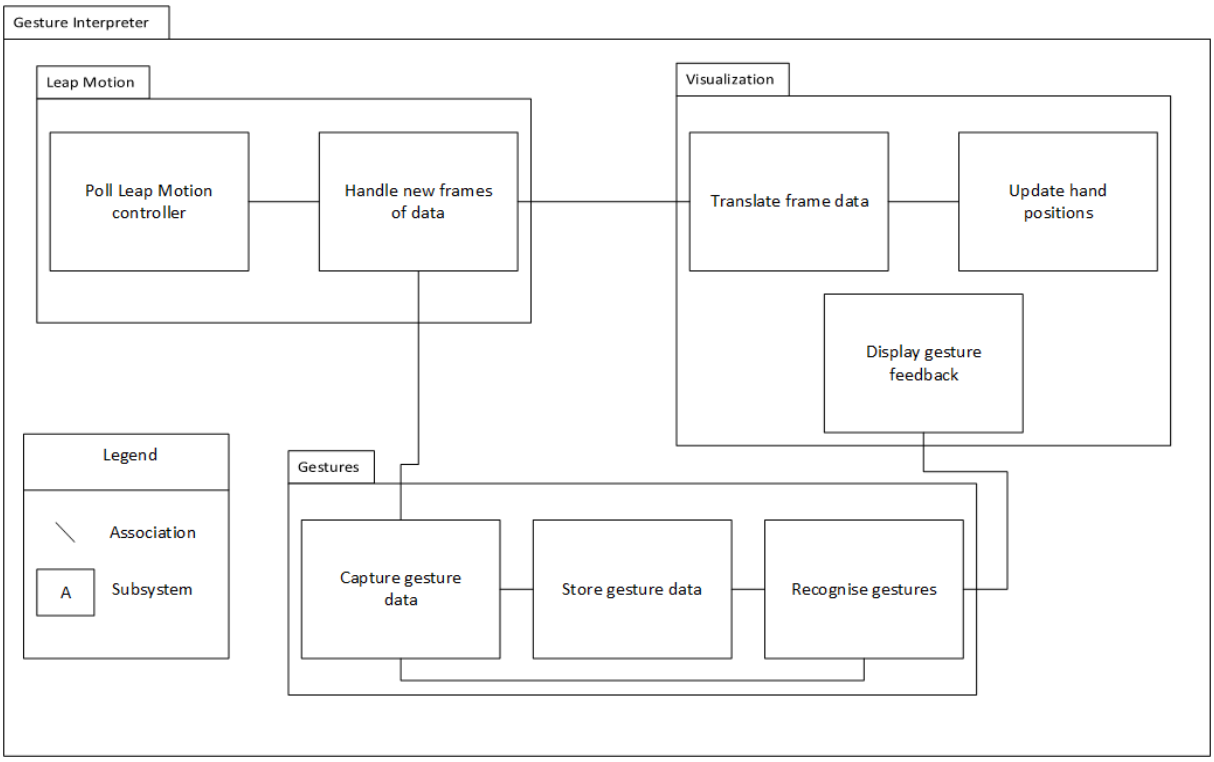



**Figure 3.6:** Overview of subsystems within the application

Following on from this, figures 3.7, 3.8, and 3.9 show sequence diagrams representing the series of events which take place in these subsystems. These aim to demonstrate how sections of the application interact with one another through the exchange of messages over a period of time. Three diagrams provide an overview of the process of displaying a user's hand on screen, recording a gesture, and recognising a gesture, respectively.

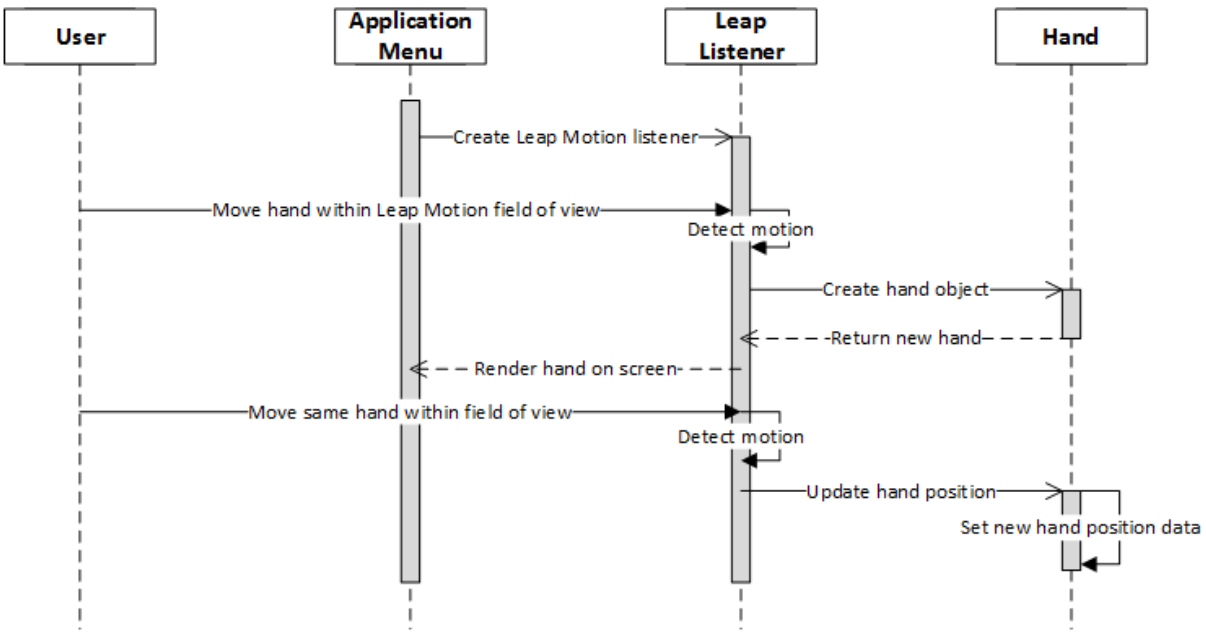



**Figure 3.7:** Sequence diagram representing the display of a user's hand

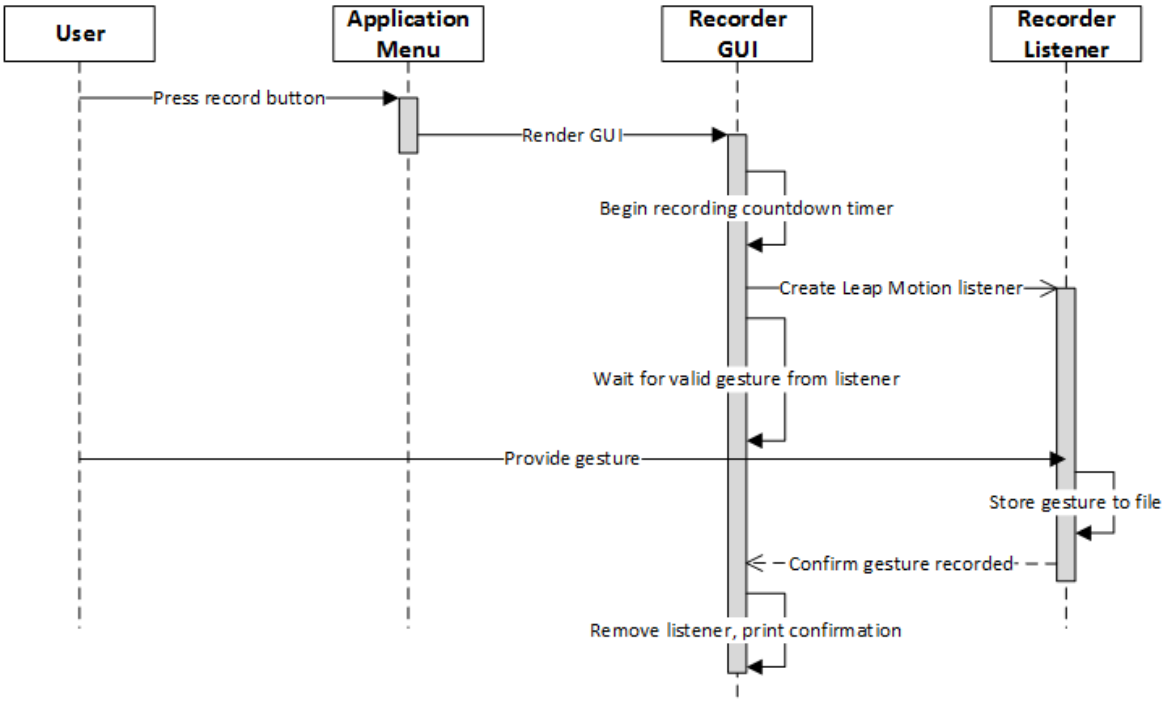



**Figure 3.8:** Sequence diagram representing the recording of a gesture

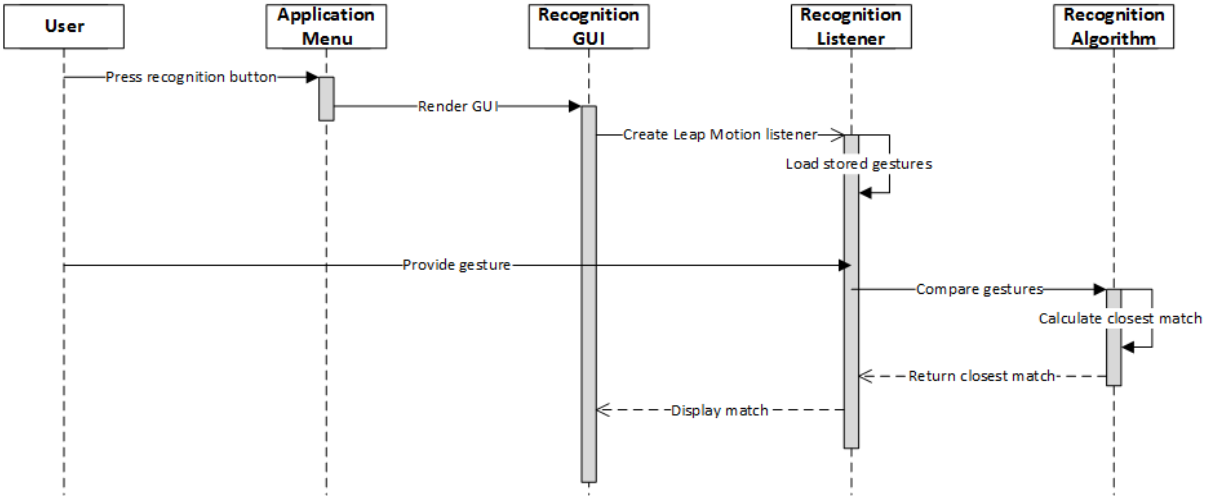



**Figure 3.9:** Sequence diagram representing the recognition of a gesture

## 3.4 Platform Selection

Although the Leap Motion supports a number of languages through its API, the application is intended to be developed in Java. Java was chosen primarily due to personal familiarity with the syntax. As there are many new or unknown technologies which will be incorporated into the application, using a familiar development language is an important first step towards development.

Due to the application requirements, some form of graphical user interface (GUI) must be included in order to represent hand motions captured by the Leap Motion. In the past, Java applications have been known to use the Abstract Window Toolkit (AWT) or Swing in order to render 2D and 3D graphical components.

More recently, JavaFX has been seen as a more appropriate choice due to its increased support from Oracle. Additionally, compared with AWT and Swing, JavaFX features more consistency as well as flexibility through the inclusion of improved event handling and animation capabilities, both of which are required for some aspects of the application. With no substantial prior knowledge of any Java based GUI libraries, I thought it to be most practical to begin development using the most modern of the identified three.

# Chapter 4

# Implementation

## 4.1 Leap Motion Integration

The Leap Motion controller records tracking data in a series of frames. Each frame contains the positions of any detected hands or other pointable objects. Frames can be acquired by simply polling the device or via a call back method from an event listener which is assigned to the device. In the latter case, the Leap Motion will create a new thread for each new frame, and will pause execution until the current thread's call back method has returned. This prevents an occurrence of thread-flooding, where threads are created at a faster rate than the device is able to process them.

Comparing the two integration choices, I found the second to be the most intuitive. The use of the event listener allowed me to manipulate or store the current frame's data without having to consider the poll rate, which could in some cases be causing frames to be skipped, or duplicate frames to be requested, depending on the rate of frame polling compared with the rate of frame returns by the Leap Motion controller.

The Leap Motion is added to a Java application through the use of controller and listener classes. The controller class is a representation of the device itself, and must be created before any data can be retrieved from the Leap Motion. At this stage, the controller can be polled in order to retrieve frames of tracking data but this is only appropriate for the first integration choice as described above. The second choice requires the creation of a listener which is associated with this controller.

The listener class defines a number of call back methods which are executed depending on the status of the Leap Motion controller. For example, the method onConnect() will be executed when the controller object connects to the Leap Motion software and the Leap Motion hardware device is plugged in. The method we're particularly interested in, however, is onFrame(). onFrame() is executed when a new frame of hand and finger data is available. This data can then be stored or transferred as appropriate within a system. For the purposes of this application, data is handled through the use of Platform.runLater() calls, which are described in the following sub-section.

## 4.2 Graphical User Interface

The application's GUI is handled by JavaFX, a library of graphics and media packages which is written as a Java API, meaning it can be referenced from any standard Java based program. The properties of JavaFX allow external devices such as the Leap Motion to be easily incorporated within an application. An adapted schema from Vos [19] describes this combination, as shown in figure 4.1.

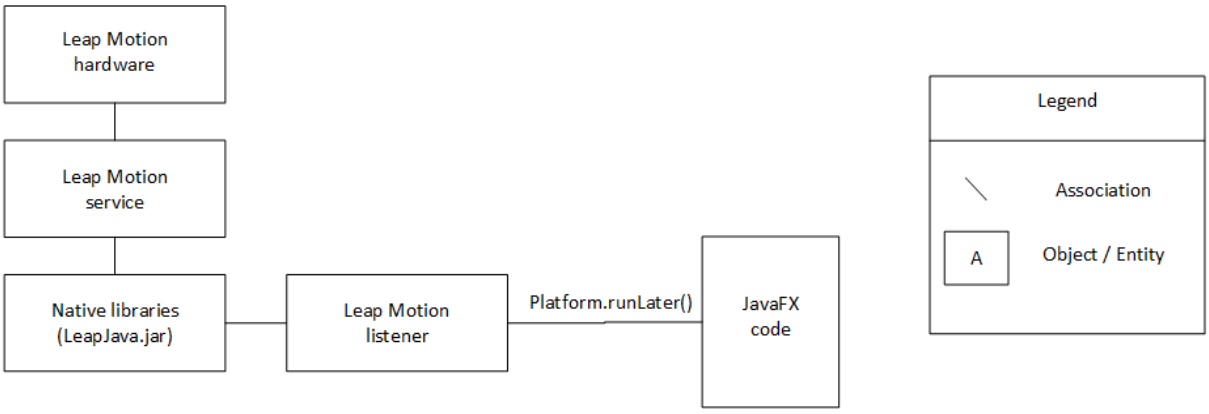



**Figure 4.1:** Interaction between Leap Motion and JavaFX

From the above schema we can see a number of key aspects detailing the relationship between JavaFX and the Leap Motion API. The physical Leap device is recognised by the operating system through a Leap Motion service. This service is subsequently linked to an application via the LeapJava.jar native library. This library allows use of the Leap's API within Java code. A listener class is extended using this API, containing one or more Platform.runLater() calls, where each of which contains specific JavaFX code used to modify the properties of objects rendered in the application window.

As described in Oracle's documentation of JavaFX [13], a system using JavaFX will run two or more of the following threads at any given time:

- **JavaFX application thread**: Primary thread used by JavaFX. Essentially any content which can be seen by the user must be managed in this thread.
- **Prism render thread**: Allows the application to perform concurrent processing. While one frame is being rendered, the next can be pre-processed to help off-load the work required.
- **Media thread**: Runs in the background and synchronizes the latest frames using the application thread.

With JavaFX, any property that modifies a window's live content can only be changed through a Platform.runLater() call – this ensures that these modifications only occur on the application thread. The combination of this system and the Leap Motion listener lead to an approach where each call of onFrame() on a listener includes one or more Platform.runLater() calls in order to modify the content shown in the JavaFX window, either directly or indirectly. This ensures the application runs safely in regards to multithreading so as not to attempt to modify values in incorrect threads.

## 4.3 Hand Visualization

The inclusion of a visualization system to represent a user's hands when they're interacting with the application was a crucial initial design consideration. The user should be able to see their actions in real time, without having to swap between the Leap and the keyboard/mouse to e.g. navigate through menus – or having no visual feedback from the Leap entirely.

The visualization is integrated through the use of a dedicated listener class (as described in section 4.1) with a number of Platform.runLater() calls (as described in section 4.2). When each new frame is received by the Leap controller, this listener's onFrame() method checks the content of the frame to find the number of hands (if any) within it. If there is at least one hand visible in the frame, the listener creates an instance of a HandFX object and adds it to the application scene in order to be displayed, all wrapped within a Platform.runLater() call – the pseudocode of which is shown in figure 4.2.

**onFrame(Controller controller)**

```
1.  Frame frame = controller.frame();
2.  Platform.runLater(() -> {            // code inside this block is ran on JavaFX main thread
3.      if (frame contains hands) {
4.          for each (leapHand in frame) {
5.              handFX = new HandFX();    // initialize a new handFX object
6.              handGroup.add(handFX);    // add to application scene to display
7.              handFX.update(leapHand);  // update the handFX object with leap co-ordinates
8.          }
9.  });
```

**Figure 4.2:** Pseudocode of interaction between Leap Motion listener and application display

A HandFX object instance is a 3D representation of a user's hand, generated by taking the raw data from the Leap Motion and converting it into co-ordinates. These co-ordinates are subsequently used to update the positions of groups of sphere and cylinder shapes, representing a hand's joints and bones respectively. This solution provides a 1 to 1 mapping of the Leap Motion data to the application, meaning the user can be certain that what they see on the screen is a true representation of what the Leap Motion 'sees' at any given time.

## 4.4 Gesture Recognition

The application builds on the work of Vatavu, et al. [17], using an adapted version of the $P recognizer (discussed briefly in section 2.3.5). The algorithm is based on the idea of machine learning, and was originally written in JavaScript and C# for use with handwriting recognition. The code used has been converted to the Java syntax and modified to support points in three dimensions in order to function correctly with the Leap Motion.

At its highest level, the $P is an "instance-based nearest neighbour classifier with a Euclidean scoring function". Breaking this down, the $P selects the most appropriate category for an object, given a selection of objects, by calculating the Euclidean difference in positions between all available objects in order to find the closest and hence the most likely match. The

$P is instance-based, meaning it compares a given unknown object against a set of known objects which are stored in memory.

In machine learning, objects are grouped into specific ‘categories’ based on a number of their ‘features’. Features refer to the properties of the object which make it unique, compared with other objects. In the context of the application, the features shown in table 4.1 are used to distinguish gestures from one another.

**Table 4.1:** Hand features used for gesture recognition.

| | **Palm** | **Fingers** |
|---|---|---|
| **Features used** | Stabilized palm position | Finger direction |
| | Palm normal | Metacarpal start position |
| | Palm direction | Metacarpal end position |
| | | Proximal end position |
| | | Intermediate end position |
| | | Distal end position |

The meaning of each of these features is as follows:

- Stabilized palm position: The distance between the centre point of a hand’s palm and the Leap Motion controller origin, in millimetres. Smoothing and stabilization is applied to this value.
- Palm normal: A vector pointing in the same direction as the palm’s normal (orthogonal to the palm). For example, if the hand is held flat the normal would point downwards.
- Palm direction: A vector pointing from the palm position towards the fingers.
- Finger direction: The direction in which the finger is pointing.
- Finger start position: The base of the bone, closest to the wrist.
- Finger end position: The end of the bone, closest to the fingertip.

Each position feature is highlighted in figure 4.3, a hand bone model from the Leap Motion API page [7].

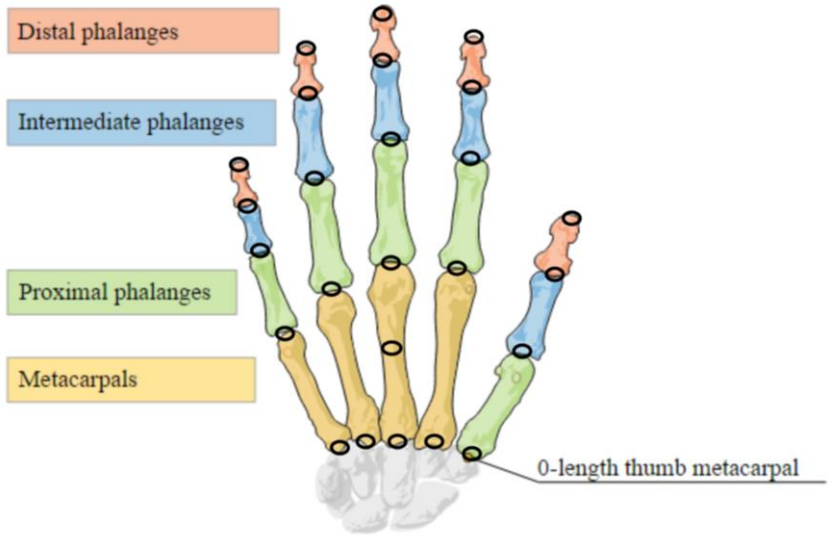



**Figure 4.3:** Bone model with highlighted position features

Gestures are recognized by the application implicitly based on a certain criteria. I had considered to fire recognition events explicitly via input from the user (e.g. pressing the spacebar to signal the start and stop points of a gesture) but felt that this would become cumbersome to operate, more so in cases where the keyboard might be difficult to access or not available entirely.

The criteria mentioned above is the velocity of each hand within the frame. When each new frame of image data is received by the Leap Motion listener, the velocity of all detected hands are checked – this is found by checking the velocity of the palm of the hand, as well as the maximum velocity of each of the fingers on the hand. If this velocity is above a threshold (300 millimetres per second was chosen as the default value) then the listener will recognize that a gesture is being performed. If this threshold is not met, the listener will instead recognize that a 'pose' is being performed.

Although the BSL is usually referred to as a set of hand gestures, this isn't necessarily the case in practice. Figure 4.4 shows an example of the alphabet interpreted in BSL where this can be seen more clearly. From the 26 letters of the English alphabet, only two ('H' and 'J') involve any hand motion. The majority of letters involve the hands remaining stationary in a specific pose, hence the split of pose and gesture categories. If a given gesture includes any degree of hand motion then it's a waste of time to compare against any stored gestures which don't, and vice versa.

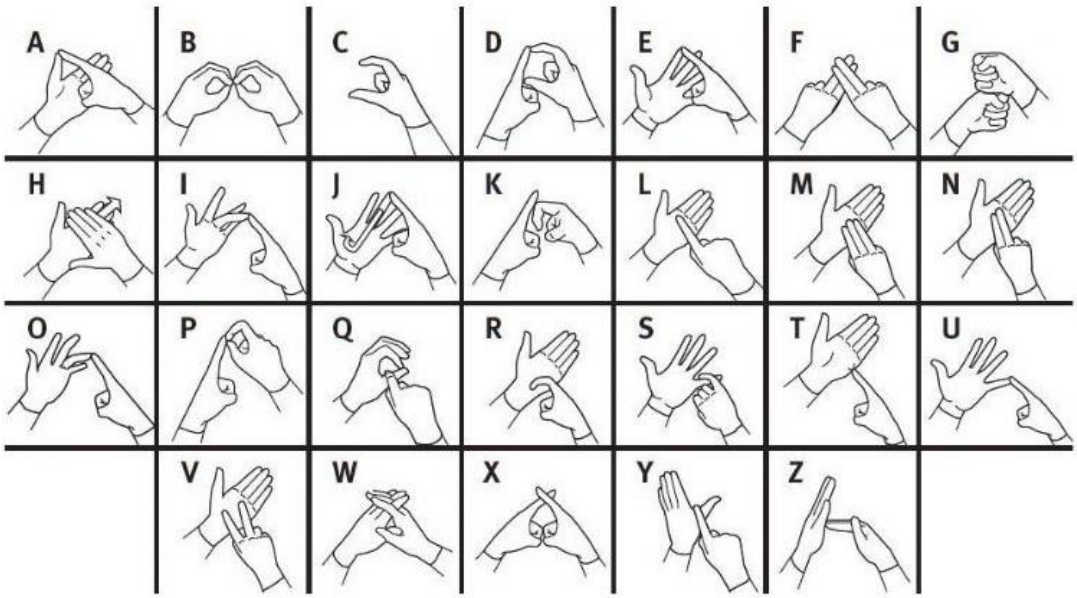



**Figure 4.4:** BSL interpretation of English alphabet

If a pose is detected, the listener will wait for 50 frames. If the hand's velocity remains below the motion velocity threshold for the entire 50 frames, a single frame will be captured to represent this pose. This frame is stored as a Gesture object in-memory, containing each of the features described in Table 4.1.

The process is essentially identical when considering a gesture which contains motion, such as for the 'H' or 'J' letters – for each frame that the hand's velocity is above the threshold, a frame

is recorded. If at least 10 frames are recorded in succession the gesture is considered to be valid. In the case of motion gestures, recognition will trigger once the hand's velocity eventually falls below the threshold. In theory this allows for gestures which continue for an indefinite length of time, though in the case of the BSL they typically last a second or two at most.

## 4.5 Gesture Comparison

### 4.5.1 Adaptation of the P-Dollar Recognizer

As previously mentioned in subsection 2.3.3, The P-Dollar Recognizer ($P) is an algorithm originally designed to facilitate the recognition of handwriting (2D) gestures. Whilst conducting background research in the area of gesture recognition, my thoughts were that the $P would be reasonably straightforward to adapt for use with 3D gestures.

The standard $P algorithm treats a given 2D gesture as a series of evenly spaced points on a flat plane (the authors use the term 'point-cloud', hence the algorithm name). Each point is assigned a stroke ID, assigned depending on the current pen stroke – i.e. the first pen stroke would generate a stroke ID of 1 for all points it created, and so on.

In order to support 3D gestures, the algorithm was adapted to include the z-axis in any calculations which previously included both the x and y axis. The problem with this approach, however, is that we're no longer dealing with points from single pen strokes as with a handwriting gesture – instead we have points from multiple hands, fingers, bones etc., we therefore can't apply this idea of stroke IDs. This issue was addressed by assigning the same stroke ID to all generated points – not the optimal solution but given that the algorithm wasn't designed for use with 3D data, these kinds of problems were to be expected. Assigning a single stroke ID to all points in a gesture had no discernible effect on the recognition outcome.

From the testing conducted throughout the development of the application, I found that simply recording the positions of finger tips, as we would record the position of the pen tip in a 2D gesture, would result in highly inaccurate gesture comparisons. With a 3D gesture we must consider a number of other factors, rather than just these fingertip points. This was addressed by storing additional features for each gesture, as previously shown in table 4.1.

There is the potential to store further feature data for a given gesture in order to improve the accuracy of gesture recognition. Unfortunately we're essentially stuck in this regard as we're reliant on the Leap Motion's API, which is currently lacking in terms of available data which can be extracted from a given image frame.

### 4.5.2 Normalization

Before a given gesture can be compared against a saved gesture, they must both be normalized. This helps to ensure that fair and accurate comparisons are made between all gestures when determining a match. Gestures undergo a series of three normalization steps before they are compared. To increase comparison performance, all stored gestures are normalized between being loaded from file and stored in memory, whereas new or unknown gestures are normalized lazily (immediately prior to being passed to the comparison algorithm).

When dealing with gestures, it's likely that they won't always be performed in the same position. The feature values composing each gesture are therefore translated to an origin of (0, 0, 0). This is otherwise known as translating to a known origin.

Additionally, the actual size of the users hands must be considered – gestures performed with varying hand sizes may not contain the same feature values. Points are therefore rescaled in the range [0-1].

Finally, each gesture is resampled to a specific number of points. This is helpful when dealing with gestures that contain varying amounts of motion, as it wouldn't be accurate to compare one gesture that lasts 5 seconds against another that lasts 10 seconds, for example. Gestures that are comprised of a single frame of hand data are also resampled to the same number of points but remain essentially unchanged as far as comparison is concerned.

A single frame of a gesture using two hands contains 3 features for each palm, and 6 features for each finger, totalling 66 feature points. Considering the rate at which frames are captured by the Leap Motion, the number of feature points can quickly increase into the 1000s for gestures lasting only a few seconds. This is therefore an additional performance measure to improve the time taken to compare these types of gestures.

### 4.5.3 Machine Learning

Gestures are evaluated via the sum of the Euclidean distance between each of their feature points. For a given point in a given unknown gesture, the distance is calculated to all other points in a given stored, known gesture until the closest point is found. This process repeats until all point distances have been found and recorded. This is then repeated for all stored gestures - and the gesture with the smallest overall distance is returned as the closest match to the unknown gesture.

Machine learning is relevant here as it relates to the idea of an algorithm becoming 'better', or more accurate, over time, provided it is continually supplied with new data. Any data that is provided to the algorithm (when not actively used) is known as training data. Each set of training data will contain a number of feature values comprising an object, as well as a specific label which categorizes the object. The algorithm should then be able to categorize unknown objects based on its knowledge of this training data. This process is otherwise known as supervised learning.

For example, supposing a gesture for the letter 'A' is stored with feature point values X, Y, and Z. If we then wanted to recognize this letter we would need to provide a gesture containing these same feature values, or values which were closer to these than to those of any other gesture.

If we then added an extra recording of the letter 'A' with feature values X+1, Y+1, and Z+1, there would be a wider degree of tolerance for any provided gesture to match it. We might subsequently perform the gesture for 'A' slightly differently than our first recording, but closer to our second recording (e.g. X+2, Y+2, Z+2) – in that case a match could still be generated, which might not otherwise have been the case.

An algorithm which can learn based on input is therefore crucial to the success of applications which rely on this kind of pattern matching or differentiation. As the number of possible

categories increases, having more samples of each category should, in theory, greatly improve the accuracy of recognition.

## 4.6 Application UML Diagram

Figure 4.5 shows a more technical overview of the complete application. This helps to clarify the relationships between classes, as well as the data and features each class contains.

The diagram is split around the ‘Menu’ class which handles the rendering of the application’s main menu and initializes the Leap Motion for use. This can therefore be considered the main class of the application, after first being initialized by the ‘Driver’ class.

The left side of the diagram relates to the visualization of hands within the application – the primary class being ‘HandFX’ which controls the positions of detected hands and loads the shapes representing the hand via the ‘ShapeCreator’ helper class, receiving updates from an associated ‘LeapListener’ instance. An additional ‘LeapButton’ class handles the creation of GUI buttons which are designed to be interacted with using only the Leap Motion, satisfying a criteria of the application’s design specification.

The right side of the diagram focuses on the recording and recognition of gestures, with each containing an associated GUI and Leap Motion listener class. All gestures are categorized as an instance of the ‘Gesture’ class, with each gesture being composed of an array of ‘Point’ objects. Each point represents a feature of the gesture, as described in table 4.1. The recognizer calls the $P in order to generate gesture matches, returning the result in a ‘Recognizer Results’ object which is forwarded to the GUI in order to be displayed.

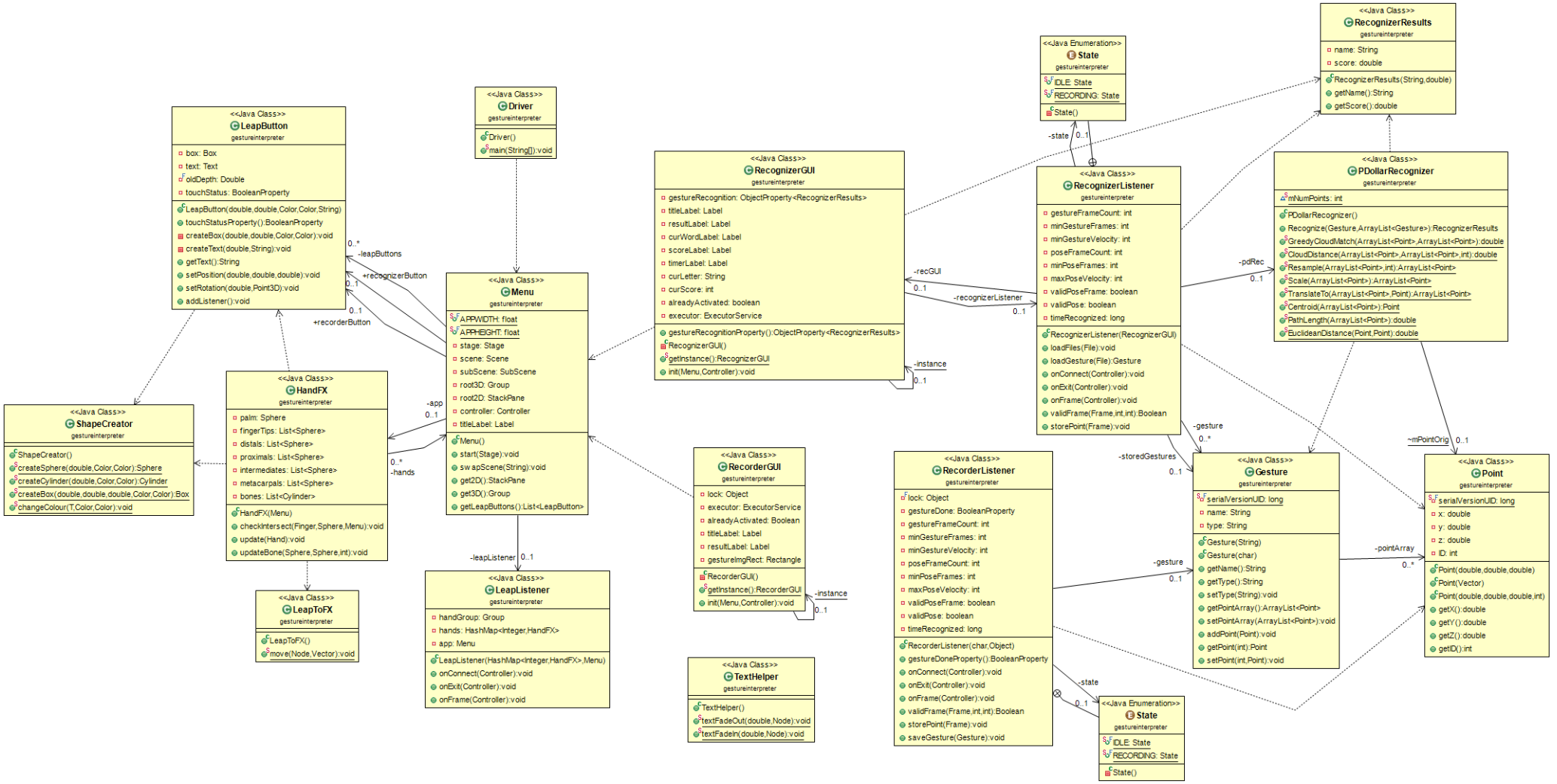



**Figure 4.5:** Application UML class diagram

# Chapter 5

# The System in Operation

## 5.1 Initial Application State

Upon starting the application, the window displays the content shown in figure 5.1. This screen acts as a main menu of the application, allowing access to the recognition and calibration screens. The user is able to access either of these screens by making a tap motion with their hand in the Leap Motion controller's field of view, as if they were actually pressing the button. As discussed in chapter 3, I had aimed to design the user interface to be over-simplistic in order to promote ease of use with the Leap Motion.

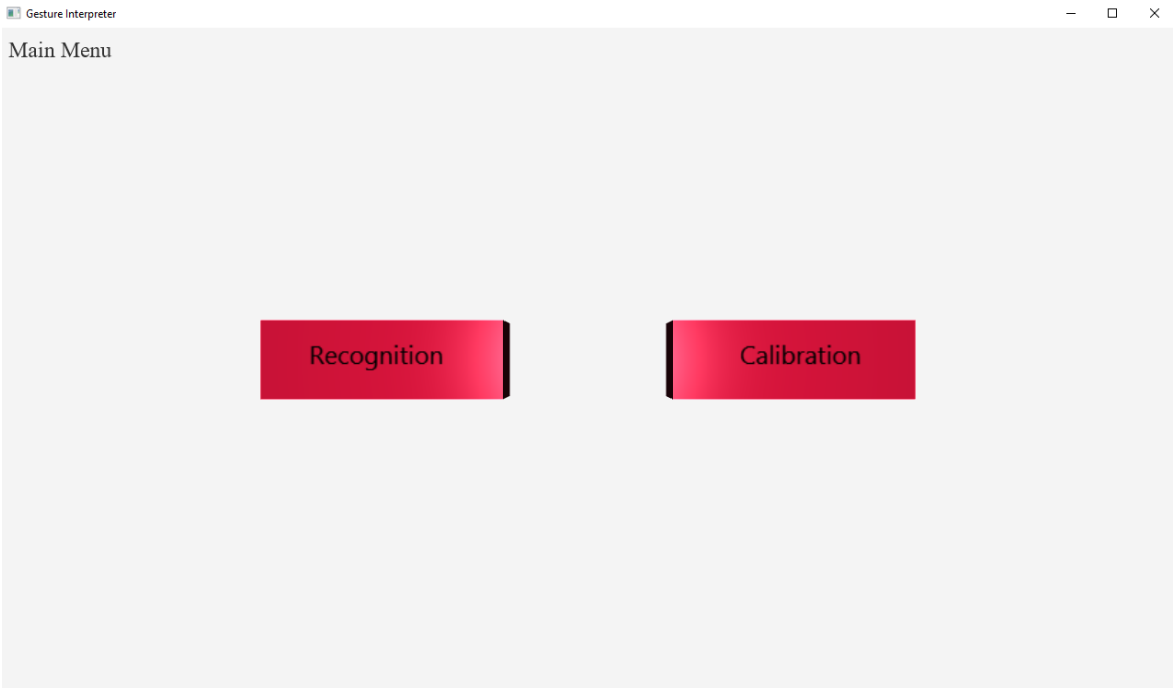



**Figure 5.1:** Application when it is first opened, with no user input

## 5.2 Hand Enters Field Of View

When a hand enters the Leap Motion's field of view, a skeletal model is shown in the application window, as seen in figure 5.2. This model mimics the motion of the user's hand in a 1 to 1 fashion. A side effect of this is that any blemishes in the Leap Motion's accuracy are also represented in the application. For example, if a hand is unable to be correctly identified by the Leap Motion it will appear distorted in the application. This is beneficial as it highlights cases where the Leap Motion is at fault, rather than the user.

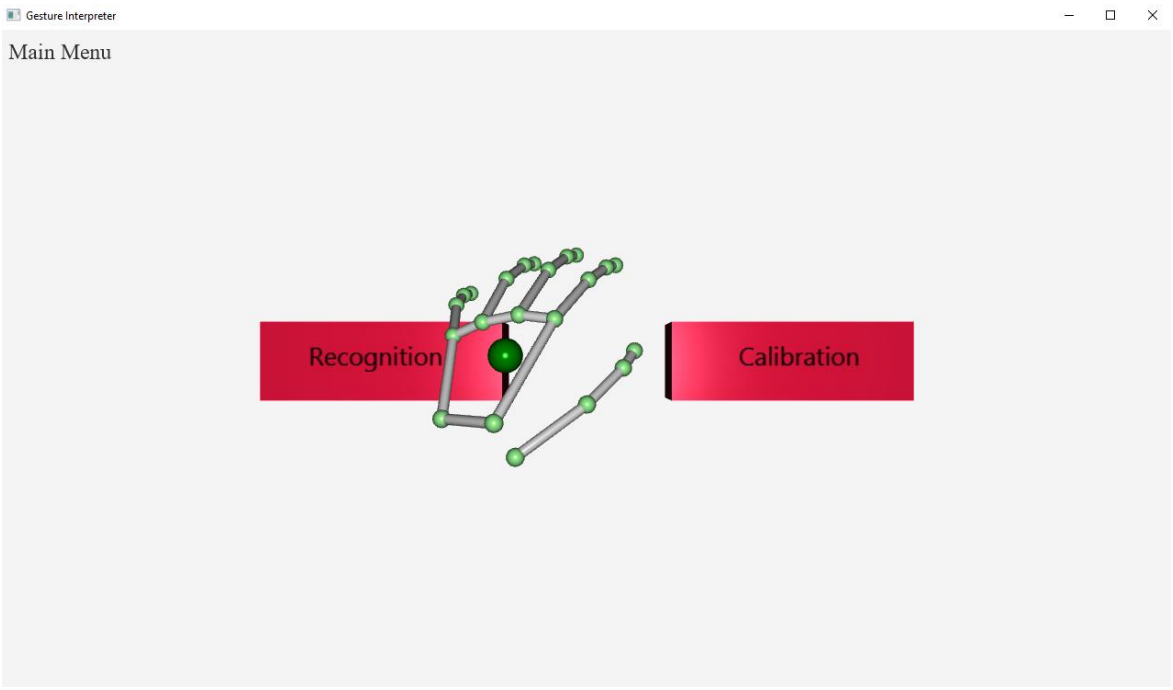



**Figure 5.2:** Application state when a hand enters the field of view of the Leap Motion.

## 5.3 Recognition

When the user touches the left-most ‘recognition’ button, the screen shown in figure 5.3 will be displayed. The user is asked to perform a given gesture from the set of all English alphabet letters. When a gesture is performed, the closest match and associated accuracy score is displayed to the user.

For each successful gesture match, the score value shown in the top-right is increased by 10. The goal is to correctly match the most amount of gestures within the time limit provided. As shown in figure 5.3, the user was asked to perform a ‘C’ gesture and upon recognition the application displays a similarity rating of 66%.

The objective of this section is to provide an example as to how gesture recognition could be integrated into an application. There are of course a variety of uses which could be expanded upon but this provides a useful starting point and proof of concept.

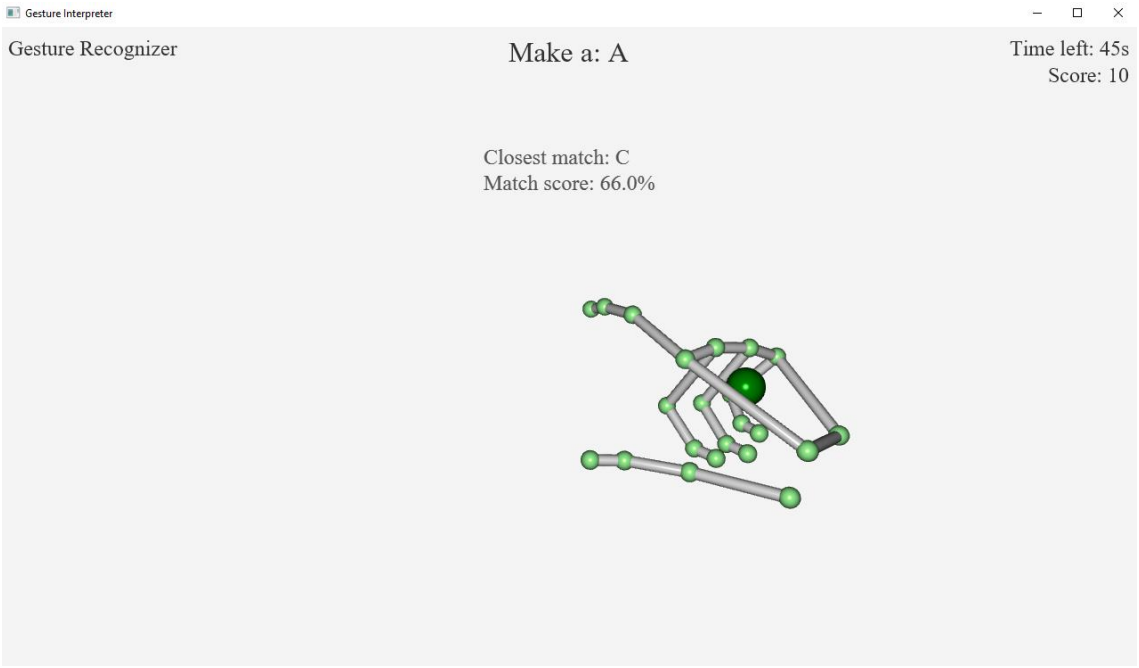



**Figure 5.3:** Application state when the ‘recognition’ button is pressed

## 5.4 Final Score

When the timer expires, the user is shown their final score, as seen in figure 5.4. This screen remains for 5 seconds before returning the user to the main menu screen.

This score is derived from the number of gestures which were successfully matched by the user within the time limit, with each correct gesture match granting 10 points. In the figure below, the user correctly matched 3 gestures and was therefore presented with a final score of 30.

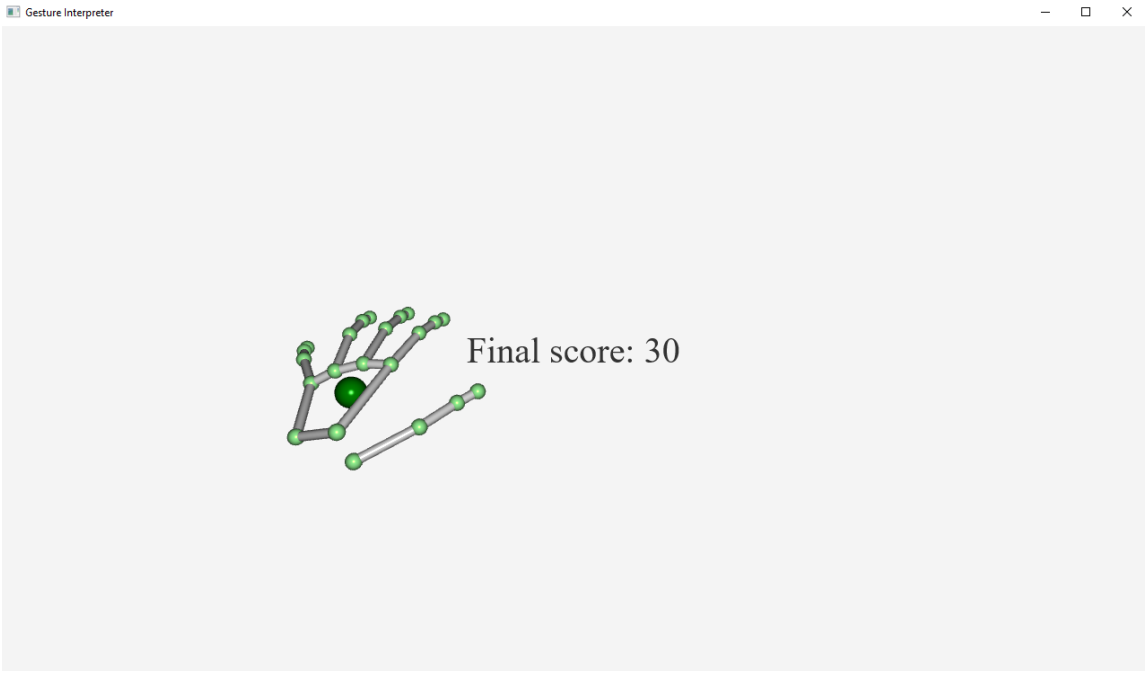



**Figure 5.4:** Application state when recognition timer expires

## 5.5 Calibration

When the user touches the right-most ‘calibration’ button, the screen shown in figure 5.5 is displayed. Calibration is a means of improving the application’s recognition via a set of gestures which are categorized from A through Z. For each alphabet letter, an image is displayed in the centre-top of the screen, displaying its recognized BSL equivalent. Users are able to perform their own gestures for each letter if desired, the image only serves as a suggestion. Additional labelled sets added in this manner improve the accuracy of the recognition algorithm as there is a wider variance of objects to select for each given category.

The user is given a three second countdown timer before recording begins, at which point they are able to store a gesture. Upon successful storage of a gesture, the application will move onto the next letter and the gesture guidance image will update accordingly. When the last letter (Z) has been recorded, the application automatically returns to the main menu.

The implemented system provides a degree of flexibility as well as extensibility. For example, users may wish to record gestures for words from a dictionary list as well as the alphabet letters – this is possible by adding additional word sources to the system. The only limitation in this case is the scope of the Leap Motion’s recognition capabilities with regard to gestures that include the face or body etc.

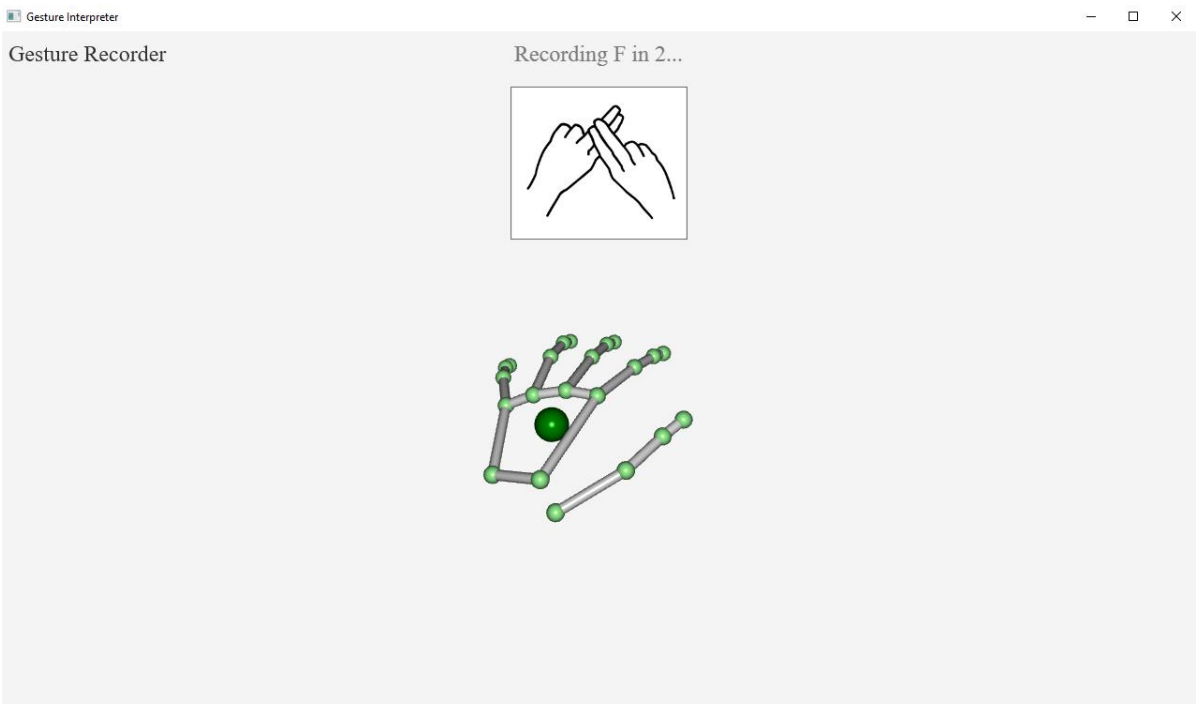



**Figure 5.5:** Application state when the ‘calibration’ button is pressed

# Chapter 6

# Testing and Evaluation

## 6.1 Testing Procedure

Testing was conducted as a multi stage process in order to analyse every section of the application independently.

The application was first checked against the user requirements specified in chapter 3, in order to evaluate whether it was successful in satisfying all of the criteria or not.

The next test looked at the quality of the application's recognition – this included accuracy, time taken, and the effect of modifying the number of points used to represent a gesture.

The performance of both the Leap Motion device and the application in general were also evaluated, as these could potentially be problematic in some cases.

Finally, feedback on the application was collected from a group of user participants and subsequently evaluated.

## 6.2 User Requirements Analysis

Table 6.1 shows a range of tests completed in order to verify that the application satisfies each of the user requirements originally specified in chapter 3. For each requirement, a test and an accompanying result is shown.

**Table 6.1:** Tests and results against each user requirement

| ID | Requirement | Test | Result |
|---|---|---|---|
| R1 | The system shall display a real time interpretation of the user's hands during operation | When the user's hand enters the Leap Motion controller's field of view, it should be displayed in the application | **Successful** – Hands are displayed when recognized by the Leap Motion |
| R2 | A user shall be able to record and store their own data for a given gesture | When performing a gesture in the calibration section of the application, the gesture data should be stored to disk | **Successful** – the object representing the gesture is stored in a unique text file on disk |
| R3 | The system shall recognize a gesture provided by a user | When a gesture is performed by a user in the recognition section of the application, the application | **Successful** – The closest matching gesture is shown on screen when a gesture is performed |

| | | | |
|---|---|---|---|
| | | should present the user with a recognition result | |
| R4 | The system shall present user feedback, a normalized score, based on the similarity between a given gesture and stored gestures | When a gesture is recognized, it should be accompanied by a user score | **Successful** – A percentage score in the range 0%-100% is returned with the closest matching gesture, signifying how close the match is |
| R5 | The system's real time display of a user's hands shall be updated with a latency of 5ms or less | When a user's hand is inside the Leap Motion's field of view, the position update method should return in less than 5ms | **Successful** – The hand's update method takes between 0.16ms and 0.53ms to return, well below the requirement target of 5ms |
| R6 | The recognition of gestures shall take no longer than 200ms to complete | When called, the gesture recognition method should return in less than 200ms | **Successful** – The recognition method's average time taken was 14.57ms (Observed in figure 6.2) |
| R7 | The system shall recognize gestures with an accuracy of at least 80% | A given gesture should result in a correct match at least 80% of the time | **Successful** – Recognition accuracy rates of 100% were achieved with 9 training sets (Observed in figure 6.1) |
| R8 | The system shall implement a simplistic interface which doesn't rely on mouse or keyboard input and can be understood by a user within 5 minutes of use. | The application should be fully navigable using only the Leap Motion. User testing should provide insight into the interface's usability metric. | **Successful** – All sections of the application can be accessed through input only from the Leap Motion. All users who provided feedback on the application agreed it was easy to understand and use after 5 minutes of use. (subsection 6.6). |
| R9 | The system shall have a reliability of 100%. (Should never crash or otherwise exhibit failure) | The application should not crash, close, or fail when performing any task | **Successful** – Application will only close upon a user's request. Application never crashed or otherwise showed signs of failure throughout all development and testing. |

## 6.3 Recognition Analysis

Achieving a high recognition accuracy is the primary and most important goal of the application. If the system is unable to distinguish between gestures then it's not fit for purpose no matter how well it performs in other areas.

Testing was carried out with a data set of 10 of each of the 26 BSL alphabet letters, for a total of 260 samples. From this, 1 set of letters was used as the test data, with the remaining sets used as training data. This test employs the theory of machine learning as described in subsection 4.5.3.

The amount of training sets used in the tests varied from 1 set to 9 sets in order to check how the accuracy of the recognition scaled as a result.

32 position points were used to store each gesture for every test, unless otherwise stated.

The time taken to generate each recognition was recorded by taking the time difference between calling the recognition method and receiving a result. This time value was calculated via Java's nanoTime() method, then converted to milliseconds and rounded to 2 decimal places. It can be assumed that this time value is therefore accurate to the nearest nanosecond.

Figure 6.1 shows the performance of the application when recognising all 26 letters of the test set, with a varying number of training sets. Figure 6.2 shows the average time taken to complete all 26 recognitions of the test set, with a varying number of training sets. Figures 6.3 and 6.4 show the performance and time taken (respectively) with a varying number of points, with both using 9 training sets.

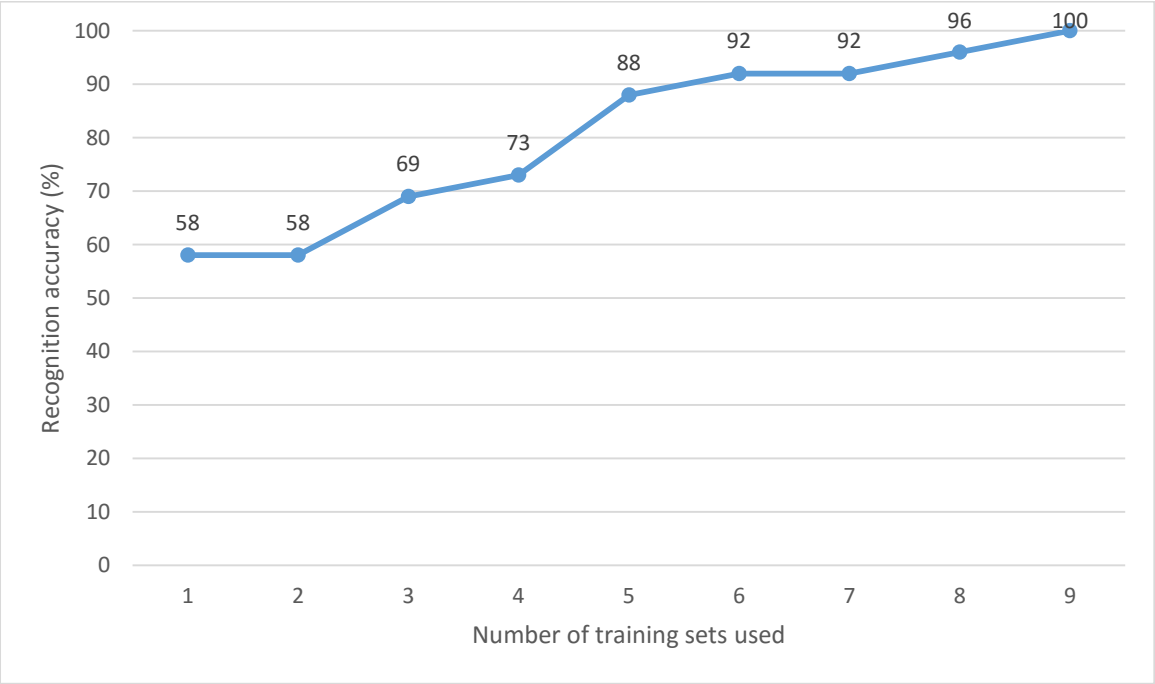



**Figure 6.1:** Recognition accuracy of all 26 alphabet letters, with a varying number of training sets

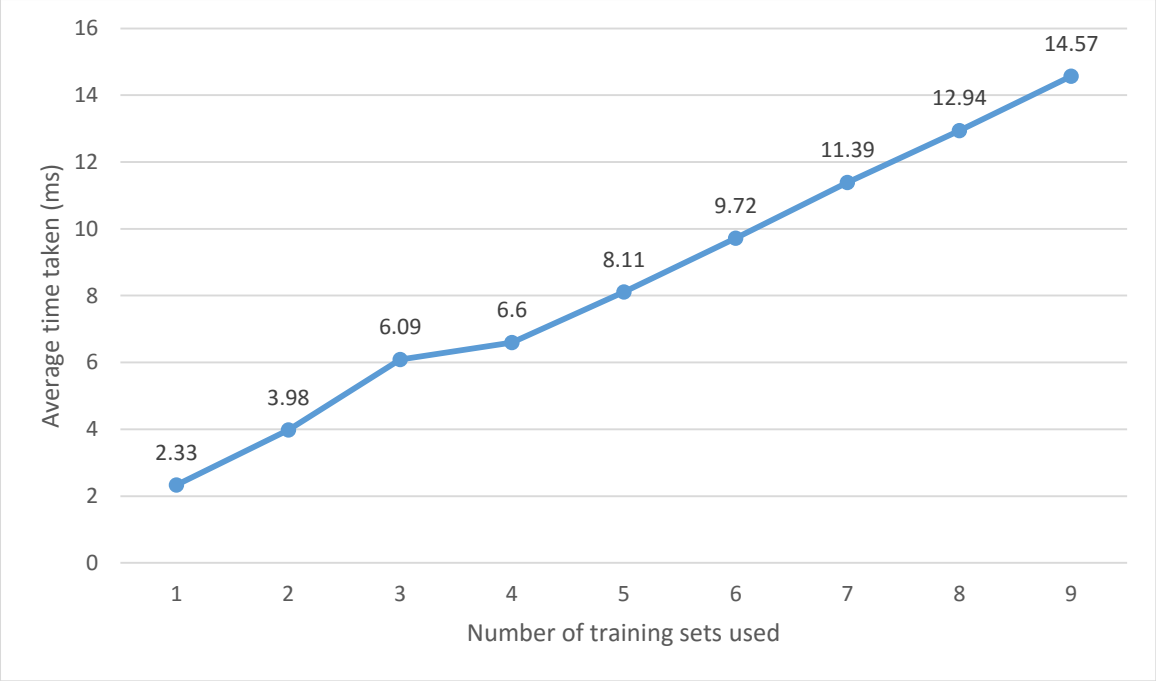



**Figure 6.2:** Average time taken to recognise all 26 alphabet letters, with a varying number of training sets

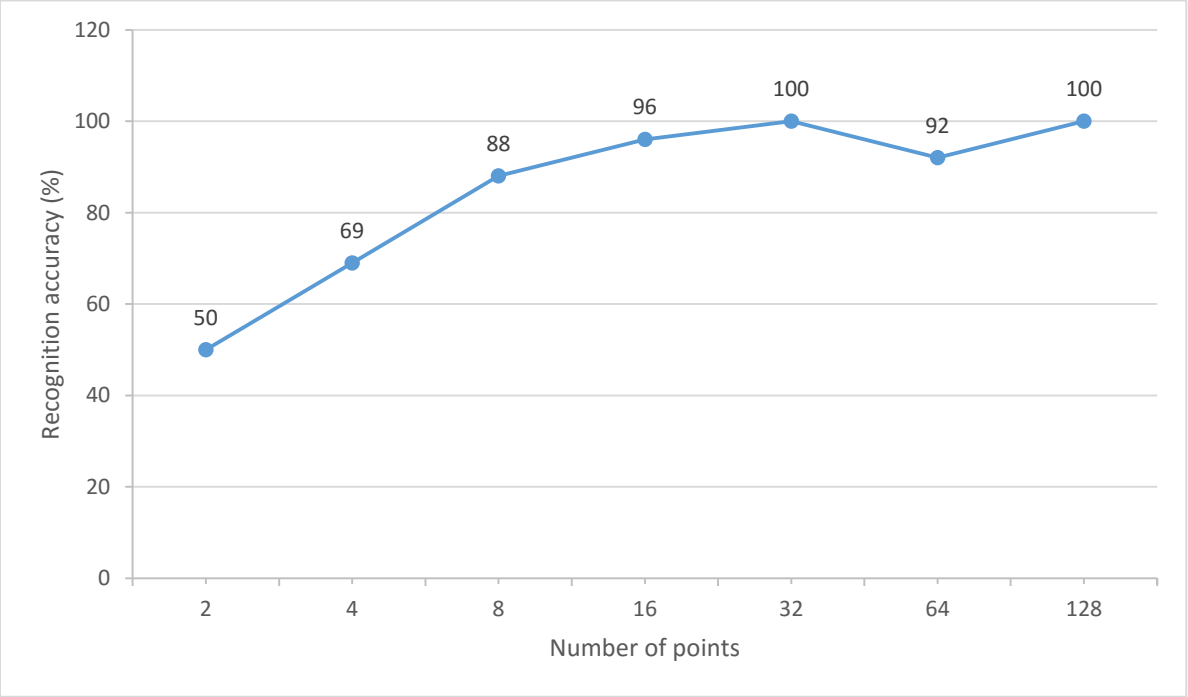



**Figure 6.3:** Recognition performance (of all 26 alphabet letters) with a varying number of points (9 training data sets used for each test)

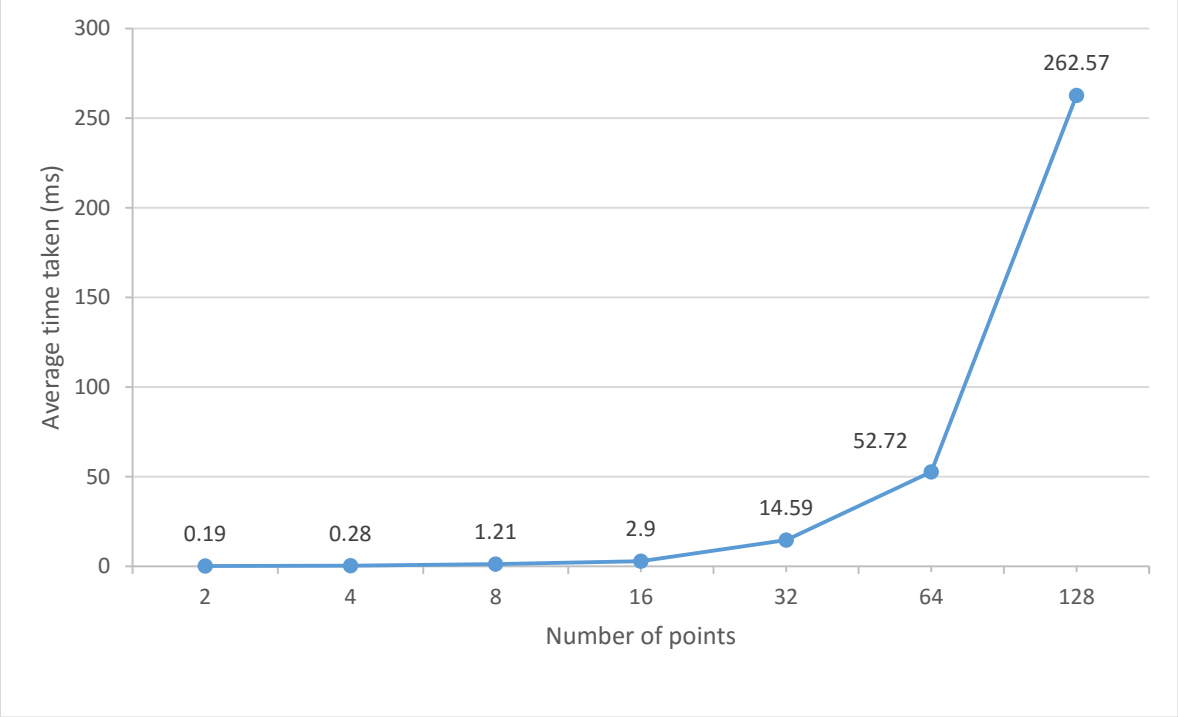



**Figure 6.4:** Average time taken to recognise a set of all 26 alphabet letters with a varying number of points (9 training data sets used for each test)

Firstly, looking at figure 6.1 – there's a clear correlation between the system's recognition accuracy and the number of training sets used. As the number of training sets used increases, the range of acceptable values for each feature of a gesture widens. This means a given gesture has more margin for error when comparing to stored gestures. In comparison, with a single training set there is only a single accepted value for each feature. As a result, the testing data set has to match the training data set much more closely in order to generate a match. This is particularly problematic when considering 3D gestures, where it's unlikely that a given gesture will be performed in exactly the same manner every time. Section 6.4 highlights the concerns regarding the recognition of features themselves by the Leap Motion controller.

Figure 6.2 shows a linear increase in time taken (with the time taken with 3 sets being a slight outlier) as the number of training data sets increases. It's important to note that even with 9 training sets used, the recognition of all BSL letters is still completed in only 14.57ms. One non-functional requirement of the application, R1.6, specified that recognition should take less than 200ms, in this regard the application is therefore successful. s

Figures 6.3 and 6.4 show that using 32 position points provides the optimum result in terms of accuracy and speed. Figure 6.4 shows a near exponential increase in time taken as the number of points used is increased. This is most evident with at least 16 points being used (2.9ms), compared with 32 points (14.59ms). Past this point the time taken increases at a much higher rate.

If speed of recognition is more of a concern than absolute accuracy, the use of 16 points could be considered as it still resulted in a 96% accuracy rating, meaning it misrepresented only a single alphabet letter. The use of more than 32 points may only be relevant for data which

contains more feature values as there is therefore more data available to manipulate. Splitting features into too many points can actually have a negative effect, as seen with a 92% recognition rate when using 64 points – not to mention that the time taken also greatly increases.

## 6.4 Leap Motion Performance

The recognition algorithm is certainly capable of providing accurate results, given the maximum accuracy of 100% as shown in figure 6.1. The issue lies in the capture of these test gestures to recognise in the first place, due to the frequent detection problems that the Leap Motion exhibits.

Unfortunately the Leap Motion software in general leaves a lot to be desired. The recent v2 firmware upgrade helped alleviate some issues through the inclusion of unique bone tracking, but many more still remain. The Leap Motion recognizes hands based on a best guess system so can very easily misrecognize hand motions – or fail to recognize them entirely. This is evident in the case of this application due to the complex hand poses required to represent some BSL letters.

Figure 6.5 shows three rows of gestures, with the gestures in each row bearing many similarities. In practice a new user is unlikely to generate correct matches with these gestures unless they make a deliberate attempt to exaggerate or modify the appearance of each. Looking at the first row, ‘I’ and ‘O’, the features available from the Leap Motion API make it impossible to reliably distinguish between them. If we consider the differences between them, ‘I’ requires the user to raise the middle finger on their left hand against the index finger on their right hand. ‘O’ is nearly the same pose, the only difference being the left hand’s ring finger is raised instead.

It’s a similar scenario when looking at the second row of figure 6.5. When we consider the features that are recorded to represent each gesture, the majority are the same for both gestures. Looking at ‘F’ and then ‘X’, the middle fingers on each hand are different but the palm and other finger positions are very similar (clasped in the palm). A test gesture representing ‘F’ or ‘X’ is therefore close in distance to both ‘F’ and ‘X’. Ideally we would access to a larger variety of feature types through the Leap Motion API, or store more features to represent each gesture - though the latter would have a negative effect on the recognition speed.

The third row is related more to the lack of inference capability in the Leap Motion. When one hand is above another, the Leap Motion is essentially blind to the motion of the upper hand due to occlusion by the back of the lower hand. This makes it difficult to reliably recognize ‘L’, ‘M’, ‘N’, and ‘V’ without generating false positive results. I would usually have to make these gestures several times and adjust finger positions slightly each time before the correct match is given.

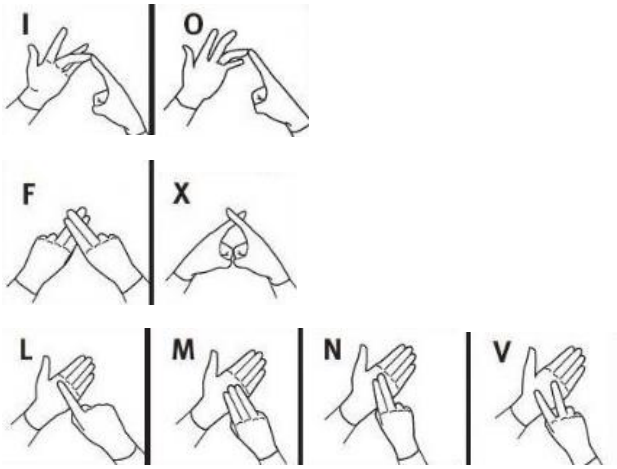



**Figure 6.5:** Frequently mismatched gestures

A second problem with the Leap Motion relates to its misrepresentation of hand position and motion. Figure 6.6 shows two examples of letters which are difficult to recognize due to this. As mentioned earlier, the Leap Motion functions using a best guess system. In some cases this system will fail to accurately represent hand motions. This will cause hands to become distorted or disappear entirely as the Leap Motion is unable to process them. This usually occurs when hands are pressed close together, fingers are interleaved, or hands are placed on top of one another.

The 'G' gesture in particular is difficult for the Leap Motion to recognise due to the complete occlusion of the right hand when it is placed above the left hand. It usually requires several attempts to perform the gesture. Rotating the hands backwards / forwards aids in recognition by allowing more of the upper hand to be seen by the Leap Motion sensors - but then it is not the true representation according to the BSL.

The 'W' gesture sees the fingers on both hands being interleaved / locked together. This makes it difficult for the Leap Motion to distinguish which fingers belong to which hand and will frequently result in the hands becoming undetected until they are separated again.

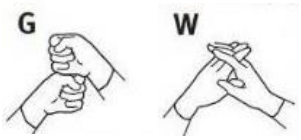



**Figure 6.6:** Frequently misrepresented gestures

Along with complex gestures, a few other factors are likely to cause problems for the device:

Firstly, lighting conditions must be considered due to the dual infrared cameras used to detect the position of hands and fingers. External infrared light can and will be detected by the cameras, resulting in reduced tracking performance.

Additionally, any objects close to the device will obscure the view of these tracking cameras, also resulting in reduced performance.

Finally, the visible range of the tracking cameras may prove to be problematic. From my own experience with the device, and from the device's software control panel, the infrared cameras will only reliably track hand positions at a maximum range of approximately 25cm. As hands move further than this distance away from the device, they will typically be misrecognized by the device, eventually becoming entirely undetected.

The combination of all of these factors means the environment in which the Leap Motion is used must be carefully considered – particularly so when using an application which is significantly reliant on the device's recognition capabilities.

## 6.5 Application Performance

The performance of the application could be a concern for some users. The Leap Motion is advertised as a compact, portable device – it would therefore not be unreasonable to assume that some users will pair it with low-power laptops, netbooks and so on. Although the application is relatively simple, it surprisingly has a fairly high resource usage.

Figure 6.7 shows the resource usage of a system on which the application was ran for approximately 90 seconds. Before the application is started, we can see a system memory (RAM) usage of 4646MB and a video memory (VRAM) usage of 757MB. After running the application for around 90 seconds, we see a usage of 5114MB RAM and 1306MB VRAM, a difference of 468MB and 549MB respectively. For systems which use a shared pool of VRAM and RAM this could be problematic as the application therefore uses just over 1GB of combined memory. With extended use this is liable to increase, depending on the efficiency of the JVM's garbage collection.

The application was tested on two platforms: The first, a desktop with 6GB VRAM and 8GB RAM. The second, a laptop with 2GB combined memory. Throughout development the latter system was noticeably slower, with hands appearing delayed as they moved across the application window. Performance on the desktop, on the other hand, was never a problem. This performance difference is evident in the provided video demo – the first system shown (laptop) is noticeably slower than the second system (desktop). When looking at the extensibility of the application, its resource usage should therefore be considered a potential limitation.

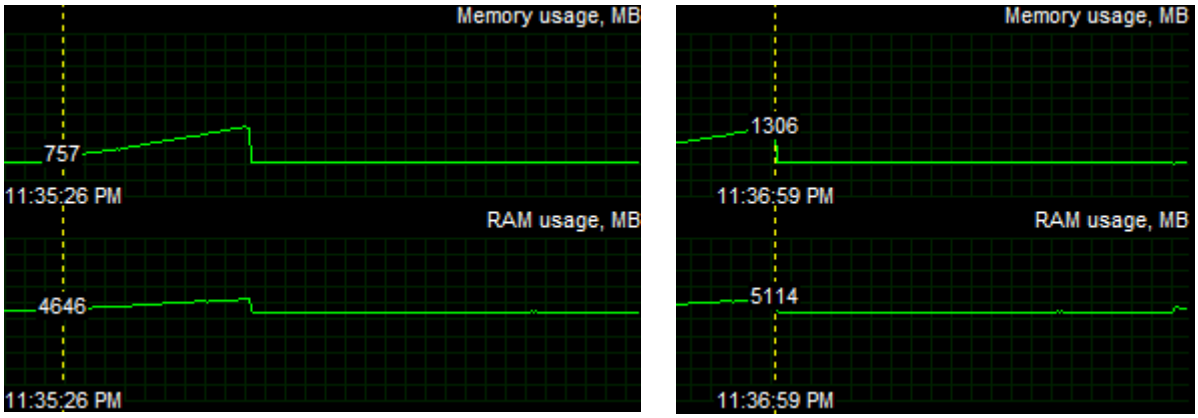



**Figure 6.7:** Comparison of video memory and system memory used after the application was ran for approximately 90 seconds

## 6.6 User Feedback

Although the application has a deliberately simple user interface, there are other aspects of the system on which user feedback is helpful. In order to gather feedback on the application's interface, and the application in general, the questionnaire shown in figure 6.7 was given to 5 participants. Each participant was asked to try the calibration and recognition features of the application, without any external guidance. After 5 minutes of experience with the application, participants were asked to answer the questionnaire and prompted for any additional feedback if possible.

The questionnaire is based on the Likert scale question format [10]. The use of a Likert scale allows users to select an option from a multiple point scale (in this case five) which represents how much the user agrees or disagrees with a particular statement. The scale is best used when measuring the attitudes of participants – this is therefore a good match for my study as I'm most interested in what users opinions of the application are after having used it. From 1 to 5, the Likert scale represents 'Strongly Disagree', 'Disagree', 'Neutral', 'Agree', and 'Strongly Agree'.

The results of the questionnaires are shown in Tables 6.2 and 6.3. Table 6.2 shows the Likert responses, tallied for all users with an average response for each question. Table 6.3 shows the additional feedback comments provided by each user. With the results of the user study stored, each of the questionnaire statements are evaluated in turn in combination with the additional feedback provided.

Firstly, statement 1, "The application was easy to understand and use", considers the accessibility of the application overall. When designing an application it's important to consider both experienced users and users who may have not come across a Leap Motion device before. The application should therefore be straightforward and simple to use, without having to consult walkthroughs or guides. Additionally, two users added relevant comments as extra feedback – user 2 stated the UI was "simple" and "easy to use without experience". User 5 stated the interface was "simple to use with just my hands". The average response to this statement was 4.4, representing an average opinion between 'Agree' and 'Strongly Agree'.

Statement 2, "The interface buttons were easy to interact with", looked at the responsiveness of the interface, particularly the buttons used to manoeuvre around the application. This aspect was probably the greatest unknown as I wasn't sure whether users would approve of it over simply clicking buttons with a mouse. As with statement 1, however, the average response was 4.4. This implies that users were satisfied with the Leap Motion based interface.

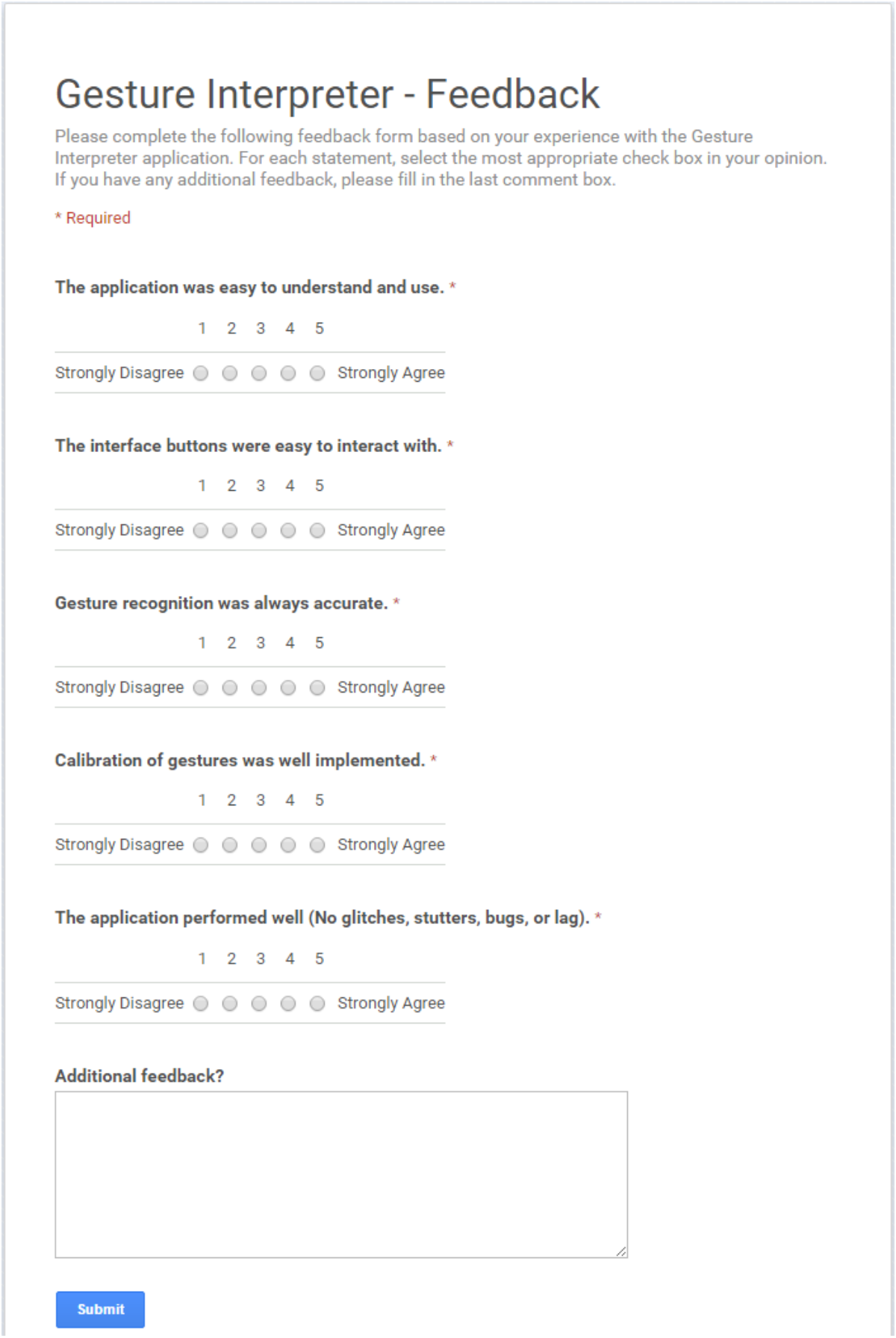
From observing users with the application, all managed to press each button on their first or second attempt. Out of the 5 participants, every user attempted to press the buttons using the Leap Motion first, rather than attempting to click it with a mouse. This suggests that each user subconsciously expected the interface to function using the Leap Motion due to the reliance on it for gesture recognition.

Statement 3, "Gesture recognition was always accurate", was the lowest scoring statement with an average response of 3.8. This was in line with my expectations as there were a number of recognition issues with some gestures, as documented in section 6.4. Users found it most difficult to match letters 'I' and 'O', with 2/5 users mentioning this in their additional feedback. Other problematic gestures mentioned included 'G', 'L', 'M', N', and 'V'. One user mentioned

“Hands sometimes not representing my movements properly”, which was another issue noted during development.

Statement 4, “Calibration of gestures was well implemented”, saw the highest average response of 4.8. In addition, two users included additional feedback relating to this statement: “Images were helpful for calibration”, and “Recording images were convenient to show me how to make each gesture”. As none of the users had previous experience with a sign language, it was agreed that the inclusion of a sample image for each gesture during recording was helpful as a guide.

Finally, statement 5, “The application performed well (No glitches, stutters, bugs, or lag)”, considered the application’s stability. The statement was met with an average response of 4.0. Throughout observation of each user, there were several times where the display of the user’s hands did not fully match their actual positions, which occurred when fingers were pressed together or interleaved. Unfortunately this is a limitation on the device’s capabilities so cannot be easily resolved. The application as a whole, however, maintained a regular performance level throughout each test, and did not close unless a request was explicitly sent by a user (e.g. by closing the application window manually).



# Gesture Interpreter - Feedback

Please complete the following feedback form based on your experience with the Gesture Interpreter application. For each statement, select the most appropriate check box in your opinion. If you have any additional feedback, please fill in the last comment box.

* Required

**The application was easy to understand and use.** *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ○ | ○ | ○ | ○ | ○ | Strongly Agree |

**The interface buttons were easy to interact with.** *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ○ | ○ | ○ | ○ | ○ | Strongly Agree |

**Gesture recognition was always accurate.** *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ○ | ○ | ○ | ○ | ○ | Strongly Agree |

**Calibration of gestures was well implemented.** *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ○ | ○ | ○ | ○ | ○ | Strongly Agree |

**The application performed well (No glitches, stutters, bugs, or lag).** *

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| Strongly Disagree | ○ | ○ | ○ | ○ | ○ | Strongly Agree |

**Additional feedback?**

Submit

**Figure 6.8:** User feedback questionnaire

**Table 6.2:** Combined numeric results of user feedback questionnaires

| Gesture Interpreter – Questionnaire Feedback | | | | | | |
|---|---|---|---|---|---|---|
| **Question** | **Response Given** | | | | | **Average Response** |
| | 1 | 2 | 3 | 4 | 5 | |
| 1 | | | | 3 | 2 | 4.4 |
| 2 | | | | 3 | 2 | 4.4 |
| 3 | | | 2 | 2 | 1 | 3.8 |
| 4 | | | | 1 | 4 | 4.8 |
| 5 | | | 1 | 3 | 1 | 4.0 |

**Table 6.3:** Additional feedback provided in feedback questionnaire from each user (if any)

| Gesture Interpreter – Additional Feedback | |
|---|---|
| **User** | **Feedback Given** |
| 1 | Images were helpful for calibration.<br><br>Difficulty making gestures I,O,G. |
| 2 | Simple UI, easy to use without experience.<br><br>Hands sometimes not representing my movements properly. |
| 3 | Had trouble with gestures like l, m, n and sometimes v all being detected as each other. |
| 4 | None given |
| 5 | Hard to recognise gestures like I and O, kept getting confused between them.<br><br>Hands glitching out when I moved sometimes.<br><br>Same problem with other gestures like M and N.<br><br>Interface was good to use with just my hands.<br><br>Recording images were convenient to show me how to make each gesture. |

# Chapter 7

# Conclusion

## 7.1 Aims Analysis

In the introductory chapter, a number of aims were defined for the project application. By revisiting these it can be established whether or not the application is successful in fulfilling its requirements. The aims were identified as follows:

- Record sign language gestures performed by a user by storing image data from a Leap Motion device
- Recognise the gestures representing British Sign Language alphabet characters and distinguish between them
- Given an unknown gesture by a user as input, output an identified matching gesture with an associated similarity score

The first aim of being able to store gestures was successful. The application allows a user to save either motion-based or stationary gestures for a given alphabet character. The user is able to perform a gesture of their own choosing, or a copy of one provided in the example image shown for each character.

The only restrictions on the user are the limitations of the Leap Motion device itself. The user is limited to gestures which do not involve the face or body (these would not be detected by the Leap Motion) and which can be fully captured within the Leap Motion's field of view. Gestures which are exceedingly similar to one another may prove difficult to distinguish during recognition, due in part to the restricted set of features that can be extracted from a Leap Motion data frame, but also due to the limitations of the Leap Motion software as covered in section 6.4.

The second aim looks at the explicit range of categories which can be observed by the application. The system is successful in this regard as it is able to recognise all 26 BSL representations of alphabet characters, though with some caveats. The hand positions of similar gestures should be exaggerated in some way, for example performing the gesture at a different angle than you normally would. The user must consider the degree of similarity of all gestures they record using the application, in order to improve the reliability of the system's recognition.

The third aim looks at providing feedback to a user in order to confirm that their actions are recognized by the application. This third aim was also successful. When the application's recognition method is called, the returning object contains the closest matching gesture as well as a similarity score. This score is calculated by normalizing the distance between this closest matching gesture and the input gesture as a percentage value between 0 and 100. With this, the user is provided with feedback on the quality of the recognition, as well as the recognition itself.

## 7.2 Future Work

There are a number of areas which could be explored further in order to enhance both the application in general and the accuracy of its recognition.

### Data inference capabilities

In its current state, the Leap Motion uses a best-guess system in order to calculate the position, rotation, and other characteristics of hands detected in its field of view. Improved inference capabilities would allow the Leap Motion to make better judgements given a frame of data which contains unclear hand positions. One example of this would be BSL gestures which require hands or fingers to be stacked on top of each other. The upper hand is blocked by the lower hand and thus the Leap Motion is unable to categorize the data.

A possible solution to this issue could be the use of multiple Leap Motion devices. One device could be positioned below the hands as usual, with the other mounted above or to the side of the interaction area. This would limit the portability of the application but may assist in clarifying obscured hand positions. However, this would introduce an additional challenge of synchronizing multiple Leap Motion devices as well as combining the data received by each device on each new frame of image data.

### Alternate recognition algorithm

As described in chapter 2, there are a number of machine learning algorithms currently available and probably many more under development. Possible areas of improvement could be found in the speed of the algorithm, as well as its performance with a restricted number of training sets. As seen in chapter 6, the implemented algorithm required 9 training sets before an accuracy rating of 100% was reached.

Ideally this would've been possible with only 1 training set, though this also relates to the quality of the features that can be selected from the Leap Motion frame data – which although reasonably varied at the moment, has the potential for further improvement.

The current recognition algorithm uses custom Point objects which represent co-ordinates in 3D space. The implementation of additional algorithms with support for more standard data types (floats, booleans etc.) would allow for more varied features to be recorded for gestures as they would not be limited to only types which could be easily converted to Point objects, e.g. 3D vectors.

### Improved Leap Motion software

As discussed in chapter 6, the Leap Motion's current software and API is limited in some regards. An expanded API with a wider range of hand feature tracking would allow for more concise categorization of gestures. A major software update (V3), 'Orion', was released in a beta state in February 2016 [9]. Orion boasts lower latency and better tracking of hands, as well as reduced CPU usage.

Unfortunately the update has several known issues, including an issue with the tracking of hands which overlap, stating that they're currently handled worse in firmware V3 than in V2. From experience with the new firmware, compared with V2, the inference capabilities of V3 are greatly reduced, so much so that it is not suitable to recognise gestures which contain any degree of hand overlapping. Ultimately until this is resolved V3 isn't suitable for use with this

application. For applications where hands aren't required to overlap, Orion's improved tracking in all other aspects is definitely noticeable.

## 7.3 Lessons Learned

Probably the most substantial thing to take away from this project would be the experience of working independently to solve a problem that I had no prior knowledge of, within a specific time span.

The project required me to interact with a new device (Leap Motion) as well as several new APIs (Leap Motion, JavaFX) and integrate them appropriately into a single application. Despite some base knowledge of the Java programming language, it was the first time I had been exposed to many new technologies at once and tasked with developing software using them all appropriately.

Although hearing of terms such as machine learning in the past, the project required me to explore the area in greater detail in order to implement a recognition algorithm which would satisfy the primary objective of the application.

If I was to undertake the project again, I would definitely have started development earlier. Looking back, I should've spent more time on the project over the summer break, so as to not have to reduce the scope of the application in order to actually complete it in time. At the time I did not consider the challenges I would face during development, and the impact these would have on the project's overall timeline.

With more time available I would've liked to expand on the uses of the application. Learning a sign language is likely to be more effective with a variety of activities which the user can participate in, all of which involve the recognition of gestures in some way. The current implementation shows that gesture recognition is possible, but its utilization is more of a proof of concept than anything else at this point.

## 7.4 Project Conclusion

Overall I consider the project to be a success. The application successfully met all of its original aims, as well as all of the requirements I had originally defined. Undertaking the project helped develop both my technical and development skills as well as other skills such as time management and strategic planning. The completion of the application has shown that the Leap Motion is certainly capable of gesture recognition, with the caveat that the device's software requires further development. The recent firmware V3 update, Orion, although currently in a beta state, looks to be very promising for the Leap Motion's future.

# References


[1] Chen, M. (2013). Universal motion-based control and motion recognition. On line publication, Georgia Institute of Technology, http://www.dtic.mil/dtic/tr/fulltext/u2/a344219.pdf. Last accessed 1 November 2015.

[2] Chuan, C. H., Regina, E., & Guardino, C. (2014, December). American Sign Language Recognition Using Leap Motion Sensor. In Machine Learning and Applications (ICMLA), 2014 13th International Conference on (pp. 541-544). IEEE, 2014.

[3] Etherington, D. (2013, April). Leap Motion Controller Ship Date Delayed Until July 22, Due To A Need For A Larger, Longer Beta Test. Website, http://techcrunch.com/2013/04/25/leap-motion-controller-ship-date-delayed-until-july-22-due-to-a-need-for-a-larger-longer-beta-test/. Last accessed 4 December 2015.

[5] Human Benchmark. (2016) Reaction Time Statistics. Website, http://www.humanbenchmark.com/tests/reactiontime/statistics. Last accessed 9 February 2016.

[6] Leap Motion. (2014) Leap Motion V2 Tracking Now in Public Developer Beta. Website, http://blog.leapmotion.com/leap-motion-v2-tracking-now-in-public-developer-beta/. Last accessed 16 November 2015.

[7] Leap Motion. (2015) Introducing The Skeletal Tracking Model. Website, https://developer.leapmotion.com/documentation/java/devguide/Intro_Skeleton_API.html. Last accessed 5 February 2016.

[8] Leap Motion. (2015) Menu Design Guidelines. Website, https://developer.leapmotion.com/documentation/java/practices/Leap_Menu_Design_Guidelines.html. Last accessed 9 February 2016.

[9] Leap Motion. (2016). Orion Beta. Website, https://developer.leapmotion.com/orion. Last accessed 17 February 2016.

[10] McLeod, S. A. (2008). Likert Scale. Retrieved from www.simplypsychology.org/likert-scale.html. Last accessed 17 February 2016.

[11] Mohandes, M., Aliyu, S., & Deriche, M. (2014, June). Arabic sign language recognition using the leap motion controller. In Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on (pp. 960-965). IEEE. Chicago, 2014.

[12] MotionSavvy. (2015) UNI. Website, http://www.motionsavvy.com/. Last accessed 9 October 2015.

[13] Oracle. (2014) JavaFX: Getting Started with JavaFX. Website, https://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm. Last accessed 2 February 2016.

[14] O'Leary R. (2013) LeapTrainer.js. GitHub repository, https://github.com/roboleary/LeapTrainer.js. Last accessed 9 October 2015.

[15] Potter, L. E., Araullo, J., & Carter, L. (2013) The leap motion controller: a view on sign language. In Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration (pp. 175-178). ACM, 2013.

[16] Signature. (2015) British Sign Language. Website, http://www.signature.org.uk/british-sign-language. Last accessed 1 November 2015.

[17] Vatavu R.D., Anthony L., & Wobbrock J. O. (2012) Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes. On line publication, University of Washington, http://faculty.washington.edu/wobbrock/pubs/icmi-12.pdf. Last accessed 13 October 2015.

[18] Vikram, S., Li, L., & Russell, S. (2013) Writing and sketching in the air, recognizing and controlling on the fly. In CHI'13 Extended Abstracts on Human Factors in Computing Systems (pp. 1179-1184). ACM, 2013.

[19] Vos, J. (2014) Leap Motion and JavaFX. Website, http://www.oracle.com/technetwork/articles/java/rich-client-leapmotion-2227139.html. Last accessed 2 February 2016.

[20] Weichert, F., Bachmann, D., Rudak, B., & Fisseler, D. (2013) Analysis of the Accuracy and Robustness of the Leap Motion Controller. On line publication, PubMed Central, http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3690061/pdf/sensors-13-06380.pdf. Last accessed 16 October 2015.