

Kyro: A Text to Sign Language Translator

Charles Patrick Lyttle – H00318301

Final Year Dissertation
Computer Science BSc (Hons)

Supervised by Alistair McConnell




Heriot-Watt University
School of Mathematics and Computer Sciences
Department of Computer Science

April 2023

Declaration

I, Charles Lyttle, confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: 

Date: 26/04/23

Abstract

Over 430 million individuals worldwide suffer from incapacitating hearing issues. (*Vaughan, 2021*) Many of these people receive education that is of subpar quality and, as such, hearing-impaired literacy rates are much lower than the average. The acute absence of deaf inclusive educational resources is one of the causes of this.

This paper presents Kyro: A system that converts any text input into Sign Exact English using British Sign Language signs. The system utilises body tracking of real sign language videos in order to automatically generate BSL animations from textual input. As such it aims to generate sign language videos that can be used alongside written resources to improve the comprehension for hearing-impaired individuals.

The system has been evaluated on its performance on common hardware, finding it to be usable even on lower end computers. And it's perceived visual quality has been evaluated by a group of participants taken from HWU's computer science course who found the output animations to be readable and fluent.

The results demonstrate the potential of the system to support communication and accessibility for the deaf and hard-of-hearing community.

Standard Declaration of Student Authorship

Course code and name:	F20PA – Research Methods and Requirements Engineering
Type of assessment:	Individual
Coursework Title:	Dissertation Deliverable 2
Student Name:	Charles Lyttle
Student ID Number:	H00318301

Declaration of authorship. By signing this form:

- **I declare** that the work I have submitted for individual assessment OR the work I have contributed to a group assessment, is entirely my own. I have NOT taken the ideas, writings or inventions of another person and used these as if they were my own. My submission or my contribution to a group submission is expressed in my own words. Any uses made within this work of the ideas, writings or inventions of others, or of any existing sources of information (books, journals, websites, etc.) are properly acknowledged and listed in the references and/or acknowledgements section.
- I confirm that I have read, understood and followed the University's Regulations on plagiarism as published on the [University's website](#), and that I am aware of the penalties that I will face should I not adhere to the University Regulations.
- I confirm that I have read, understood and avoided the different types of plagiarism explained in the University guidance on [Academic Integrity and Plagiarism](#)

Student Signature (*type your name*): Charles Lyttle

Date: 26/04/23

Acknowledgements

Foremost I want to thank my supervisor Alistair for providing feedback, help and insight throughout the process of writing this dissertation and for tolerating the whirring laptop in the corner of his office for hours at a time (sorry!).

I also want to thank all the team at Signs@HWU for not only the amazing work they do, but also the eye-opening insight into hearing-impaired issues that I gained from speaking with them. A specific thanks to the interpreters that allowed such a conversation to take place.

And finally, I want to thank my parents for helping develop the name Kyro, and my girlfriend, Viv, for supporting me along the way.

Thank you all!

Table of Contents

Declaration	2
Abstract	3
Standard Declaration of Student Authorship	4
Acknowledgements	5
1. Introduction	8
1.1 Context	8
1.2 Aim	8
1.3 OpenPose	8
1.4 Other Technologies	9
1.5 Input Data	9
1.6 The Name	10
2. Background	11
3. Risk Analysis	12
3.1 Purpose	12
3.2 Defining Risk	12
3.3 Identifying Risks	13
3.4 Risk Mitigation	14
4. Potential Issue Analysis	15
4.1 Legal	15
4.2 Ethical	15
4.3 Social	15
5. Requirements Analysis	16
5.1. Use Case	16
5.1.1 Actors	16
5.1.2 Use Case Scenario	16
5.1.3 Use Case Diagram	16
5.2 Functional Requirements	16

5.3 Non-Functional Requirements	17
6. Development Timeline.....	19
7. Implementation	20
7.1 Development Environment	20
7.2 System Design	20
7.2.1 Final Design	20
7.2.2 Initial Design.....	20
7.3 System Layout	21
7.4 Scraper	22
7.5 Converter.....	23
7.5.1 Overview	23
7.5.2 OpenPose Command Analysis.....	23
7.5.3 Improvements	25
7.6 Dictionary	25
7.7 Tokeniser.....	25
7.8 Assembler	26
7.8.1 Lookup Script.....	26
7.8.2 Displaying Avatar	29
7.8.3 Convert to Video.....	39
7.9 System Validation.....	39
8. Evaluation	40
8.1 Requirement Evaluation.....	40
8.1.1 Functional Requirements	40
8.1.2 Non-Functional Requirements	41
8.1.3 Results.....	42
8.2 Visual Quality Evaluation	42
8.2.1 Methodology	42
8.2.2 Design	42
8.2.3 Results.....	43
8.3 Performance Evaluation.....	45
8.3.1 Methodology	45
8.3.2 Results.....	45
8.4 BSL User Evaluation	47
8.4.1 Results.....	47
9. Conclusions.....	48
9.1 Achievements.....	48
9.2 Limitations	48

9.3 Future Work	49
Bibliography	50
Appendices.....	52
Appendix Item A – Evaluation Information Sheet and Consent Form	52

1. Introduction

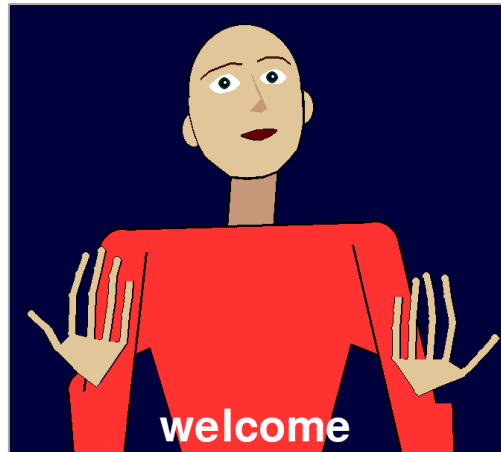


Figure 1. The Kyro system displaying the BSL sign for ‘welcome’.

1.1 Context

Sign languages are visual languages that use body movements to convey meaning and context. These body movements are most notably hand articulations and facial expressions. Sign languages have their own grammar and vocabulary which are dependent on the specific language spoken.

British Sign Language (BSL) is the preferred sign language used in the United Kingdom with an estimate 13,000 users in Scotland alone. (*Scotland's Census, 2011*) It's dictionary contains over 2500 defined signs. (*BSL SignBank, 2014*) BSL has a different syntactic order when compared to spoken English as this order conveys meaning faster, an example being, if a BSL user wished to sign “What’s your name?” the sign order would be “your”, “name”, “what?”.

Sign Exact English (SEE) is widely used as a technique to help deaf (or BSL using) individuals learn English (*Stephenson et al., 2020*). It is often used alongside spoken English to aid comprehension for individuals who use BSL, as such it doesn't follow the usual grammatical structure of BSL but, instead, the structure of spoken English, only borrowing the signs from BSL. Although SEE isn't as rich as BSL it can still help to get across key information to a hearing impaired (HI) learner. Automatic generation of an SEE animation would allow any text that a hearing-impaired user wished to read to become accessible without a human signer present and without the requirement of high-level English understanding. It could be used to translate educational resources, such as textbooks and academic papers potentially improving the quality of educational resources for hearing-impaired individuals.

1.2 Aim

This project aims to create a software that deaf learners can use alongside a written resource in order to understand the resource more clearly, to do so I propose a system that takes any text input, be it from an educational resource, website, or otherwise, and generates a video displaying an avatar signing the corresponding British Sign Language signs in Sign Exact English syntax order.

1.3 OpenPose

OpenPose is an open-source pose detection software developed by Carnegie Mellon University (*Cao et al., 2018*). OpenPose can track the position of body keypoints such as hands, the limbs, and specific joints from images. This can subsequently be used to track the keypoints of videos when ran frame-by-frame giving detailed and accurate body key point data of entire videos. More recent versions of the software allow key point detection of facial features allowing the tracking of facial movements from videos.

OpenPose is used to extract the movement data from real-life sign language videos and this data is then used to animate a signing avatar.

1.4 Other Technologies

The output of the system is built in Python's PyGame library, a simple, yet robust 2D game engine (Pygame, 2021). PyGame is used to display the signing avatar to the screen using python code.

As well as PyGame, the 'Requests: HTTP for Humans' library is used. Requests 'is an elegant and simple HTTP library' (Requests Team, 2022). This library is used to download the sign videos from SignBSL.com. SignBSL.com is further explained below.

1.5 Input Data

OpenPose is a versatile and robust system managing to handle undesirable data quite effectively. Even on videos with poor lighting, low resolution, or generally poor video quality, OpenPose still creates effective and accurate keypoints if configured correctly. However, even though this is true, OpenPose still works best on clean videos. Clean, in this context, meaning smooth, high resolution, low compression videos with a high level of contrast between the subject and their surroundings. (Cao, *et al.*, 2018) As such, the input videos used in the system, ideally, have these given characteristics.

In order to get such data SignBSL.com has been used. SignBSL.com is an online British Sign Language dictionary of over 21,000 sign language videos (SignBSL, *n.d.*). SignBSL.com is a collation of sign videos from many different sources from across the internet. It provides high quality videos of many signs in the BSL vocabulary along with their definitions, regional variations, and other helpful information.

Within SignBSL.com there are multiple creators that make sign language videos of words for use on the site. The most common of these creators being SignStation, with a large number of the videos available on SignBSL.com being from this creator. The videos posted by SignStation were ideal for this application. They all have a single subject, against a blue screen with studio lighting. Each video is uploaded in the exact same resolution making their use with OpenPose simpler. This is due to how OpenPose saves keypoints, simply put, the keypoints generated are the X,Y values of the pixels from the input video. Therefore, if each input video is of the same resolution, there is consistency between the keypoints generated and no processing has to be done to scale each video (or keypoints) to the same size.

For this application the SignStation videos are a brilliant resource, but they suffer from being in a low resolution of 320 x 240 pixels. This will affect the quality of the keypoints generated by OpenPose, especially in the more detailed areas such as the facial features and finger positions. However, due to the impressive lighting and consistency these videos are still the best openly available for this use case.



Figure 2. SignStation video screenshot of BSL sign depicting “hello”. (*SignBSL, n.d*)

1.6 The Name

The name ‘Kyro’ comes from ‘chiro’ a Latinization of the Greek word ‘kheiro’ meaning ‘hand’(*Harper, n.d.*). It also pays homage John Bulwer, one of the first English researchers into deaf communication and his adoptive deaf daughter, Chirothea (*Wollock. 1996*).

Throughout this report the name ‘Kyro’ will be used to refer to both the name of the avatar, and the name of the system itself.

2. Background

Literature on the use of sign language alongside written texts is varied and has garnered different results. Joy (2021) implemented a system called ‘SignText’ which would display in-browser videos of an instructor signing in Indian Sign Language alongside educational texts. The study found that having a video present during the learning process improved test scores significantly. This finding was validated through the use of a control group that did not use the system, which demonstrated little to no improvement. Thus, this study showed that there is merit to accompanying sign language visuals alongside text-based learning materials. However, the study was limited by the use of only a few chapters of learning materials, so it can not be assumed that the programme would be beneficial for all educational disciplines. Joy (2021) advises that there is a need for automation of the content generation process in future studies which they believe would be beneficial for both instructors and students. In an earlier study, McConnell (2020) implemented automation by creating a text to sign language synthesis system that employed the use of OpenPose in generating keypoints. McConnell’s approach had a small dictionary of signs, restricted to only the finger sign alphabet of BSL, this meant that McConnell’s output lacked the ability to translate a full fluent sentence. As well as this, McConnell’s approach didn’t take facial keypoints and was restricted to only the arms and hands.

Another approach by Yahia (2013) animated a 3D avatar by creating a descriptive language called SML (Sign Modelling Language), a dictionary that held geometric descriptions of sign movements. This geometric synthesis created stiff, inhuman movements and caused limbs and extremities to clip within the 3D model at times which broke immersion. As well as this, the generation of the descriptive sign data was cumbersome as this was done manually rather than with motion tracking. Chakladar (2021) created a similar system to that of Yahia (2013) but converted English sentences into Indian Sign Language (ISL). Similarly, to Yahia’s study, the sign movements were defined manually, and their implementation lacked the animation of facial expressions. Furthermore, Chakladar did not tackle blending between sign words therefore their implementation had stiff transitions between signs.

Ebling (2013) created a signing avatar for use in train announcements, it translated German train announcements to Swiss-German Sign Language. However, as in Yahia’s (2013) case, signs were developed geometrically so had the same shortcomings that come with most geometric synthesis methods such as unnatural looking movements and having to manually generate the sign data. Also, they were limited to set announcements and fingerspelling and their implementation couldn’t automatically generate signs from any given text.

Arguably, Shyam (2021) presents the most impressive and relevant finding amongst the literature. The study explored another system using Indian Sign Language (ISL), however this system used OpenPose and a 3D estimation algorithm to convert the 2D OpenPose points to 3 dimensions. This method estimated joint rotations in order to generate the 3rd dimensional value which allowed them to track these points to a realistic 3D avatar. The downsides presented by their method was a lack of facial animations and jittery animations.

In general, the available literature provides evidence that the use of sign language displayed alongside written resources is beneficial to learning. Many studies have successfully produced working systems, but most suffer setbacks from the use of geometric synthesis, or small data sets. Studies that have used 3D representations successfully have had specific limitations either due to their limited use case or flawed approach to sign data generation. This suggests that there is a need for sign synthesis to be explored and developed further.

3. Risk Analysis

3.1 Purpose

We analysed potential risks before beginning the development process so that if and when they occurred mitigation plans were already developed and the impact of said risks was lessened.

3.2 Defining Risk

Risk is defined by the likelihood of something happening multiplied by the impact of it occurring. The severity of a risk can therefore be calculated and analysed.

If we take a rating for each of these factors:

Likelihood:

Very Unlikely	Unlikely	Possible	Likely	Very Likely
1	2	3	4	5

Table 1. Likelihood Rating Table

Impact:

Negligible	Minor	Moderate	Significant	Severe
1	2	3	4	5

Table 2. Impact Rating Table

We can then use a risk rating matrix to calculate a risk value and, as such, how much effort should be put into mitigating said risk:

		Impact				
		Negligible	Minor	Moderate	Significant	Severe
		1	2	3	4	5
Likelihood	Very Unlikely	1	2	3	4	5
	Unlikely	2	4	6	8	10
	Possible	3	6	9	12	15
	Likely	4	8	12	16	20
	Very Likely	5	10	15	20	25

Figure 3. 5x5 risk calculation matrix

3.3 Identifying Risks

To identify possible risks, the following table was created with the risk, the likelihood rating, consequence rating, risk rating and a description of what the risk is to understand which risks require the most mitigation.

Risk	Likelihood	Consequence	Rating	Description
1. Too low technical specifications	Unlikely	Moderate	6	The possibility that my home PC isn't powerful enough to run OpenPose would mean that I would have to use a university owned machine, although this isn't a huge hinderance it would mean that all development must be done on campus and therefore could slow down development time.
2. Poor performance	Possible	Minor	6	Game engines, specifically when rendering out, can be slow and cumbersome programs, this would make the software much less user friendly if the user had to wait for an extended period for their video output.
3. Change in Requirements	Likely	Negligible	4	There is a risk that during the design process of the system I have missed essential requirements/features that will become apparent during development. These can be added to the list of requirements later but may take up more time.
4. Underestimated Timeline	Possible	Moderate	9	The time that each requirement will take (see Fig. 2) is difficult to estimate accurately, if the timeline isn't followed then the entire project could get behind schedule.
5. General Software Problems	Possible	Significant	12	The software that are planning to be used may cause general technical issues (OpenPose being inaccurate on the input data, throwing errors etc.) which could significantly slow down expected development speed.
6. Input video quality issues	Unlikely	Minor	4	There is a chance that the quality of the collected sign videos won't be good enough for OpenPose to accurately track possibly leading to inaccurate pose data
7. Loss of work	Unlikely	Severe	10	Loss of work can occur due to a number of technical issues, which could set back progress significantly.
8. Sign Language Oversights	Possible	Significant	12	It is possible, due to not having a background in sign language, that I have made an oversight about how sign languages work possibly making the proposed system unusable in a real setting.

Table 3. Table of risks, risk ratings and descriptions.

3.4 Risk Mitigation

In order to mitigate risks, a strategy was created for each to minimise either the likelihood or the consequence of occurrence.

1. Too Low Technical Specifications

The computer used has a recent, high-end GPU meaning it should be more than enough to run the OpenPose framework. In case it is not, a university computer would be used for development. This would only affect the converter stage of development and therefore shouldn't cause significant slow downs. As well as this, the Heriot-Watt Keyserver would allow access the university PC from home and therefore time wouldn't be lost travelling to campus.

2. Poor Performance

To mitigate the effects of poor performance, lots of the processing (the creating of the sign dictionary) is occurring before runtime, this means that the OpenPose software will not have to run when the software is used as that processing has taken place beforehand. This, however, may mean that the software is large as the sign information will be held locally instead of fetched on runtime.

3. Change in Requirements

Change in requirements is likely, however, it is mitigated by lots of time being allocated for the sections of the system in which I don't have extended prior knowledge, for example, I have made very similar programs to the Dictionary and the Tokeniser in the past and therefore the time allocated for these is short, however the Assembler and the Converter have a much longer timeframe allocated and therefore a change in requirements in these sections should be handled within the dedicated time.

4. Underestimated Timeline

As with the previous risk, the timeline has been decided based on my prior programming/software development history and therefore more time has been allocated to sections which I am unfamiliar with. This should mitigate the potential negative consequences of tasks being harder than first perceived.

5. General Software Problems

Mitigating the risk of general software issues is very difficult, to do so more time has been allocated for unfamiliar software, as well as this, a small amount of preliminary testing took place in order to develop a richer understanding of the software before development began.

6. Input Video Quality Issues

If the quality of the input videos taken from SignBSL.com is of subpar quality I intend to film sign videos myself in an OpenPose friendly environment. It should be noted, however, that OpenPose has been proven to work effectively in substandard environments and therefore the likelihood of this risk occurring is very low.

7. Loss of Work

To mitigate the chance of work loss I am storing all work on both a cloud service (OneDrive) and locally on my home machine, as well as this I intend to save a backup to an external USB drive at intervals throughout the development.

8. Sign Language Oversight

To mitigate the chance that my project is unfeasible I interviewed a Heriot-Watt sign language researcher to discuss my project. They informed me that SSE follows the same grammar as English and therefore very little prior sign language knowledge would be necessary to create the desired product.

4. Potential Issue Analysis

This section will note any professional, legal, ethical and social issues that may relate to the project.

4.1 Legal

A legal issue that may be presented by the project is the use of external video sources, the videos that are used are from SignBSL.com are therefore not my own. They could have copyright restrictions. Although, as this is a research study, non-commercial and the use of the videos is transformative in nature, the use case complies as an exception to copyright laws. (*Intellectual Property Office, 2014*) If this work was later commercialised, the videos from SignBSL.com would have to be licenced in order to sell the product.

As well as this all General Data Protection Regulations (GPDR) (*Intersoft Consulting, 2018*) are being followed during my evaluation study, consent is taken via a consent form (*Appendix Item 1*) and all data will be anonymised to avoid negative consequences of any potential data leaks. Holding the personal data of the participants for any extended amount of time isn't required and therefore any regulations regarding this do not apply. Data will be removed from the study at the request of the participant, and we will be fully transparent with the aims and goals of the study to avoid confusion or deception.

4.2 Ethical

During this project I will follow the British Computer Society Code of Conduct for IT Professionals as a code of ethics. This is a well regarded, widely used code of ethics that considers public interest, professional competence, and one's duty to the profession. (*British Computer Society, 2022*) This will act as an ethical framework to follow as I conduct my research.

As my evaluation study may use hearing-impaired individuals as participants, I will attempt to make the evaluation procedure as accessible as possible, however, a moderate level of English comprehension is required to take part in the questionnaire. I could potentially mitigate this by having a sign language interpreter present at the evaluation. The usefulness of this would be purely determined by the reading level of the participants and, as most will be academics, I think it is safe to assume moderate English competency.

4.3 Social

One social issue regarding my product is that it is attempting the automation of an already established job. For example, if the proposed project is used as the basis for a further expanded software that could live translate, it could be used in place of current sign language interpreters as a cheaper method to make content accessible to the hearing impaired. Businesses would be inclined to use a software approach to sign language interpretation over a human as it would allow a faster, cheaper, automated way of providing accessibility to almost any resource. This could potentially lead to the loss of jobs for sign language interpreters.

The project, however, is focused on adding accessibility to resources that currently aren't accessible, this means that the resources used don't have sign language interpretations yet associated with them and therefore my project only add to the accessibility rather than taking away or automating someone's job. As such, it is only an extension of my work that would cause potential harm and loss of jobs.

5. Requirements Analysis

5.1. Use Case

5.1.1 Actors

The actors that will be interacting with the system are, mainly, hearing-impaired individuals and as such the system should not rely on audio for usability.

5.1.2 Use Case Scenario

The successful use case scenario for my system is as follows:

The user generates a successful animation:

- A user is struggling to understand a text resource online,
- The user launches the developed software,
- The user copies the text that they are struggling with into the input box,
- The system generates a video of an avatar signing the given text in SSE,
- The user plays this video alongside the written resource to gain a better understanding,

5.1.3 Use Case Diagram

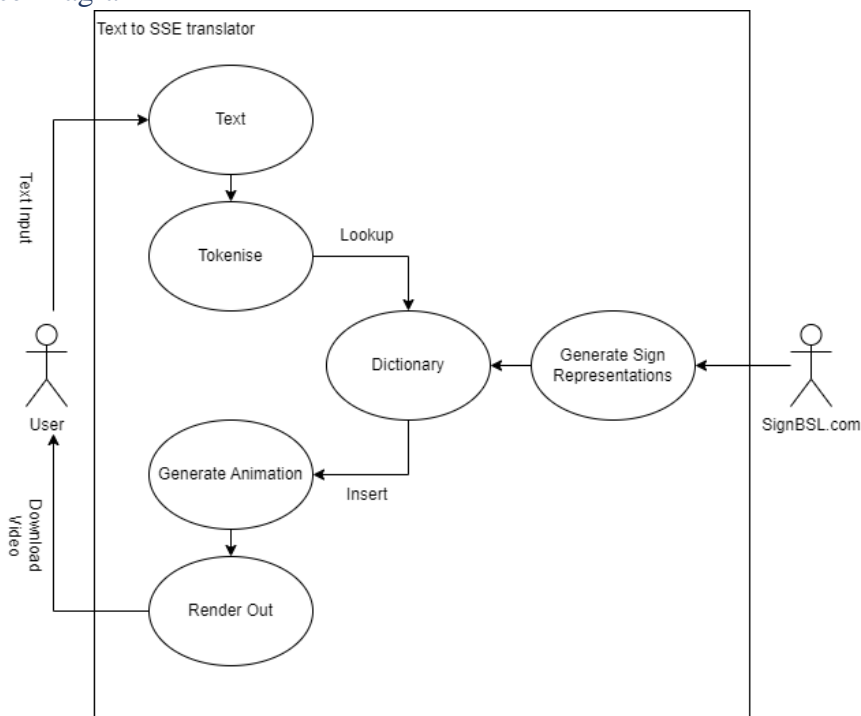


Figure 4. Use case diagram of the proposed system.

5.2 Functional Requirements

The following is a table of the functional requirements of the proposed system with MoSCoW analysis of each:

Table of Functional Requirements	
F-UR 1	Tokeniser
F-UR 1.1 Must	The tokeniser must take the input text and split it into words (or tokens),
F-UR 1.2 Must	The tokeniser must then look up the tokens in the dictionary,
F-UR 1.3 Could	The tokeniser could recognise multi-word signs (single signs that depict more than one English word),

F-UR 1.4 Could	The tokeniser could take other options rather than just plain text as input (.pdf, .docx, web links etc.)
F-UR 2	Converter
F-UR 2.1 Must	The converter must convert the sign videos to JSON format,
F-UR 3	Dictionary
F-UR 3.1 Must	The dictionary must be an object capable of holding sign words and their JSON format
F-UR 3.2 Must	The dictionary must contain the BSL finger sign alphabet and their JSON representations,
F-UR 3.3 Should	The dictionary should contain 10% of the SignBSL.com dictionary and their JSON representations,
F-UR 3.4 Could	The dictionary could contain 100% of the SignBSL.com dictionary and their JSON representations,
F-UR 3.5 Will Not	The dictionary will not hold sign representations for any sign languages other than BSL,
F-UR 4	Assembler
F-UR 4.1 Must	The assembler must fetch the required JSON data from the dictionary for the input text,
F-UR 4.2 Must	The assembler must animate an avatar using the fetched JSON data ,
F-UR 4.3 Should	The assembler should automate the avatar animation process without intervention,
F-UR 4.4 Could	The assembler should blend the generated animations together so transitions between them are smooth,
F-UR 4.5 Could	The assembler could display the corresponding word with the sign
F-UR 4.6 Must	The assembler must be able to render out the generated animation in an accessible video format (.mp4, .avi, .mov, etc.),
F-UR 5	User Interface
F-UR 5.1 Must	The UI must be, at least, presented in an easy to understand, text-based format
F-UR 5.2 Could	The UI could be graphical and work independently as a runnable .exe
F-UR 5.3 Will Not	The application will not be web integrated

Table 4. Table of functional requirements and their descriptions

5.3 Non-Functional Requirements

The following is a table of the non-functional requirements of the proposed system with MoSCoW analysis of each:

Table of Non-Functional Requirements	
NF-UR 1	Performance
NF-UR 1.1 Must	The application must be able to run on high-end consumer technology (Workstation/Gaming PC),
NF-UR 1.2 Could	The application could be able to run on low/mid-end consumer technology (Mid-range Laptop),
NF-UR 1.3 Will Not	The application will not be able to run on mobile,
NF-UR 1.4 Will Not	The application will not be able to run on any non-windows OS,

NF-UR 1.5 Must	The application must be able to complete processing (text input to output render) in 30 seconds per word,
NF-UR 1.6 Should	The application must be able to complete processing (text input to output render) in 5 seconds per word,
NF-UR 1.7 Could	The application must be able to complete processing (text input to output render) as fast as the words would be spoken (sub 1 second per word),
NF-UR 1.8 Could	The application could predict how long processing will take and report this to the user

Table 5. Table of non-functional requirements and their descriptions

6. Development Timeline

Figure 5 displays a Gantt chart showing the developmental timeline that was followed when the system was developed. It shows key dates, requirements and the time in which they were completed.

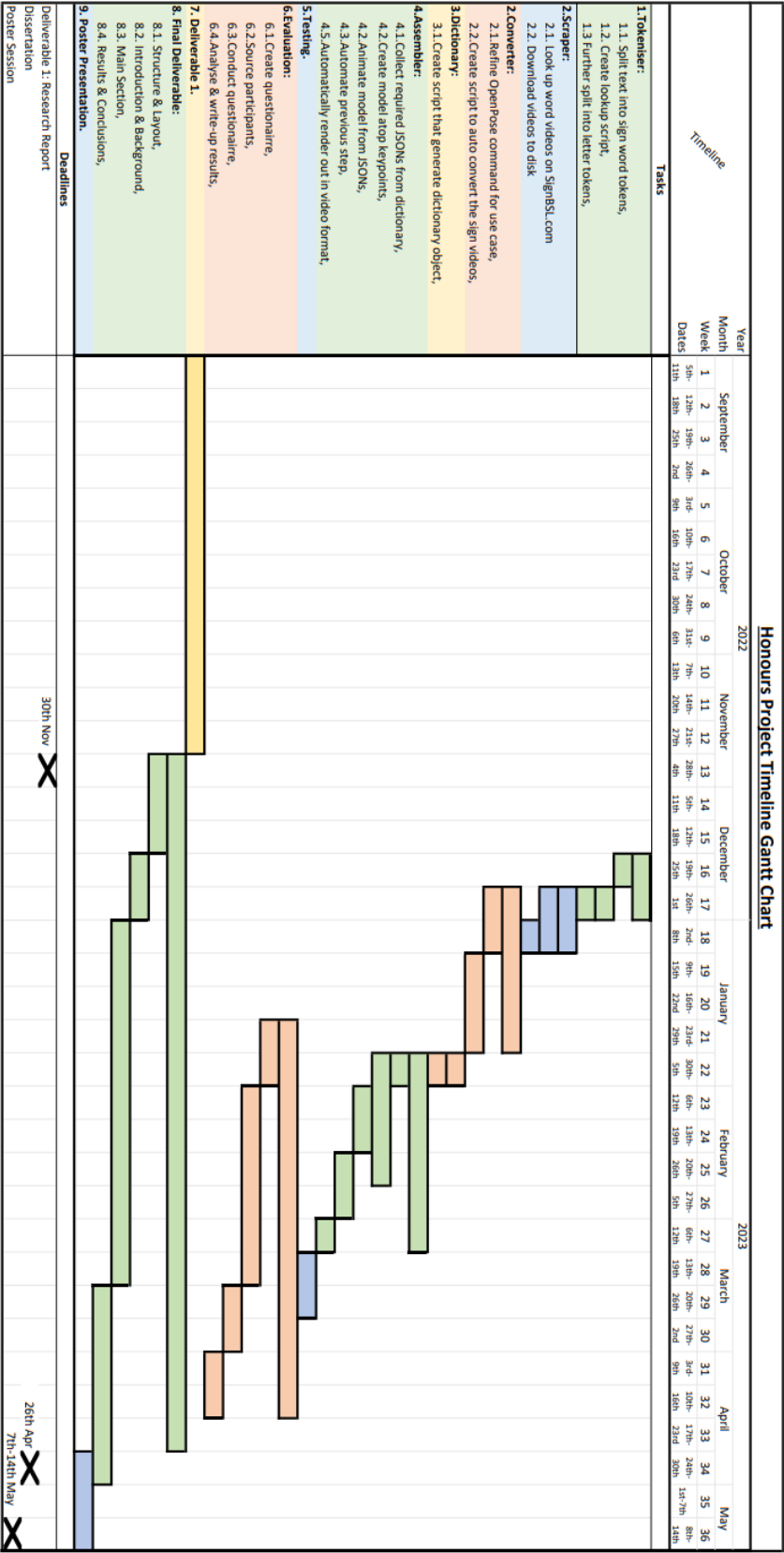


Figure 5. Development timeline of the system displayed as a Gantt chart.

7. Implementation

7.1 Development Environment

The system used to develop the application is a home PC system running Windows 10, it contains an NVIDIA GeForce RTX 2060 with 6GB of VRAM, 16GB system memory and a RYZEN 5 3600 6-Core Processor. This system is reasonably high spec and will be higher than most end users, especially those using portable systems such as notebooks, laptops, or tablets. As such the system must be tested on these lower-end machines.

VS Code has been used as text editor and any code screenshots used are taken directly from this.

7.2 System Design

7.2.1 Final Design

The design of the system is as follows, the system is split into two parts, one part is completed before runtime and the other part is processed at runtime. Before runtime there are two steps. First, the scraper system scrapes SignBSL.com for sign language videos that are then saved to the disk. Secondly, the converter uses OpenPose to convert the sign videos to their JSON keypoint format and saves them to the disk. These two systems create the dictionary of sign words and their JSON representations for use at runtime. At runtime the system first takes user input of the sentence they wish to convert. A tokeniser routine then splits the text into each individual word. Each words corresponding JSONs are then fetched from the dictionary and finally the assembler displays their movements on a 2D avatar to the screen.

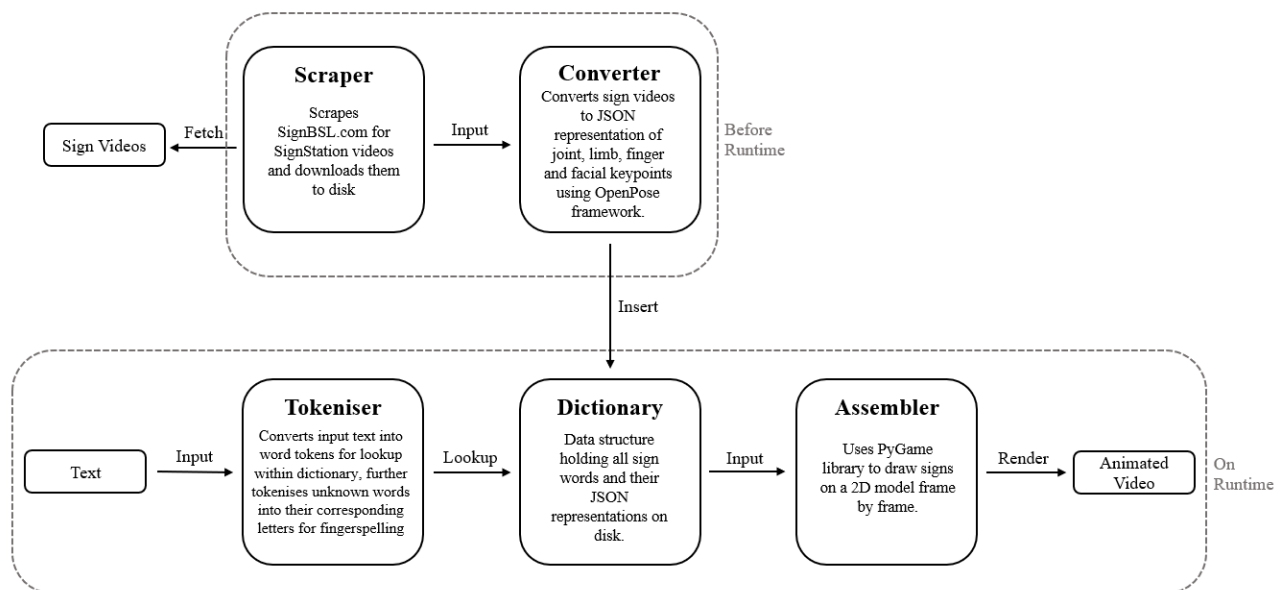


Figure 6. Design of the Final System

7.2.2 Initial Design

The system's design has changed since the original design of the system proposed in D1. The original system's design can be seen below. The original system design didn't have a scraper as it had not yet been decided how the sign videos would be collected. The original idea was that the videos would be downloaded manually from SignBSL.com however this approach would've been very time consuming, and the final approach used was a significantly better way of doing it, as well as this the assembler program has been changed from 3D to 2D. OpenPose, by default, only generates 2D keypoint data, meaning that what OpenPose outputs is X,Y values of each keypoint. Therefore, to develop a 3D approach the Z or depth value would have to be calculated. After extensive research it was found that a 3D implementation wasn't viable without extra equipment, for example, OpenPose

can actually handle 3D reconstruction however it requires an IR camera and as the videos are taken from the internet and not recorded by myself it would be impossible to use this approach without re-recording the signs with both a visible and IR camera. The other method I thought could work is using a system such as OpenMMD (Liu, 2021). OpenMMD is a deep learning project build on OpenPose, this project uses deep learning models to take an input video and from it estimate the depth values of the subject in the video. It can therefore assign depth to each keypoint to be used in a 3D software, in this case a 3D animation software called MikuMikuDance (hence MMD) (Liu, 2021). This approach, however, is limited in the accuracy of its generated depth values and doesn't handle hand or facial keypoints being used. As such these approaches weren't suitable and instead the choice was made that a robust 2D approach would make a more usable product, it also allows the end product to be less resource heavy and therefore gives it the ability to run on lower-end hardware.

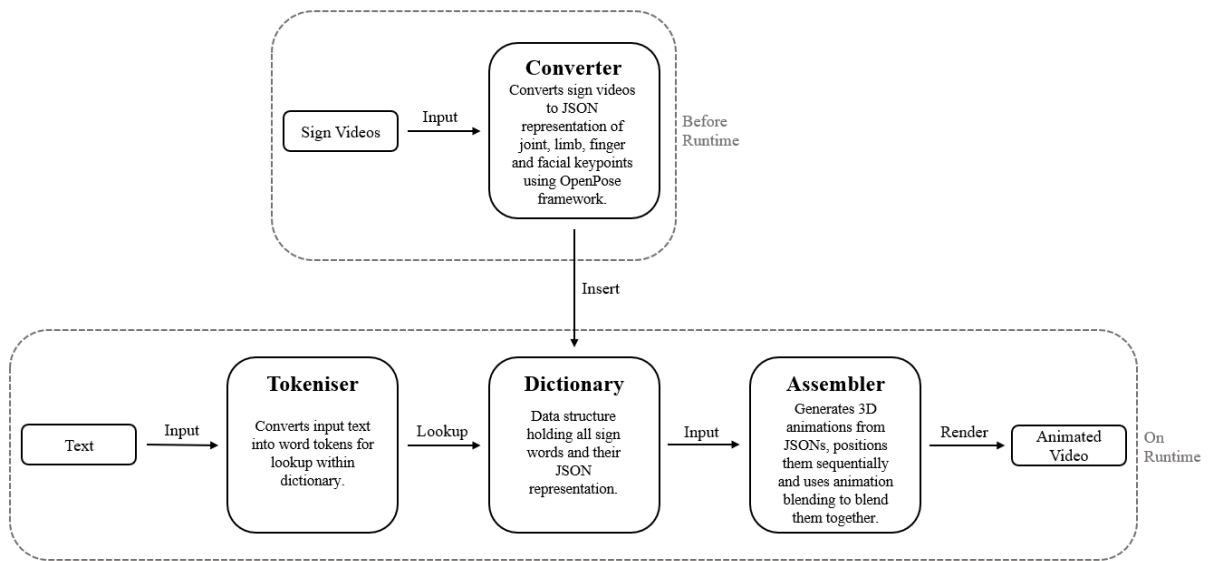


Figure 7. Initial Design of the System

7.3 System Layout

The system is split amongst 6 python files. With only either 2D.py or 2DVideoOut.py being ran at runtime. The layout is as below:

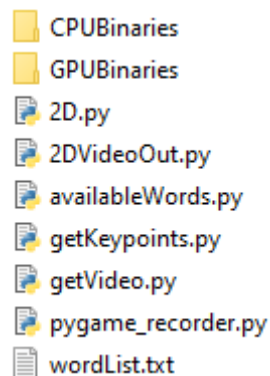


Figure 8. Layout of project files in the file system.

The CPUBinaries and GPUBinaries folders both hold specific versions of OpenPose. The CPU binaries of OpenPose run the pose estimation software using the CPU and the GPU binaries run it on the GPU. The converter system (getKeypoints.py) is configured for using the GPU binaries however this can be switched if the PC running the converter has no dedicated GPU or one of low spec. These are both the windows portable binaries of OpenPose and are not compiled from C. The scraper

(getVideo.py) takes words from wordList.txt and these are then scraped from SignBSL.com and are stored in GPUBinaries/openpose/examples/signs. Therefore wordList.txt must be populated for new words to be added to the system. availableWords.py is a small helper program that simply searches for and prints all the words available to the system simply for visualisation of the dictionary. The converter system (getKeypoints.py) reads in the videos generated by the scraper in GPUBinaries/openpose/examples/signs and outputs the generated JSON files into GPUBinaries/openpose/raw_jsons. 2D.py is the main assembler program, this allows the user to input their desired sentence and will display the rendered output to the screen, 2DVideoOut.py is the exact same as 2D.py however it records the screen producing a video of the PyGame screen called output.avi. This is done using the pygame_recorder.py library (*tdrmk, 2019*).

To summate, 2D.py and 2DVideoOut.py are the only files ran on runtime. And these only require the raw_jsons folder and the pygame_recorder.py library to have full functionality.

7.4 Scraper

The scraper system is the system designed to fetch the sign language videos from SignBSL.com, in order to do so it uses the python request library to make download requests to SignBSL.com.

All links to sign videos on SignBSL.com come in a standard format, this being:
[https://media.signbsl.com/videos/bsl/\[Creator's Name\]/\[Sign Word\].mp4](https://media.signbsl.com/videos/bsl/[Creator's Name]/[Sign Word].mp4). A request can then be sent to this URL from python in order to download the corresponding video. As only videos from SignStation are being fetched, [Creator's Name] can be replaced with "signstation" and then the [Sign Word] must be set to whichever word is desired to be fetched. In order to do so, a text file (wordlist.txt) was created containing 10,000 words taken from an online word list provided by MIT (*Price, n.d*) this text file is loaded into the scraper program then iterated through, sending a request to download each sign video from SignBSL.com. If the video does not exist, then the response returned is of a very small size (under 500 bytes) and nothing is saved to the disk.

```

#For every word in the wordlist
for word in wordList:

    #URL to each given video
    URL = "https://media.signbsl.com/videos/bsl/signstation/"+ word + ".mp4"
    #output path for downloaded videos
    outPath = "../GPUBinaries/openpose/examples/signs/" + word + ".mp4"

    #If the file doesn't exist at outPath
    if (pathlib.Path(outPath).is_file() == False):
        #Send request to URL
        print("requesting...")
        response = requests.get(URL)
        print("request complete.")

        #If video exists:
        if (len(response.content) > 500):
            # Write the request response (video) to file at outPath
            print("writing...")
            open(outPath, "wb").write(response.content)
            print("writing complete.")
        else:
            print("file does not exist")

```

Figure 9. Main function of the scraper program. Iterating through the word list and sending the download request to SignBSL.com.

This approach to downloading the sign videos is, unfortunately, not exhaustive. This means that every single SignStation video is not downloaded, instead, only the videos for the words defined in the wordList.txt are searched for. This can be countered by increasing the number of words in wordList.txt file to cover the entire BSL dictionary meaning that every possible BSL word is requested. However, this also adds inefficiencies as it sends requests for videos that may not exist. SignStation don't have videos for every single BSL sign, the dictionary is just too large for one user to do so. This means that, as the system looks for specifically SignStation videos, and many of them do not exist, the system wastes request calls to non-existent videos frequently. These inefficiency issues aren't particularly important as the scraper program is only run before runtime meaning that this program probably wouldn't be run on an end user's machine and instead the sign videos (in fact only their JSON representations) would be pre-downloaded and shipped with the end product.

7.5 Converter

7.5.1 Overview

The converter is a simple python program designed to use OpenPose to generate keypoints. At its core it simply runs OpenPose commands on the command line using the Python 'OS' library (*Python Software Foundation, 2021*). Doing so from python rather than from command line allows the process to be automated. The program iterates through all the videos downloaded by the scraper program and runs a specific OpenPose command on them one by one.

7.5.2 OpenPose Command Analysis

The command used to generate the keypoints can be seen below with the 'fileNameMp4' variable being changed to the new sign video each time it is ran.

```
OpenPoseDemo.exe --video examples\\signs\\" + fileNameMp4 + " -
number_people_max 1 --net_resolution -1x160 --face_net_resolution 224x224 --
hand --face --display 0 --render_pose 0 --write_json raw_jsons
```

Figure 10. OpenPose command used to generate keypoints.

This command is tailored specifically for an NVIDIA GeForce RTX 2060 graphics card. An RTX 2060 is a mid-range card and 2 generations old (at time of writing). As such it fails to run OpenPose at maximum accuracy, mainly due to the limitations of its 6GB of VRAM. For OpenPose to run at maximum accuracy it requires a GPU with 16GB of VRAM available to it. (Cao *et al.*, 2017) Therefore, in order to generate keypoint data the accuracy of OpenPose had to be sacrificed.

The following section aims to explain the OpenPose command used, and its given parameters.

OpenPose has the ability to track the keypoints of many subjects within a video simultaneously, however, for this use case this is not necessary. Therefore, to create a minor increase in performance the `number_people_max` parameter is set to 1, this specifies the maximum number of subjects that OpenPose will attempt to track and therefore after the first subject's keypoints being tracked OpenPose will not waste resources looking for any others. After this parameter are the two 'net_resolution' parameters, one for the body keypoints and one for the facial keypoints. These define the accuracy of the keypoints generated, higher net_resolution uses more memory and is more accurate, lower net_resolution uses less memory but is less accurate. The default net_resolution is -1x368 (Cao *et al.*, 2017). And therefore, anything lower than this will speed up performance. The net_resolution parameter determines the resolution of the body keypoints. As these are larger than the details on the face or hands this resolution can safely be lowered without too much inaccuracy occurring in the output keypoints. On the other hand, the details on the face are small in the input videos and therefore there needs to be a higher net_resolution used for accurate facial keypoints. Note that there is no change in the 'net_resolution' for the hands. Due to the nature of the sign language use case, the movement of the hands and fingers are the most crucial to capture accurately and therefore losing any accuracy in the hands would be a significant detriment to the output results and subsequently the usability of the product as a whole. The face and hand parameters simply make OpenPose detect face and hand keypoints, otherwise it would run on only the body keypoints. The 'display' and 'render_pose' parameters are both set to 0, these make the OpenPose command run with no visualisation. The default configuration can be seen in Figure 11 below, this displays the current frame to the screen and renders the keypoints generated over the top of the input video. This was useful when tweaking the 'net_resolution' values in order to visually see the inaccuracies of the generated keypoints in real time, however, as the converter program is running this command many times over many input videos it is best suited to show no visualisation in order to improve performance.



Figure 11. An example of OpenPose being run with visualisation turned on.

The final parameter is the 'write_json' parameter, this, as the name suggests, manages writing the keypoints generated to JSON files and where they will be stored.

7.5.3 Improvements

The JSONs that were originally generated using a RTX 2060 were acceptable and worked reasonably well, however, there was a significant loss of accuracy due to the lowering of 'net_resolutions'. This meant that the system wasn't able to make use of OpenPose's full potential. A short way though the development timeline, and during the creation of the assembler system, the keypoints were re-generated again using a different GPU. My supervisor allowed the use a laptop fitted with a mobile 2080 GPU. The main draw of using this GPU was the 2 extra gigabytes of VRAM available to it. This allowed the net_resolution to be scaled up a small amount. The facial net_resolution was left unchanged but the body net_resolution was upped to 224x224 to match that of the face. This meant that the body movements were tracked more accurately and meant there were fewer visual bugs in the final program.

7.6 Dictionary

The dictionary of all the words and their corresponding JSON values is kept on the disk in GPUBinaries/raw_jsons, each frame of each sign is kept in corresponding JSON. This is the default output method of OpenPose when ran on a video.

This wasn't the only implementation experimented with, originally a master JSON dictionary file was generated. This is where a single file held all the data for every sign, this implementation was faster when displaying the output however it used a huge amount of RAM as the master JSON was over 2GB and had to be loaded into RAM all at once. This meant that there was a large pause before the system began to render out the output. This may even cause a complete system crash if it was run on a low power system that had a small amount of RAM available to it, or if the system was running other programs at the same time that had a large RAM requirement.

7.7 Tokeniser

The tokeniser is a very small, simple program used to split the given input sentence into its words, it also deals with some formatting so that the assembler can lookup the correct word in the dictionary. It is situated at the start of the assembler program (2D.py and 2DVideoOut.py).

The tokeniser first takes the user's desired sentence, strips it of punctuation, then converts the sentence to lowercase and splits it into words using the python inbuilt split() function. This then gives

us a list (sentenceList) of each word that the user wishes to be displayed. However, as mentioned before, there is not a sign video for every word, as well as this there may not be a BSL sign for every word either. For example, there is no dedicated BSL sign for every single name, be it of a person, an organization, or a street etc. There are simply too many names for BSL to account for. Instead, BSL users ‘fingerspell’. This is where the word is spelled in English letter by letter using the BSL representation of each letter. This also allows my program to spell words that the scraper program missed to avoid it getting stuck with words not present in the dictionary. To implement this the words generated previously in ‘sentenceList’ are iterated through and checked against the dictionary of available words with JSON representations. If the word is not present in the dictionary it is split further into its individual letters and uppercased and then added back into the ‘sentenceList’ in its appropriate place so that it can be looked up effectively.

```
--Generates sentenceList--

#Takes user input
sentence = input("\nEnter sentence for translation:\n")
#Remove Punctuation
sentence = sentence.translate(str.maketrans('', '', string.punctuation))
#Splits by whitespace into list
sentenceList=sentence.lower().split(" ")
#For every word in sentenceList
for i in sentenceList:
    #If it's an available word
    if i in availableWordList:
        #Do nothing
        pass
    #If not
    else:
        #Generate the missing word in letters for fingerspelling
        insert = list(i.upper())
        idx = sentenceList.index(i)
        #Replace in the sentenceList
        sentenceList = sentenceList[0:idx:] + insert + sentenceList[idx+1:len(sentenceList):]
```

Figure 12. Splitting input into sign words, and further into letters if word is not present in dictionary.

7.8 Assembler

The assembler program first handles library imports and defines various variables for use later. It also initialises the PyGame environment and screen. The assembler then takes a user input in which the user specifies what resolution they want the PyGame screen output to be. Then the user is prompted to enter the sentence to be translated. The tokeniser defined above is then ran on the input text. Below this all the functions for displaying the avatar are defined. Finally, the main loop (held in a ‘while True’ statement) is ran every frame and uses PyGame to display the avatar to the screen. When the drawing of the avatar to the screen for each sign is complete the program (if running 2DVideoOut.py instead of 2D.py) saves the video recording of the PyGame screen to the disk and exits.

7.8.1 Lookup Script

Before the avatar is displayed, first the signs JSON representations need to be pulled into the program for use. The lookup script is what does so, generating 4 list objects for the body, face, left hand and right hand keypoints.

The lookup script first generates the filename of the current JSON file to be looked up. The JSON’s which are generated from OpenPose are automatically named in the following manner:

[Current Word]_[Frame Number]_keypoints.json

As such, in order to be called into the program at the correct time, the correct word and the correct frame must be calculated, then, using the JSON python library, the specific JSON can be brought in.

The [Current Word] variable is calculated by keeping track of the number of the current word the system is rendering at that time and simply looking for that index in the 'sentenceList' generated by the tokeniser system previously. As long as the number of the current word is incremented every time word is complete, the [Current Word] variable is always easy to define.

The [Frame Number] variable defined above is in a twelve-digit format and increments backwards when another digit is added. For example, the first frame will have the [Frame Number] value of "000000000001" the second will be "000000000002", however the tenth will be "000000000010". This presents a rather unique problem as this number cannot be incremented by simply summing 1 to it. The whole value must be generated for each frame. To do so the system not only has a 'current frame' variable (curFrame) but also a 'digits' variable (digits) that holds the number of digits in the current frame. As such the correct [Frame Number] value is created by appending 12-digits number of 0's to a string followed by the value of the 'current frame' variable. This meant that the correct extension (jsonExt) is appended to each JSON file name every frame. This extension is also reset when the program moved onto the next word.

```

#--Calculate the current filename--
for i in range(12-digits):
    jsonExt=jsonExt+"0"
jsonExt=jsonExt+str(curFrame)
jsonDir = "GPUBinaries/openpose/raw_jsons/" + sentenceList[curWord] + "_" + jsonExt + "_keypoints.json"

```

Figure 13. Calculating the current filename at the start of each frame.

```

#If we have reached the final frame
except FileNotFoundError:
    #Increment the current word
    curWord+=1
    #If the last word was the final word, quit
    if curWord == wordsInt:
        quit()
    #Current frame / current JSON file reset
    curFrame = 1
    #Digits within the current frame number reset
    digits=1

```

Figure 14. Incrementing to the next word and resetting variables at the end of a word.

Once the JSON is loaded for the current frame each type of keypoint generated is extracted and put into corresponding lists. This has its own unique issues. The first of which being that OpenPose doesn't only save the X,Y coordinates of the current keypoint but also a confidence value. The confidence value is how sure OpenPose is that the X,Y coordinates it has generated are correct. It saves this confidence value as a float from 0-1. 0.9 being 90% certainty for example. For this implementation the confidence value will not be used therefore when generating the keypoint lists every third value had to be removed. This is done by having a count variable that, whilst going through each value, counts to 3, when it reaches 3 the value is disregarded, and the count is reset.

In doing so the lookup script generates 4 separate lists. ‘bodyKeypointList’, holding all the body keypoints, ‘faceKeypointList’, holding all the facial keypoints, ‘rHandKeypointList’ and ‘lHandKeypointList’ holding the keypoints of each corresponding hand.

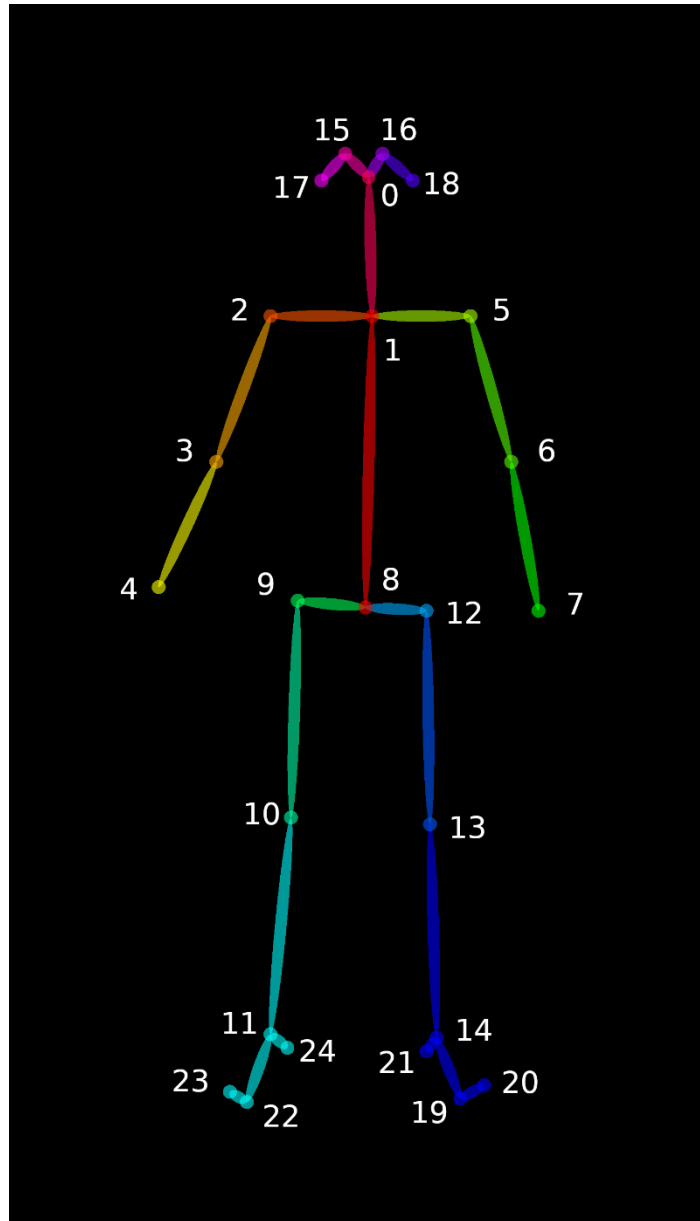


Figure 15. The guide image for OpenPose body keypoints (*Cao et al, 2018*)

The keypoints that OpenPose generates are ordered and the OpenPose documentation (*Cao et al, 2018*) provides images that act as a key to accurately identify each keypoint. Above is the key for each bodily keypoint. In this case these are the keypoints held in the ‘bodyKeypointList’ list that was generated by the lookup script. Therefore, the item at ‘bodyKeypointList’s 0th index is the point labelled 0 in Figure 15. The same images have been provided for the facial and hand keypoints seen in Figures 16 and 17.

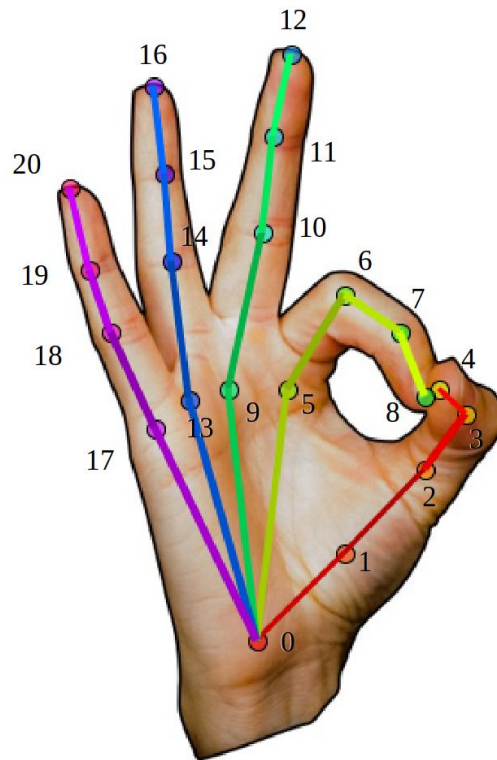


Figure 16. The guide image for OpenPose hand keypoints (Cao et al, 2018)

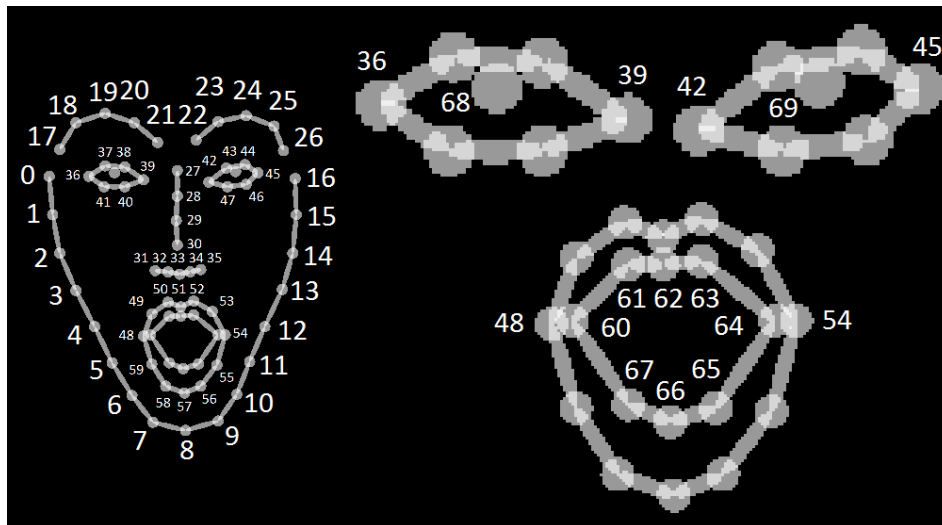


Figure 17. The guide image for OpenPose facial keypoints (Cao et al, 2018)

These images were extremely useful when rendering the avatar and help with understanding how it is drawn in the 'DisplayAvatar' function.

7.8.2 Displaying Avatar

7.8.2.1 Methodology

The PyGame engine doesn't have functionality to allow models to be imported into it. As such the way in which the avatar is created and used is rather unique. A usual approach to this problem would be that a rigged model is chosen beforehand and then the animations are transposed onto that model so that it moves. In this case the approach is quite different. The model (avatar) is drawn atop the keypoints every frame. For example, the coordinates of the elbow joint and the wrist joint are known and then a line is drawn between them to simulate the forearm. The model isn't predetermined, it is

generated at runtime. In this use case this works as the movements of the avatar are reasonably limited, for example, there would be many problems using this approach if, say, the avatar turned around. But as the subject is sat facing the camera in all examples and, mostly, only moving their hands and arms, this approach works well.

The following outlines some of the most crucial functions defined for rendering the character.

```
def DrawEllipse(coordList: list, index: int, col: tuple, weight: int, width: int, height: int):
    try:
        #Filters out untracked keypoints (if the keypoints are successfully tracked)
        if coordList[index][0] > 10*multiplier:
            #Defines a rectangle PyGame object within which the ellipse will be drawn
            r = pygame.Rect(0,0,width,height)
            #Centres the rectangle
            r.center = (coordList[index][0],coordList[index][1])
            #Draws ellipse
            pygame.draw.ellipse(screen,col,r,weight)
    except IndexError:
        pass

def DrawPoint(coordList: list,index: int,col: tuple,weight: int,xMod: int,yMod: int):
    #Converts weight parameter from diameter to radius
    weight = math.ceil(weight/2)
    try:
        #Filters out untracked keypoints (if the keypoints are successfully tracked)
        if coordList[index][0] > 10*multiplier:
            #Applies x and y modifiers
            coordList[index][0] = coordList[index][0]+xMod
            coordList[index][1] = coordList[index][1]+yMod
            #Draws circle
            pygame.draw.circle(screen,col,coordList[index],weight)
    except IndexError:
        pass
```

Figure 18. The ‘DrawEllipse’ and ‘DrawPoint’ functions. Used to draw ellipses and circles.

The first two, ‘DrawEllipse’ and ‘DrawPoint’ can be seen in Figure 18. These are reasonably similar functions with slight differences.

‘DrawEllipse’ takes 6 parameters. ‘coordList’ is one of the lists generated by the lookup script defined earlier. So, either the list of body keypoints, facial keypoints, or one of the hands’ keypoint lists. The ‘index’ parameter is the index of the point within the ‘coordList’ of which the ellipse should originate from (it’s centre). The ‘col’ parameter is a 3-item tuple defining the RGB colour values of the colour in which the ellipse should be drawn ((255,255,255) would be pure white). The ‘weight’ parameter is the thickness of the ellipses outline if it is not to be filled. Any ‘weight’ value over 0 means the ellipse will not be filled and the ‘weight’ will be the line width in pixels. The ‘width’ and ‘height’ parameters each define the width and height of the ellipse in pixels. The PyGame ‘draw.ellipse’ function works by first defining a PyGame rectangle object that determines the bounds of the ellipse and then draws the ellipse within those bounds. DrawEllipse is used to draw the top of the head and also the ears.

‘DrawPoint’ also takes 6 parameters, ‘coordList’, ‘index’ and ‘col’ are all identical to ‘DrawEllipse’, in this case ‘weight’ refers to the diameter of the circle being drawn in pixels and both ‘xMod’ and ‘yMod’ refer to modifiers for the X and Y values. For example, if xMod is 1 the circle’s origin will be shifted 1 pixel right on the X axis. The ‘DrawPoint’ function is used to draw the eyes as well as many of the joints on the avatar.

```

#Draws a misc. polygon
def DrawPoly(coordList: list, pointsList: list, col: tuple, weight: int):
    #Initialises list of all x,y points to be drawn
    drawList=[]
    try:
        for i in pointsList:
            #Filters out untracked keypoints (if the keypoints are successfully tracked)
            if coordList[i][0] > 10*multiplier:
                drawList.append(coordList[i])
            #Draws polygon connecting all the vertices in drawList
            pygame.draw.polygon(screen,col,drawList,weight)
    except IndexError:
        pass
    except ValueError:
        pass

#Draws a misc line
def DrawLine(coordList: list, pointsList: list, col: tuple, weight: int):
    #Initialises list of all x,y points to be drawn
    drawList=[]
    try:
        for i in pointsList:
            if coordList[i][0] > 10*multiplier:
                #Filters out untracked keypoints (if the keypoints are successfully tracked)
                drawList.append(coordList[i])
            #Draws a line connecting all the vertices in drawList
            pygame.draw.lines(screen,col,False,drawList,weight)
    except IndexError:
        pass
    except ValueError:
        pass

```

Figure 19. The ‘DrawPoly’ and ‘DrawLine’ functions. Used to draw polygons and lines.

The ‘DrawPoly’ and ‘DrawLine’ functions are also similar to each other and can be seen in Figure 19. The ‘DrawPoly’ function takes 4 parameters, The first being ‘coordList’ which is the same as the previous two functions. It defines which keypoint list (generated by the lookup script) to use. The second is the ‘pointsList’ parameter. This is an ordered list of integers. These integers refer to the indexes of items within the ‘coordList’. As such each item in ‘pointsList’ refers to a keypoint within the ‘coordList’ and these are then connected in the order they appear in ‘pointsList’. This is necessary as when drawing polygons, the order the keypoints appear in the ‘coordList’ may not be the correct order in which the vertices need to go. As such they are re-ordered by using the ‘pointsList’ parameter. The next parameter is ‘col’ referring again to the colour of the polygon. And finally, ‘weight’. The ‘weight’ parameter in ‘DrawPoly’ follows the same rules as ‘DrawEllipse’ being that the value 0 means the polygon is filled and any value over 0 meaning that the polygon is unfilled and instead the value determines the line’s thickness. ‘DrawPoly’ is used extensively in the drawing of the avatar’s clothing as well as the bottom half of the head, the palms of the hands, whites of the eyes and the nose.

‘DrawLine’ uses the exact same parameters as ‘DrawPoly’ but draws a line between each specified keypoint. The ‘weight’ parameter strictly defines line thickness. ‘DrawLine’ is used for rendering the fingers, eyebrows, arms, and neck.


```
#Multiplies every point by the multiplier value
def Expand(pointsList):
    for i in range(len(pointsList)):
        pointsList[i][0]=pointsList[i][0]*multiplier
        pointsList[i][1]=pointsList[i][1]*multiplier
    return(pointsList)
```

Figure 20. The ‘Expand’ function. Used to multiply every keypoint by the multiplier value.

The next notable function is the ‘Expand’ function shown in Figure 20. This is used to handle the variable resolutions that Kyro can produce. The ‘Expand’ function takes all values within a keypoint list and multiplies it by the ‘multiplier’ variable which is determined by the user. This means that the position of the keypoints can be expanded to fit the different PyGame screen resolutions rather than being limited to the resolution of the input videos.

```
#Returns the distance between two points
def CalcDistance(p1: tuple, p2: tuple):
    #d=√((x2 - x1)² + (y2 - y1)²)
    return math.ceil(math.sqrt(((p2[0] - p1[0])**2)+(p2[1]-p1[1])**2))

#Returns midpoint between two points
def CalcMidpoint(p1: tuple, p2: tuple):
    #(x₁ + x₂)/2, (y₁ + y₂)/2
    return ((math.ceil((p1[0]+p2[0])/2),math.ceil((p1[1]+p2[1])/2)))
```

Figure 21. The ‘CalcDistance’ and ‘CalcMidpoint’ functions. Used to calculate the midpoint or distance between two keypoints.

Both the ‘CalcDistance’ and ‘CalcMidpoint’ functions are simple mathematical functions. ‘CalcDistance’ is used to return a value in pixels of the distance between two keypoints. It uses the following formula with x_{p1} referring to the x coordinate of point 1 etc.

$$d = \sqrt{(x_{p2} - x_{p1})^2 + (y_{p2} - y_{p1})^2}$$

(BYJU’s, n.d)

The ‘CalcMidpoint’ function is used to return a new keypoint exactly in between two given keypoints. It uses the following formula with $p3$ referring to the new point.

$$(x_{p3}, y_{p3}) = \left(\frac{x_{p1} + x_{p2}}{2}, \frac{y_{p1} + y_{p2}}{2} \right)$$

(Cuemath, n.d)

The entire rendering of the avatar is done using the ‘DisplayAvatar’ function, it uses all of the functions previously defined to generate the avatar seen in the outputs. Instead of displaying the entire function here (as it is very long and repetitive) I have created visualisations of what the ‘DisplayAvatar’ function does. Figure 22 shows some of the rendering methods used to display the avatar’s face.

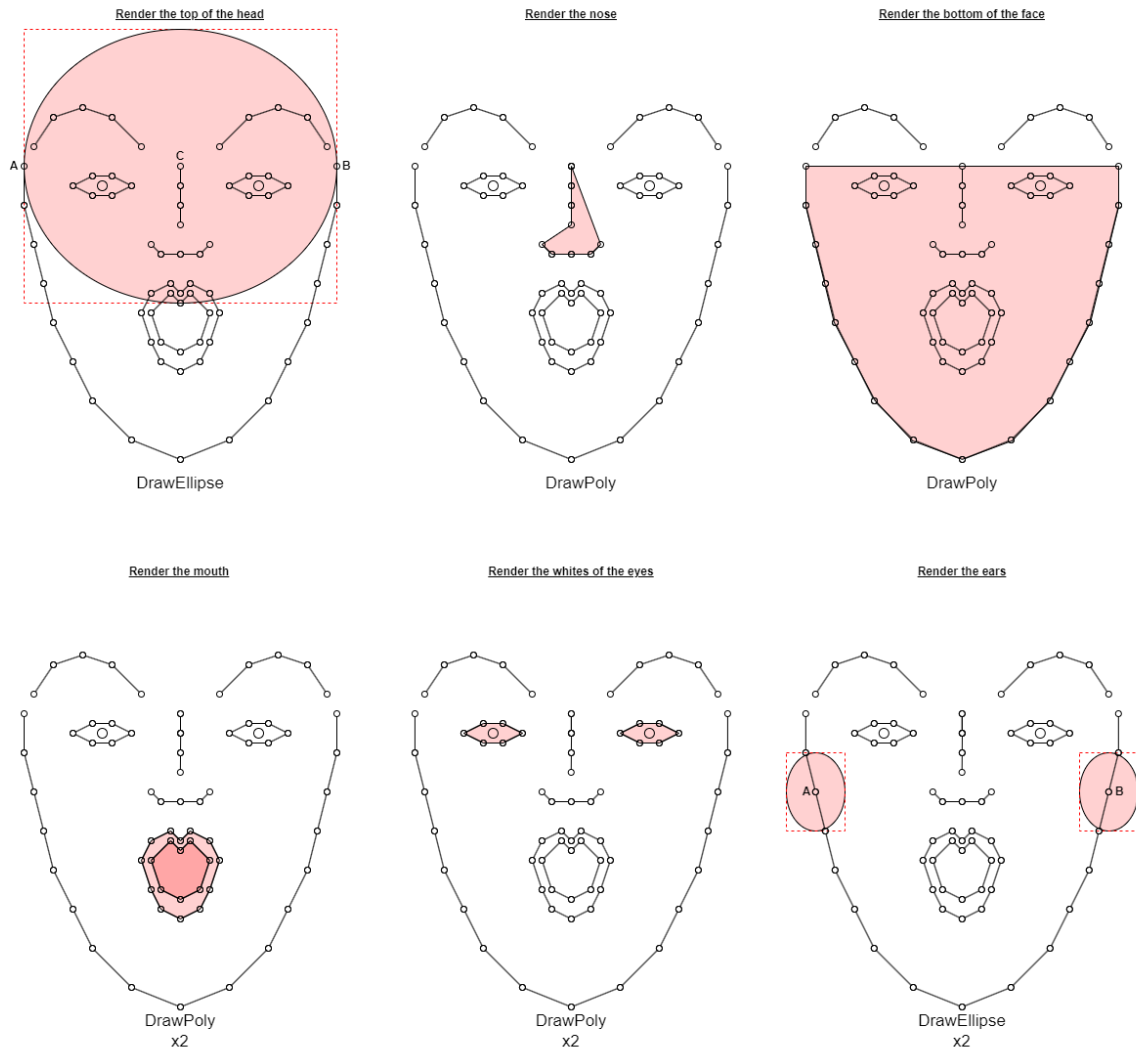


Figure 22. Visualisation of the ‘DisplayAvatar’ function rendering the face.

Refer back to Figure 17 to better understand the specific keypoints being used. The top of the head is rendered using the ‘DrawEllipse’ function, it first creates a rectangle, the width of which is calculated using the ‘CalcDistance’ function between points A and B on the diagram. It is then centred about point C, which is the midpoint between points A and B, calculated using the ‘CalcMidpoint’ function, the ellipse is then drawn within the constraints of said rectangle. All the uses of ‘DrawPoly’ are reasonably simple, they use the keypoints shown as the vertices of a polygon. This is used for the whites of the eyes, the bottom of the head, both the inside and lips of the mouth, and the eyes. The ears are fixed size ellipses drawn on the points A and B. As well as these, the eyebrows are drawn with the ‘DrawLine’ function and the irises, pupils and eye reflection are each drawn from the centre keypoints in the eye using the ‘DrawPoint’ function with differing parameters.

The hands are drawn very simply, referring back to Figure 16, the ‘DrawPoly’ function is used to connect the 0th, 1st, 2nd, 5th, 9th, 13th and 17th keypoints creating a polygon that fills the palm of the hand in one colour. From this each finger is drawn using the ‘DrawLine’ function. As well as this a circle is drawn on every hand keypoint, this smooths out the lines drawn by the ‘DrawLine’ function by rounding the ends of the lines drawn.

```

def DrawClothes():
    #--Display Body--
    try:
        lHip = (((width/2) + bodyKeypointList[2][0])/2,height)
        rHip = (((width/2) + bodyKeypointList[5][0])/2,height)
        lBicep = CalcMidpoint(bodyKeypointList[2],bodyKeypointList[3])
        rBicep = CalcMidpoint(bodyKeypointList[5],bodyKeypointList[6])
        lHigh = CalcMidpoint(bodyKeypointList[2],lBicep)
        rHigh = CalcMidpoint(bodyKeypointList[5],rBicep)
        mCore = CalcMidpoint(bodyKeypointList[1],(width/2,height))
        mFloor = (bodyKeypointList[1][0],height)
        mStomach = CalcMidpoint(mCore,mFloor)
        mNeck = CalcMidpoint(bodyKeypointList[0],bodyKeypointList[1])
        mNeckLower = CalcMidpoint(mNeck,bodyKeypointList[1])
        #See diagram
        clothingKeypointList=[bodyKeypointList[2],lHigh,lBicep,bodyKeypointList[5],rHigh,rBicep,mCore,mStomach,mFloor,lHip,rHip,mNeck,mNeckLower]
        DrawPoly(clothingKeypointList,[0,9,10,3],black,2*multiplier)
        DrawPoly(clothingKeypointList,[0,1,7,4,3],black,2*multiplier)
        DrawPoly(clothingKeypointList,[0,2,3],black,2*multiplier)
        DrawPoly(clothingKeypointList,[0,5,3],black,2*multiplier)
        DrawLine(clothingKeypointList,[0,3],black,22*multiplier)
        for i in range(2,7):
            DrawPoint(bodyKeypointList[i],black,22*multiplier,0,0)

        DrawPoly(clothingKeypointList,[0,9,10,3],red,0)
        DrawPoly(clothingKeypointList,[0,1,7,4,3],red,0)
        DrawPoly(clothingKeypointList,[0,2,3],red,0)
        DrawPoly(clothingKeypointList,[0,5,3],red,0)
        DrawLine(clothingKeypointList,[0,3],red,20*multiplier)
        for i in range(2,7):
            DrawPoint(bodyKeypointList[i],red,20*multiplier,0,0)
    except IndexError:
        pass

```

Figure 23. The ‘DrawClothing’ function

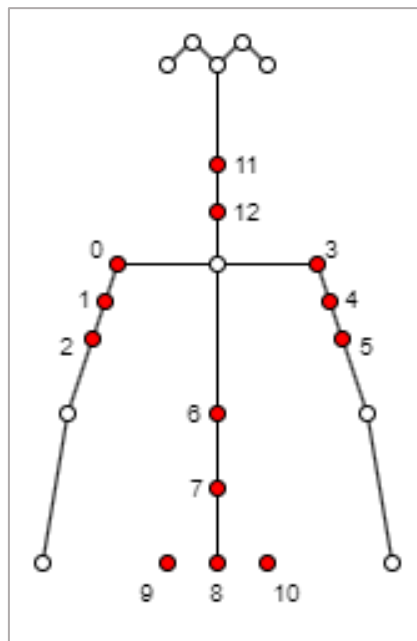


Figure 24. Visualisation of ‘clothingKeypointList’.

In order to render the clothing to the body more keypoints had to be generated, the ones in ‘bodyKeypointList’ aren’t enough to create realistic clothing. As such a new keypoint list, ‘clothingKeypointList’, had to be generated. This extensively used the ‘CalcMidpoint’ function to generate the new keypoints seen in Figure 24. It should be noted that keypoint 8 shown in Figure X is not the same as the keypoint 8 shown in Figure 15. In ‘clothingKeypointList’ the base of the avatar (point 8) is fixed at the bottom-centre of the screen. This is because OpenPose didn’t always pick up point 8 when tracking the input videos as this keypoint was just out of frame.

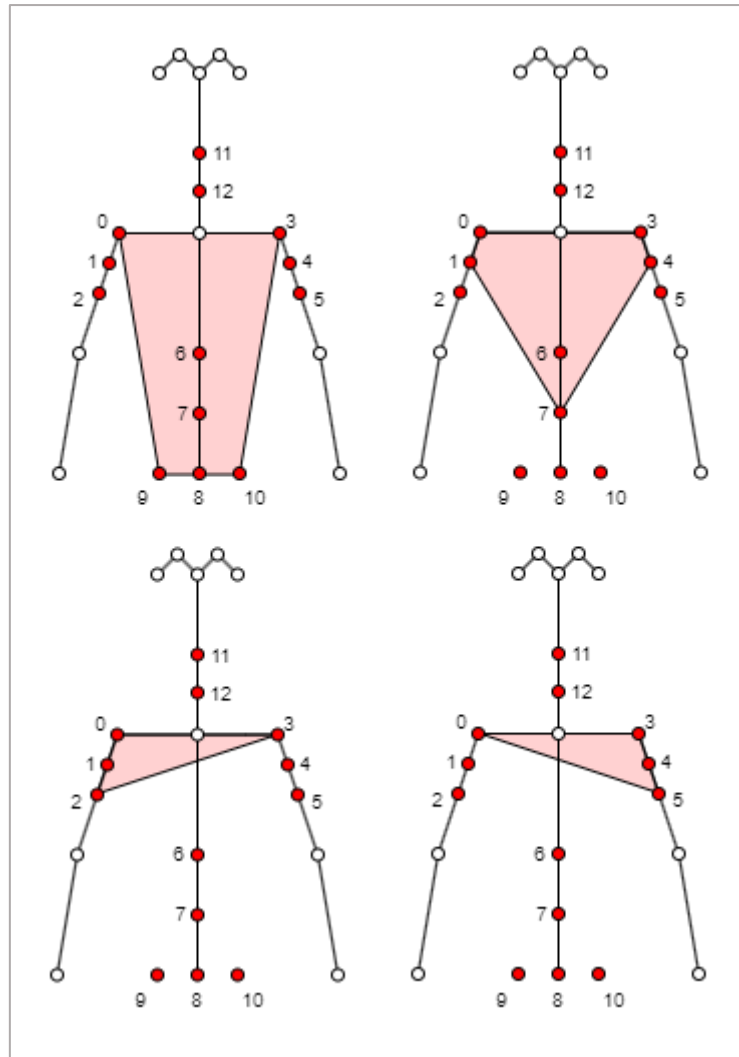


Figure 25. Diagram of how the clothing is drawn with the ‘DrawPoly’ function.

The ‘DrawClothing’ (called within ‘DisplayAvatar’) function uses these newly generated keypoints in order to draw the shirt onto the avatar. The ‘DrawPoly’ function is used to draw the four polygons shown in Figure 25.

The only other draw functions of note are ‘drawLine’ used to draw both the neck and the arms. This is done by connecting the specific keypoints from ‘bodyKeypointList’.

For all objects that have a black outline in the final output, these draw functions have been run twice, first in black and slightly larger, then the draw functions described are drawn on top, this gives the illusion of the objects being outlined in black. This causes the ‘DisplayAvatar’ and ‘DrawClothing’ functions to be very long and repetitive and leads to poor readability of the code. This is difficult to remedy as PyGame has no in-built way to add borders or outlines to drawn objects (Pygame, 2021).

7.8.2.2 Design Choices

The design of the character went through a few phases, originally the character was going to be much more ‘cartoonish’ however this approach was scrapped as it made facial expressions difficult to read. As well as this the hands of cartoon characters are usually large and have thick fingers, not ideal for displaying BSL signs. As such it was determined that a more realistic approach would be better suited.

The final design of the avatar is strongly inspired by the AI-powered interactive story game *Faade* (*Playable, n.d*). This design was decided upon for its simplicity, the simple design allowed it to be easily recreated using PyGame whilst keeping resource costs low and maintaining the readability of facial expressions and hand movements. In *Faade* the characters react to what the player says, showing emotions in, not only their dialogue, but their facial expressions too. The characters of *Faade* are semi-realistic without triggering the ‘uncanny valley’ effect. This is where realistic representations of humans appear creepy, eerie, or uncomfortable when they become too close to reality (*Wong, S. 2017*). The simplistic nature of the character’s features allows this not to be the case whilst still maintaining all the required facial information to express emotion. For these reasons a similar style of character was ideal for this application.



Figure 26. Image of characters Trip and Grace from videogame *Faade* (*Playable, n.d*)

7.8.2.3 Visual Iterations

The avatar’s visuals went through many evolutions and improved incrementally with the development process.

Originally, only the keypoints were drawn to the screen to make sure that they had been extracted correctly from the JSONs. Only the body keypoints were displayed as below:



Figure 27. PyGame screen displaying only body keypoints as circles.

This was simple, but successfully visualised the keypoints in space and showed that they were being fetched correctly by the lookup script. The next step was to add the hand and facial keypoints, as below:



Figure 28. PyGame screen showing all keypoints as circles.

Figure 28 shows what data there was to work with. The dots seen here are the entire set of keypoints generated by OpenPose. These are rather limited when faced with creating an entire avatar from just these points. For example, there are no keypoints for the top of the head, the ears, or any definition of the limbs. These values had to all be estimated to create a realistic avatar.

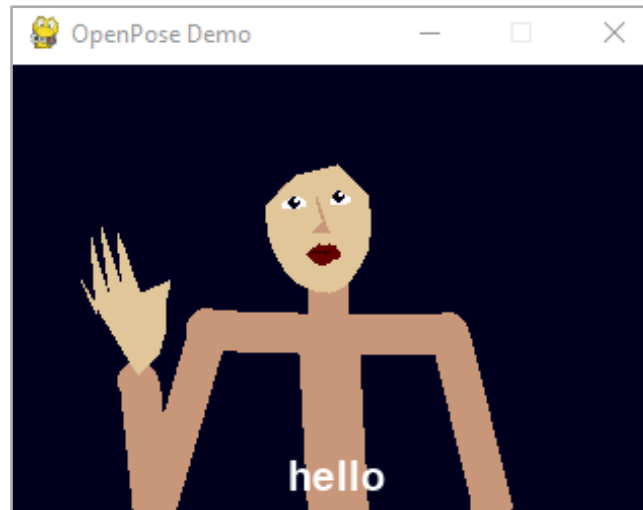


Figure 29. PyGame screen showing first draft of facial features, body, and hands.

Figure 29 shows the first attempt at making the character somewhat humanoid. The facial features displayed here are comparable to those seen in the final render and an attempt at generating the top of the head has been made, however, the head is sharp as only two new keypoints had been generated. PyGame's `draw.polygon` function was used on the hands (simply using every hand keypoint as the vertex of a polygon) as a 'quick and dirty' placeholder for a real, 'finger by finger' hand rendering. As well as this, text was added to the screen displaying the current word being signed. This stage was a significant milestone in the development as the rendered output was starting to look like a usable product. Signs could be interpreted from the avatar successfully if they weren't too reliant on complex finger movements.

In fact, this was the stage where I began to inadvertently learn some BSL signs when testing the system.

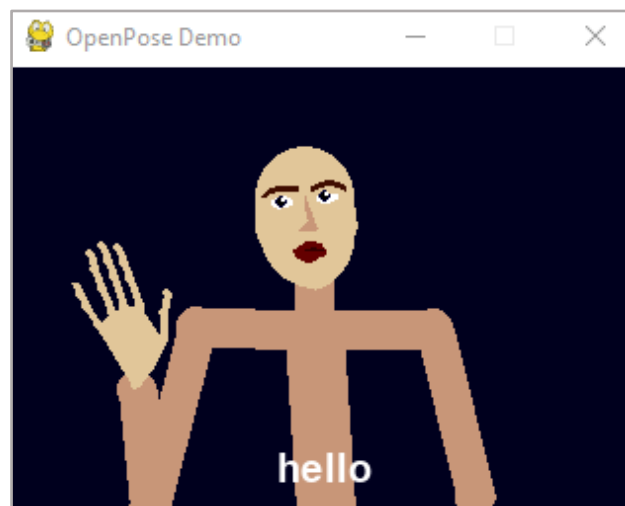


Figure 30. PyGame screen showing more complex hands and facial rendering.

Figure 30 displays further visual improvements, such as the rendering of eyebrows and individual fingers. As well as this the top of the head was changed to be a circle, originating from the top of the character's nose. The changes to the head shape drastically improved the avatar's look, however, the circle was of a fixed size and therefore didn't scale or change with perspective, for example when the head is rotated. This created some visual glitches where the top of the head wouldn't connect flush with the jaw.

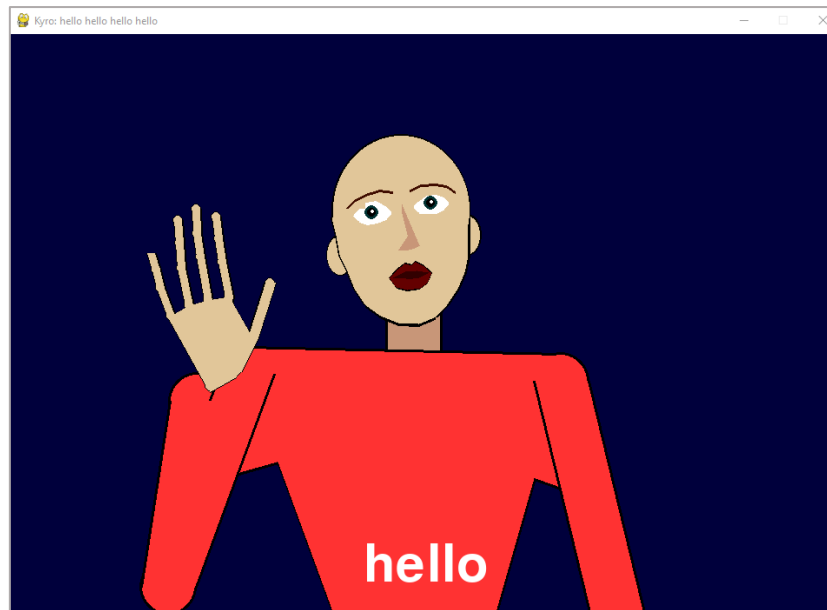


Figure 31. PyGame screen showing the final look of the avatar.

Figure 31 shows the final look of the system. Almost every aspect of the visual was tweaked or refined, ears were added, a black outline is drawn to improve contrast and help with readability of the fingers. The top of the head now scales dynamically with the distance between the jaw keypoints and is therefore always connected and accurate. The body is now rendered with clothing. And the PyGame screen is scalable to higher, variable resolutions determined by the user.

7.8.3 Convert to Video

The `pygame_recorder` library (*tdrmk, 2019*) was used to record the PyGame screen. It takes a screenshot of the PyGame screen every frame then compiles them into a video once the program is complete. The library is first imported into the program, then a 'recorder' object must be made, this object takes the width and height of the screen (note the recorder object is created after the resolution multiplier is defined and therefore the video scales with the screen's resolution) and the FPS of the output video (30 in this case as the input videos are at 30 FPS). Then after each frame the `capture_frame` function is called which takes the PyGame screen object as a parameter (defining which PyGame screen to capture) and finally, at the end of the program, before the quit function is called, the `end_recording` function is called which collates all the previous frames and outputs them to a video file. The library outputs the video as a .avi file which is a widely used video format and a standard on the Windows OS (*Microsoft., 1998*).

7.9 System Validation

Due to the nature of the system being developed normal testing procedures couldn't be used in order to validate the system, as such, visual validity tests were done in order to validate certain aspects of the system. For example, as can be seen in Figure 27, in order to test that the correct keypoints had been extracted from the JSON files they were drawn to the PyGame screen. This allowed validation of a large set of keypoints all at once. Significant outliers from the expected data would be easy to see and would stand out from the surrounding keypoints.

Also throughout, to test specific functions, various prints were made to the console in order to display where problems arose from. This allowed me to pinpoint issues easily and therefore fix bugs and problems before continuing with development.

8. Evaluation

8.1 Requirement Evaluation

In order to evaluate the effectiveness of the final system the requirements defined earlier can be revisited. By doing so we can determine how effective the final system was and if it reached the goals set for it.

8.1.1 Functional Requirements

Below is the table of functional requirements and their status, whether they have been Completed, Not Completed or Partially Completed.

Table of Functional Requirements		
F-UR 1	Tokeniser	Status
F-UR 1.1 Must	The tokeniser must take the input text and split it into words (or tokens),	Completed
F-UR 1.2 Must	The tokeniser must then look up the tokens in the dictionary,	Completed
F-UR 1.3 Could	The tokeniser could recognise multi-word signs (single signs that depict more than one English word),	Not Completed
F-UR 1.4 Could	The tokeniser could take other options rather than just plain text as input (.pdf, .docx, web links etc.)	Not Completed
F-UR 2	Converter	
F-UR 2.1 Must	The converter must convert the sign videos to JSON format,	Completed
F-UR 3	Dictionary	
F-UR 3.1 Must	The dictionary must be an object capable of holding sign words and their JSON ,format	Completed
F-UR 3.2 Must	The dictionary must contain the BSL finger sign alphabet and their JSON representations,	Completed
F-UR 3.3 Should	The dictionary should contain 10% of the SignBSL.com dictionary and their JSON representations,	Completed
F-UR 3.4 Could	The dictionary could contain 100% of the SignBSL.com dictionary and their JSON representations,	Partially completed
F-UR 3.5 Will Not	The dictionary will not hold sign representations for any sign languages other than BSL,	N/A
F-UR 4	Assembler	
F-UR 4.1 Must	The assembler must fetch the required JSON data from the dictionary for the input text,	Completed
F-UR 4.2 Must	The assembler must animate an avatar using the fetched JSON data ,	Completed
F-UR 4.3 Should	The assembler should automate the avatar animation process without intervention,	Completed
F-UR 4.4 Could	The assembler should blend the generated animations together so transitions between them are smooth,	Not Completed
F-UR 4.5 Could	The assembler could display the corresponding word with the sign	Completed

F-UR 4.6 Must	The assembler must be able to render out the generated animation in an accessible video format (.mp4, .avi, .mov, etc.),	Completed
F-UR 5	User Interface	
F-UR 5.1 Must	The UI must be, at least, presented in an easy to understand, text-based format	Completed
F-UR 5.2 Could	The UI could be graphical and work independently as a runnable .exe	Not Completed
F-UR 5.3 Will Not	The application will not be web integrated	N/A

Table 6. Table of functional requirements, their descriptions and status.

Here in Table 6, we can see that all of the necessary functional requirements (Must) were met, including some of the ‘should’ requirements. However, both requirements 4.4 and 5.2 were uncompleted and would greatly improve the end program. These are only ‘could’ requirements, but the usability of the final program is significantly hindered by their lack of completion.

8.1.2 Non-Functional Requirements

Below is the table of non-functional requirements and their status, whether they have been Completed, Not Completed or Partially Completed.

Table of Non-Functional Requirements		
NF-UR 1	Performance	Status
NF-UR 1.1 Must	The application must be able to run on high-end consumer technology (Workstation/Gaming PC),	Completed
NF-UR 1.2 Could	The application could be able to run on low/mid-end consumer technology (Mid-range Laptop),	Completed
NF-UR 1.3 Will Not	The application will not be able to run on mobile,	N/A
NF-UR 1.4 Will Not	The application will not be able to run on any non-windows OS,	N/A
NF-UR 1.5 Must	The application must be able to complete processing (text input to output render) in 30 seconds per word,	Completed
NF-UR 1.6 Should	The application must be able to complete processing (text input to output render) in 5 seconds per word,	Completed
NF-UR 1.7 Could	The application must be able to complete processing (text input to output render) as fast as the words would be spoken (sub 1 second per word),	Partially Completed
NF-UR 1.8 Could	The application could predict how long processing will take and report this to the user	Not Completed

Table 7. Table of non-functional requirements and their descriptions and status.

Here in Table 7, we can see that, again, all the essential non-functional requirements have been met as well as all the ‘should’ requirements. Requirement 1.7 is marked as partially completed as when no output video is being generated the assembler runs at sub 1 second per word and is very smooth,

however the PyGame recorder library is quite taxing and causes the program's real-time render to slow down. This slowdown isn't displayed in the output video which still will run at a smooth 30 FPS.

8.1.3 Results

It is clear to see that the final program is an extremely usable product by the fact that it successfully completes all of the necessary requirements set out for it both functional and non-functional.

8.2 Visual Quality Evaluation

In order to evaluate the visual quality of Kyro a questionnaire was devised. Participants of the evaluation are fellow Computer Science students from Heriot-Watt. The sample size is 8 participants. This evaluation attempts to return data on how successful Kyro's visual output of the signs are.

8.2.1 Methodology

The participants will first read and understand the Information Sheet), this will supply the participant with information on what the evaluation entails and why they have been chosen. After doing so they will complete a consent form consenting to taking part in the evaluation. Both the information sheet and consent form are supplied in the appendix of this document. These are both modified versions of Heriot Watt's field research consent form (*Heriot-Watt, n.d*) and 'information for participants in academic research projects' form (*Heriot-Watt, n.d*). Then they will watch a selection of videos output by the system and complete the questionnaire. The results will be entirely anonymised, and the participants will be able to remove themselves from the study at any time.

8.2.2 Design

The questionnaire is designed to produce both qualitative and quantitative data, as such each question has both a 1-5 likert response and the ability for the participant to expand on their reasonings. The design is as follows:

1. How engaging did you find the animations of the avatar?

Not engaging at all	1	2	3	4	5	Extremely engaging
---------------------------	---	---	---	---	---	-----------------------

2. Anything else you'd like to add about the engagingness of the animations? (Answer is not required)

--

3. How clearly could you see the movements of the avatar's hands and arms during the signing?

Not clearly at all	1	2	3	4	5	Extremely clearly
-----------------------	---	---	---	---	---	----------------------

4. Anything else you'd like to add about the clarity of the animations? (Answer is not required)

--

5. How visually appealing did you find the overall design and colour scheme of the avatar?

Extremely unappealing	1	2	3	4	5	Extremely appealing
--------------------------	---	---	---	---	---	------------------------

6. Anything else you'd like to add about the appeal of the avatar? (Answer is not required)

--

7. How easy was it to understand what the avatar was signing, based solely on the visual aspect of the animations?

Not understandable at all	1	2	3	4	5	Extremely easy to understand
---------------------------------	---	---	---	---	---	------------------------------------

8. Anything else you'd like to add about the readability of the avatar? (Answer is not required)

--

9. How satisfied were you with the level of detail and nuance in the avatar's facial expressions during the signing?

Not satisfied at all	1	2	3	4	5	Extremely satisfied
----------------------------	---	---	---	---	---	------------------------

10. Anything else you'd like to add about the facial animation of the avatar? (Answer is not required)

--

11. How smoothly did the avatar's signing movements transition from one sign to another?

Not smoothly at all	1	2	3	4	5	Extremely smoothly
---------------------------	---	---	---	---	---	-----------------------

12. Anything else you'd like to add about the transitions between signs? (Answer is not required)

--

13. Anything else you'd like to add in general? (Answer is not required)

--

8.2.3 Results

The questionnaire returns both qualitative and quantitative data, the odd numbered questions (minus question 13) are all the likert responses and the even questions (plus question 13) are all the long text answers. These questions will be reviewed separately.

8.2.3.1 Quantitative

The likert responses have been visualised on the stacked bar graph below, the stacked bars represent the value of the responses, and their height reflects that also, the bars to the right of each stacked bar display the percentage of participants that gave that value response.

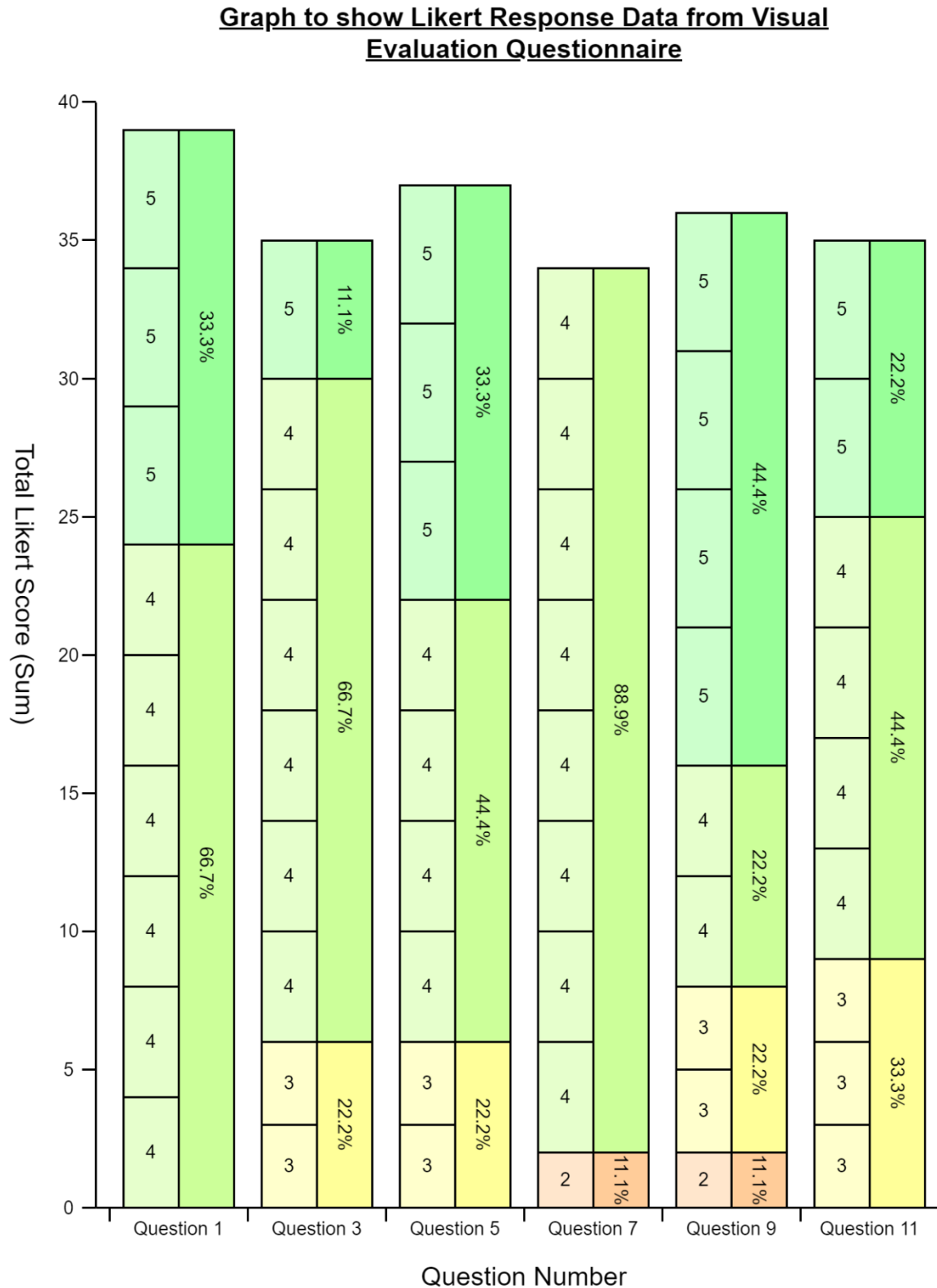


Figure 32. Graph to show likert response data from visual evaluation questionnaire.

Figure 32 shows us that the responses to the questionnaire were mostly positive, with only 4 responses in total below the neutral score of 3. Overall, this shows us that the visual aspect of the application was received well, and the animations were readable and appealing to the participants. It is worth

noting that, as the participants are not BSL users, this evaluation only draws conclusions about purely the visual appeal of the program's output.

8.2.3.2 Qualitative

This questionnaire produced some beneficial qualitative data about the Kyro's visual quality. For example, two users noted that the hand/finger movements were difficult to read stating "The fingers are hard to follow" and "Hard to distinguish the fingers movements sometimes". This is most likely due to the quality of the finger keypoints generated by OpenPose as there is a very common failure case when keypoints overlap (Cao *et al.*, 2018). This sometimes causes the hands to visually glitch or for fingers to be mixed up. However, it should be noted that another user stated that "the fingers are distinct to recognise" so at least in some cases the finger movements are easy to follow. Many of the participants also noted the appeal of the avatar stating, "it's pleasant", "expressive", "very clear" and "human-like", this coincides with the likert responses gathered and again, goes to show that the avatar is attractive and engaging for the large majority of users.

8.3 Performance Evaluation

In order to evaluate the performance of the program the main function will be timed using the python time library. This function will then be timed on differing hardware, with or without generating output videos. This will give us insight into which parts of the program could be optimised and what is potentially causing bottlenecks.

8.3.1 Methodology

PyGame isn't GPU accelerated and therefore the quality of the GPU shouldn't affect the performance. Instead, the CPU is used for rendering. Therefore, in order to test the performance of the system both a mid to high-end CPU (AMD Ryzen 5 3600), that is likely to be used in a desktop PC, and a low-end mobile CPU (Intel Core i3-7100U), which is likely to be used in a laptop, will be compared. As well as this, running the program on both a high-end SSD (Samsung SSD Pro) and a low-end HDD (Western Digital 5400RPM) will be compared. This is due to the JSON's being fetched from the disk therefore the disk speed may have an effect on performance. These tests should allow us to make a conclusion as to what may cause slowdowns in the program and what potential optimisations could be made in order to improve the system's performance.

Each test will be carried out 5 times and an average mean time will be calculated for comparison.

9.3.2 Results

The original length of the 'hello' sign video taken from SignBSL.com was 1.326s, the closer Kyro gets to this value, the more realistic the sign movements.

First is the comparison of the SSD to the HDD.

Iteration	SSD		HDD	
	Video Out	No Video Out	Video Out	No Video Out
1	1.728s	1.363s	1.733s	1.360s
2	1.710s	1.358s	1.730s	1.359s
3	1.706s	1.361s	1.731s	1.356s
4	1.709s	1.360s	1.737s	1.353s
5	1.708s	1.357s	1.785s	1.363s
Mean	1.712s	1.360s	1.741s	1.358s

Table 8. Table showing time taken when running Kyro on both an SSD and an HDD.

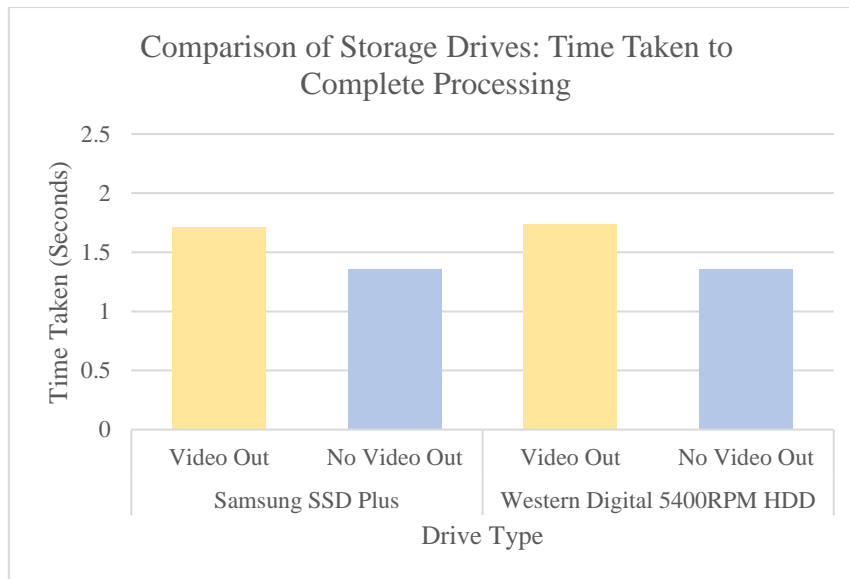


Figure 33. Grouped bar chart displaying storage drive comparison results.

As seen in figure X the program displayed no significant differences in time taken when run on either an SSD or an HDD. With even the HDD running marginally faster on average for the ‘No Video Out’ tests. The difference in time taken for each case is less than 5% with a difference of 1.68% between the ‘Video Out’ tests and a miniscule difference of 0.147% between the ‘No Video Out’ tests. The results show that the program will run the same no matter the quality of the hard drive in the system. Or even if there is a difference it is so marginal that it will not affect the usability of the software.

The next test is the comparison between the Ryzen 5 3600 and the Intel i3-7100U CPUs.

	Ryzen 5 3600		i3-7100U	
Iteration	Video Out	No Video Out	Video Out	No Video Out
1	1.728s	1.363s	2.237s	1.608s
2	1.710s	1.358s	2.163s	1.594s
3	1.706s	1.361s	2.183s	1.608s
4	1.709s	1.360s	2.173s	1.643s
5	1.708s	1.357s	2.208s	1.607s
Mean	1.712s	1.360s	2.193s	1.612s

Table 9. Table showing time taken when running Kyro on both a Ryzen 5 3600 and a i3-7100U.

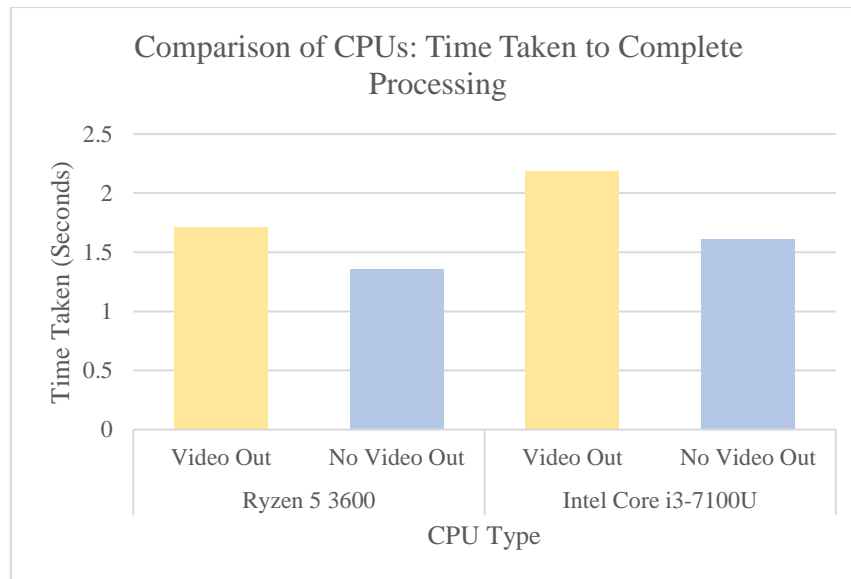


Figure 34. Grouped bar chart displaying CPU comparison results.

These results show that although the difference in CPU speeds does have an obvious effect on performance, the difference doesn't make the system unusable. Using the low-spec CPU, even whilst producing the video output, the time taken to display the sign and save the sign video as output is 49% slower than the original input video. This is definitely noticeable at runtime, however, as the 'pygame_recorder' library works by taking a screenshot every frame and compiling them into a video at the end, the output video that is saved to disk is unaffected by slowdowns and therefore the program is still usable.

8.4 BSL User Evaluation

In order to evaluate the effectiveness of the system in use, a BSL interpreter was asked about their opinions on the software in an open interview style, the use of an actual BSL participant allowed for evaluation of the clarity and accuracy of the signs as this participant uses these signs daily as opposed to the participants who took part in the visual evaluation. The same information form and consent form was provided (*Appendix Item 1*). Some of the results of their opinions are shown below:

8.4.1 Results

The participant stated that the animations were "attractive and modern". This coincides with the results from the visual evaluation and shows that Kyro's design is pleasant and is usually responded to positively. The participant also stated that the visuals were "jumpy" and "not easy to watch". This is most likely due to some of the questionable keypoint detection by OpenPose that sometimes places keypoints in unusual locations causing immersion breaking visual jumps where a keypoint may move too far in a single frame. The main issue, which was mentioned multiple times, was with the semantics of the signs. For example, the participant stated "The sign used for 'little' is incorrect. This sign means little in the sense of 'not much' (i.e. 'I have a little sugar left') - it doesn't mean little in terms of size.". They also stated: "The sign used for 'to' as in 'to eat', means 'to' as a preposition (i.e. 'I am going to the cafe') - it doesn't mean 'to' as in the infinitive of a verb. Similar issue with 'to' in the phrasal verb 'listen to' - it is also signed as a preposition.". These semantic issues are due to the lack of context detection in Kyro's implementation. Kyro's dictionary only contains a single sign representation per word. As such it has no way of determining the difference between a geological 'rock' and 'rock' music.

9. Conclusions

9.1 Achievements

As seen in the evaluation of my system, Kyro successfully completes all of the necessary functional requirements originally set out for it, it successfully covers 10% of the SignBSL.com alphabet (2,185 available words of the ~21,000 on SignBSL.com (*SignBSL, n.d*)), creates a real-time visualisation of an avatar signing in Sign Exact English, and renders out in a widely used video format. It also manages to fulfil all the required non-functional requirements, completing processing at a similar speed to the input videos therefore, it can be determined that the project as a whole was successful.

The system, as seen from the performance evaluation, also works on low spec computers and therefore its accessibility is increased. With Kyro being accessibility software the importance of this is paramount and the fact that it can run on lower end machines is a significant merit as more people can access and use it. As well as this a significant advantage of the performance of the system is the fact that the video output is unaffected by the actual time taken to run. Therefore, even on extremely low spec machines, the processing may take some time, but the output videos will always be clean and smooth.

As can be determined from the visual evaluation the general response to the looks of the system is positive with users finding the animations, clear, enjoyable, and engaging.

9.2 Limitations

However, Kyro is usable it still has some significant limitations, first and foremost being its primitive handling of BSL syntax. BSL uses different syntax than spoken English, this means that the word order of BSL isn't the same as English. This isn't accounted for and therefore the readability of the sentences Kyro produces suffers. Instead of BSL Kyro produces sentences in Sign Exact English (SEE) using BSL signs. This is where the signs are signed in normal spoken English word order. SEE isn't as widely used for communication as it's counterparts such as BSL or ASL (*Stephenson, 2020*), however, is still used in the teaching of hearing-impaired individuals English. This limits the use of Kyro as a communicative tool, however, should still allow improved understanding of texts for hearing-impaired individuals.

As well as the word order there is another problem with BSL's syntax that Kyro doesn't account for is the lack of certain word signs. For example, BSL doesn't have signs for some extremely common words like 'the' or 'is'. When signing BSL these words are omitted but the intent of the sentence is still held. However, this poses a problem for a translator as English text contains these words frequently. Currently Kyro will fingerspell any word not available in it's dictionary rather than removing them from the signed sentence, even when this may be a more fluent, natural approach.

A major issue with Kyro's current implementation is also context-free, this means that the program cannot determine the context of the current word being signed. This was mentioned extensively in the BSL user evaluation. For example, the sign for 'I' referring to self and 'I' referring to the letter are the same or 'rock' referring to a stone and 'rock' referring to a music genre, there is only one representation of each word in the dictionary that has been created for Kyro. Whereas, in BSL, there is a separate sign for each. This causes significant inaccuracies in the visualisation of the signs as incorrect signs are displayed quite often. This wouldn't necessarily be a problem if the user is fluent in both BSL and English as they would still understand the meaning, however, as this system is designed to help improve comprehension for those who may not fully understand English this is a significant problem.

Kyro also suffers from visual glitches; these are due to the quality of the keypoints produced by OpenPose which are sometimes inaccurate putting limbs in unusual and unnatural positions. This is mentioned in the BSL user evaluation stating that the "fingerspelling [of] 's' and 'l' is visually

indecipherable”. Usually, the visual glitches are not bad enough to break the readability of the sign, however, with signs containing complex finger positioning, and/or overlapping keypoints the signs can sometimes become unreadable. These traits are very common in the BSL alphabet and therefore Kyro’s fingerspelling ability is quite poor. This could be potentially alleviated with future developments in OpenPose.

9.3 Future Work

Although Kyro is a usable system, it could be notably improved by additions or developments in the future.

A significant improvement to the system would be the implementation of context detection to detect the semantics of each word. This can be done with various Natural Language Processing (NLP) methods many of which have been developed for chat-bot / conversational agent use cases and theoretically could be applied to Kyro in order to display the semantically correct sign based off the context of the sentence as a whole. This would greatly improve the accuracy of the results produced by Kyro as the semantic issues were the main negative brought up in the BSL user evaluation.

Using maximum accuracy when generating the OpenPose keypoints would be ideal and would significantly increase the quality of the outputs, as well as the render being smoother and more accurate it would also mitigate the visual bugs seen in the final system. Visual issues such as the fingers being ‘hard to distinguish’ as brought up in the visual evaluation and the ‘jumpy’ animations as brought up in the BSL user interview could be greatly mitigated by the use of maximum accuracy keypoints. This is a very simple improvement that could be made to the system easily, it would simply require a high-end GPU with 16GB VRAM (*Cao, 2018*) and a significant amount of time to run the `getKeypoints.py`.

As well as this, I think using a 3D avatar would be a great improvement. This could be done with current OpenPose functionality by refilming the input sign videos with an FLIR camera (or a multiple cameras approach) in order to generate depth data for the keypoints. This would allow data to be transposed onto a hyper realistic 3D model and therefore create more natural believable output animations.

Bibliography

- [1] Ben Yahia, N. and M. Jemni (2013). Animating signing avatar using descriptive sign language, IEEE.
- [2] British Computer Society (2022) BCS Code of Conduct. Available at: <https://www.bcs.org/membership-and-registrations/become-a-member/bcs-code-of-conduct/> (Accessed: 29/11)
- [3] BSL Signbank (2014) The BSL SignBank Dictionary. Available at: <https://bslsignbank.ucl.ac.uk/about/dictionary/> (Accessed: 29/11)
- [4] BYJU's. (n.d.). Distance Between Two Points Formula. BYJU's. <https://byjus.com/maths/distance-between-two-points-formula/#:~:text=The%20formula%20to%20find%20the,coordinate%20plane%20or%20x%20Dy%20plane>
- [5] Cao, Z., Hidalgo, G., Simon, T., Wei, S. E., & Sheikh, Y. (2018). OpenPose Documentation. Carnegie Mellon University. Retrieved from <https://github.com/CMU-Perceptual-Computing-Lab/openpose/documentation/>
- [6] Cao, Z., Hidalgo, G., Simon, T., Wei, S.E. (2018). OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields. IEEE Transactions on Pattern Analysis and Machine Intelligence, 43(1), 172-186. doi: 10.1109/TPAMI.2018.2850734
- [7] Cuemath. (n.d.). Midpoint Formula. Cuemath. <https://www.cuemath.com/geometry/midpoint/>
- [8] Das Chakladar, D., et al. (2021). "3D Avatar Approach for Continuous Sign Movement Using Speech/Text." Applied sciences 11(8): 3439.
- [9] Ebling, S. and J. Glauert (2013). Exploiting the Full Potential of JASigning to Build an Avatar Signing Train Announcements.
- [10] Harper, D. (n.d.). Chiro- | Etymology Online. Retrieved from <https://www.etymonline.com/word/chiro->
- [11] Heriot-Watt University. (n.d) CONSENT FORM FOR FIELD RESEARCH. https://www.ros.hw.ac.uk/bitstream/handle/10399/2855/LoK-KJ_0515_std%20%2826%29.pdf?sequence=28&isAllowed=y
- [12] Heriot-Watt University. (n.d) YOUR PERSONAL DATA:INFORMATION FOR PARTICIPANTS IN ACADEMIC RESEARCH PROJECTS. <https://www.hw.ac.uk/uk/services/docs/information-governance/PrivacyNoticeResearch-V4Finalversion.pdf>
- [13] Intellectual Property Office (2014) Exceptions to Copyright. Available at <https://www.gov.uk/guidance/exceptions-to-copyright#non-commercial-research-and-private-study> (Accessed: 29/11)
- [14] Intersoft Consulting (2018) General Data Protection Regulation. Available at: <https://gdpr-info.eu/> (Accessed: 29/11)
- [15] Joy, J. (2021). "SignText: a web-based tool for providing accessible text book contents for Deaf learners."
- [16] Liu, J. Q. (2021). OpenMMD. GitHub. <https://github.com/peterljq/OpenMMD>
- [17] McConnell, M. (2020). "Two Dimensional Sign Language Agent."
- [18] Microsoft Corporation (1998). WAVE and AVI Codec Registries - RFC 2361. IETF. doi:10.17487/RFC2361.
- [19] Playabl Studios. (n.d.). Façade. Playabl Studios. <https://www.playablstudios.com/facade>
- [20] Price, E.C. (n.d.). Wordlist.10000. Massachusetts Institute of Technology. <https://www.mit.edu/~ecprice/wordlist.10000>

- [21] Pygame Contributors. (2021). Pygame Documentation. 2.0.1. Pygame.
<https://www.pygame.org/docs/>
- [22] Python Software Foundation. (2021). os — Miscellaneous operating system interfaces. Python 3.10.0 documentation. Python Software Foundation.
<https://docs.python.org/3/library/os.html>
- [23] Requests Team. (2022). Requests: HTTP for Humans™ documentation. Latest version. Read the Docs.
<https://requests.readthedocs.io/en/latest/>
- [24] Scotland's Census (2011) 2011 Census: Key Results from Releases 2A to 2D. Available at: <https://www.scotlandscensus.gov.uk/documents/2011-census-key-results-from-releases-2a-to-2d/> (Accessed: 29/11)
- [25] SignBSL. (n.d.). About SignBSL.com. SignBSL.
<https://www.signbsl.com/about>
- [26] Stephenson, P., Zawolkow, E.(2020). Communication Considerations: Signing Exact English (SEE). Retrieved from [https://www.handsandvoices.org/comcon/articles/see.htm#:~:text=Signing%20Exact%20English%20\(SEE\)%20is,to%20be%20published%20\(1972\).](https://www.handsandvoices.org/comcon/articles/see.htm#:~:text=Signing%20Exact%20English%20(SEE)%20is,to%20be%20published%20(1972).)
- [27] tdrmk. (2019). Pygame Recorder (v1.0) [Software]. GitHub.
https://github.com/tdrmk/pygame_recorder
- [28] Wollock, J (1996). John Bulwer's (1606–1656) place in the history of the deaf. *Historiographia Linguistica*, 23, 1/2, p34.)
- [29] Wong, S. (2017). The Uncanny Valley Effect: Implications on Robotics and A.I. Development

Appendices

Appendix Item A – Evaluation Information Sheet and Consent Form



PARTICIPANT INFORMATION SHEET

Project Title:

Kyro: A Text to Sign Language Translator

Invitation

I would like to invite you to take part in a research study. Before you decide you need to understand why the research is being done and what it would involve for you. Please take time to read the following information carefully. Ask questions if anything you read is unclear or you would like more information. Take time to decide whether or not to take part.

The project is supervised by Alistair McConnell and the researcher is Charles Lyttle. The project has been approved by the Heriot-Watt University School of Computing ethics committee.

What is the purpose of this project?

The aim of this study is to create a software that deaf learners can use alongside a written resource in order to understand the resource more clearly, to do so I have created a system that takes any text input, be it from an educational resource, website, or otherwise, and generates a video displaying an avatar signing the corresponding British Sign Language signs in Sign Exact English syntax order.

Why have I been invited to take part?

You have been invited to participate as you are a computer science student and therefore have the ability to evaluate the visual quality of the video output generated.

Do I have to take part?

Taking part is voluntary. You may choose not to take part or to decide to stop taking part at any time without giving a reason. This will not affect your studies/employment/access to services/treatment. You also have the right to omit or refuse to answer or respond to any question that is asked of you.

What will happen if I take part?

The research will take place on Heriot-Watt Edinburgh Campus

You will be asked to answer a questionnaire about the accuracy, fluency, coherence and overall quality of the generated output animation.

Time commitment

This will take approximately 15 minutes.

Compensation and payments

Participation is voluntary. There is no compensation for time or expenses or other payments for taking part in this study.

What are the possible benefits of taking part?

Whilst there are no immediate benefits to you participating in the project, we hope that the information you provide in this study will help contribute to our understanding of sign language generation programs.

What are the possible risks of taking part?

There are no known risks to you in taking part in this study.

What will happen if I don't wish to carry on with the study?

If you decide that you no longer wish to take part, you can withdraw at any time without giving a reason. Please inform the project researcher *Charles Lyttle*, cl157@hw.ac.uk

Privacy and confidentiality

Heriot-Watt University is the data controller for the personal data collected in this project. This means that we are responsible under data protection law for making sure your personal information is kept secure, confidential, and used only in the way you have been told it will be used.

We will keep your personal information securely and ensure that no one will be able to link the research data you provide to any identifying information you may supply.

No information that could identify you will appear in any report on the results of this research.

We will keep records of your consent to participate forms and contact details separately in a password protected folder on our University IT system.

We will collect and use your personal data for this project only with your consent. You can withdraw your consent at any time until the data is fully anonymised by contacting the researcher, supervisor, or the data protection team. If you withdraw from the study all personal data collected from you will be securely destroyed.

However, it will not be possible to remove your data from the project once it has been anonymised as we will not be able to identify your specific data. This does not affect your data protection rights.

If you would like to know more about what Heriot-Watt University does with your personal data and your rights under privacy law, please visit our data protection web pages at <https://www.hw.ac.uk/about/policies/data-protection.htm> or contact our Data Protection Officer by email at dataprotection@hw.ac.uk.

What happens at the end of the project?

The researcher will use the findings of the research to produce a dissertation.

We will not include any information that could identify you in any report or publication.

Once we have analysed your questionnaire responses the researcher will completely anonymise your data and confidentially destroy any information that could identify you as a participant in this study.

After the project ends, we may use the anonymous dataset for research outputs such as articles and conference presentations to contribute to academic research in the public interest.

What if I have questions or concerns?

If you have questions about any aspect of this study, please contact

Charles Lyttle, cl157@hw.ac.uk - Researcher

Alistair McConnell, alistair.mcconnell@hw.ac.uk – Project Supervisor

If you have a concern or complaint about the study, please contact:

Patricia A. Vargas, p.a.vargas@hw.ac.uk – Director of Ethics, School of Computing

If you have any questions or concerns about the use of your personal data or your rights under data protection law, please contact dataprotection@hw.ac.uk

Thank you for reading this information sheet and for your interest in this research.

If you decide to take part, you will be given a copy of this information sheet and a signed participant agreement form to keep.



PARTICIPANT CONSENT FORM

Project Title:

3D Sign Supported English Animation Generation from Text Input

I have read and understand the Participant Information Sheet and Consent Form.

I have been given the opportunity to consider the information provided, ask questions, and have had these questions answered to my satisfaction.

I understand that there are no expected potential risks to me in my participation.

I am taking part in this research study voluntarily (without coercion or payment) and that I can refuse to answer any questions or stop taking part at any time without any of my rights being affected.

I understand that my anonymised data may be kept indefinitely for use in future ethically approved research.

Please check YES or NO to state whether you agree or do not agree to each statement	YES	NO
I agree to take part in this study		
I agree to being [video/ audio] recorded for the purpose of collecting data for this study as well as a means of verifying results from other data collected.		
I agree that my words may be quoted in publications, reports, web pages and other research outputs but not in a way that could identify me.		
I agree that my real name and words can used in these outputs.		

Name of person giving consent	Date	Signature

Name of Researcher	Date	Signature
Charles Lyttle		

Participant ID number:.....