JBD99: AI-Driven British Sign-language Translation

A dissertation submitted in partial fulfilment of

the requirements for the degree of

BACHELOR OF ENGINEERING in Computer Science

in

The Queen's University of Belfast

by

Kristina Geddis

23rd April 2021

SCHOOL OF ELECTRONICS, ELECTRICAL ENGINEERING and COMPUTER SCIENCE

CSC3002 – COMPUTER SCIENCE PROJECT

Dissertation Cover Sheet

A signed and completed cover sheet must accompany the submission of the Software Engineering dissertation submitted for assessment.

Work submitted without a cover sheet will **NOT** be marked.

Student Name: Kristina Geddis Student Number: 40198325
Project Title: AI-Driven British Sign-language Translation

Supervisor: Dr John Bustard

Declaration of Academic Integrity

Before submitting your dissertation please check that the submission:

1. Has a full bibliography attached laid out according to the guidelines specified in the Student Project Handbook
2. Contains full acknowledgement of all secondary sources used (paper-based and electronic)
3. Does not exceed the specified page limit
4. Is clearly presented and proof-read
5. Is submitted on, or before, the specified or agreed due date. Late submissions will only be accepted in exceptional circumstances or where a deferment has been granted in advance.

By submitting your dissertation you declare that you have completed the tutorial on plagiarism at http://www.qub.ac.uk/cite2write/introduction5.html and are aware that it is an academic offence to plagiarise. You declare that the submission is your own original work. No part of it has been submitted for any other assignment and you have acknowledged all written and electronic sources used.

6. If selected as an exemplar, I agree to allow my dissertation to be used as a sample for future students. (Please delete this if you do not agree.)

Student's signature Kristina Geddis *Date of submission* 23/4/2021

Acknowledgements

I would like to thank my supervisor, Dr John Bustard, for his well-appreciated support and guidance.

Abstract

This paper presents a comprehensive solution for automatically translating BSL (British Sign Language) footage into text captions and demonstrates the capability to translate continuously signed sequences.

The project followed a systematic approach, experimenting with a range of different pose estimation libraries, machine learning models and action recognition algorithms. It used state-of-the-art libraries including FrankMocap [21] and Dynamic Time Warping [16] to maximise the performance of the system. These concepts are brought together in an interactive web application that produces a subtitle file containing the translated labels of a given video.

Contents

1. Introduction

Sign language is a visual form of communication that involves the use of gestures, hand signs and finger-spelling to represent words, letters and phrases. Several modern sign languages are in use throughout the world, but in the United Kingdom, the most common is BSL (British Sign Language). There are approximately 11 million people in the UK who are deaf or hard of hearing and 151,000 people use BSL as part of their everyday life [4].

However, sign language's usefulness is limited due to its minimal adoption by non-deaf individuals. This makes communication challenging for deaf and hard of hearing individuals who have limited practical alternatives.

This is also accentuated in respect to modern forms of media. For example, in video communication (i.e. video calling and YouTube) sign language is seldom catered for and the addition of captions is a labour-intensive process. However, if a solution that automatically translated sign language into text was available, this would greatly enhance the users' experience on digital platforms and allow sign language to be used in similar scenarios.

This project aims to provide an automatic sign language-translation tool that will distinguish words in sign language footage from pre-recorded videos and translate them into text captions.

This will provide the foundation for integrating real-time sign language translation tools into video-based platforms, for example, video calling applications, in the future.

2. Researching the Problem area

2.1 Overview

In the case of BSL, words are expressed using upper body movements (i.e. hands, face, arms etc.) and their respective positioning on the person's body. The change in this movement is also important as different words can share the same poses at given moments in time.

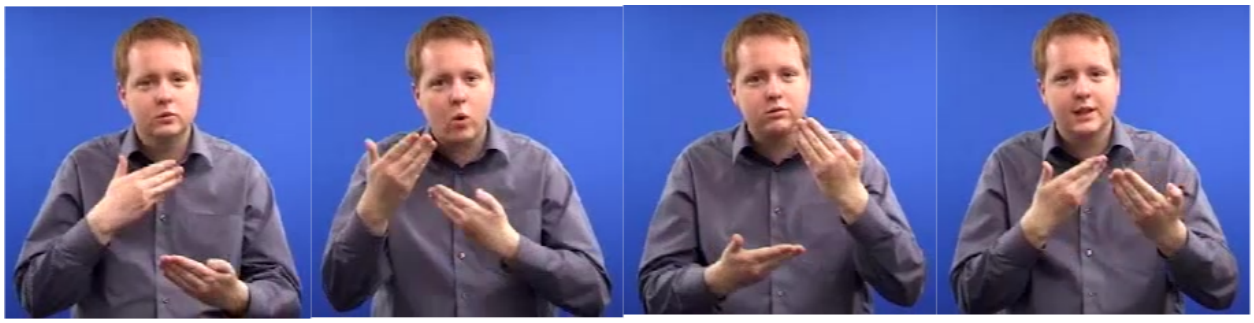

Figure 1: Sequence of frames of a signed word

To teach a machine to interpret these movements, the system needs to have a reliable process for extracting position information from the video footage. It then needs to use this information to distinguish consistently between the spectrum of signed words. This is very challenging as signed words are complex due to their motion over time and changed meaning based on human emotion and context (*see Fig 1*).

However, recent research has made machine learning algorithms for identifying human activity recognition possible, and these have been used as a foundation for this project [5].

2.2 Action Recognition

Sign language translation is a human action recognition problem. This refers to problems where the machine needs to recognize human actions in images or videos. Human activities are normally grouped into four categories: gesture; action; interaction; and group activities [6]. These are determined depending upon which body parts are engaged in the action and its overall complexity.

There are a variety of existing projects exploring this research area. However, they do not solve the problem this project aims to tackle as they are either incomplete, closed source or not capable of translating sentences:

1. “DeepASL” [7]: This paper explored the idea of supporting sentence-level translation but did not make its code public.

2. “Real-Time Sign Language Detection Using Human Pose Estimation” [8]: This paper presented a solution for detecting when sign language was or was not present in footage, but did not translate it. The project used pose estimation and optical flow techniques to achieve detection.

3. “Real-time Action Recognition Based on Human Skeleton Video” [9]: This paper implemented an action recognition system that was able to distinguish between 9 different types of everyday action using pose estimation.

4. “Sign Language Interpreter using Deep Learning” [10]: This project implemented basic gesture translation via hand motions or fingerspelling on a frame-by-frame basis. It is not capable of translating continuous signing sequences.

A common aspect of these papers was that the datasets they used were very limited (containing around 10 words/actions) and were regulated to use a standard format.

Additionally, they demonstrated that it was possible to distinguish between poses, but none explored the concept of distinguishing between poses over a finite time frame. This is a necessity for a successful project due to the fact the sign language words are conveyed over multiple seconds, rather than a single action in time.

It was self-evident from these research papers that the extraction of time-based pose information from the video data was crucial, and had a significant impact on the end quality of these systems.

2.3 Feature Extraction

Sign language data could come in a variety of formats such as sensor information [37]. However, this project aims to use existing data sources, of which video data is the most common.

Before the system can use this data, it must extract the relevant information. The majority of image classifiers would simply use the image data and post-processing techniques to extract the key shapes in the image. However, as sign language focuses purely on the position of the body, pose estimation would be a more reliable extraction process.

2.3.1 Pose Estimation

Pose estimation is the process of extracting human joint and limb positions from images. This information is typically represented as a "skeleton" made up of a series of lines and can be overlaid on the original footage (see *Fig 2*).

Pose estimation coordinates are normally calculated in two dimensions (x,y coordinates) but there are also more complex algorithms for three-dimensional analysis (x,y,z coordinates). All pose estimation libraries aim to accurately predict the position of each joint, and use a variety of techniques, such as computer vision, to do so.

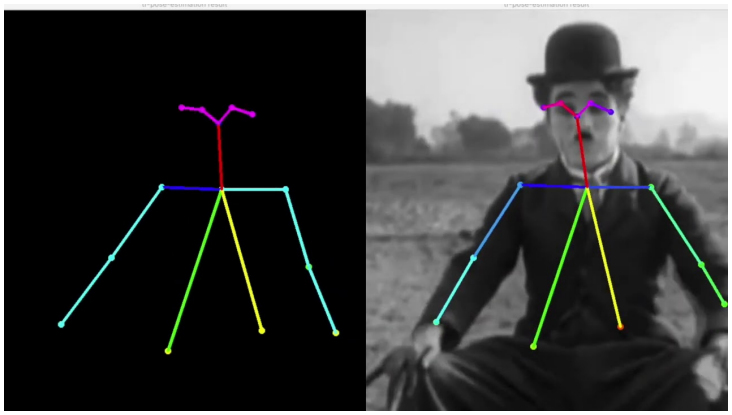

Figure 2: Example of 2D pose estimation

Numerous open-source pose estimation libraries have been created in recent years [11]. Currently, OpenPose [12] is the most popular. It accommodates many different features such as real-time 2D multi-person keypoint detections [13].

However, there are limitations of pose estimation libraries. For example, many of them struggle to detect joints that overlap or are difficult to distinguish due to environmental factors i.e. lack of background contrast. Additionally, these libraries cannot see joints that are out of the frame and will likely estimate an incorrect prediction. Finally, as this process is purely estimation, it will not be 100% accurate and this depends entirely on the quality of the library incorporated.

2.4 Classification

Different approaches are available to classify each signed word. One approach would be to produce a ruleset that defines each action. However, this is not viable when actions are complex or there are too many similar actions to distinguish between. This would be the case for sign language, as the variety of possible words and minor variations would be too complicated to write a reliable rule-driven system for.

For this reason, machine learning algorithms have become a choice of preference for action recognition.

2.4.1 Machine Learning

Machine learning is the science of teaching a computer to learn automatically without having to explicitly program individual patterns. By exposing specific data to a model, it is possible to improve itself through repetitive experience. Additionally, the more data the model is given, the better it will become at learning and recognising different patterns.

Action recognition is a classification problem which means that the model will use the data to understand what patterns separate each action. In the case of action recognition for sign language, a dataset of signed words would be converted to a numerical representation (such as by using pose estimation) and used as training classes that the model learns to classify.

This kind of problem also falls into the category of supervised learning. Supervised learning is a process in which the dataset provided is fully labelled and categorised. In estimating the performance of a system, a small percentage of the data is left out of training to be used as testing samples. This allows the model to analyse how well it is doing by scoring what percentage of testing samples were correctly predicted.

Additionally, how the data is structured will impact what behaviour the model learns to classify. One approach would be to identify actions from individual frames. However, this may not generalise well for sign language as the action is based on the motion of different body parts over multiple frames, which this data would not have the context to.
Another approach involves applying a fixed-length sliding window. This would extract a sequence of windows for each sample and means that it would capture the motion over multiple frames. Unfortunately, this would be constrained by having a fixed window frame size, which in turn would impact its ability to adapt to dynamic length sequences.

2.4.2 Dynamic Time Warping

Dynamic Time Warping (DTW) is an algorithm that compares the similarity between two arrays or time series of different lengths [14].

It works by converting the data into vectors and calculating the Euclidean distance for every point in the two-time series [15]. It can be used in combination with a classifier (e.g. K-Nearest Neighbour) to predict which sequence the sample was closest to (*see Fig 3*).

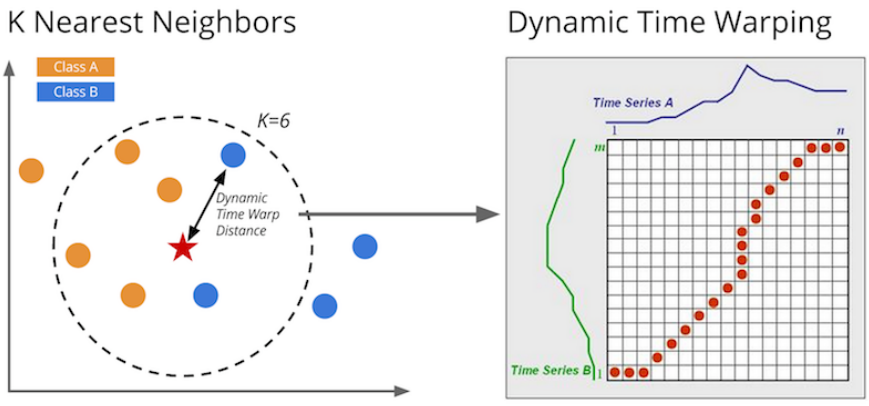

Figure 3: Best alignment path illustration for DTW using KNN [16]

The path of best alignment (red path) is computed by measuring the distance between each cell in time-series A and time-series B (*see Fig 4*)

$$DTW_{AB} = SUM(shortest\ paths_{AB})$$

Figure 4: Dynamic Time Warping best alignment path expression [16]

The advantage of using DTW over a simple comparison is that it can handle arrays of different lengths and where values could also be offset. For this reason, it is commonly applied to problems such as speech recognition, where words may be spoken at different speeds. Human actions can also vary in length and, therefore, DTW would be the most suitable application for these problems.

Dynamic Time Warping uses a warping window for comparisons (*see Fig 5*). This window is what searches surrounding elements to find the closest sample where data is offset. To configure this, a max warping window size must be set. This parameter constraints how far the window can deviate from the samples and restricts the number of cells that can be used to compute the DTW distance matrix.

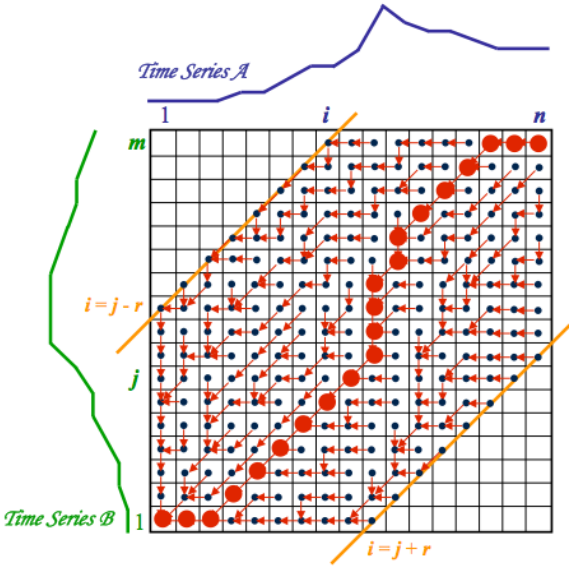

Figure 5: Different warping window constraints for DTW [16]

The optimal warping window size is generally different depending on the dataset size and its structure, meaning that it may not transfer well between different datasets [17]. Therefore, an optimal value must be calculated for each dataset.

3. System Requirements and Specification

Before conducting the experiments, the requirements of the system needed to be thoroughly evaluated. The descriptions of these requirements are given below.

3.1 Sign Language Translation

The overarching goal of the project was to develop a system that could translate BSL video footage to text. In doing so, the project investigated a range of methods to maximise the translation accuracy of the proposed system.

To deliver this, the project specification was to provide an application that allowed the user to upload a video and output the word predictions and timings in a suitable format.

3.2 Dataset Collection

The project needed to create a dataset of BSL videos, which were then used to facilitate the development and testing of this system, as such a dataset was not available.

The project requirement was to develop specific tools that made it possible to extract data from existing data sources (BSL websites). These tools aimed to maximise the available data, but also to ensure that the dataset produced was accurate and correctly labelled.

3.3 Estimation Analysis

Due to the complexity and experimental nature of the problem area, the prediction process was required to be transparent, easy to configure and fine-tunable. Additionally, the system needed to be designed in such a way that steps could be easily fault-traced to provide essential diagnostic information and elucidate the decision process behind the predictions made.

3.4 Further Development

As this project was focused on becoming the framework for a potentially wider solution, it was a requirement that the system was easily maintainable and expandable.

To this end, the project was structured to allow for future development. It was also a requirement to be fairly dynamic to incorporate changes, such as new larger datasets and changes to various processes in the pipeline. This needed to be interchangeable to facilitate more focussed development of the system going forwards.

4. Design

An overview of the design is presented in this section and was utilised in the final implementation.

4.1 Application

To meet the requirement outlined in 3.1, the problem was analysed in considerable detail to produce the most logical sequence of steps for translation through a user interface. This resulted in the following web application design (*see Fig 6*).

Figure 6: Visualisation of the user-interface

The web application allows the user to upload a video of choice. The configuration of the prediction can be modified by selecting the “Advanced Options” button, but this is hidden by default to make the interface more user friendly.

On submit, the application will then initiate the preprocessing steps and prediction on the video. The progress bar informs the user of the loading time of this process.

Once the loading is complete, the user can play the video which has captions of the predicted translation overlaid using a subtitle file. The video and subtitle file can also be downloaded, via the individual download buttons, to be saved locally, if desired.

Selecting the advanced options on the first screen allows the user to customise various features of the process. These options include supporting a wireframe overlay of the key skeleton points to analyse what the system extracted from the video, as well as setting a minimum score and confidence level to customise the prediction process (*see Fig 7*). This allows advanced users to experiment on the system to satisfy the requirements in objective 3.3.

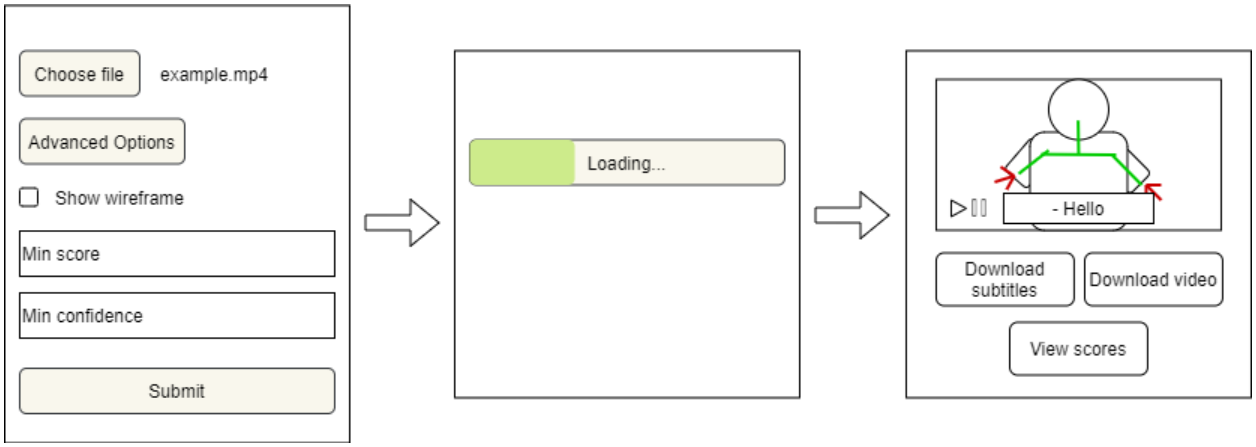

Figure 7: Visualisation of advanced option user-interface

On the results screen, an option to view the scoring of the predicted words can also be selected (*Fig 8*). This displays the confidence and timings of each word on a separate page in the form of a table. This information can be used to fine-tune the prediction process via tweaking the advanced options with updated values. The higher the score, the better the confidence is at the translation.

Figure 8: Visualisation of the scoring table

4.2 Pipeline

The pipeline in *Fig 9,* was initially constructed and through a series of trials and experiments, was developed to be sufficiently robust for the final solution.

In stage 1, a web scraper retrieves a list of video URLs from various BSL websites. Afterwards, another script will download these videos and group each word into folders. To ensure the dataset quality, videos that do not meet a set of requirements will be removed.

In stage 2, preprocessing prepares the dataset for training. This starts by extracting pose data using a pose estimation library. This data is then normalised to make it consistent across all videos.

In stage 3, training uses the normalised pose data to produce a model weights file.

In the final stage, the application allows the user to upload a video which the system predicts using the pre-trained model weights file. Using its prediction algorithm, the sentence words and timings are estimated. A subtitle file with the predicted words and their synchronized timings is produced and displayed to the user.

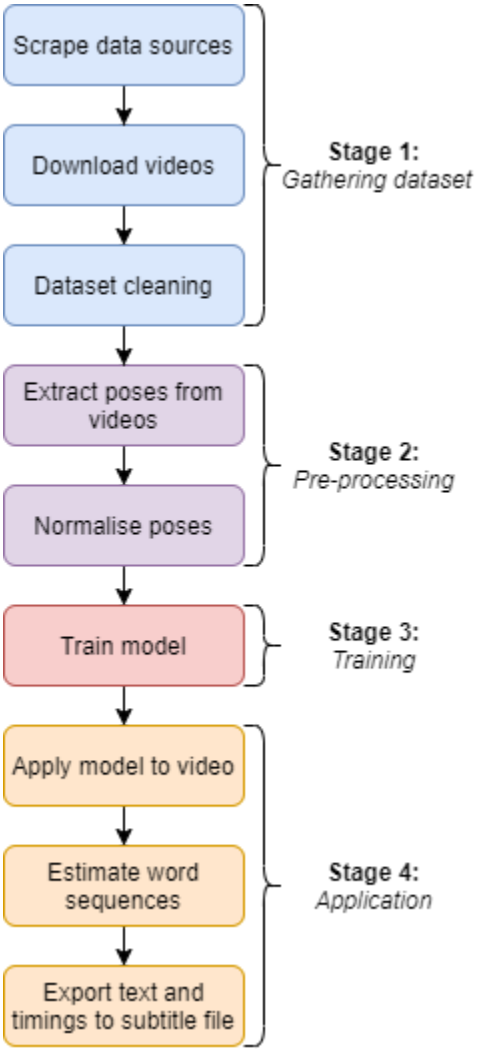

Figure 9: Pipeline of the end-to-end process

5. Experimental Results

Experiments were undertaken to identify the optimal approach to each step of the pipeline. Data gathered from all experiments, both successful and unsuccessful, were thoroughly evaluated and the most promising results implemented in the final solution. This was a process of iteration to reach a rational and viable outcome.

5.1 Experimentation setup

The pipeline was applied to a standard dataset with validation videos to fairly and consistently evaluate each experiment. This dataset consisted of 6-word classes sourced from the BSL Signbank [18] and SignBSL [19] websites and defined as: I; car; true; story; mechanic and my.

The training dataset was intentionally simplified to 6-word classes to minimise computational cost and simplify the verification process. Each class had between 2 and 10 videos varying from 1 to 4 seconds in duration. This was typically representative of the inconsistency across the spectrum in available data from the current data sources.

Two validation videos were generated from the BSL Corpus [20] BF3n video to score the validation accuracy of the models detailed in *Table 1*.

Table 1: Validation video information for the small dataset

Video name	Content	Words to validate	Duration
BF3n_9595.mov	“I have a true story”	“I”, “true”, “story”	3 seconds
BF3n_23640.mov	“He’s my car mechanic”	“My”, “car”, “mechanic”	1 second

These were validated through a series of validation scripts found in the *research* folder [1] which calculated the validation accuracy of the correctly predicted words and then produced an average of the two validation videos. As expressed in *Fig 10*, where “n” is the total number of validation samples and “correct” is the correctly identified samples.

$$\frac{1}{n} \sum correct$$

Figure 10: Validation accuracy expression

Initial parameters were set as the first basis for comparison of results from the experiments (*see Table 2*).

Table 2: Initial parameters set for experimentation

Pose Estimation Library	Training Method	Frame Sample Rate	Normalisation	Augmentation	Classifier
OpenPose	Single-frame	5	Ratio	None	Gaussian Process

As the experiments progressed, it was found necessary to update the parameters accordingly to maximise performance.

5.2 Pose Estimation

The project initially experimented with pose estimation libraries. Selecting a strong pose estimation library would maximise the accuracy in the extracted pose data, and should therefore improve the performance of the model.

5.2.1 OpenPose

OpenPose [12] was used in the early experimental stages as a means of extracting key body points. It was chosen due to its popularity within the community. Initially, eight key body points were extracted: nose; neck; shoulders; elbows; and wrists (*see Fig 11*). These body points were chosen as a reliable starting point to reduce the complexity and computational cost of the training.

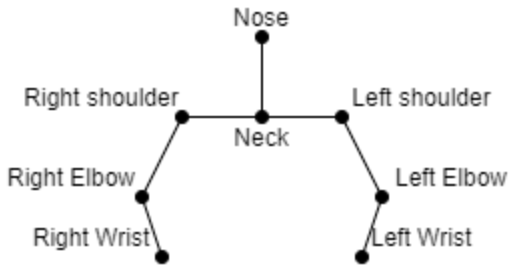

Figure 11: Key body points visualisation

Surprisingly, this model did not perform well and produced rather inconsistent results. It tended to output high confidence predictions for the wrong classes. For example, in the figure below, the man was signing the word “true”, but the classifier predicted the word to be “mechanic” as it scored the highest confidence value of 27%.

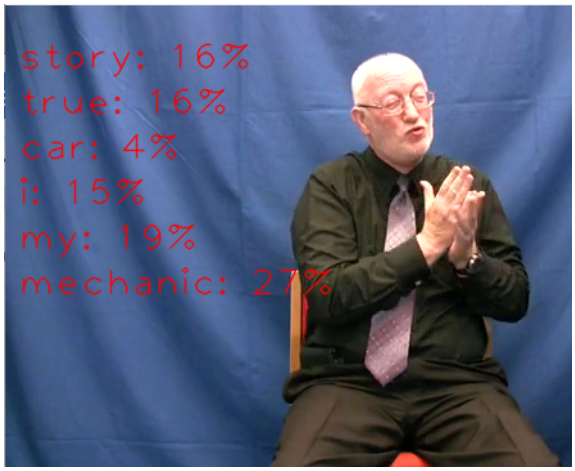

Figure 12: Screenshot of single-frame validation video prediction

This was not satisfactory and led to the observation of failing cases to better understand why the classifier was incorrectly predicting each signed word. Consequently, a fault-tracing visualizer (*research/visualisation/visualiser.py*) was created. This converted the recorded pose information of each frame to a Scalable Vector Graphics (SVG) which allowed the data to be visualised (*see Fig 13*).

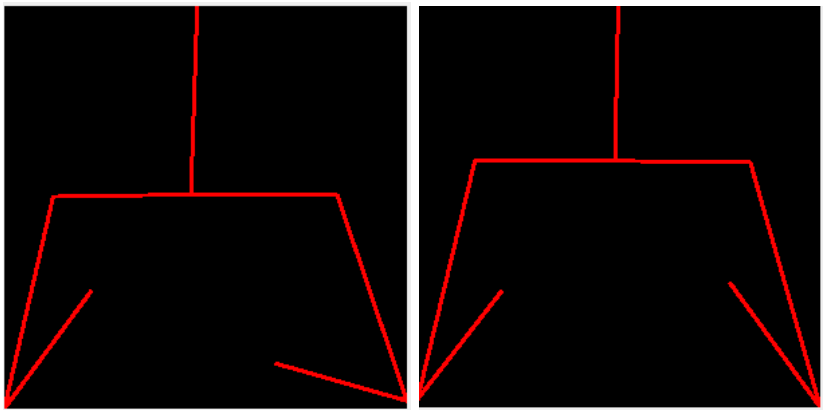

Figure 13: SVG visualiser displaying signed words; "true" (left), "mechanic" (right)

It was evident that for these two words, "true" and "mechanic", the eight key body points were very similar except for the left wrist point in *Fig 13*. Therefore, it was concluded that small pose changes were having a dramatic effect on the predictions.

From observing the OpenPose skeletons overlaid on the videos shown in *Fig 14*, it was evident that the biggest differences between the two poses were the hand positions and finger orientation. Thus, the addition of hand key points to the data would in theory help improve the results.

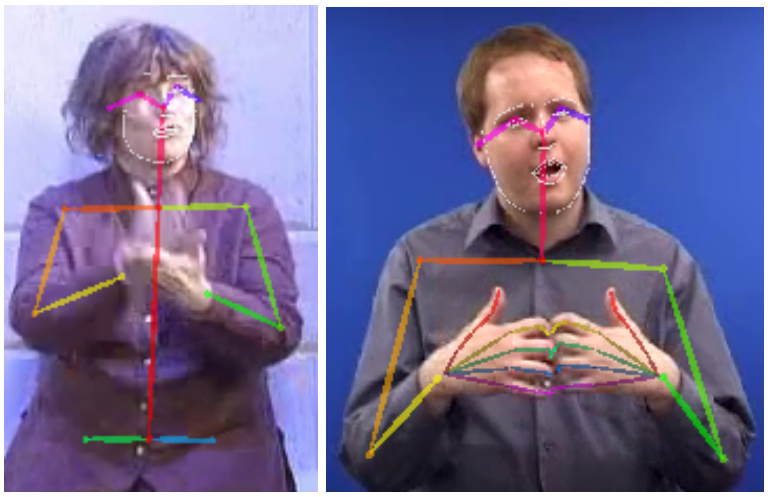

Figure 14: OpenPose wireframe displaying signed words; "true" (left), "mechanic" (right)

5.2.2 Addition of Hand Key Points

From the earlier experimentation, it was found necessary to add hand key points to the data. As shown below (*Fig 15*), fortunately, OpenPose provides an additional 21 key points for each hand.

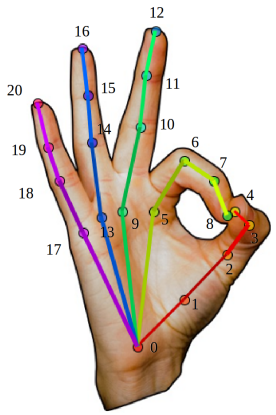

Figure 15: Visualisation of 21 hand key points

As a result, the size of the dataset increased significantly to 50 key body points (8 body, 21 left-hand and 21 right-hand) per frame. This data was then retrained and reapplied to the same standard validation data to verify whether it was successful in improving the model validation accuracy.

However, OpenPose struggled to identify the hands consistently as shown in *Fig 16*. This was a particular problem for the project as one of the major aspects of sign language is the movement of hand and finger positioning.

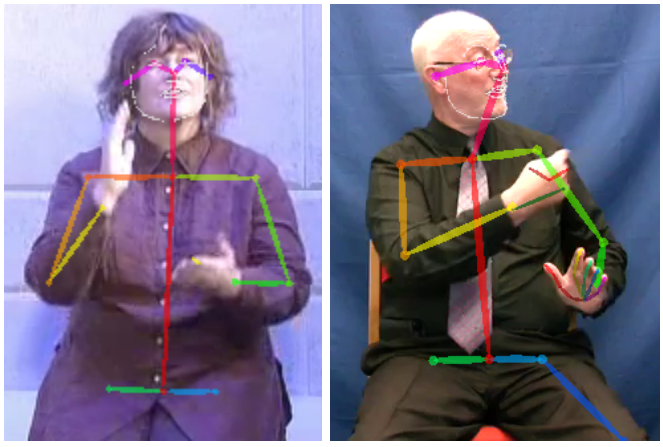

Figure 16: OpenPose examples of inconsistent hand detections

After further experimentation, there was little improvement in the reliability of the results and thus it was judged that OpenPose was not a suitable candidate for this project.

5.2.3 FrankMocap

From further research of the available libraries, the project experimented with FrankMocap [21]. FrankMocap is a state-of-the-art library produced by Facebook researchers which render accurate 3D poses of the body and hands (*see Fig 17*).

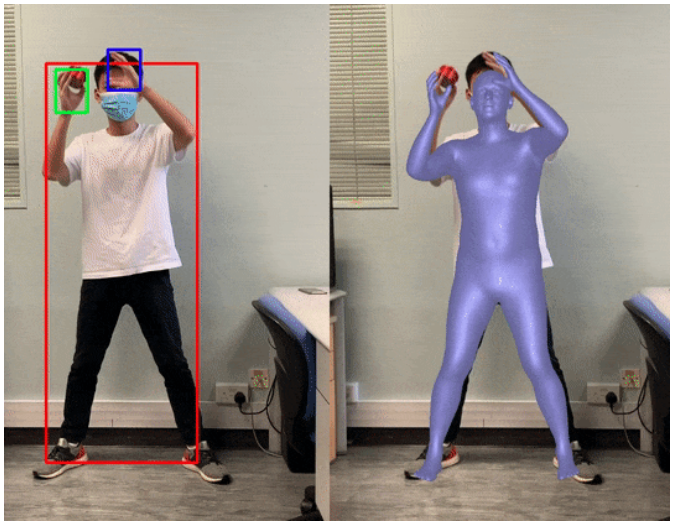

Figure 17: FrankMocap 3D body pose visualisation

Although the 3D visualisation was not relevant for this project, the underlying key point detection appeared very promising. Early tests showed that FrankMocap was more consistent at extracting valid hand key points than OpenPose as visualised in *Fig 18* below.

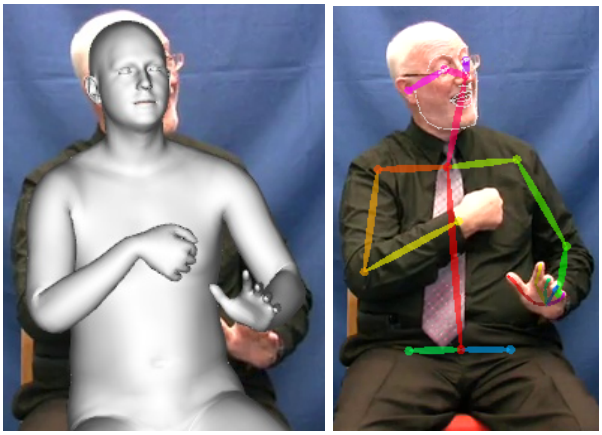

Figure 18: Difference in hand detection for FrankMocap (left) and OpenPose (right)

Unlike OpenPose, FrankMocap will make a pose prediction when it cannot find certain body points in the frame (*Fig 19*). This means it may artificially position hand points when a hand is off-screen, which could lead to misclassification in certain instances.

Figure 19: FrankMocap predicting hand placement

5.2.4 Results

The results of the two pose estimation models are summarised in the table below. These values were run using the initial parameter set outlined in section 5.

Table 3: Pose estimation library results comparison

Pose Estimation Library	Training Accuracy (%)	Validation Accuracy (%)
OpenPose	46.8	28.4
FrankMocap	53.4	34.7

Based on the above results, FrankMocap was 6.3% better on validation data and 6.6% better on training data when compared to OpenPose. This indicated that FrankMocap was the stronger of the two and was used going forwards.

5.2.5 Sampling Rate

The sampling rate controls how many frames are sampled per second, as illustrated in *Fig 20*.

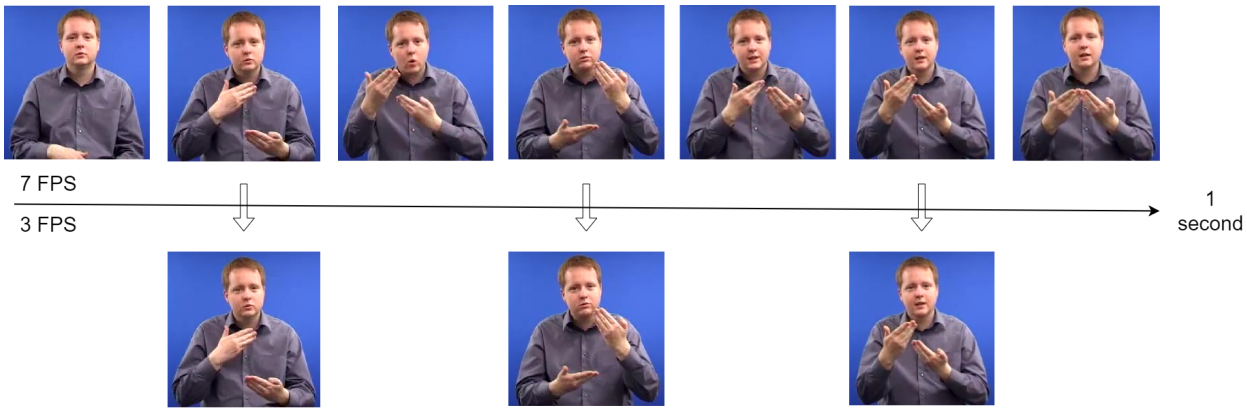

Figure 20: Visualisation of how sampling works

A low sample rate reduces the volume and complexity of the data in comparison to a high sample rate. Therefore, reducing the sample rate would reduce the complexity of the dataset.

The table below shows the results of experiments conducted with varying sampling rates.

Table 4: Sampling rate results

Sampling rate (fps)	5	10	15	25
Training Accuracy (%)	53.4	65.0	65.3	82.3
Validation Accuracy (%)	34.7	33.4	38.5	39.5

From these results, it is evident that maintaining a high sample rate appreciably increased the accuracy. This was not surprising, as in this case, more data was present and therefore exposed the model to more data to learn from. Therefore, a sample rate of 25 was implemented.

5.3 Normalisation

Normalisation converts data to a common scale so that it can be better processed. In the case of this project, normalisation was necessary to ensure that the data purely focused on the positioning of the limbs of the person and removed any other features such as video resolution and human body size.

An unnormalised sample (*Fig 21*) would produce different pixel values for the same action because of the distance and body size of the individual.

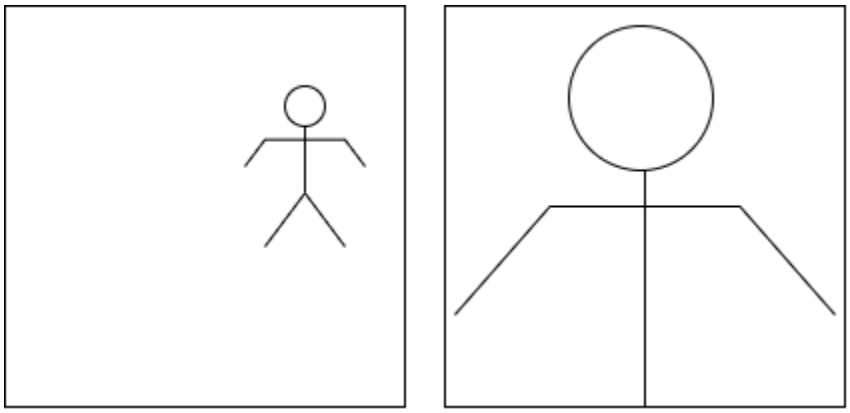

Figure 21: Visualisation of unnormalised frames

A normalised sample would remove these factors so that both were evaluated purely on the positioning of their body and no other factors. From analysing the data available, this project found two possible approaches to normalisation.

5.3.1 Ratio

"Ratio" is the process of scaling coordinates relative to the total area of all available coordinates. In simple terms, this involves calculating the minimum and maximum x and y points and converting each coordinate to the percentage of that overall dimension.

For example, in *Fig 22*, the point (125, 100) is at 50% of the total horizontal distance (from the furthest left point to the furthest right point) and 60% of the total vertical distance (from the highest point to the lowest point). This means that the point (125, 100) would be converted to (0.5, 0.6) after this normalisation process.

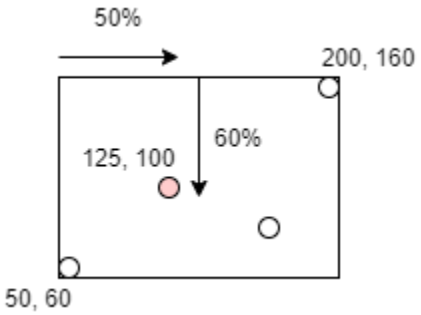

Figure 22: Ratio illustration

This process eliminated the effect of video resolution and body size because it normalised the percentage of the overall width and height. For example, doubling the width of the video resolution would still output the same normalised values because each point is still accounting for its position relative to the global width.

5.3.2 Joint angles

Calculating the angles of joints removed the need to keep track of the specific joints and instead focused on the relative position of each joint (*see Fig 23*).

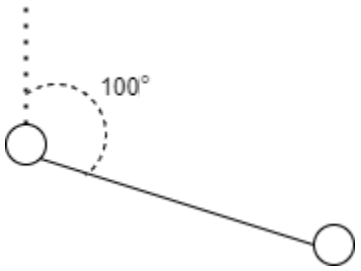

Figure 23: Joint angle illustration

This removed the effect of video resolution and human body dimensions, as values were represented relative to each other and were therefore unaffected by changes in scale.

5.3.3 Results

Comparing the two normalisation techniques showed that the ratio processed dataset produced an accuracy score of 82.3% with a validation score of 39.5%, and joint angle scored 74.6% accuracy with a 23.5% validation score.

This showed that the ratio normalisation technique provided more useful information for the model to learn from, leading to better accuracy. This was likely attributed to the joint angle relying on more of the key body points being detected, as each angle needed two points to calculate.

5.4 Augmentation

Augmentation is the process of artificially generating more training data from existing data. There are many different ways of augmenting image data including rotation, flipping, cropping etc [22]. Many of these techniques could not be used for this problem as they did not preserve key body details. However, it was found through experimentation that adding small random noise variations ranging from 0.1 to 0.3 to the pose coordinates was a possible approach (*see Fig 24*).

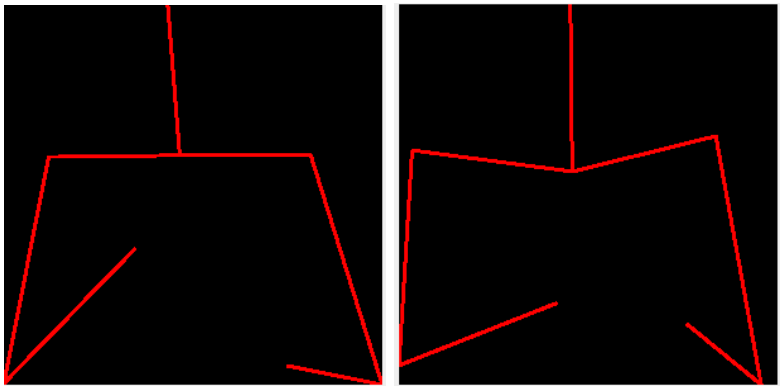

Figure 24: SVG visualisation of the word "true": non-augmented (left), augmented (right)

Table 5: Augmentation results

	Training Accuracy	Validation Accuracy
Augmented data (%)	97.9	33.7
Non-augmented data (%)	82.3	39.5

As seen in these results, augmentation improved the training accuracy of the Gaussian Process classifier by 15.6%. This was because it provided more sample variations for the model to learn from. On the other hand, there was a small detrimental effect on the validation accuracy by 5.8%. This did not improve because the augmented samples did not provide additional variety and therefore did not help in places where the classifier had failed before. It may have also caused minor overfitting, which is where the model learns to fit the training data too well and cannot generalise well to the problem.

In conclusion, it was better to use the non-augmented data as the validation accuracy is the most crucial factor.

5.5 Training

To maximise performance, a range of training methods were evaluated. For all of the training methods, a range of popular Sklearn [23] classifiers were applied to select the one which was most suitable for this project.

Additionally, the validation videos were labelled with timestamps of when a certain word was being signed. This allowed each classifier to be scored on the percentage of frames it correctly predicted for each word.

5.5.1 Single-frame

This approach involved classifying each frame of each video as a separate sample. This was a basic approach but also meant that the understanding of change between frames would be lost.

Table 6: Single-frame training accuracy/validation accuracy results

KNN	Gaussian Process	Decision Tree	Random Forest	Neural Net	Adaboost	Naive Bayes	QDA
81.6%/ 30.3%	82.3%/ 39.5%	63.7%/ 35.3%	59.8%/ 26.1%	70.1%/ 30.9%	45.3%/ 26.5%	32.9%/ 18.6%	75.9%/ 38.7%

The difference in accuracies between the training and validation data showed that the classifiers were not as effective when applied to the problem. The best classifier was Gaussian Process which had an 82.3% training accuracy, but only correctly predicted 39.5% of the frames in the validation videos. Adaboost and Naive Bayes produced the least desirable results.

To identify the poor validation accuracy, a tool that showed the nearest neighbour sample for the KNN classifier was developed (*Fig 25*). This helped in understanding where the classifier was mislabelling samples and more importantly why. *Fig 25* shows a screenshot of the tool providing the nearest neighbour for that specific frame.

Figure 25: Validation video (left), nearest neighbour sample (right)

From this, it was evident that looking at a single-frame method was not sufficient to reliably classify each word. This was because some words shared similar positions at different moments in time and therefore were only clearly separated by their motion over multiple frames. For this reason, the single-frame approach was deemed not viable for this problem.

5.5.2 Multi-frame

The next training approach involved taking a fixed sequence of frames from each video as training samples. The use of a sequence of frames meant that the motion between frames was maintained at the expense of additional complexity.

For this experiment, two different ways of extracting the sequences of frames were tested; sliding window and overlapping sliding window.

5.5.2.1 Sliding Window

The first approach involved cutting each video into slices of frames. This process is known as a sliding window and works by scanning over videos with a window consisting of a fixed length of frames.

For example, as shown in *Fig 26*, a video of 6 frames would produce 2 window samples each containing 3 frames.

Figure 26: Sliding window method using a fixed window size of three

The extracted windows were merged into one array and became the samples that the model trained on. The same extraction process was applied to the validation videos before prediction.

To handle the variety in video length, the last window extracted from each video needed to be zero-padded. For example, a video that is 6 frames long cannot be divided equally into a window size of 4. To deal with this, the leftover 2 frames at the end are padded with values of zero so that they are the same length as the other windows (4 frames). This ensures that all samples are of the same length.

A range of window sizes were experimented with to identify which worked best are presented in *Table 7*.

Table 7: Sliding window training accuracy/validation accuracy results

Window size	KNN	Gaussian Process	Decision Tree	Random Forest	Neural Net	Adaboost	Naive Bayes	QDA
3	68.6%/ 29.2%	59.7%/ 33.3%	60.4%/ 29.2%	52.8%/ 22.2%	67.3%/ 29.2%	37.1%/ 5.5%	30.2%/ 16.7%	40.3%/ 5.6%
5	61.9%/ 29.2%	33%/ 16.6%	41.2%/ 0%	52.6%/ 12.5%	58.8%/ 18.1%	42.3%/ 13.9%	28.9%/ 11.1%	29.9%/ 16.7%
10	45.1%/ 11.1%	45.1%/ 29.2%	29.4%/ 5.5%	39.2%/ 13.9%	21.5%/ 16.7%	25.5%/ 19.4%	19.6%/ 16.6%	27.5%/ 13.9%
15	42.8%/ 8.3%	17.1%/ 12.5%	48.6%/ 13.9%	37.1%/ 8.3%	40%/ 0%	37.1%/ 16.7%	25.7%/ 15%	22.9%/ 12.5%

These results showed that the Gaussian Process classifier with a window size of 3 produced the best validation accuracy of 33.3%, but interestingly, KNN produced the highest training accuracy of 68.6%. However, this did not improve on the results from the single-frame runs. This was because the sliding window increased the complexity of the data and reduced the available samples, making it more difficult for the classifiers to learn the behaviour.

5.5.2.2 Overlapping Sliding Window

Overlapping sliding window builds on the sliding window approach by moving the sliding window in single-frame increments (*see Fig 27*). This means that frames from each video appear in multiple samples at different positions. This provided the classifiers with more data than the traditional sliding window approach and therefore improved results.

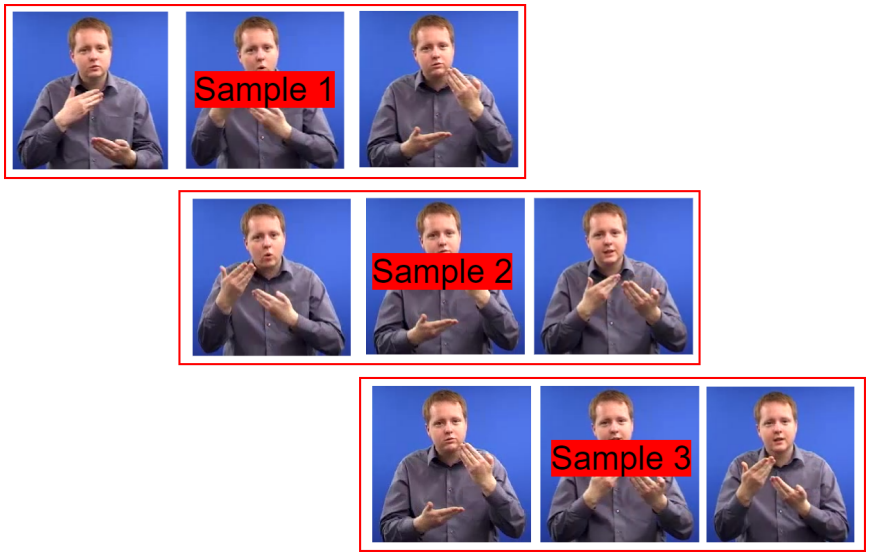

Figure 27: Overlapping window producing three different samples of the same word

Table 8: Overlapping window training accuracy/validation accuracy results

Window size (with overlap)	KNN	Gaussian Process	Decision Tree	Random Forest	Neural Net	Adaboost	Naive Bayes	QDA
3	83.3%/ 29.2%	86.8%/ 34.7%	67.8%/ 12.5%	60.6%/ 0%	71.4%/ 29.2%	46.9%/ 33.3%	31.5%/ 16.7%	54.8%/ 16.7%
5	85%/ 29.2%	85.6%/ 41.7%	64.9%/ 16.7%	62.2%/ 0%	81.3%/ 29.2%	49.4%/ 27.8%	30.5%/ 16.7%	33.9%/ 16.7%
10	82.6%/ 29.2%	91.6%/ 29.2%	61.5%/ 12.5%	63.3%/ 20.8%	80.6%/ 29.2%	44.6%/ 0%	38%/ 16.7%	43.4%/ 33.3%
15	91.6%/ 29.2%	34.1%/ 16.7%	51.2%/ 12.5%	63.8%/ 8.3%	93.5%/ 29.2%	45.2%/ 5.5%	39%/ 16.7%	46.9%/ 29.2%

Gaussian Process classifier performed best with a validation score of 41.7% when applying an overlapping window size of 5. A reason for the results peaking at around a window size of 5 was most likely due to it effectively balancing a greater number of samples with the detail provided in a single sample.

5.5.3 Dynamic Time Warping

Even though the validation accuracy of multi-frame was better than single-frame, it was restricted by only being able to see a certain portion of a word at a time. It was also unable to differentiate the same word being signed at different speeds.

As discovered in the research section, Dynamic Time Warping can compute the similarity of two different length arrays. For this project, DTW could be used to compare how similar two signed words were depending on the body movements in the sequence. Unlike single and multi-frame approaches, DTW can deal with variations in signing speed and evaluate the full sequence of each word.

Research has shown that KNN was the most suitable classifier used together with DTW [16] and for this rationale, DTW was implemented using the KNN classifier. This was because poses can be compared by their Euclidean distance, which KNN supports, unlike other classifiers. Additionally, KNN is a straightforward and transparent classifier that lends itself to uncomplicated computations and easy debugging.

Two different approaches of calculating the distance matrix were experimented with; standard implementation of DTW [16] and the FastDTW [28] library. The results are tabulated below.

5.5.3.1 Standard

Table 9: Standard DTW training accuracy/validation accuracy results for different parameters

	Max warping window size			
K value	1	5	10	15
1	69%/43.8%	70%/37.7%	70%/37.7%	70%/37.7%
3	68%/48.5%	69%/47%	69%/44.9%	69%/44.9%
5	68%/50.5%	70%/47.5%	69%/49.5%	70%/49.5%

The results indicated that using a max warping window size of 1 and a K value of 5 was the best, with a validation accuracy of 50.5%.

5.5.3.2 FastDTW

Running the same optimal parameters that were discovered in the standard matrix calculation with the FastDTW library produced a training accuracy of 67% and a validation accuracy of 40.8%. This did not improve the performance of the standard approach and was slower to run, taking 51 minutes compared to the standard implementation runtime of just 4 minutes.

5.5.4 Evaluation

In conclusion, the standard implementation of DTW was chosen due to its superior validation accuracy with an improvement on the previous best by 8.8%.

5.6 Post Processing

Post-processing is a technique used to refine or evaluate data after generation. In the case of this project, it was necessary to take the predictions of the classifier and convert them to the word and timing predictions (to be saved to a subtitle file). This involved using the DTW classifier values from the previous experiment in section 5.5 to decide which words were present and for what durations.

Experiments were conducted using validation videos that were scored based on the total words correctly identified. This aimed to maximise the percentage of True positives (words that were predicted and were in the video) and minimise the False positives (words that were predicted but were not in the video). Furthermore, the average deviation in seconds from the real word timings was also recorded.

5.6.1 Best Class Filter

This filter takes the highest probability class for each frame of the prediction and outputs words where they were predicted as the most likely over multiple consecutive frames. In the case of the validation videos, the results were generated using a minimum word frame length of 3.

Table 10: Optimal confidence and word length applied to validation videos

Video	Total words in video	Number of words predicted	Minimum confidence	Minimum Word Length	Number of True Positives	Number of False Positives	Average timing deviation (seconds)
BF3n_9575.mov	3	4	0.4	3	2	2	0.55
BF3n_23640.mov	3	5	0.4	3	2	3	0.36

5.6.2 Confidence Filter

This filter takes the probability of each class for each frame of the prediction. It then scores sequences of frames by summing the word confidences where it was above a minimum value (for example 5 frames of confidence 0.5 would generate a score of 2.5). The lowest scored words were then removed.

In the table below, the optimal minimum confidence and minimum score were found for each validation video.

Table 11: Optimal confidence and score values applied to validation videos

Video	Total words in video	Number of words predicted	Minimum confidence	Minimum Score	Number of True Positives	Number of False Positives	Average timing deviation (seconds)
BF3n_9575.mov	3	2	0.2	18	2	0	0.64
BF3n_23640.mov	3	3	0.4	4	2	1	0.6

Further experiments were run to generalise the system to all videos, see *Table 12*. A minimum confidence of 0.4 and a minimum score of 4.5 were found to be the best default values. Whilst these values did not generate perfect results for both videos, they produced the best average across both.

Table 12: Universal confidence and score values applied to validation videos

Video	Total words in video	Number of words predicted	Minimum confidence	Minimum Score	Number of True Positives	Number of False Positives	Average timing deviation (seconds)
BF3n_9575.mov	3	3	0.4	4.5	2	1	0.44
BF3n_23640.mov	3	3	0.4	4.5	2	1	0.29

5.6.3 Evaluation

After experimenting with both filters, it was concluded that the confidence filter could more consistently generate the correct word labels and with better timing predictions. It was therefore selected as the filter of choice for the final system. The minimum confidence and score values were implemented as the defaults in the final application.

6. Implementation

This section details the final implementation based on the experiments run in section 5. The project was developed in a Linux environment using Anaconda [24].

Python [25] was chosen as the primary development language for this project due to its simplicity, wide use and support for a host of pertinent libraries. The steps involved in the implementation of this project are detailed below.

6.1 Dataset Generation

The dataset was generated using the tools created in the *dataset* directory.

6.1.1 Webscraper

The webscraper tools (*dataset/webscraper*) were developed using Python and Selenium [26] to automate the process of gathering videos. The following scripts were written to accomplish this purpose and they are run in the following order:

- *Signbank_url_fetcher.py*
- *Signbank_video_url_fetcher.py*
- *Signbank_clip_fetcher.py*
- *Signbsl_url_fetcher.py*
- *Signbsl_video_url_fetcher.py*
- *Signbsl_clip_fetcher.py*
- *Sentence_url_fetcher.py*
- *Sentence_clip_fetcher.py*

The “clip_fetcher” scripts read the URL text files retrieved from the “url_fetcher” scripts and download these videos to a specified directory. As the videos are being downloaded, the script automatically sorts them into subfolders grouped by word. It also changes the filenames to be the word followed by a unique number (e.g. the third cat video would be called “cat_3”).

6.1.2 Dataset Reduction

After the new dataset has been generated, the dataset trim script (*dataset/dataset_trim.py*) cleans the generated subfolders and their contents by removing word folders and videos that were unsuitable to use for translation purposes. This encompassed folders with a poor naming convention, such as those containing invalid characters and invalid videos, such as those that were too long.

To standardise the length of videos, the distribution of video durations were analysed. To do this, *research/dataset_trim_duration.py* was produced to generate a duration histogram (*see Fig 28*). The majority of videos were between 1 and 3 seconds in duration. There were also some outliers from 6 to 60 seconds in duration, which were likely erroneous by having multiple words in a video rather than one. For the final implementation, the project used only videos of up to 3 seconds in duration.

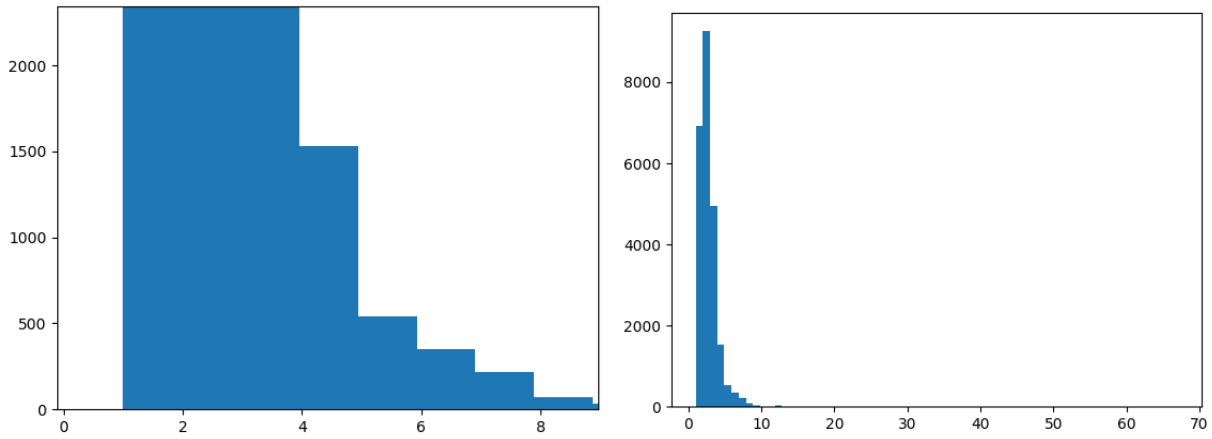

Figure 28: Graph detailing how many samples (y-axis) per video duration (x-axis)

Additionally, it was vital to scrutinise the spread of data across different word classes as there was extreme variance in videos per class, ranging from 1 to 35. This variation had the potential to skew the results, as classes with more samples were more likely to be chosen than those with fewer. This was addressed in the *dataset/dataset_trim.py* script which balanced each of the subfolders by ensuring each folder had exactly 5 videos.

In effect, folders that contained 5 or more videos were trimmed down to just 5 videos, and those with less than 5 were deleted. An added benefit of balancing the data was that it minimised the computational cost concerning training and reduced the training runtime from 1000 hours to 7 hours.

6.2 Preprocessing

The *preprocessing* folder contained the relevant scripts required to action the preprocessing stage. Because multiple steps were required, a single script (*preprocessing/run_preprocessing.py*) was created to automatically run all of the preprocessing steps outlined below.

6.2.1 Extracting Pose Estimation

The pose estimation script (*preprocessing/frankmocap_pose_estimation.py*) used FrankMocap to extract the wireframe pose coordinates from each video. First, the *preprocessing/run_preprocessing.py script* opened each video through OpenCV [27] and fed the frames through to the pose estimation library. OpenPose was also supported in *preprocessing/openpose_pose_estimation.py* and could be used if enabled through the command line arguments.

After running the script, a JSON file was produced for each class. These files contained the name, width, height and extracted pose coordinates for each video. The structure of the JSON file is detailed below where each object represents a different video.

[{“video”: <string>, “width”: <int>, “height”: <int>, “ body”: [[<int>,...]], “left _hand”: [[<int>,...]], “right_hand”: [[<int>,...]]},...]

A sampling rate could also be set through the command line arguments (*preprocessing*/*arguments.py)*. By default, it will sample 25 frames per second (as outlined in experimentation).

6.2.2 Normalisation

The pose data (body, left_hand and right_hand) in the JSON file produced from pose estimation, was then normalised using the ratio normalisation technique and flattened into a single array. The joint angle normalisation technique is also supported and can be used instead if enabled in command line arguments.

The results of the normalisation are appended to an extra field of each JSON object called “normalised” which contains the normalised data.

6.3 Training

The final model used was Dynamic Time Warping with KNN. This model was implemented in *training/dynamic_time_warping.py* and was based on an existing DTW notebook [16]. The matrix calculation process had been extended to support a standard calculation approach as well as the FastDTW [28] library.

The *training*/*train.py* script was used to split the ‘normalised’ data of the preprocessed JSONs into training and testing data, by the ratio 80:20. The training data was then fed into the DTW model to produce the distance matrix. The model was validated using the testing data which calculated the accuracy score.

The final training model was then saved to a specified file using Pickle [29], which meant that it could be used later without needing the dataset or to re-run the training again.

6.4 Application

The web application was delivered through Flask [30] with the main script being *app.py*. It was based on the design outlined in section 4 and implemented the page designs using HTML and CSS. All of the files for the application are stored in the *application* folder.

The application was designed to be as straightforward as possible. Therefore, on the first screen (*Fig 29*) the user can easily upload a video and press the submit button to start the process. They also have the choice to select "Advanced options" to enable additional configuration features outlined in section 4.

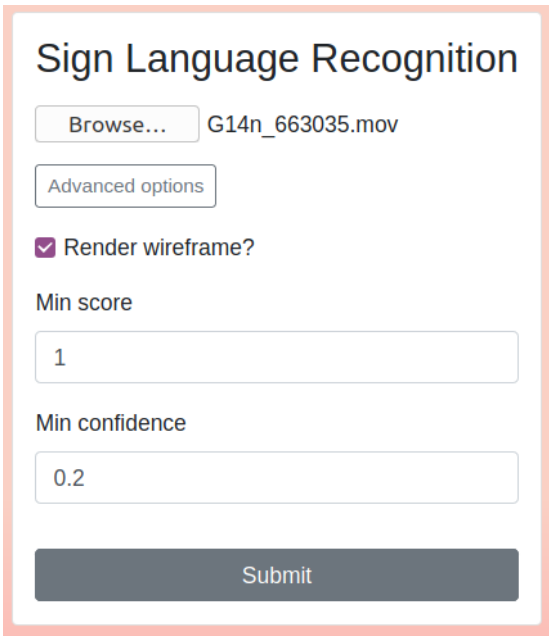

Figure 29: Screenshot of the user-interface

Flask SocketIO [31] was then used to handle the preprocessing and prediction as a background task whilst a progress bar was displayed to inform the user of the remaining runtime (*see Fig 30*).

Logs were produced by the Python Logging [32] module to send messages to the frontend, which a Javascript client used to update the page (*Fig 30)*.

Log information messages were displayed ("Converting video…" and "Predicting video labels") below the progress bar. During this process, it applied post-processing steps to the predictions which implemented the confidence filter from 5.6.2 (*application/predict.py*). When the background process finished, the app sent a "Complete" message to the frontend which redirected to the results page (*Fig 31*).

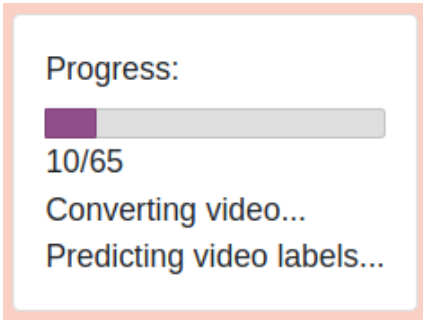

Figure 30: Screenshot of the progress bar indicating training progression

The results page automatically rendered the video with the predicted captions (*Fig 31*). The wireframe of the pose would also be displayed if the option had been selected on the first page (*Fig 29*). Options to download the subtitle and video files were provided via the buttons below the video.

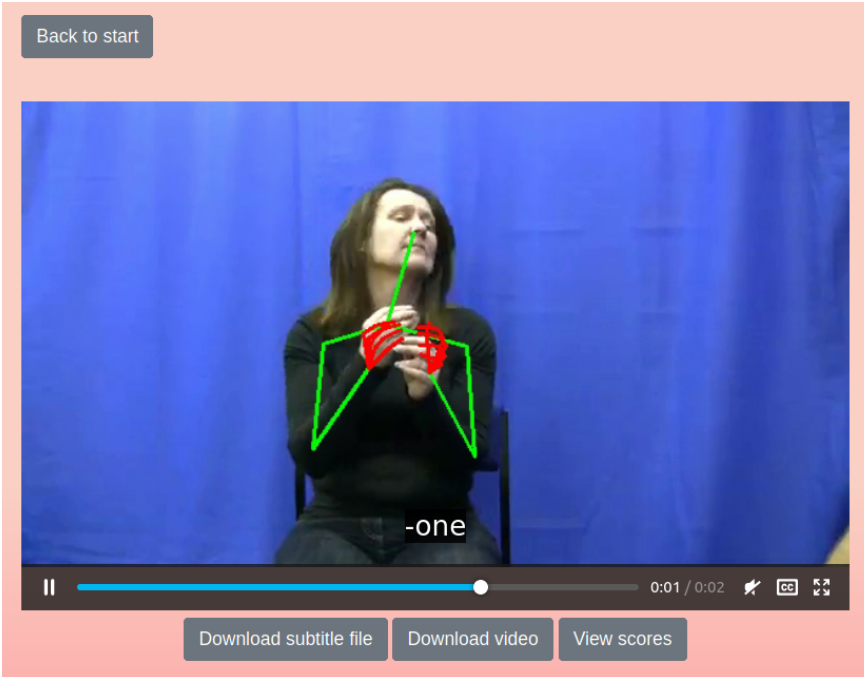

Figure 31: Screenshot of the results page displaying the video with captions

Selecting the “View Scores” button displayed the score post-processing generated for each predicted word as well as the start and end timings of when it thought the word appeared in the video (*Fig 32*).

Figure 32: Screenshot of the scores page detailing word predicted, score, start and end timings

7. Testing

7.1 Manual Testing

Due to the nature of this project, each step of the pipeline relied on manual testing. This involved running each script from the command line using argparse [33] and validating the results produced.

Validation included comparing the results of outputted scripts and also assessing the accuracy of the model and system to identify possible mistakes.

7.2 Usability Testing

Usability testing was applied to the web application to ensure that the required user experience was met. This involved observing users interacting with the system and receiving feedback from them to introduce design improvements to the system.

7.3 Unit Testing

To ensure the functionality of the system remained intact, a range of unit tests were implemented covering each step of the pipeline using PyTest [34]. The *test* folder contained the testing scripts along with additional testing files. The tests are described below:

test_preprocessing.py:

- Checks that a JSON file is produced for each video class
- Checks that a JSON object is present for each video
- Checks that the required JSON fields are present
- Checks that normalisation has been correctly applied

test_training.py:

- Checks that the class labels are correctly identified
- Checks that a model file is produced after training is complete
- Checks that predictions on a new video are valid

test_application.py:

- Checks that predictions under a certain score are ignored
- Checks that valid predictions are kept
- Checks that predictions with a small gap are combined
- Checks that predictions with a large gap remain independent
- Check that OpenCV can successfully extract video fps
- Check that it correctly builds a subtitle file (.vtt)

7.4 Extended Data Testing

To move towards a more realistic representation of the final system, it was necessary to increase the dataset size and expose the model to more words.

The larger dataset contained 27 words as opposed to the 6 words used in the experimental section. These words are defined as: go; have; football; small; alright; number; but; good; home; one; thought; I; before; always; finish; when; five; children; easter; better; summer; laugh; winter; spring; about; remember; story. These classes had 5 videos each and were chosen as they were the most common words found in the validation sentences (*dataset/sentence_split.py*).

The number of validation videos was also increased to cover as many of the words as possible. 6 validation videos were generated (as opposed to 2 used in the experimental stage) from the same BSL video sources (G27n, G14n, BF23n, G15n, G19n and G24n) which were trimmed to include relevant sentences only using the script, *dataset/sentence_file_reader.py*.

8. System Evaluation

The outcome of this project was a pipeline and application that was capable of producing and learning from a sign language dataset to predict words in sign language videos. The results provide a foundation for continued research in this area, which could ultimately facilitate tools that aid the use of digital communication between signers and non-signers.

The success of this project was evaluated against the system requirements set in section 3.

8.1 Final Results and Conclusions

The final extended dataset produced a training accuracy of 43% and validation accuracy of 30.4%. These scores were inadequate for a viable final application and demonstrated the complexity of the task. This showed that there is scope for further refinement in future research.

8.1.1 Sign Language Translation

This project has fulfilled the requirements set in 3.1 by producing an app that enables the user to upload a video and produce a subtitle file containing the translation. Additionally, a range of classifiers and data processing techniques have been explored in the experimentation section to produce a possible baseline system with room for further improvement.

8.1.2 Dataset Collection

This project has fulfilled the requirements set in 3.2 by generating tools that have sourced videos from the BSL Corpus [20], BSL Signbank [18] and SignBSL [19] websites. It has also provided cleaning methods to ensure the dataset is of high quality and fit for purpose.

8.1.3 Estimation Analysis

This project has fulfilled the requirements set in 3.3 by enabling parameters such as scoring to be easily accessed and configured through the app. It has also provided diagnostic information such as SVG visualisation and scoring data to help identify and enhance each step of the system.

8.1.4 Further Development

This project has fulfilled the requirements set in 3.4 by designing the system to facilitate interchangeability of tools including pose estimation and normalisation libraries. Additionally, design choices, such as selecting the KNN classifier, have been incorporated to maximise transparency in the system and facilitate its maintenance in the future.

8.2 Recommended Future Improvements

Although this project achieved its goal of being able to translate sentence-level sign language, there was still room for future improvements. Due to the time constraint, only a limited number of libraries were able to be explored. Additional research is needed to discover and experiment with new libraries in hopes to improve performance.

Continued investigation into the preprocessing and classification of data is recommended and should lead to better accuracy in the translation process. For example, a more complex deep learning classifier (i.e. LSTM) may improve performance.

A tool that could convert the predicted words into grammatically correct English (such as by using Natural Language Processing [35]) is recommended to be developed and would aid in the clarity and understanding of the translated results. Additionally, to date, no open-source projects incorporate facial expressions and human emotions during sign language translation. This could complement the system by better conveying the emotion of words [36].

Due to the system's structured design, it could be extended to cater for other sign languages such as American Sign Language (ASL). It could also be extended to support other action based classification problems such as semaphore flag signalling.

9. References

[1] K. Geddis. “Gitlab Project Repository.” Queen’s Gitlab Repository. [Online]. Available: https://gitlab2.eeecs.qub.ac.uk/40198325/sign-language-recognition

[2] K. Geddis. “Weights folder for FrankMocap.” Google Drive. [Online]. Available: https://drive.google.com/file/d/133FiIgbH_q3p5G2mm55OumTH89Zq4qzm/view?usp=sharing

[3] K. Geddis. “Pre-trained model weights.” Google Drive. [Online]. Available: https://drive.google.com/drive/folders/1L5lPZk-Of4qfPNYXYNlIMDUX5SsEtc8S?usp=sharing

[4] British Deaf Association. “BSL Statistics.” [Online]. Available: http://signlanguageweek.org.uk/bsl-statistics

[5] Wikipedia, “Activity Recognition,” [Online]. Available: https://en.wikipedia.org/wiki/Activity_recognition

[6] Allah Bux, “Vision-based Human Action Recognition using Machine Learning Techniques,“ 2017. [Online]. Available: https://core.ac.uk/download/pdf/141542312.pdf

[7] Biyi Fang, Jillian Co and Mi Zhang, “DeepASL: Enabling Ubiquitous and Non-Intrusive Word and Sentence-Level Sign Language Translation,” 2018. [Online]. Available: https://arxiv.org/abs/1802.07584v3

[8] Amit Moryossef, Ioannis Tsochantaridis, Roee Aharoni, Sarah Ebling, Srini Narayanan, “Real-Time Sign Language Detection using Human Pose Estimation”, 2020. [Online]. Available: https://arxiv.org/abs/2008.04637

[9] Feiyu Chen and Qiancheng Zhao, “Real-time Action Recognition Based on Human Skeleton Video”, 2019. [Online]. Available: https://github.com/felixchenfy/Data-Storage/blob/master/EECS-433-Pattern-Recognition/FeiyuChen_Report_EECS433.pdf

[10] Harsh Gupta, Siddharth Oza, Ashish Sharma, and Manish Shukla, “Sign Language Interpreter using Deep Learning,” 2019. [Online]. Available: https://github.com/harshbg/Sign-Language-Interpreter-using-Deep-Learning

[11] “The Top 62 Human Pose Estimation Open Source Projects.” [Online]. Available: https://awesomeopensource.com/projects/human-pose-estimation

[12] Gines Hidalgo, Zhe Cao, Tomas Simon, Shih-En Wei, Hanbyul Joo, and Yaser Sheikh, “OpenPose,” 2017. [Online]. Available: https://github.com/CMU-Perceptual-Computing-Lab/openpose

[13] Mohit Maithani, “Guide to OpenPose for real-time human pose estimation,” [Online]. Available: https://analyticsindiamag.com/guide-to-openpose-for-real-time-human-pose-estimation/

[14] Wikipedia, “Dynamic Time Warping.” [Online]. Available: https://en.wikipedia.org/wiki/Dynamic_time_warping

[15] Jeremy Zhang. “Dynamic Time Warping.” [Online]. Available: https://towardsdatascience.com/dynamic-time-warping-3933f25fcdd

[16] Mark Regan, “Timeseries Classification: KNN & DTW.” [Online]. Available: https://nbviewer.jupyter.org/github/markdregan/K-Nearest-Neighbors-with-Dynamic-Time-Warping/blob/master/K_Nearest_Neighbor_Dynamic_Time_Warping.ipynb

[17] Hoang Anh Dau, Diego Furtado Silva, François Petitjean, Germain Forestier, Anthony Bagnall, Abdullah Mueen and Eamonn Keogh, “Optimizing dynamic time warping’s window width for time series data mining applications,” 2018. [Online]. Available: https://link.springer.com/article/10.1007/s10618-018-0565-y

[18] UCL. “BSL Signbank.” [Online]. Available: https://bslsignbank.ucl.ac.uk/dictionary/

[19] SignBSL “British Sign Language Dictionary.” [Online]. Available: https://www.signbsl.com/

[20] Economic & Social Research Council. “British Sign Language Corpus Project.” [Online]. Available: https://bslcorpusproject.org/project-information/

[21] Yu Rong, Takaaki Shiratori, Hanbyul Joo, “FrankMocap: A Fast Monocular 3D Hand and Body Motion Capture by Regression and Integration,” 2020. [Online]. Available: https://penincillin.github.io/frank_mocap

[22] Renu Khandelwal. “Data augmentation techniques in python.” [Online]. Available: https://towardsdatascience.com/data-augmentation-techniques-in-python-f216ef5eed69

[23] Sklearn. “Classifier Comparison.” [Online]. Available: https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html

[24] Anaconda. “Python distribution library.” [Online]. Available: https://www.anaconda.com/

[25] Python. “Programming language documentation.” [Online]. Available: https://www.python.org/

[26] Selenium. “Selenium API documentation.” [Online]. Available: https://selenium-python.readthedocs.io/

[27] OpenCV. “Open Source Computer Vision Library.” [Online]. Available: https://opencv.org/

[28] FastDTW. “Python approximate Dynamic Time Warping library.” [Online]. Available: https://pypi.org/project/fastdtw/

[29] Pickle. “Python object serializer library.” [Online]. Available: https://docs.python.org/3/library/pickle.html

[30] Flask. “Flask API documentation.” [Online]. Available: https://flask.palletsprojects.com/en/1.1.x/

[31] Flask SocketIO. “Flask SocketIO library.” [Online]. Available: https://flask-socketio.readthedocs.io/en/latest/

[32] Logging. “Python logging library.” [Online]. Available: https://docs.python.org/3/howto/logging.html

[33] Argparse. “Python parser for command-line options.” [Online]. Available: https://docs.python.org/3/library/argparse.html

[34] PyTest. “Python unit testing library.” [Online]. Available: https://docs.pytest.org/en/6.2.x/

[35] Wikipedia. “Natural Language Processing.” [Online]. Available: https://en.wikipedia.org/wiki/Natural_language_processing

[36] Gov.uk, “Research and analysis Saleem: profoundly deaf user,” [Online]. Available: https://www.gov.uk/government/publications/understanding-disabilities-and-impairments-user-profiles/saleem-profoundly-deaf-user

[37] ResearchGate, “Sign language recognition using sensor gloves,” 2002. [Online]. Available: https://www.researchgate.net/publication/4014533_Sign_language_recognition_using_sensor_gloves

10. Appendices

1. The user installation guide can be found in the README.md of the Github project [1]
2. Data used for the experiments in this project can be found in the following OneDrive folder (*link: https://qubstudentcloud-my.sharepoint.com/:f:/g/personal/40198325_ads_qub_ac_uk/EotuxQZ8A0FOrdCfU5pT5Y0B8SLcJmYESiJ669u7fH_WKA?e=THRMlF*)
    a. Folder /small_dataset contains the testing videos and JSON dataset used for the experimentation
    b. Folder /large_dataset contains the final videos used to produce the final results in the application
    c. Folder /screenshots contain various screenshots used to analyse the systems performance and decision making