Dylan Demolder

Registration number 100297926

2022

Gesture Interpreter with Leap Motion

Supervised by Jaejoon Lee

University of East Anglia
Faculty of Science
School of Computing Sciences

Abstract

In this project I will use the Leap Motion Controller to interpret the British Sign Language (BSL). The system recognises the gesture of hands for BSL through the utilisation of the Leap Motion Controller and displays the classified phrase using UnrealEngine 4.

Acknowledgements

I would like to thank all of my friends and family who have supported me during this project and report writing. They all know how much of a struggle this has been for me due to my dyslexia, but have all taken time out of their own schedules to lend a hand. I would also like to thank Debbie Taylor for going above and beyond with resources and advice for my report. But a special thanks to Dr. Jaejoon Lee for running this project and giving me guidance and suggestions throughout the whole development and also for providing the LeapMotion Controller.

Contents

List of Figures

1. Introduction

British Sign Language is the standard for Sign Language in Britain. There are estimated to be around 60 thousand people in the UK that use BSL are their preferred language SignSolutions (2021). There has never been a universal sign language and each country/community has developed their own variants – much like spoken languages or dialects. As BSL is most definitely a minority language in Britain this project aims to be able to translate BSL into written / spoken English in real time. This project intends to assist BSL users to communicate with people that do not know how to use the language.

In the next few sections I will be analysing systems that utilise the same technologies or designs that I plan to be using throughout my development. I will also be discussing some ideas and structures that need to be understood before any development or further understanding can happen.

1.1. Similar Systems Analysis

In the following sections I will be explaining various other hand tracking technologies. Some will be related to the Sign Language translation while others may just be for gesture control. I will be using these similar systems as a baseline for my development and testing of the project; they will act as inspiration and an idea base.

1.1.1. On-Device, Real-Time Hand Tracking with MediaPipe

"This approach provides high-fidelity hand and finger tracking by employing machine learning (ML) to infer 21 3D keypoints of a hand from each frame of video"(vidursatija , 2019). This project uses a palm detection model called BlazePalm, a landmark model that processes information based on the location of the palm - an adaptive processing region. (Valentin Bazarevsky, Fan Zhang , 2019)

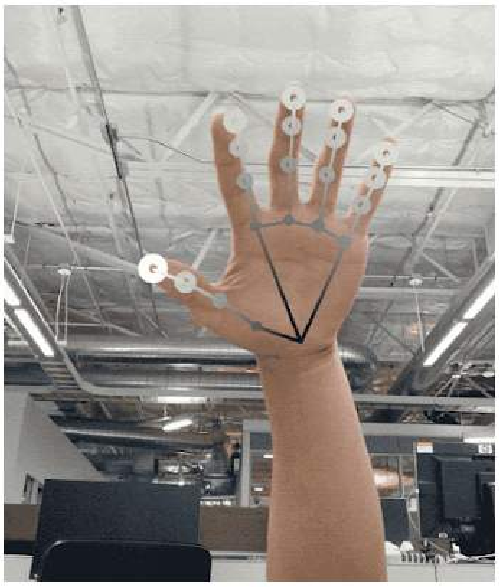


Figure 1: 3D hand perception in real-time on a mobile phone via MediaPipe. Depth is indicated in grayscale.(Valentin Bazarevsky, Fan Zhang , 2019)

This project also had some gesture recognition included. The project applied an algorithm to derive the gestures. "First, the state of each finger, e.g. bent or straight, is determined by the accumulated angles of joints"(Valentin Bazarevsky, Fan Zhang , 2019). Then, the set of finger states is mapped to a set of pre-defined gestures. This straightforward, yet effective, technique allows for the approximation of basic static gestures with reasonable accuracy.

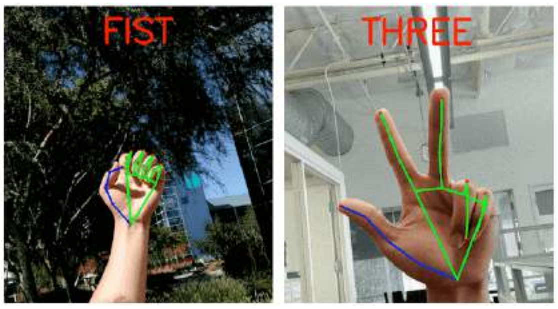


Figure 2: Gesture recogition using MediaPipe.(Valentin Bazarevsky, Fan Zhang , 2019)

1.1.2. GANerated Hands for Real-Time 3D Hand Tracking from Monocular RGB

The aim of this project was to create a real time 3D hand tracking system using just RGB (Red, Green, Blue) video. This would void the need for marker based tracking, pose estimation or depth cameras - which can become expensive and time consuming. This system uses a monocular method, meaning that the information that is processed is limited to a specific region that is dynamically changed throughout the tracking - making the program much more efficient.

The overall tracking is completed by locating the joints and tips[1] of the fingers, then creating a Joint Heatmap. These joints will then have a colour coded line drawn between them, and a visualisation will be placed on top of the original picture or video. (Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, Christian Theobalt , nd)

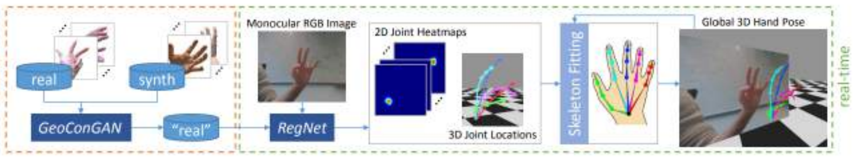


Figure 3: Pipeline of real-time system for monocular RGB hand tracking in 3D. Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, Christian Theobalt (nd)

Comparing the results from this project to the Zimmermann and Brox project Christian Zimmermann, Thomas Brox (2017), the results are clearly more accurate and have much better prediction accuracy when the hands are occluded or obstructed. In one of the figures given in the report, it can be seen that the location of the fingers are almost completely unknown using the Zimmermann and Brox method, but the method proposed calculated the locations much more accurately. Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, Christian Theobalt (nd)

[1]Can also be referred to as "tops".

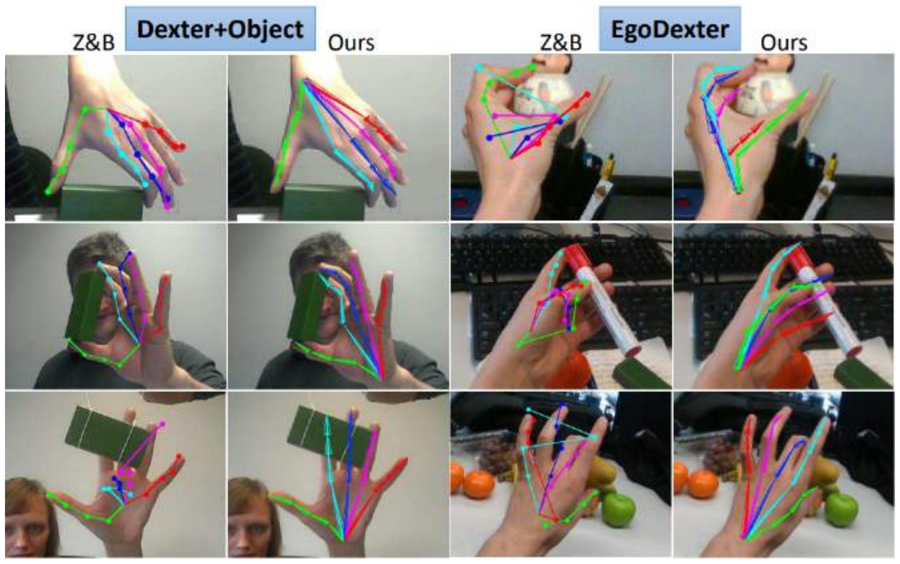


Figure 4: A comparison of accuracies from the Zimmerman and Brox method, to the one proposed. Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, Christian Theobalt (nd)

1.1.3. Improved Face and Hand Tracking for Sign Language Recognition

This tool uses feature extraction to detect the skin colour of the user, it will then remove noise and then try to differentiate the face and the hands - by calculating their size. This raised an edge case where some users may be wearing short sleeve tops as their arms interfered with the differentiation technique. No mitigation was added to this project but the issue was noted. After the face has been isolated, the eyes, nose and mouth can be detected and used for data extraction. The hands can also be isolated, supporting more accurate tracking and recognition.

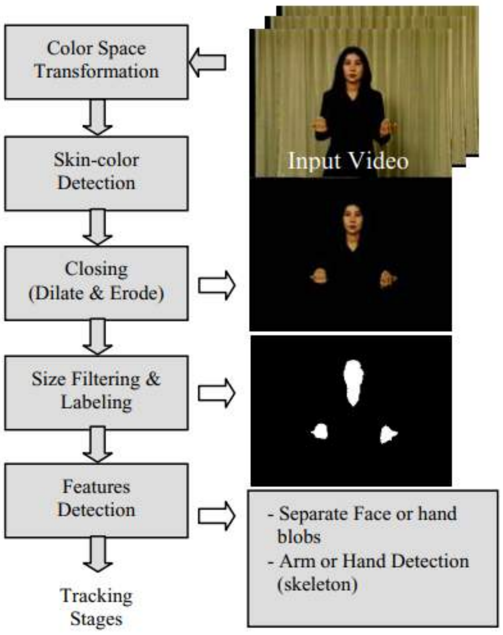
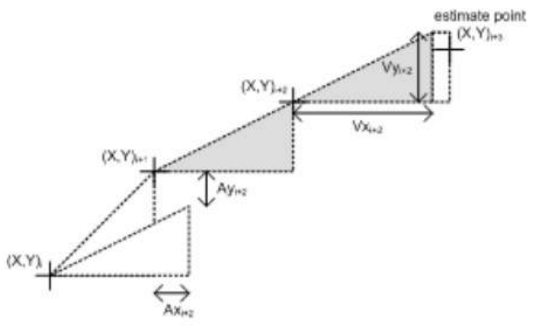

Color Space Transformation
Skin-color Detection
Closing (Dilate & Erode)
Size Filtering & Labeling
Features Detection
Tracking Stages
Input Video
- Separate Face or hand blobs
- Arm or Hand Detection (skeleton)

Figure 5: Initial Stages of Facial Tracking Narut Soontranon, Supavadee Aramvith, Thanarat H. Chalidabhongse (2005)

The hand tracking for their system has been estimated using the velocity and acceleration from consecutive frames of video. This uses the premise that previous data can predict future information, which can be accurate across large data sets, but also has issues with over predicting positions during velocity changes - such as user movement alongside Sign Language gesturing. This estimated position would move the detection frame window, so that the hands remain inside the search area. This limits the amount of data that the program needs to process to identify the hand position and gesture, increasing efficiency. This window can change size and location as needed, adapting to the region calculated from the acceleration and velocity of the hands.



$$\begin{bmatrix} x(i+1) \\ y(i+1) \end{bmatrix} = \begin{bmatrix} x(i) \\ y(i) \end{bmatrix} + \frac{1}{2} \begin{bmatrix} V_x(i) + V_x(i-1) + A_x(i) \\ V_y(i) + V_y(i-1) + A_y(i) \end{bmatrix}$$

(a) Motion Estimate Visualisation (b) Motion Estimate Formula

Figure 6: Motion Estimation Narut Soontranon, Supavadee Aramvith, Thanarat H. Chalidabhongse (2005)

By using an adaptive search area Narut Soontranon, Supavadee Aramvith, Thanarat H. Chalidabhongse (2005) can keep errors to a minimum whilst also providing fast and relevant information. The program works at an estimated 15fps and runs in a 320x240-pixel environment. This is relatively small to other systems, but provides more than enough information to calculate gesture information.

2. Related Work

In order to produce a full report and project, I will be reviewing related areas that maybe useful during the development. This will involve research into the development of Sign Language over the years, how Sign Language differs in various countries and the Leap-Motion Hand Tracker.

2.1. Sign Language

According to British Sign (nd) Sign Language is a visual means of communicating using gestures, facial expression and body language. Sign Language is used mainly by people who are deaf or who suffer from hearing impairments. SignSolutions (2021) estimate that there are over 300 variations of Sign Language used around the world , each having their own colloquialisms and unique gestures. Sign Languages definitely have strong regional variations. The reason for this is that the variants are more about what you say

- i.e. signs; equivalent to words in spoken languages, rather than how you say it - i.e. pronunciation (UCL, nd).

2.2. British Sign Language

"As its name suggests, British Sign Language (BSL) is the most widely used Sign Language in the UK" (SignSolutions, 2021). There are around 151,000 BSL users in the UK, and approximately 87,000 users are deaf. It is also used by the families and relatives of deaf people, Sign Language interpreters and BSL learners. "BSL has its own vocabulary, grammar and syntax and, as a language, is not dependant on spoken English" SignSolutions (2021). BSL is a rich combination of hand gestures, facial expressions and body language. Sign Health (nd) In 2003, the Government recognised BSL as an official minority language. BSL is part of BANZSL, which is also made up of Australian Sign Language (Auslan) and New Zealand Sign Language (NZSL). All are derived from the same Sign Language system used in 19th Century Britain. However, national variations exist, meaning that a deaf person from Australia or New Zealand may have difficulties communicating with a BSL user and vice versa. SignSolutions (2021)

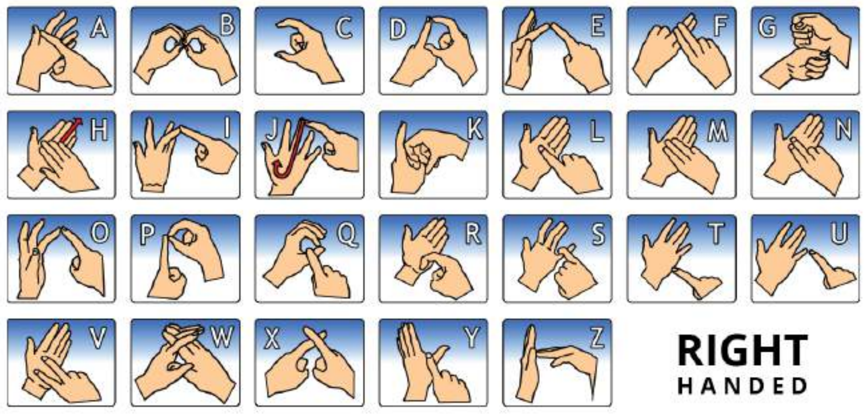


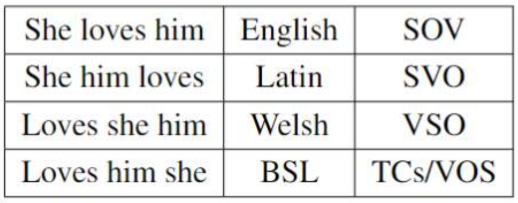
Figure 7: Finger Spelling in BSL (Right Handed) British Sign (2013)

2.3. Learning and Understanding Sign Language

Much like a spoken language, those who do not understand BSL will need to learn the language before being able to recognise or use it properly. This was proven during an

Austrian study. Non-signers were able to recognise pauses and breaks performed by the signers hands, but were unable to notice the head movements of the signer (Sign Non Manuals, nd). This study shows that Sign Languages, much like any language, are not intuitive and need to be learnt and practised before a user is able to understand it fully.

The syntax of grammar also varies from written or spoken English in BSL. In written English we use Subject-Object–Verb word order Rodney Huddleston (nd), but in BSL the word order is Topic-Comment-structure Amie Harrison (2016), for example:



She loves him	English	SOV
She him loves	Latin	SVO
Loves she him	Welsh	VSO
Loves him she	BSL	TCs/VOS

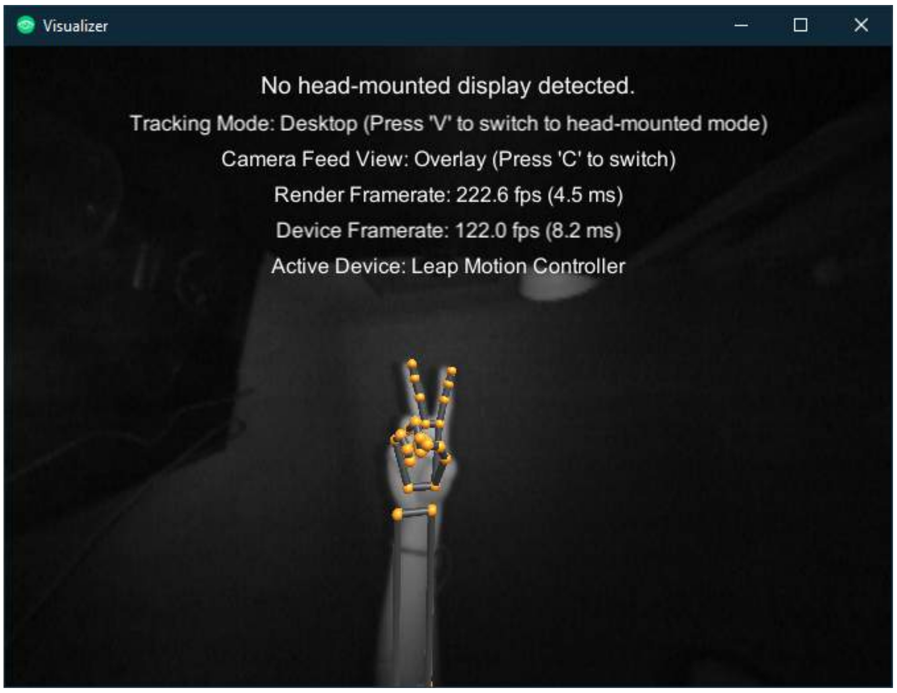
Figure 8: Grammatical Syntax Russell S. Tomlin (1986)

Figure 8 shows the differences between the same sentence when just the syntax is changed. The key for this table is:
S = Subject, O = Object, V = Verb, T = Time, C = Comment, s = Structure
The variation of grammar structure and syntax between languages give them unique properties.

2.4. LeapMotion Hand Tracker

"The Leap Motion Controller from Ultraleap is an optical hand tracking module that captures the movement of users' hands and fingers so they can interact naturally with digital content. Small, fast, and accurate, the Leap Motion Controller can be used for productivity applications with Windows computers, integrated into enterprise-grade hardware solutions or displays, or attached to virtual/augmented reality headsets for AR/VR/XR[2] prototyping, research, and development" (LeapMotion, nd).

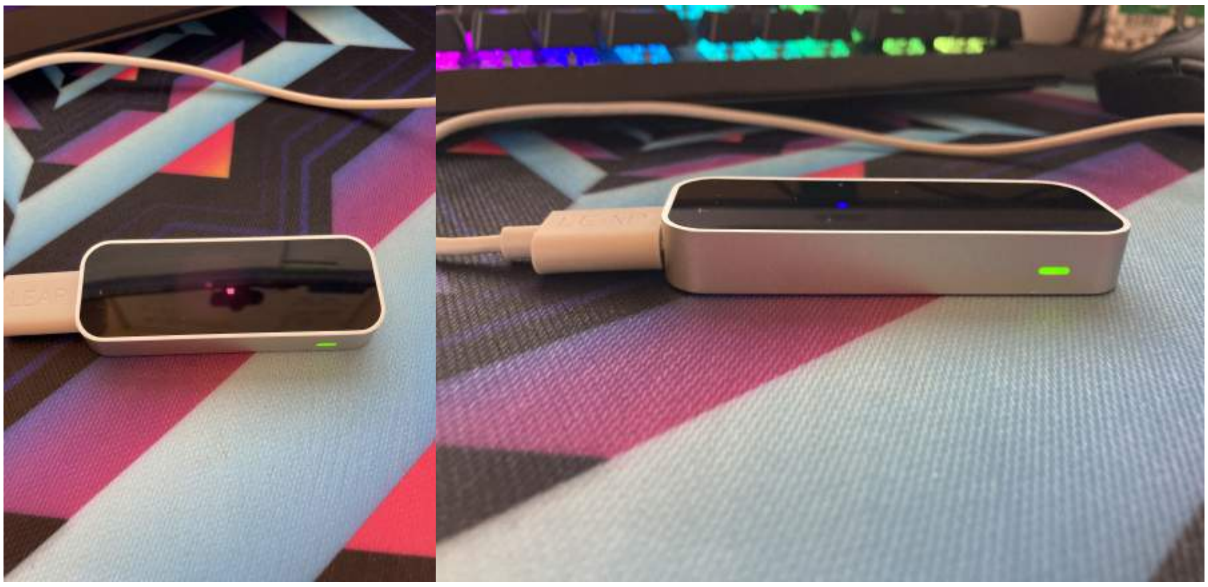
[2]Augmented Reality, Virtual Reality and Extended Reality.



No head-mounted display detected.
Tracking Mode: Desktop (Press 'V' to switch to head-mounted mode)
Camera Feed View: Overlay (Press 'C' to switch)
Render Framerate: 222.6 fps (4.5 ms)
Device Framerate: 122.0 fps (8.2 ms)
Active Device: Leap Motion Controller

Figure 9: My hand shown in the LeapMotion Visualisation Tool

LeapMotion advertise their product as "Small. Fast. Accurate. World-class hand tracking for anyone, anywhere" LeapMotion (nd).
"The controller is capable of tracking hands within a 3D interactive zone that extends up to 60cm (24") or more, extending from the device in a 140x120° field of view"(LeapMotion, nd). Leap Motion‘s software is able to discern 27 distinct hand elements, including bones and joints, and track them even when they are obscured by other parts of the hand. It can also detect at a nominal speed of 120hz (LeapMotion, nd).



(a) (b)

Figure 10: The LeapMotion Controller LeapMotion (nd)

3. Context

Before starting my project I decided to spend a lot of time researching the needs of the users, the project's requirements and possible limitations. During this time I also started to understand my motivations towards this project and why I felt driven to complete it. In the following sections I will be explaining the reasons and causes for certain choices and problems that I may encounter, and what their mitigations may be.

3.1. Motivation

Communication has always been difficult for me, especially written – due to my dyslexia. Trying to convey information or language has always been a challenge. However, as I see challenge as an opportunity rather than a barrier I have always tried to find new ways to overcome this difficulty. After learning about the struggles of people that rely on BSL as their main communication method I felt motivated to help give those people a new way to relay information.

SignSolutions (2021) state that approximately 157,000 people use Sign Language, there are 68.5 million people in the UK (Worldometers , 2022) (as of May 2022) which therefore means that BSL users are limited to the amount of people they can communicate with. My goal is to develop my project in such a way that allows for an increase in that number by allowing non-signers to understand signers.

3.2. Approach

I decided to use the development paradigm of Agile as I have be treated my project as a Software Development project. I have be created diagrams, charts and graphs along the way to aid my development and future maintenance of this piece. This will allow me to break my project up into small manageable chunks to ease development and lessen the potential of feeling overwhelmed with a large amount to do all at once. It would also allow me to keep a detailed log of all the progress I have made, and what I would need to do next. I will be keeping a log of all the steps I take in a Google Docs [3]. I will mark down the date, and the action I have done that day. I will be sticking to the general time frame that I have laid out in my Gantt chart, see Section 4.

3.3. Objectives

The objectives of this project are to assist the user in real-time translation of BSL into written English. This would be done by using the LeapMotion Hand Tracker to interpret gestures using UnrealEngine4 with the LeapSDK. The design of this project is simple and does not require any specific conditions – aside from fair lighting. The computation behind the actual interpretation would be completed in UrealEngine4 using the LeapMotion SDK in C++. The use of blueprints and C++ in UnrealEngine4 (see Section 16) makes it a very powerful tool for visual development. I felt it was important for the user to have an accurate representation of their hands in a 3D space while utilising this tool. Fortunately the LeapMotion Hand Tracker has a well defined API[4] for C/C++ integration.

[3]A word processing application.

[4]Application Programming Interface.

3.4. Risks

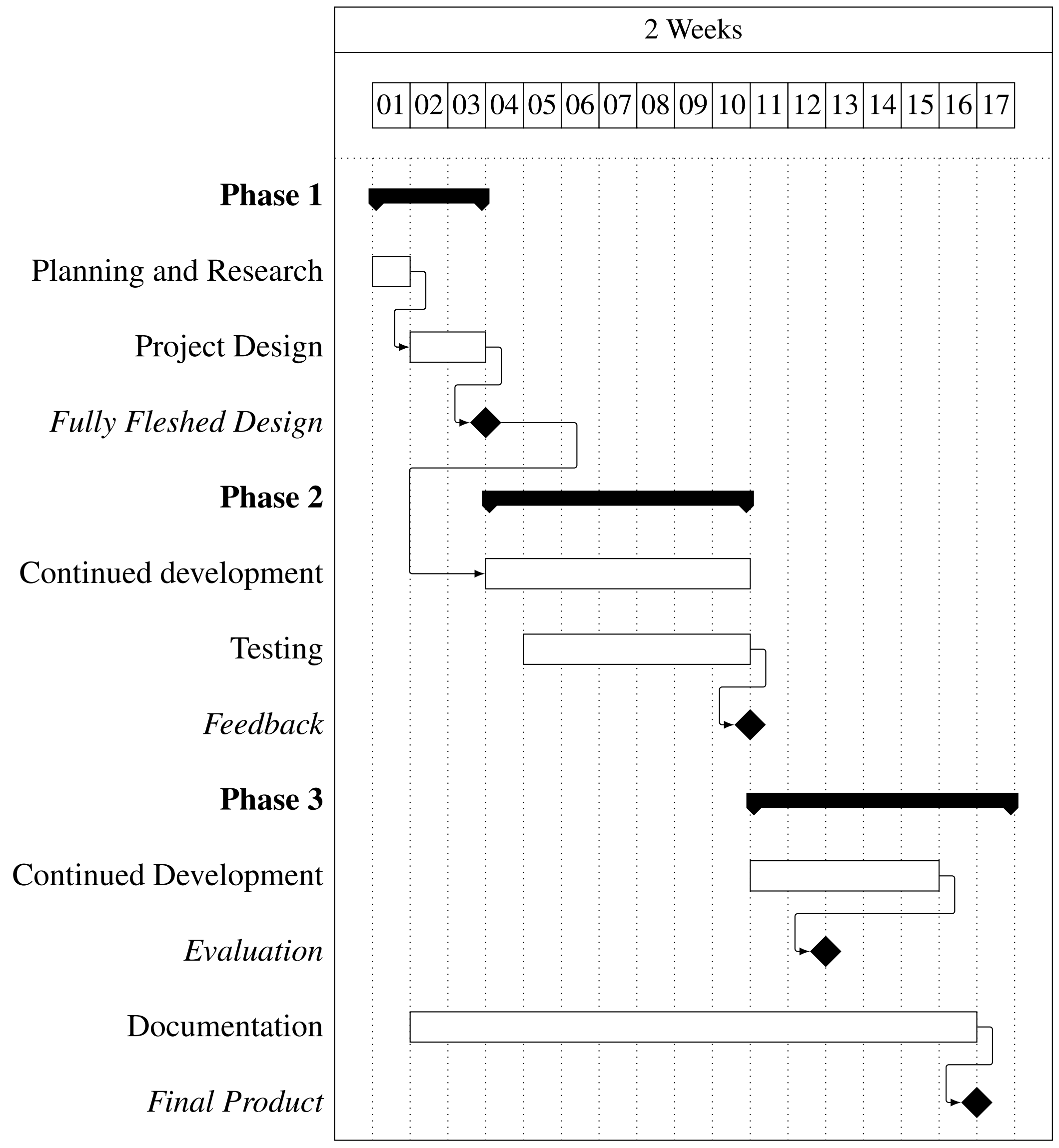
The main risk of my project is that I am heavily relying on the third-party motion sensor to complete the task. If the sensor breaks, has limitations, reliability issues or bugs then my final product is at risk of failure – which is out of my control.
Unfortunately, there is no way to mitigate this issue unless I use another sensor. However, as this project is dedicated to using the LeapMotion Hand Tracker it cannot be avoided.
Another risk of my project is that the solution may provide an inaccurate translation of the gesture given. This could lead to miscommunication if more than one party is using this product.

3.5. Limitations

A large limiting factor of my project is the hardware. Leap Motion develop a fantastic hand tracking kit, useful for developing games and general motion tracking. It works in real-time and is very lightweight. However, as this is a single view tracker there are some immediate limitations. Such as; the inability to work in a true 3D space, loss of tracking when a hand is behind an object and lighting/reflection sensitivity. A solution to all of these problems would be to have two or more trackers working at the same time in conjunction. Unfortunately the Leap Motion API does not support more than one tracker running simultaneously – so this cannot be done without major changes to their API, which is out of scope for this project.

Another limitation of the project is that the program can only recognise gestures that is has been previously taught. This could cause issues if the user wanted to translate a sign/gesture that had not yet been classified, as there would not be a translation to give. The program must be trained and labeled before an accurate translation can occur.

4. Gantt chart



5. Design

5.1. User Experience Design

I want the user to have a very easy time whilst using my program. So I will plan out everything so that there is as little interaction and setup as possible. This will allow the user to have a much smoother experience and a much more enjoyable session. I also intend to create a small delay in the tracking between gestures. This will stop the program assuming information before the user has finished their gesture. This time delay will be 3.4 seconds[5], as this has proven to be suitable without discomfort or the feeling of slowness.

As this project focuses on accessibility, I will be taking extra precautions to make sure that all of my fonts, pictures and assets are readable, viewable and as simple as possible. As this is a visual project, I will be making sure that all the colours I use can be seen by people of all sight types, this includes colour blindness. I will achieve this by using Primary colours and lots of contrast. This will help those that have issues reading as well as seeing to understand the contents of their screen more clearly.

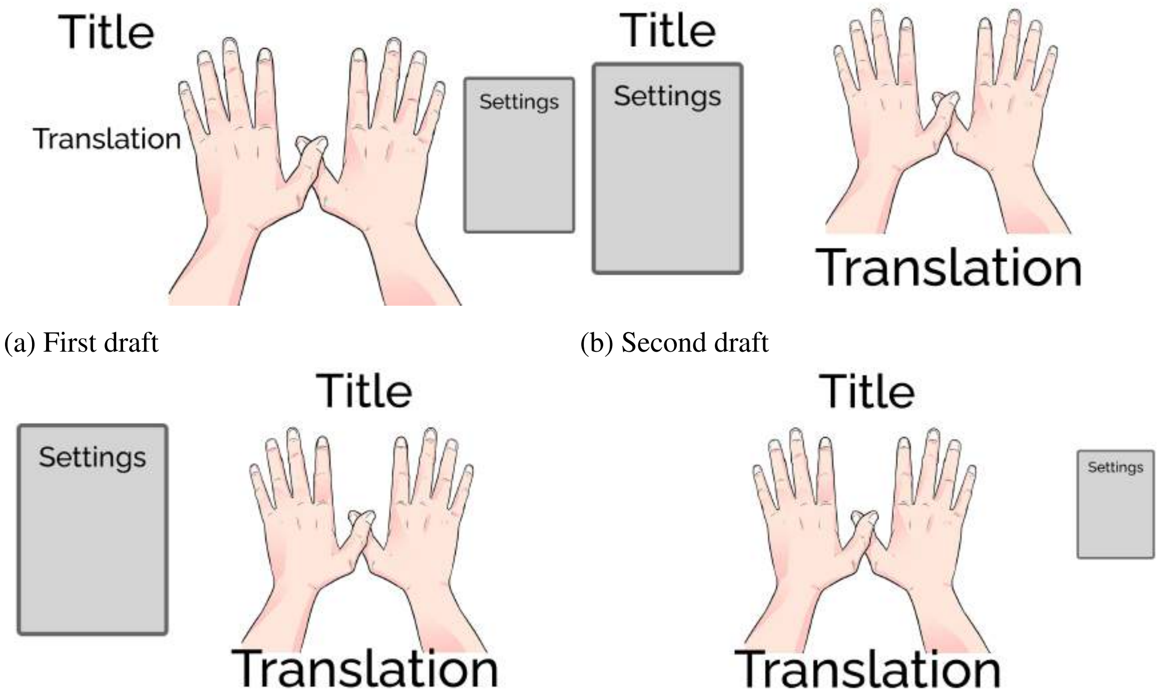
I would also like to implement an AutoFilling NLP[6] System. This will act as a predictive text for the user – reducing the number of signs/gestures that they need to do. I feel that this is not an essential part of the project, but it would definitely add to the user experience. As this is not a key feature, I will not be planning it into my GUI design, as explained in the section below.

5.2. Graphical Design

In the following sections I will be explaining the thought process behind my overall GUI[7] Design and structure. I have completed a few Lo-Fi designs before starting my final design plan.

[5]Found through empirical testing.
[6]Natural Language Processing.
[7]Graphical User Interface.

5.2.1. Lo-Fi



(a) First draft (b) Second draft

(c) Third draft (d) Final draft

Figure 11: Lo-Fi Designs

I decided to draft some designs before selecting my final structure and UI[8] layout. I did this by creating some Lo-Fi mock-ups in an image editing software. I will use the Final Draft (Figure 11) during my full design mock-up, where I will make a higher quality design. As explained in User Experience (Section 5.1), I have focused the accessibility of the information. Making sure that all of the information is readable and in a clear position. During the Lo-Fi I have only focused on the positions of the features, the colours, sizes and names, which may be changed in a final design.

5.2.2. Hi-Fi

After completing my Lo-Fi designs I am now able to create the full GUI mock-up, I have done so using Adobe XD (Adobe, nd) as it is a professional standard for UI and

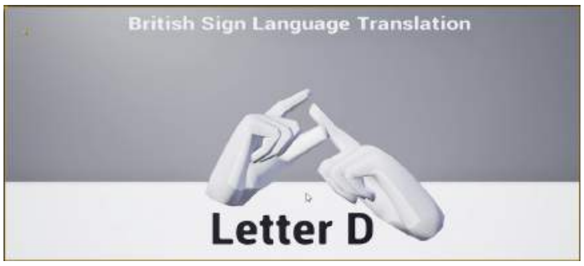
[8]User Interface

UX[9] development. I have decided to keep the hands in the centre of the screen and have the translation at the bottom of the screen. I have done this to comply with the Apple Human Interface Guidelines (Apple, nd). This will keep the users attention on the main aspects of the program - their hands and the translation. The title will stay at the top as a static text widget, and the settings window at the side. Both of these are not as important and so can be out of the main view. The settings window has been renamed to "Options" and been made slightly bigger than my Final Lo-Fi (Figure 11).

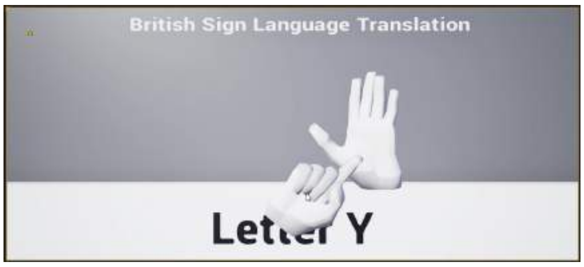
This is the general idea I am trying to achieve for my overall UI. This will give the user lots of options on the side, whilst still giving them lots of screen space for the translation and sign language. Their hands will be the main section of the screen taking up around 80%. The translation bar at the bottom is just the output and will display the user's translation in plain written English. This will take up around 10% of the screen, the other 10% will be the options and other information displayed on the screen at the time of translation.

I feel this general UI design (Figure 22) will give the best user experience as it provides a clear-cut area for each feature. It is also intuitive as there is no need to explain each area – there will be titles and examples throughout usage. The "Translation" area will have a real-time updating piece of text to show the user what sign they are doing. The hands will also be shown in real-time to help the user understand the orientation of their signs in the program, and so they can visualise the development of their words/sentences.

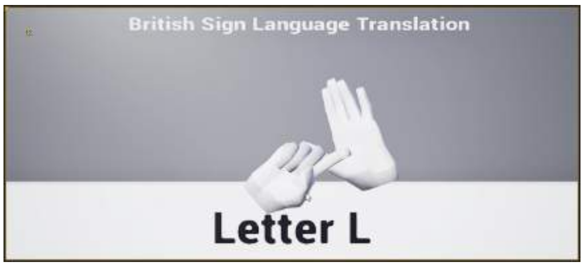
[9]User Experience



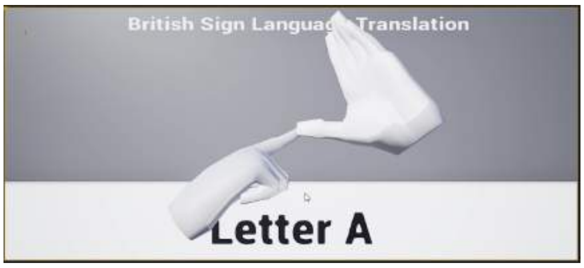
(a) Letter D



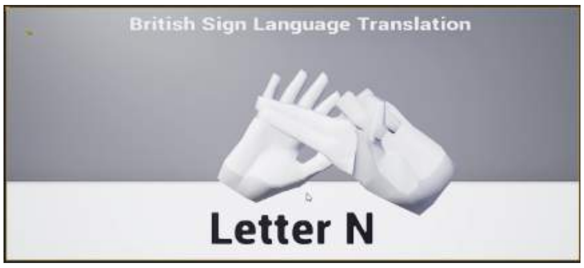
(b) Letter Y



(c) Letter L



(d) Letter A



(e) Letter N

Figure 12: The letters of my name.

5.3. MoSCoW Analysis

After completing the Similar Systems Analysis I can now produce a MoSCoW Analysis. This will help me to determine the key features of the project. It will also help me to determine what is inside the scope, and what is outside the scope of my project. Knowing where features fall in terms of scope is vital to prioritisation and time management.
Must:

- The user must be able to translate their gestures into written English accurately.
- The user must be able to translate numbers and letters, as well as words and phrases.

Should:

- The user should be able to view their hands in the 3D environment. This is to assist the user in knowing the orientation of their hands.
- The user must be able to use the program from their own device (requiring the LeapMotion Tracker)

Could:

- The user could translate their gestures into other languages, eg French, Spanish or Manderin.
- The user could gesture using other Sign Language variants. Such as American Sign Language or Greek Sign Language.
- The user could be able to see a rigged character model mirror their movements. To check that the program received their gesture correctly.

Won't:

- The user will not be able to change the font of the program. This is a feature that I wanted to add, but as far as I'm aware, it is not feasible in UnrealEngine.
- The user will not be able to learn sign language from the rigged character. This is outside the scope of the project, however it is a feature that I feel would be very beneficial in this type of program.

5.4. Use Case Diagram

After completing the MoSCoW Analysis, I am now able to create the Use Case Diagram. This will show how the user and the program will interact with each other. see figure 13.

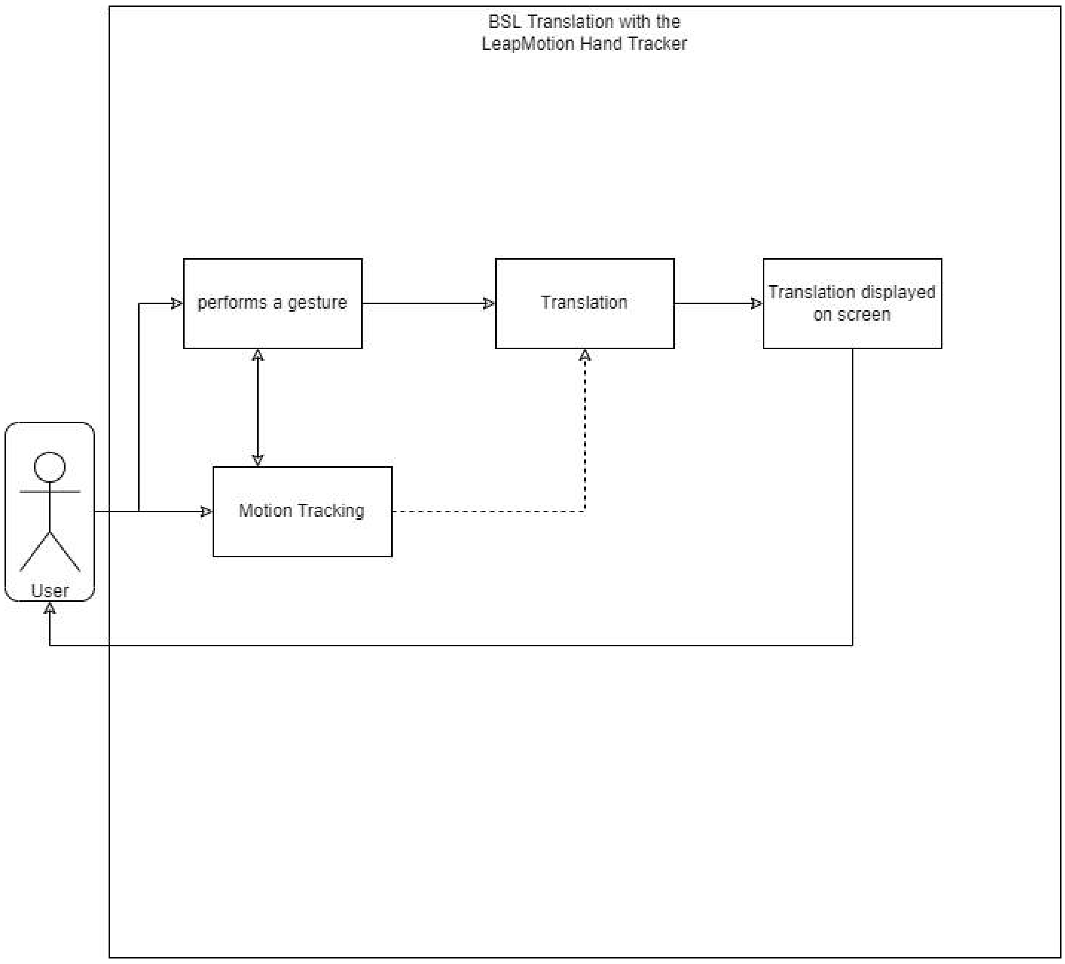
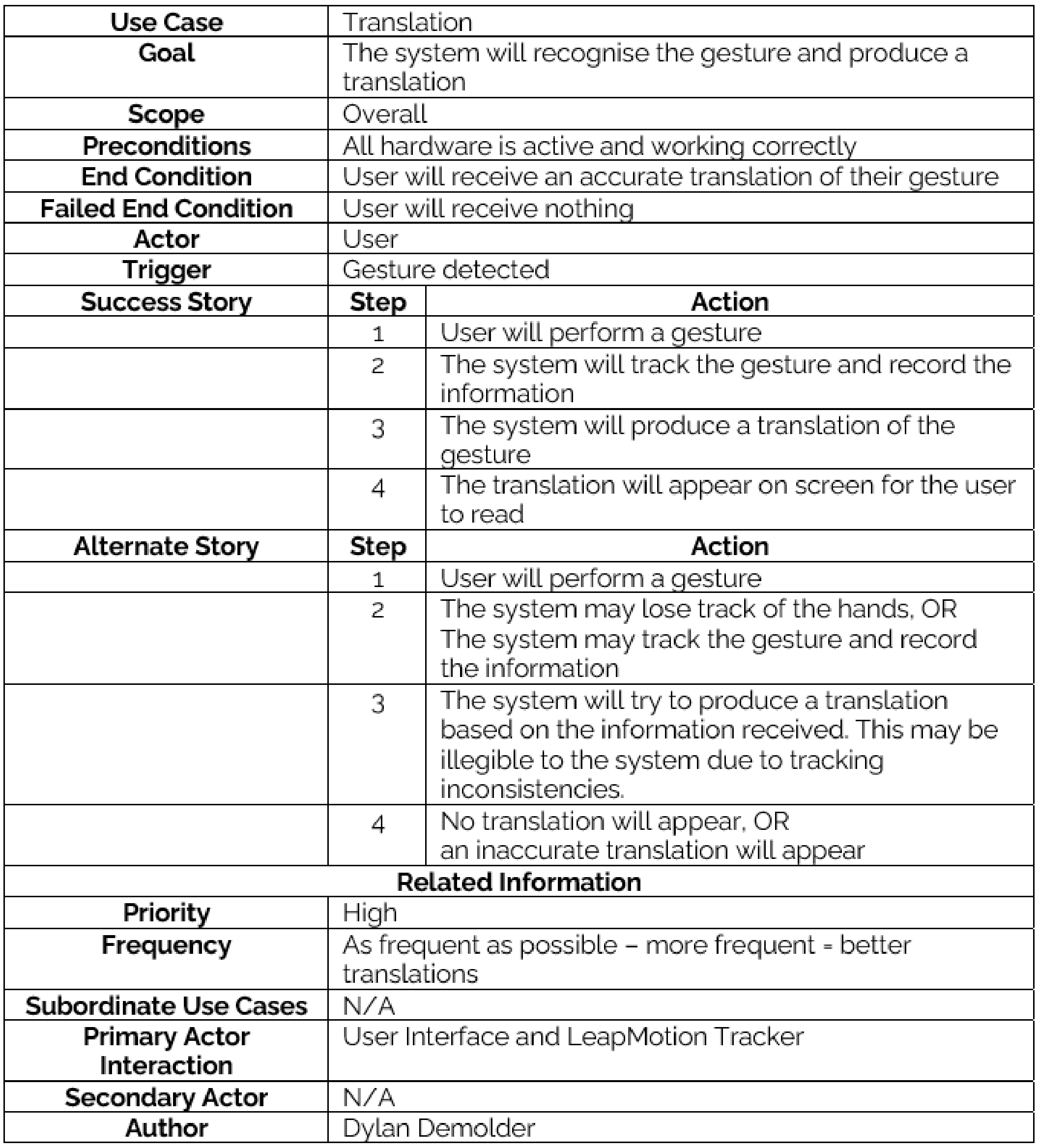

BSL Translation with the LeapMotion Hand Tracker
performs a gesture
Translation
Translation displayed on screen
Motion Tracking
User

Figure 13: A Use Case Diagram

5.5. Use Case Description

The following is the Use Case Description (Figure 14):



Use Case	Translation	
Goal	The system will recognise the gesture and produce a translation	
Scope	Overall	
Preconditions	All hardware is active and working correctly	
End Condition	User will receive an accurate translation of their gesture	
Failed End Condition	User will receive nothing	
Actor	User	
Trigger	Gesture detected	
Success Story	**Step**	**Action**
	1	User will perform a gesture
	2	The system will track the gesture and record the information
	3	The system will produce a translation of the gesture
	4	The translation will appear on screen for the user to read
Alternate Story	**Step**	**Action**
	1	User will perform a gesture
	2	The system may lose track of the hands, OR The system may track the gesture and record the information
	3	The system will try to produce a translation based on the information received. This may be illegible to the system due to tracking inconsistencies.
	4	No translation will appear, OR an inaccurate translation will appear
Related Information		
Priority	High	
Frequency	As frequent as possible – more frequent = better translations	
Subordinate Use Cases	N/A	
Primary Actor Interaction	User Interface and LeapMotion Tracker	
Secondary Actor	N/A	
Author	Dylan Demolder	

Figure 14:

5.6. Sequence Diagram

After completing the Use Case Diagram (Section 5.4) and Descriptions (Section 5.5) I have been able to create a Sequence Diagram. This diagram (Figure 15) shows the general structure of user throughput and how the user will interact with the program.

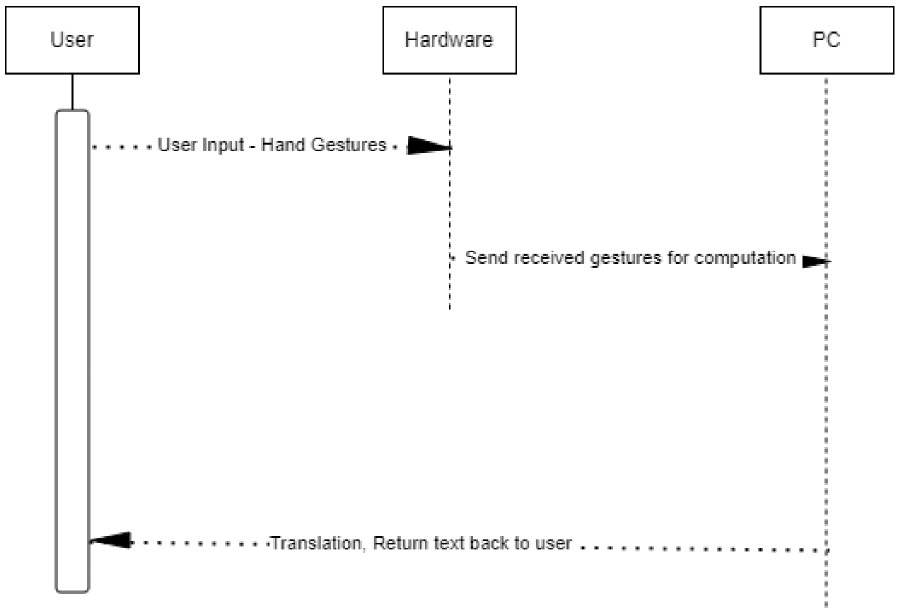

User
Hardware
PC
User Input - Hand Gestures
Send received gestures for computation
Translation, Return text back to user

Figure 15: The Sequence Diagram

6. Implementation

In the following sections I will be explaining the development and thought process behind this project. Throughout there will be changes of ideas from what I have discovered to work and when other ideas have failed/not produced the right outcome. I will be explaining my initial plan, the general structure that I would like to follow and each development stage of this project. I will be referring to my notebook where I have kept a log of my day-by-day activities related to this project.

6.1. Plan

I chose to use Unreal Engine 4 (UE4) as I wanted this project to be primarily visual. I thought that a game engine would be the best place to support that. UE4 will allow me to create real-time interaction from the user to visual representations on screen. I can then use these representations to interprete gestures and sign language. These gestures will then be compared to a database/ML structure to translate them into written English.

I will be able to use the blueprints in UE4 to create most of the logic for this project. However, I envision that I will have to use C++ for some of the more specific features of my project, as this engine was not built for this specific task. I will also be able to use the LeapMotion plugin for UE4, as well as the LeapC API to better track the hands and convert the user movements into actions to trigger events, eg translation.

I will be using this to develop a form of API that can extract the user hand data (position, finger data, roll, pitch, yaw and velocity) for each hand. From here it can communicate with a database that has information about all the hand positions and can reply with a translation. I have chosen to develop the project in this way so that It can be used to translate other sign languages, such as American Sign Language (ASL) or Greek Sign Language (GSL). This will allow the project to expand past the scope of my target. And it also gives a good basis for learning more complex signs later in the development.

I also have the idea for a machine learning (ML) data set using reinforcement learning. As this will allow the users to teach the database new words and correct ones that may have been wrong. The addition of ML would also dramatically increase the scalability of this project. As it would allow users to teach it new or different sign language variants that may have not been supported before.

Creating a Machine Learning algorithm is outside of the original scope of the project, and so I do not intend to build it. However, I do believe that it would be a very useful asset to the program. This will be a secondary part to the project - as there are other methods to complete the same task.

6.2. Structure

My intention for the development of this code is to modify the current LeapC files to be more representative of what I need to use them for. I will also be adding new features to this plugin so that I can more easily and accurately determine the position and orientation of the hands. These modifications will include the ability to determine the X, Y, Z coordinates of each finger joint, their quaternization (Section 7.3) and their velocities. With this information I will be able to cross reference the data to a database of predefined positions/gestures that will trigger an event – in my case translation.

```
void FLeapPalmData::TranslatePalm(const FVector&
   InTranslation)
{
 Position += InTranslation;
 StabilizedPosition += InTranslation;
 Velocity += InTranslation;
}
```

The above code gathers the Vector of the palm

I will be making modifications to the "UltraleapTrackingData.h" file. This is where the bulk of hand tracking computing happens. This is also where the LeapC library interacts with the LeapMotionUE plugin. Modifying this file will allow the plugin to work as originally intended, but with the addition of my new functionality. Each of the new functions that I add can be used as blueprints in the UnrealEngine editor – allowing my code to be objective and modular.

6.3. Development

In the following section I will describe the tasks and decisions that I have undertaken and made during the development of this project. This will include the resources I have used, such as software and libraries.

6.3.1. UnrealEngine4

I chose to use UE4 as it provided a lot of visual elements that I wanted to include in my project such as 3D environments, shaders, dynamic bone rigging and events management (through the use of blueprints). All of these features are ones that I felt were essential to the user experience – but not necessarily the project goal. Utilising these features with a blend of added elements from the LeapMotionUE plugin I was able to create a real-time representation of the users' hands in the 3D space that I had created. From this point I was able to change the scale of the hands and of the environment so that I could have a user space. This user space is where all of the user interaction and translation will take place. Keeping user interaction (outside of the BSL) to a minimum is essential for a good user experience (CAROLINE WHITE , 2021).

UnrealEngine also has a great Bone Rigging Feature (UnrealEngine , 2022). I plan to utilise this along-side LeapC to create a true hand bone mesh. This mesh will be the basis of the hand data. There is already a hand data feature inside LeapC. However, it does not contain all the information that I would want to utilise during the translation. I will need the 3D vectored location of each joint, which I explain more in section 7.3. I can alter this function by modifying the LeapMotionUE plugin, or by creating my own function (Section: 7).

6.3.2. LeapC

LeapC is the LeapMotion C library and API. This is what allows the LeapMotion controller to communicate with the other programs on the PC – such as the LeapMotionUE plugin, I will be using LeapC to create and modify assets in the plugin. I will be doing this because while the plugin does provide many useful features, there are some specific use cases where I will be needing unique inputs and processes to determine the gesture the user is doing. This will be the basis of my gesture management system. I can have a map of gestures and words/phrases/numbers. This map will be the overarching database for my system.

The LeapC library can be attached to any C/C++ program and used with ease. The documentation is very detailed and even provides some examples of the features it has and what they can do. This has become very helpful over the course of this project. This library will be the basis of all sign language translation throughout this project.

Without the LeapC library I would have had to recreate all of the data handling and hand tracking logic. This would have taken more time than the project provided. So, I am very grateful for the LeapMotion team for putting together this library.

6.3.3. InteractML

InteractML (2020) is powerful Machine Learning system that can be used within UnrealEngine and utilises Blueprints. This plugin provides three Machine Learning algorithms - Classification, Regression and Dynamic Time-Warping. By simply recording the input parameters the system can be used to teach the model attributes and their properties. This plugin is very modular and a large array of data-types can be fed into the learning algorithm. I have used this Classification model to record and learn gestures within the project. I have been able to feed in the bone and joint data (Figure 18) of the hands (LeapMotion, nd) to summarise position data into letters of the alphabet. I have however only implemented my name into the project as it uses the k-NN (k-Nearest Neighbour) classifier - this classifier requires a large amount of training data to be provided for each classifiable label which is out of scope for this project.

Using the blueprints system, I have been able to train the letters D, Y, L, A, N and "Nothing"[10]. I have trained each letter three times with a standardised approach of using the 'R' Blueprint which toggles the "Recording" boolean, recording data for three seconds - which is later utilised as 'training data'.

6.3.4. Engine Variants

After researching various types of game engine that the LeapC Library supports, I originally chose to use UnrealEngine4 25.6. At the time (as of 22/10/21) this was the latest version that the LeapMotionUE plugin support. This would give me a plethora of tools and utilities that I had planned to use throughout the project. After adding the plugin to my project I was able to start the development of my first stage. Over time whilst experimenting with the tools provided by the plugin, I encountered some issues. After researching I discovered that they were issues with the current version of the plugin. One of the issues meant that I was unable to build the project to produce a working program. I later learnt that UnrealEngine does not include a C builder as standard. I was able to solve this issue by adding Cmake into the project. Fortunately this issue and many more were solved in the next release of the plugin. However, it did not support the version of UnrealEngine I was using. I made the decision to remake my progress in a

[10]No data, used as a baseline.

supported version (4.27). This did mean I had to reproduce my assets, but as they were already made I was lucky enough to port them over without wasting too much time.

6.3.5. Integrated Development Environment's (IDE)

UnrealEngine has support for many IDE's inbuilt already. I had to make a choice for which I preferred and which had the most features I would be using.

Originally I chose to use CLion as I am familiar with the JetBrains environment. However, after researching the other IDE's available to UnrealEngine I tested out ReSharper - another JetBrains IDE. ReSharper has an UnrealEngine Plugin that allows for faster development, debugging and predictive autofill. But after seeing that I would be unable to build the project using ReSharper I made the switch to Visual Studio.

Visual Studio has the ability to Build and Compile the project within the IDE. This has proved to be very helpful throughout the development. This would be in tandem with the UnrealEngine's ability to "Hot Load"[11] code changes. These two features working together allowed me to build and test code on-the-fly within the software - saving hours of time in repeated building, which can be quite time consuming.

Visual Studio has allowed me to open, edit and build the C++ files within UnrealEngine, and the LeapMotionUE Plugin that I have added to my project. With these edits I can make some significant progress on the development stages of the project. So for the remainder of this project I will be using the Visual Studio IDE - with the UnrealVS Extension (UnrealEngine, nd).

6.3.6. Blueprint Visual Scripting/C++

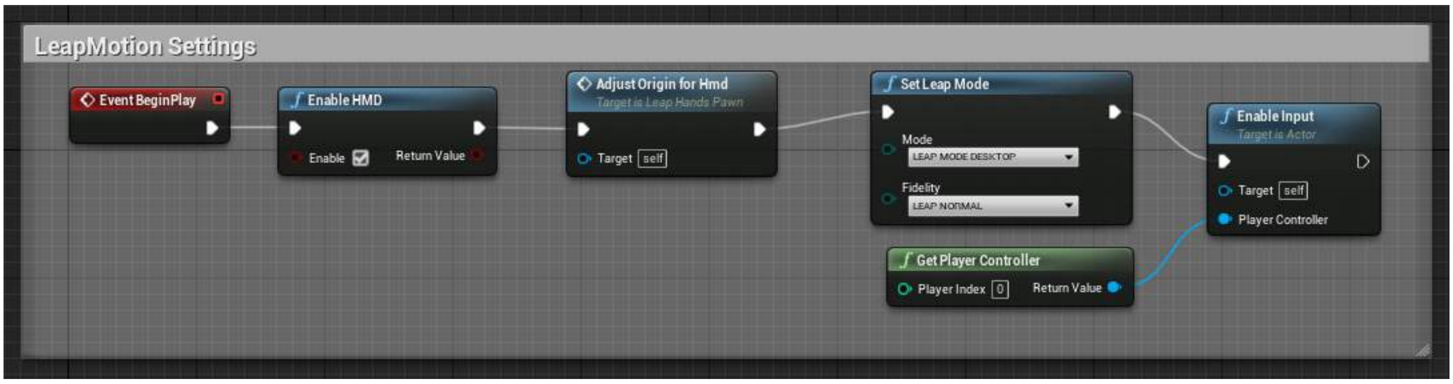
The Blueprint Visual Scripting system in Unreal Engine is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within 'Unreal Editor'. As with many common scripting languages, it is used to define object-oriented classes or objects in the engine. As you use UE4, you'll often find that objects defined using 'Blueprint' are colloquially referred to as

[11]Interchange of code without recompilation.

just "Blueprints" (Unreal Engine, nd).

A huge advantage to using UnrealEngine is the ability to utilise Blueprints with C++. These both provide a massive amount of customizability in logic and features. This is an essential feature to a game engine and so reinforced the reason I chose to use UnrealEngine. Being able to use C++ has given me complete flexibility with the logic, as now I no longer need to rely on the pre-built logic inside the engine. This is where I will be able to utilise LeapC to its fullest capabilities.

From starting this project I did not have much C++ knowledge, however I took out a few weeks to learn the basics and do experiments with it inside UnrealEngine. This proved to be valuable when I focused on this project.



Mode: LEAP MODE DESKTOP
Fidelity: LEAP NORMAL

Figure 16: LeapMotion settings being changed on the launch of the Program, using a Blueprint

This blueprint (Figure 16) can interact directly to the LeapMotion Tracker through their API. It allows for the tracker and any supported software to communicate. I can use this blueprint to guarantee that the right settings are used each time the program is launched. It will force the Tracker into using the settings of:

Leap Mode Desktop: Tracking from a table top position

Leap Normal Fidelity: The accuracy and smoothing applied to the data

7. Methods

In this section I will be describing the various methods that I have utilise during the development of the project. I will explain encountered problems and reasons to why they occurred. I will also be explaining what I believe to be the best method to complete this project.

7.1. Unreal Events Managers

Unreal Engine has an inbuilt events management system (UnrealEngine , nd). Using this allowed me to create responses to certain actions happening inside the program with the help of blueprints. I was able to use this to create a function that would display messages based on certain events performed by the user. This is the basis of the other methods that I will be describing in the following sections. This method was not able to be used as the final version, as it has many limitations. One of which being that it can only be used for static[12] events. I cannot use this method to follow a gesture - only to find the final stage of a movement. However, it could be used for most letters and some numbers for BSL.

However, I decided that using a method that only worked for *some* gestures would be inefficient and not a good use of resources. This meant that I did not use this method in the final version of my project.

7.2. Position Matching

Using the latest version of the LeapC plugin, I was able to use their new Position and Gesture management system. At first I thought this feature would be able to solve a lot of the issues I was having at the time. However, after experimenting with the feature for a while I came to realise that it was not has helpful as I had originally thought. The gestures that it was expecting were not dynamic, and it did not have much room for error. This meant that, ultimately, it could not be used for BSL translation. The gestures needed for the translation would need to be dynamically recognised and some interpretation would need to be completed; neither of which are supported in the plugin (LeapMotion , nda).

[12]Non-dynamic, typically stationary.

7.3. Quaternions

Quaternions are mathematical operators that are used to rotate and stretch vectors (Figure 17). They provide the ability to notate the 3D position and orientation of an object; This can be used to create relative positioning in the 3D environment.

$$quaternion = a + bi + cj + dk$$

(a,b,c,d are real numbers and i,j,k are basic quaternion operators.)

This will create a bone mesh map, when used on the joints of a hand.

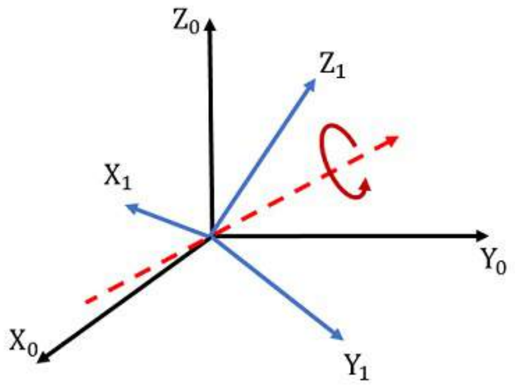


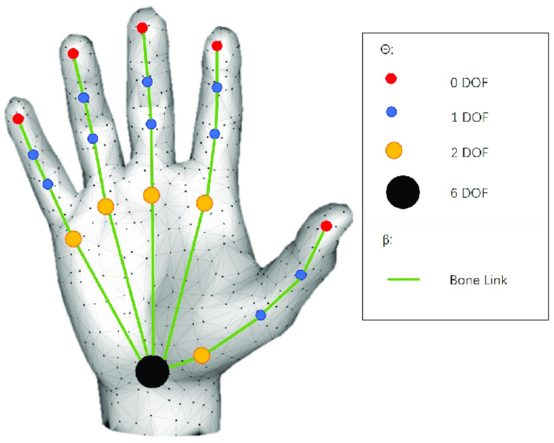
Figure 17: 3D Quaternion representation FABIO NELLI (2020)

There is good integration with LeapC for this method. This solution also matches up with my initial plan of creating a bone mesh map of the hands, allowing the program to simply check for numerical data to calculate the distances between fingers and joints. This would increase the efficiency of the program and create accurate results. This method can be utilised by using the inbuilt struct in LeapC:

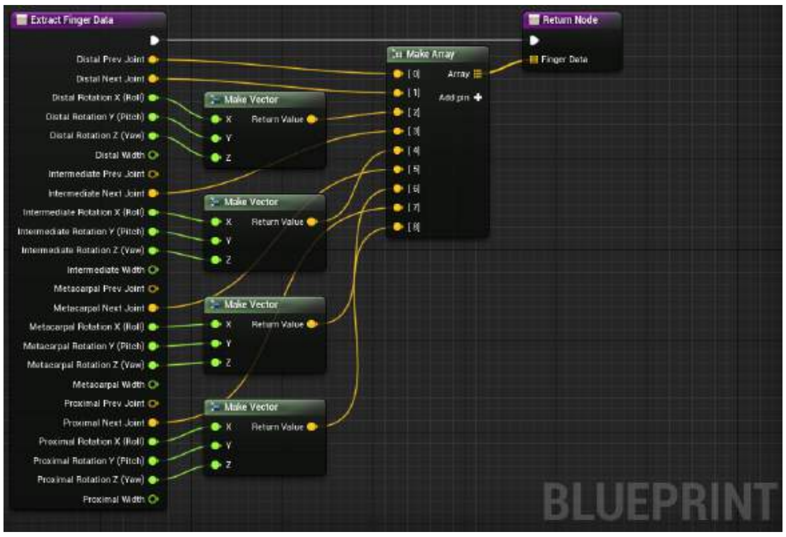
LEAP_QUATERNION: A four element, floating point quaternion. LeapMotion (nda)

Knowing the 3D positioning of the joints in the fingers (in a relative state), would be incredibly helpful when trying to create a gesture controller. It would allow for quick on-the-fly additions of gestures and possibly even the recording of gestures to be added through another program.

This method justifies the creation of a Machine Learning system. Using InteractML (Section 6.3.3) I have been able to achieve this system, in almost the exact way I planned for. I have been able to create a classification model that can be taught. It takes in the 3D positions of each bone joint, the rotation of each bone and the velocity of the hands (See Figure 18). This information is then processed by the classification model and can be detected after recording and learning.



(a) Hand Mesh showing bone and joint structure (Yunlong Che , 2020).



(b) Blueprint of all the hand tracking data made into an array of vectors.

Figure 18:

Whilst Machine Learning was out of the original scope of this project, it has created a huge opportunity for this project to become well rounded and dynamic. It has also allowed me to create a framework that supports the further development and training of other words and letters. With my current implementation, using InteractML, the system can learn any gesture and match it to a defined word/phrase or number (Figure 19).

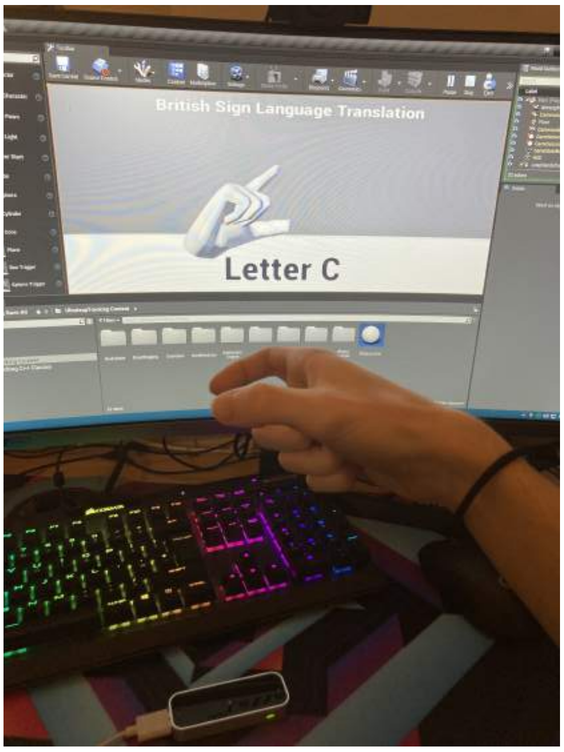


Figure 19: Using the translation program.

7.4. Process

Figure 23 shows a Flowchart for the process of the final project, I will then go on to explain the stages in more detail. These are the steps that the user will take to utilise the program. The Flowchart has been generalised for simplicity, but all of the key aspects are contained within the chart.

7.4.1. Start

Upon starting the program there will be a prompt for the user to select the mode they wish to use.

The three options available will be "Translate", "Learn" and "Teach". The "Translate" option will allow the user to translate their gestures from BSL to written English. The "Learn" option will let the user be taught sign language from a rigged character in the program, the user will then be asked to replicate the gesture and given a score based on their performance. The final option, "Teach", will utilise the previously mentioned machine learning system, which as concluded is out of scope for this project. This feature is the basis for the longevity of the project, and will also support its growth into other Sign Language variants - and possibly other written languages too (only the "Translate" option will be available at this time. Other options may become available depending on project progress).

The user will then be greeted by a screen explaining the features and options available to them for their selection. From here the user will be able to then use their LeapMotion Tracker to utilise the whole program.

7.4.2. Can see User Hands?

The program will then check to see if the user's hands are visible and able to be tracked. If at least one hand can be tracked then the gesture can be tracked and the translation can occur. Otherwise the program will keep checking for the hands each tick of the gameloop, provided by UnrealEngine.

7.4.3. Wait for gesture

The program will now track the hands, it will send the hand data to the gesture events manager - this is where the recognition will take place. The hand data includes the X, Y, Z coordinates of each joint, the angles between each joint in the fingers and the rotation of the wrist - I also intend to include the velocity of each joint so that moving gestures can be read as well (Explored in Section 7.3) - as there are few static gestures.

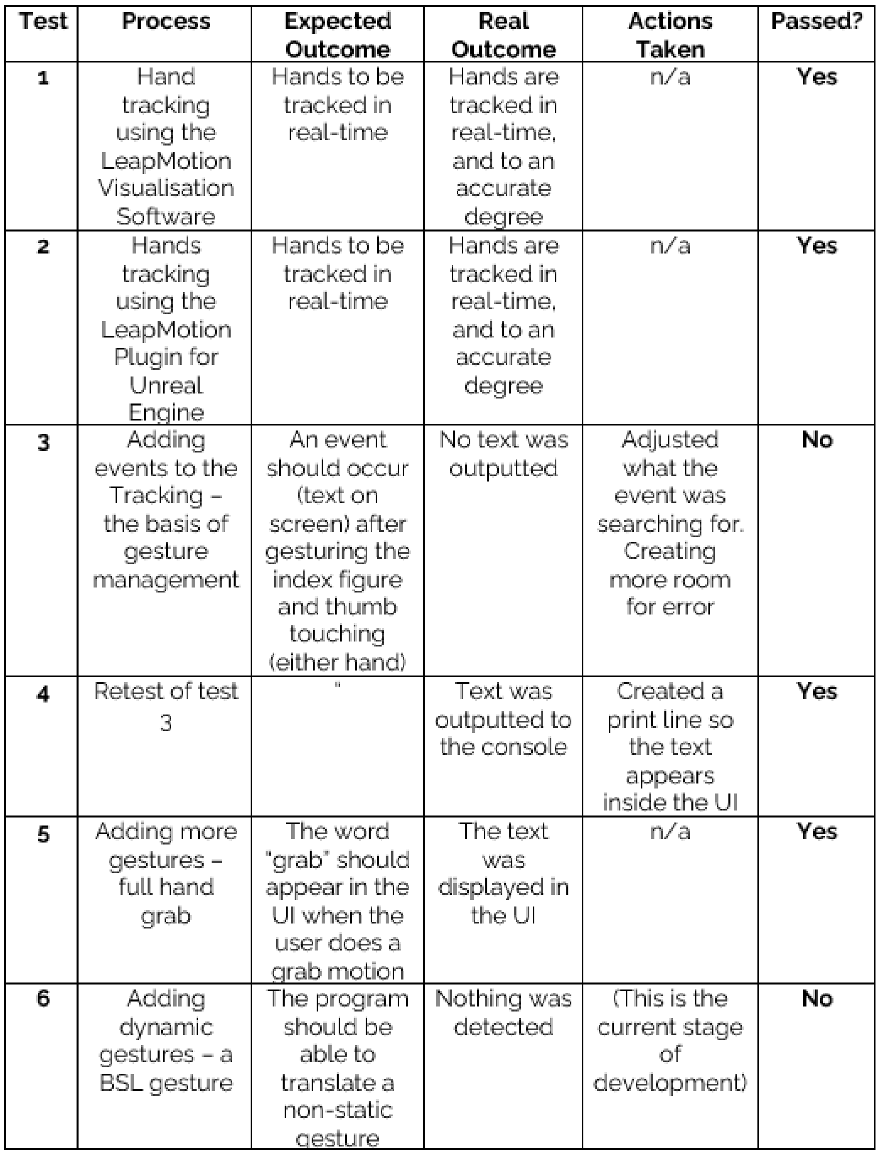
7.4.4. Recognise and Display

Once the hand data has been processed it can then be compared against a pre-recorded data set of gestures, possibly with an implementation of the k-NN or SVM[13] classifier. This is where the "translation" will occur. If no match is found then the program will not send any information back to the user. However, if a translation is found then the user will be prompted with the translation on their screen in English text. In the future I hope to include translation to other languages too, further expanding the use cases for this project.

7.5. Testing

This is the testing table (Figure 20). It shows the process being tested, the expected outcome, the real outcome and mitigation strategies taken to resolve incorrect results.

[13]k-Nearest Neighbour, Support Vector Machine.



Test	Process	Expected Outcome	Real Outcome	Actions Taken	Passed?
1	Hand tracking using the LeapMotion Visualisation Software	Hands to be tracked in real-time	Hands are tracked in real-time, and to an accurate degree	n/a	**Yes**
2	Hands tracking using the LeapMotion Plugin for Unreal Engine	Hands to be tracked in real-time	Hands are tracked in real-time, and to an accurate degree	n/a	**Yes**
3	Adding events to the Tracking – the basis of gesture management	An event should occur (text on screen) after gesturing the index figure and thumb touching (either hand)	No text was outputted	Adjusted what the event was searching for. Creating more room for error	**No**
4	Retest of test 3	"	Text was outputted to the console	Created a print line so the text appears inside the UI	**Yes**
5	Adding more gestures – full hand grab	The word "grab" should appear in the UI when the user does a grab motion	The text was displayed in the UI	n/a	**Yes**
6	Adding dynamic gestures – a BSL gesture	The program should be able to translate a non-static gesture	Nothing was detected	(This is the current stage of development)	**No**

Figure 20:

8. Outcomes

The outcomes of my testing show that I need to actively continue the development of my project to create a larger data set and reliable gestures. I need to feed more training data into my dynamic gesture manager, this will produce more accurate and representative translations. Without the extra data there cannot be a reliable and deterministic response to any of the gestures made in the program. I plan to implement a more automated training feature to further enhance the project. I have been trying many methods (7) to solve the issue of translation and gesture management - with varied success.

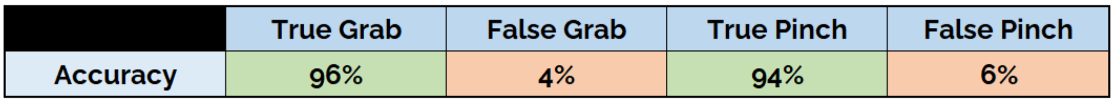
The hand tracking and visualisation works really well in UnrealEngine. I'm very pleased with the outcome of this part of the project. The user will have an accurate representation of their hands in the 3D environment and they will be able to determine the distance and shape of their hands in this virtualization. This is due to my implementation of the LeapMotion controller in UnrealEngine.

UnrealEngine has proven to be difficult to work with for this project. Looking back at my initial plan, I should've chosen to work in a language/framework that I am much more familiar with and one that has proven usage in this area. With hindsight I can say that using Python would have been a lot more productive. While I may not be able to create such a visually attractive program in Python, the base functionality would have been there - creating a more usable product. LeapMotion also have a Python library that can be used. This would save a lot of time in the development stage as most of the key functions have already been defined and tested (LeapMotion , ndb).

9. Analysis

Throughout the development of this project I have focused on utilising the format of a Software Engineering project.

I have done some analytical testing for the functionality of this project, on average there is a 95% accuracy of detection (Figure 21). Due to the small testing-set, the accuracy does not reflect the entire system, but it does give a general idea for how accurate other gestures would be once developed. I believe 95% to be within the area of human error so this is an accuracy I am happy to continue working with. This test was completed by completing a gesture 100 times, the program detected a gesture correctly for 95 of the 100. This test was orchestrated in a sub-optimal environment for the tracker (bright lights shining on the tracker and long sleeve shirt was worn), so a higher accuracy is certainly possible.



	True Grab	False Grab	True Pinch	False Pinch
Accuracy	96%	4%	94%	6%

Figure 21: Accuracy across tests of grabs and pinches.

From what I can tell, after many hours of research, there are no other projects that have taken on the idea of translating any Sign Language variant into any other language, making this project completely unique from any other. Due to my passion for this project, it will be able to assist the communication between any person, to any other person.

10. Conclusions

While this was an ambitious project, and one that needs more work, I am pleased with the progress I have made so far. I have enjoyed the development of this project, and I have learnt a lot of new skills - including some basic BSL, C++ and UnrealEngine functionality. In hindsight, the utilisation of Machine Learning over predefined code should have been utilised much sooner in my development.

In the future of this project I will reach out to the British Deaf Society (BDS) to get their thoughts on the program and to see what changes they may benefit from. With more time in the future I would love to continue the development of this project and reach out to the BDS and BSL users.

In the future, I will continue to utilise the LeapMotion, and I shall continue to research into changing over from UnrealEngine to Python. I am more familiar with Python, and so it I will be able to use my base knowledge to accelerate the development. Python also has access to lots of powerful AI and ML tools that would be very effective for the dynamic gesture manager - and also for training the program to understand more gestures outside of the predefined ones. However, a large draw back for choosing Python is how visual I want the final product to be. I would like the finished version of this project to be entirely visual and to run inside a game-like program. A major drawback for this option would be losing all of my current progress on the project. This is especially important to me as I am a visual learner and Sign Language is almost entirely visual. Even though I am more familiar with Python than I am with UnrealEngine and my progress would have been quicker had I used Python I have been able to make the project work to the same standard in UnrealEngine.

The Machine Learning system that I have made for the project has the ability to learn *any* gesture and assign any word /phrase or number to that gesture. This means that this project is not limited to either BSL or written English, but can be used for any Sign Language Variant and any written language. The system has the ability to, for example, translate BSL into German or Chinese. This means that a Signer would be able to communicate with any person given the right training data.

I am very pleased with the progress that I have made. While I may not have achieved everything that I set out to, I have learnt a lot during this development project and many of these newly learnt skills will be useful in the future. I have expanded my knowledge in many areas, such as UnrealEngine, C++ and the biology of hands. What I have made using the LeapMotion Tracker and UnrealEngine are the foundations for a world-class translation facility. The project will be able to expand to have multiple sign languages and written languages - allowing for any person to communicate with any other person (based on training data).

This report has also taken me a considerable amount of time more than I had initially expected. Being dyslexic, reading and writing has always been a struggle of mine, but I sought support from my advisors and staff and have managed to produce this report.

11. Summary of Conclusions

The development of this project has been extremely difficult, and I am very proud of the progress I have made. I have laid down the foundations for a world class technology that will facilitate the communication between any persons. I plan to continue the development of this project into the future and to complete it to my original ambitions and specification. I will research other methods of completion to accelerate the progress. I will also reach out to the British Deaf Society and the Institute of British Sign Language for ideas and accessibility methods. The development of this project has taught me a lot of new skills, and has also reinforced my current skills. Whilst I know there are areas for improvement, what I have created I can see as an achievement and overcoming a large obstacle.

References

CAROLINE WHITE (2021). Why Simplicity is so Incredibly Important in UX Design. Available at *https://careerfoundry.com/en/blog/ux-design/how-important-is-simplicity-in-ux-design/*.

Christian Zimmermann, Thomas Brox (2017). Learning to Estimate 3D Hand Pose from Single RGB Images. Available at *https://arxiv.org/pdf/1705.01389.pdf*.

FABIO NELLI (2020). Hamilton's quaternions and 3D rotation with Python. Available at *https://www.meccanismocomplesso.org/en/hamiltons-quaternions-and-3d-rotation-with-python/*.

Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, Christian Theobalt (n.d.). GANerated Hands for Real-Time 3D Hand Tracking from Monocular RGB . Available at *https://openaccess.thecvf.com/content_cvpr_2018/papers/Mueller_GANerated_Hands_for_CVPR_2018_paper.pdf*.

InteractML (2020). InteractML - UnrealEngine4. Available at *https://github.com/Interactml/iml-ue4*.

LeapMotion (n.d.a). Group Structs. Available at *https://docs.ultraleap.com/tracking-api/group/group___structs.html*.

LeapMotion (n.d.b). Setting Up a Project - Python. Available at *https://developer-archive.leapmotion.com/documentation/python/devguide/Project_Setup.html*.

Narut Soontranon, Supavadee Aramvith, Thanarat H. Chalidabhongse (2005). Improved Face and Hand Tracking for Sign Language Recognition . Available at *https://www.researchgate.net/publication/4141222_Improved_face_and_hand_tracking_for_sign_language_recognition*.

UnrealEngine (2022). Rigging with Control Rig. Available at *https://docs.unrealengine.com/5.0/en-US/rigging-with-control-rig-in-unreal-engine/*.

UnrealEngine (n.d.). Events. Available at *https://docs.unrealengine.com/4.26/en-US/ProgrammingAndScripting/Blueprints/UserGuide/Events*.

Valentin Bazarevsky, Fan Zhang (2019). On-Device, Real-Time Hand Tracking with MediaPipe. Available at *https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html*.

vidursatija (2019). BlazePalm, PyTorch & CoreML implementation of MediaPipe Hands. Available at *https://github.com/vidursatija/BlazePalm*.

Worldometers (2022). U.K. Population. Available at *https://www.worldometers.info/world-population/uk-population/*.

Yunlong Che (2020). Embedding Gesture Prior To Joint Shape Optimization Based Real-time 3D Hand Tracking. Available at *https://www.researchgate.net/figure/The-mesh-model-of-the-hand-and-corresponding-hierarchical-skeleton-The-skeleton-is-set_fig4_339315852*.

Adobe (n.d.). Adobe XD. Available at *https://www.adobe.com/uk/products/xd.html*.

Amie Harrison (2016). BRITISH SIGN LANGUAGE. Available at *https://www.linguisttraining.com/news/2016/2/10/british-sign-language*.

Apple (n.d.). Human Interface Guidelines. Available at *https://developer.apple.com/design/human-interface-guidelines/*.

British Sign (2013). BSL Finger Spelling. Available at *https://www.british-sign.co.uk/wp-content/uploads/2013/05/BSL-Fingerspelling-Right-Handed.png*.

British Sign (n.d.). What is British Sign Language? Available at *https://www.british-sign.co.uk/what-is-british-sign-language/*.

LeapMotion (n.d.). Leap Motion Controller. Available at *https://www.ultraleap.com/product/leap-motion-controller/* and *https://www.ultraleap.com/datasheets/Leap_Motion_Controller_Datasheet.pdf*.

Rodney Huddleston (n.d.). A SHORT OVERVIEW OF ENGLISH SYNTAX. Available at *http://www.lel.ed.ac.uk/grammar/overview.html*.

Russell S. Tomlin (1986). Basic word order. Functional principles. Available at *https://www.cambridge.org/core/services/aop-cambridge-core/content/view/7542AFB4A8B28D651F6E109B810F4C04/S0022226700011646a.pdf/russell-s-tomlin-basic-word-order-functional-principles-london-croom-helm-1986-pp-308.pdf*.

Sign Health (n.d.). Who uses British Sign Language (BSL)? Available at *https://signhealth.org.uk/resources/learn-about-deafness/british-sign-language-and-english/*.

Sign Non Manuals (n.d.). Segmentation of ÖGS-Texts. Available at *http://signnonmanuals.aau.at/en/node/661*.

SignSolutions (2021). What are the different types of sign language? Available at *https://www.signsolutions.uk.com/what-are-the-different-types-of-sign-language/*.

UCL (n.d.). Are there accents in bsl? Available at *https://www.ucl.ac.uk/dcal/sign-language-facts/faqs/bsl/are-there-accents-bsl*.

Unreal Engine (n.d.). Blueprints Visual Scripting. Available at *https://docs.unrealengine.com/5.0/en-US/blueprints-visual-scripting-in-unreal-engine/*.

UnrealEngine (n.d.). UnrealVS Extension. Available at *https://docs.unrealengine.com/4.27/en-US/ProductionPipelines/DevelopmentSetup/VisualStudioSetup/UnrealVS/*.

A. User Interface

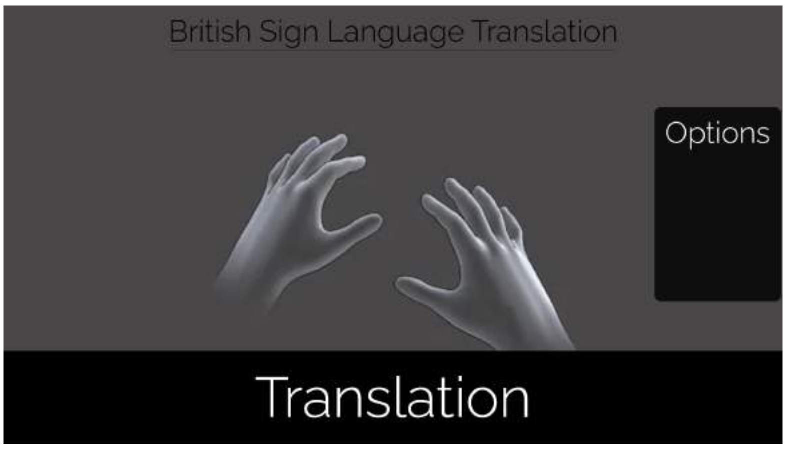


Figure 22: Graphical Design.

B. Flowchart

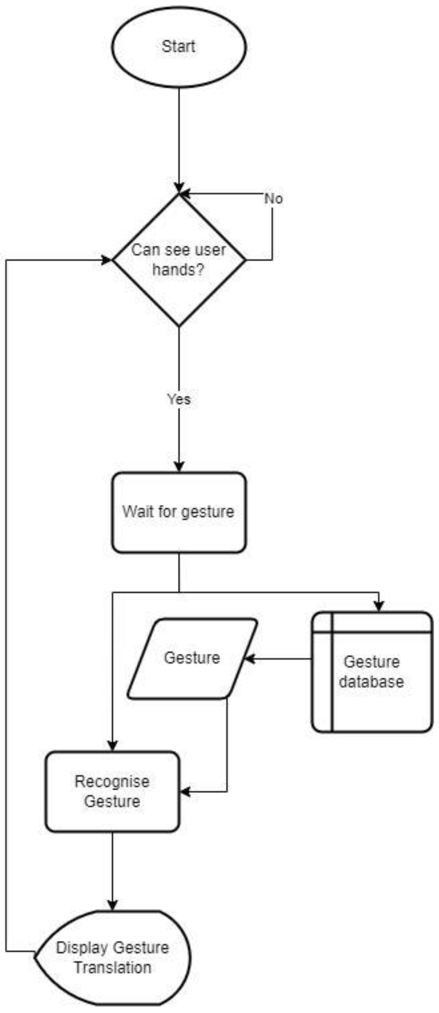


Figure 23: Flowchart of the program.