IMPERIAL

Year 3 Group Project

Group 27 – Brain Density

Department of Bioengineering

Imperial College London

A Project Report Submitted in

Partial Fulfilment of the

Beng/ MEng/ iBSC Degree

Supervisor: Dr Juan Alvaro Gallego

Department of Bioengineering

juan-alvaro.gallego@imperial.ac.uk

April 2025

Bridging the Gap Between Neural Intent and Hand Kinematics

Chatananchai Jinny, Jacob Josh, Lin Jasper, Perera Neo, Syn Felix, Thevarajah Mahish, Wasim Malaika

15th April 2025

Abstract

The loss of hand dexterity due to neurological injuries, such as spinal cord injuries or strokes, severely impacts the quality of life for millions worldwide. In the U.S., 2.5 million individuals live with upper limb paralysis, yet fewer than 10% of stroke patients fully recover hand function. Restoring natural hand movements remains a critical challenge in neurorehabilitation, as existing technologies often lack the precision to decode motor intent effectively.

This project develops a neural interface to bridge this gap by translating muscle activity into hand kinematics. Using high-density surface electromyography (sEMG), muscle signals were recorded from the forearm whilst participants performed American Sign Language (ASL) gestures. These signals are paired with motion capture data from three high-speed cameras and are processed via AI-driven pose estimation (SLEAP), which tracks 20 nodes along the hand in a 3D space. Machine learning techniques, including dimensional reductive techniques and long short-term memory (LSTM) networks, map sEMG signals to kinematic outputs, enabling real-time movement prediction.

Preliminary results demonstrate the system's potential, with the project's LSTM model explaining 19.24% of the variance in predicting the 3D displacement of hand nodes. Current efforts focus on optimising the deep learning model using principal component analysis (PCA) for dimensionality reduction, and refining 3D hand reconstruction via camera calibration such as a ChArUco board and Sleap-anipose.

This work has broad implications for neurorehabilitation, offering a foundation for intent-driven therapies, prosthetic control, and assistive technologies. By decoding motor intent with high precision and accuracy, this interface can aid individuals with paralysis to regain independence.

1. Introduction

Dexterous hand use through fine motor coordination is vital to carry out tasks of daily living such as eating, grooming, and socialising. Unfortunately, in the case of patients with spinal cord injury or stroke, they often suffer from paralysis and a loss of motor control [1]. In particular, the loss of function in the upper limbs and hands can

leave patients unable to complete basic tasks individually. Many current therapies revolve around rehabilitation through physiotherapy and exercise [2], but these rarely manage to help patients restore full function and regain complete independence. However, there are several treatments in development that leverage technologies such as Artificial Intelligence (AI), Transcranial Magnetic Stimulation and Nervous or Spinal Stimulation that promise to make substantial improvements in patient outcomes and restore higher degrees of motor control within shorter time periods [3].

Neural interfaces that can map motor intent to hand movement hold massive rehabilitative prospects and can give these patients a new lease of life. They hold the potential to restore innate hand function by being coupled with nervous stimulation technologies. Alternatively, by enabling a prosthetic to decode the motor intent, it will be able to accurately execute the intended hand movements [4] [5].

sEMG is a fast, non-invasive method used to quantitatively assess the electrical activity generated by skeletal muscle fibres contractions induced by motor neuron firing. The sEMG provides a measure of composite electrical signals produced by motor unit action potentials [6], which enable us to infer muscle activation levels during dynamic hand gestures.

In recent years, researchers have made substantial progress in developing methods for precise mapping of human arm movements using sEMG signals. Multi-channel approaches allow for improved action recognition and mapping through deep learning algorithms. Work by Sîmpetru in 2014 has shown promise in achieving highly accurate hand movement prediction and execution, laying a solid foundation for the development of more intuitive and responsive prostheses and assistive devices [7]. Although challenges exist in extracting clean signals, foundational work by De Luca et al. (2015) has established several filtering techniques to reduce noise, power line interference and motion artifacts [6].

One key milestone in the mapping process is to convert the 2D videos taken by the cameras into a 3D system. The sEMG signal can then be mapped onto the 3D system. Nath et al. (2019) demonstrated the effectiveness of DeepLabCut software in converting 2D video data to 3D pose estimations [8], with many sources of relevant literature including Simpetru et al. (2014) implementing this system. However, in 2022 Pereira et al. (2022) introduced SLEAP as a more advanced tool for animal pose tracking with flexible model architecture, enhanced data labelling tools and better performance on small datasets [9].

To further enhance the accuracy of hand gesture recognition, novel approaches have been introduced, such as integrating a lightweight sEMG-based system with a 4-stream deep learning architecture. This innovative method is strategically combined with LSTM Neural Networks. LSTMs excel at processing and learning sequential data, particularly time series data, and their architecture allows them to effectively capture and recognise sequences over extended periods of time. These advantages have been shown to aid LSTMs in effectively

classifying hand movements in the context of sEMG data [10].

The project seeks to combine motor intent, in the form of sEMG data, with a motion capture system that records the associated hand movement. By synchronising the capture of the data and feeding it through a machine learning algorithm the aim is to decode the sEMG signals into a 3D virtual hand model. The virtual hand model and decoded signals can act as the foundation for technologies that can predict and induce motor movements by stimulating nerves. Alternatively, they can act as the inputs of a prosthetic arm that can be controlled solely with motor intent.

2. Methods

The methodology starts with two concurrent data collection processes for sEMG and hand movement motion capture.

sEMG data was collected using a Quattrocento [11] (amplification device) connected to a 64-channel electrode grid placed carefully on the participant's forearm to capture precise signals of the muscles of interest. A comprehensive schematic of the entire data collection apparatus can be seen in *Figure 1(A)* below.

The participant performs specific repeated hand movements, and the raw EMG signals are collected to later be processed in MATLAB. Processing involves noise filtering, artifact removal and the use of normalisation techniques. The filtering approach implemented the notch filters and low pass filters determined by the dominant frequency in each signal [6].

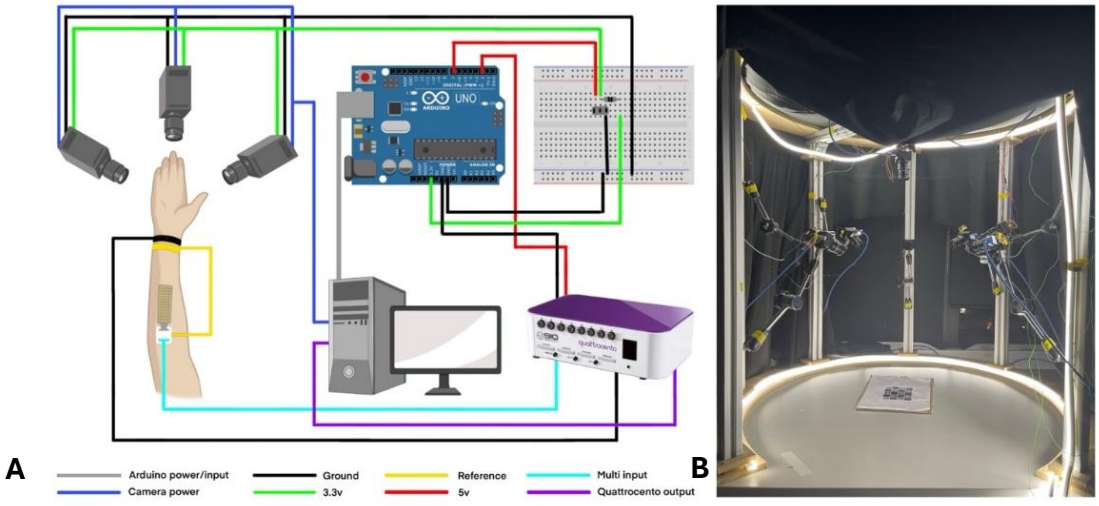


Figure 1: **(A)** Complete setup schematic for synchronised data collection of sEMG signals and Motion Capture. **(B)** Real life view of the camera system to optimise key feature acquisition during Motion Capture

Motion capture of hand kinematics involves a setup using three high speed cameras as seen in *Figure 1(B)*. The camera system was first calibrated using a ChArUco checkerboard and metadata was assessed to ensure a consistent frame count across cameras [12]. Camera footage of hand motion from multiple angles were then recorded simultaneously using the Jarvis Acquisition Tool software. The footage was exported and manually labelled with 20 2D markers, one for each joint of the hand (*Appendix A*) with the assistance of SLEAP. The 3D virtual hand model was reconstructed by triangulation with the sleap-anipose python package (*Appendix B)*.

3D data was aligned with the sEMG recordings based on the synchronisation timestamps, thereby establishing direct correspondence between muscle activation patterns and hand movements. This alignment enhanced the quality of the data used for further analysis and modelling.

Hand movement footage and sEMG signals were captured concurrently to establish temporal correspondence between the two data streams. An Arduino microcontroller generated a shared trigger pulse that synchronised the camera system with the sEMG recordings (*Appendix C*).

A LSTM setup was employed to decode the temporal dynamics between the aligned sEMG signals and the extracted hand kinematics., which provided a reliable foundation for subsequent neural decoding.

2.1. sEMG

Prior to electrode placement, a target area on the participant's forearm (as seen in *Figure 1(A)*) was localised by instructing the subject to perform repetitive finger flexion, during which the target muscle, the Flexor Carpi Ulnaris (FCU), became visually prominent.

The target area was prepared by first shaving the vellus hair to ensure maximal electrode-skin contact. An abrasive, high-chloride electrolyte gel (Abryalyt) was then applied to exfoliate the skin by removing dead cells with friction, thus reducing electrode impedance and minimising noise [13]. Final cleaning with Meliseptol rapid disinfectant ensured further optimised skin conductance and maintained participant comfort by removing any remaining impurities and removing leftover abrasive gel respectively.

For signal acquisition, a 64-channel electrode grid was prepared by first affixing an adhesive foam pad to the electrode strip. Ten20 conductive neurodiagnostic paste, which serves to enhance signal quality, was then applied evenly to cover all channels [14]. The grid was then positioned on the forearm above the FCU, due to its involvement in ulnar aspect flexion [15] as seen in the American Sign Language gestures for letters A-E (*Appendix D*). Casting tape was used to secure the electrode array firmly, thus ensuring adherence to the forearm's contours and reduced instances of motion artifacts.

The sEMG recordings were acquired using a Quattrocento system configured via the OT BioLab+ software. A wet wristband was applied to ground the sEMG signals, and a second interfaced with a monopolar filter (bandpass filter) was applied to remove baseline muscle activity and noise [16].

An Arduino board provided an external trigger to start and end the data acquisition, thus ensuring that sEMG recordings were temporally aligned with the motion capture of hand kinematics. During recording sessions, participants performed designated hand signs at 3-second intervals with interspersed relaxation periods, thereby allowing clear differentiation between muscle contraction and rest states. All procedures followed the general principles for research *(Appendix E)*, which ensure participant safety and data integrity.

Following 3D reconstruction, the corresponding sEMG data underwent pre-processing in MATLAB to enhance signal quality and reduce noise (*Appendix F)*. The MATLAB code implemented a 50Hz notch filter to eliminate powerline interference. Subsequently, a Fourier transform was conducted to identify the dominant frequency of each signal, which was then used to design an adaptive low-pass filter with a cutoff frequency set to 5Hz above the dominant frequency [17]. The transformed signal can be visualised in *Figure 2.*

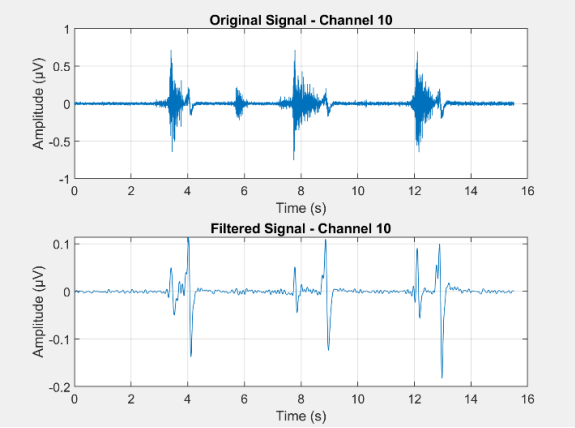


Figure 2: Original and MATLAB processed sEMG signal

To address the complexity of high-dimensional sEMG data while preserving critical neuromuscular information, Principal Component Analysis (PCA) was performed as coded in *Appendix G*. This technique reduced redundancy by identifying orthogonal axes of maximal variance, thereby isolating dominant muscle activation patterns. The top three principal components, which captured more than 95% of the total variance, were retained to encode the primary patterns of synergistic muscle coordination. Reducing dimensionality resulted in minimised noise and computational load, whilst retaining physiologically meaningful features, hence ensuring robust neural decoding. The extracted components served as efficient and interpretable inputs for machine learning models [18].

2.2. Camera Systems for 3D Hand Kinematic Analysis

Precise tracking of hand kinematics is vital for understanding dexterous movements and their relationship to muscle activity.

Traditional marker-based motion capture systems have limitations such as physical marker obstruction and movement interference [19]. This project employs a marker-less system utilising three synchronised high-speed cameras to address such issues (*Appendix H*).

2.2.1. Hardware Configuration

The set-up is mounted on a modular steel-and-wood frame that allows for flexible repositioning of the cameras to suit various experimental conditions, as seen on *Figure 1B.*

The motion capture system was designed with careful attention to camera placement, lighting and synchronisation to optimise tracking performance. Of the three cameras, two were positioned laterally with one mounted overhead. This arrangement minimises occlusions and provides a comprehensive coverage of dynamic hand movements from multiple angles. Whilst 2 camera angles are the minimum for 3D reconstruction, 3 or more provide greater accuracy [20]. To ensure consistent visibility and reduce the impact of shadows or uneven lighting, two continuous LED light strips were mounted around the upper and lower perimeters of the setup. This provided uniform illumination, ensuring contours of the subject's hand remained clearly visible throughout motion capture [21].

The cameras were controlled via FLIR's SpinView software, which was configured for manual exposure settings (8ms), to prevent exposure fluctuations [22]. A black fabric backdrop encased the setup to increase visual contrast between the hand and the environment, as well as to block out any external sources of visual noise.

2.2.2. Synchronisation and Triggering

A custom synchronisation circuit was developed to ensure all three cameras captured frames simultaneously.

An Arduino Uno generated a 5V square wave at 50 Hz (converted to 3.3V) to synchronise the FLIR cameras and sEMG recordings, ensuring frame-accurate acquisition within ±1 ms [23].

This hardware-triggered configuration provided two primary benefits. First, it ensured that each camera recorded the same number of frames during each recording session, which is essential for downstream 3D reconstruction. Second, by separating the time the camera sensor is open (exposure) from the way it reads each row of pixels one after the other (rolling shutter) and relying on external synchronisation, the system minimised temporal jitter and motion blur. This improved the spatial accuracy and reliability of the 3D point triangulation of the system.

2.2.3. Calibration Procedures

Calibration of the multi-camera system was performed in two stages: intrinsic and extrinsic. This is essential for the 3D reconstruction of hand markers [24]. Intrinsic calibration aimed to determine individual camera parameters such as focal length, principal point, and lens distortion coefficients. Extrinsic calibration establishes the spatial position and orientation of each camera view in 3D. A ChArUco board as seen in *Appendix I* was manually translated and rotated within each camera's field of view. These calibration images were processed using Sleap-anipose, to extract and optimise intrinsic and extrinsic parameters for each camera using iterative bundle adjustment [25]. This process involves minimising the projection error from the 2D image plane of each camera into 3D points.

To automate the calibration process using the sleap-anipose python package, a custom Python script was employed (*Appendix B*). The 'calibrate' method was used on the synchronised recordings of the ChArUco board, from all cameras, to implement intrinsic and extrinsic calibration. The output

was a file containing the necessary intrinsic and extrinsic parameters for 3D reconstruction with iterative bundle adjustment to find the optimal 3D position of each point. This minimised the reprojection error back to the 2D image plane of each camera [9][21]. The final optimisation yielded a reprojection error of less than 0.5 pixels, thereby indicating high geometric accuracy, which is suitable for the 3D reconstruction of fine motor movements.

2.3. Hand Kinematic Recordings

To evaluate the system's capabilities, hand kinematic data was recorded during ASL gestures – such as the representation of the letter 'A' amongst others. Each trial was captured at 25Hz across all cameras, providing sufficient resolution to analyse the movements and transitions.

2.4. Integration

2.4.1. Hand Kinematics and sEMG Data Synchronisation

To correlate muscle activity with hand kinematics, the motion capture system is synchronised with a Quattrocento sEMG system, which records data from the 64-channel electrode.

An Arduino Uno microcontroller, programmed using the MsTimer2 library (see Appendix C), generated a square wave that was converted to 3.3V using a MOSFET-based voltage divider to meet the FLIR cameras' trigger requirements. The converted square wave was distributed to each camera and used to timestamp both the video and sEMG data. This synchronisation facilitates direct mapping between muscle activation patterns and the corresponding 3D hand movements.

The integration system would enable motor unit analysis, where distinct task-specific and postural motor units are identified. It also enables neural decoding, with LSTM networks achieving up to 80% accuracy in translating sEMG signals into hand movements [26].

2.4.2. Processing Camera Footage

The post-processing workflow begins with SLEAP, a deep-learning based framework for pose-tracking. This was used to label entire hand motion recordings with 20 key hand points (nodes) on each frame. An image of a labelled frame can be seen in *Figure 3(A)*.

After labelling the synchronised recordings from all cameras, the 2D coordinates of the nodes from each camera were exported for 3D reconstruction with sleap-anipose. The 3D position of all 20 nodes on each frame was visualised with the matplotlib.pyplot python package, as seen in *Figure 3(B)*, to verify the success in 3D reconstruction (see *Appendix J*).

2.5. Neural Networks (NNs)

Classification algorithms like N-nearest neighbours or Support Vector Machine have had success in classifying hand gestures corresponding to certain sEMG signals [27]. Whilst effective, these methods struggled with interpreting signal variability and redundancy. Neural Networks (NNs) have shown up to 91% accuracy in predicting hand motion from sEMG signals [28] and account for redundant components from the input [29].

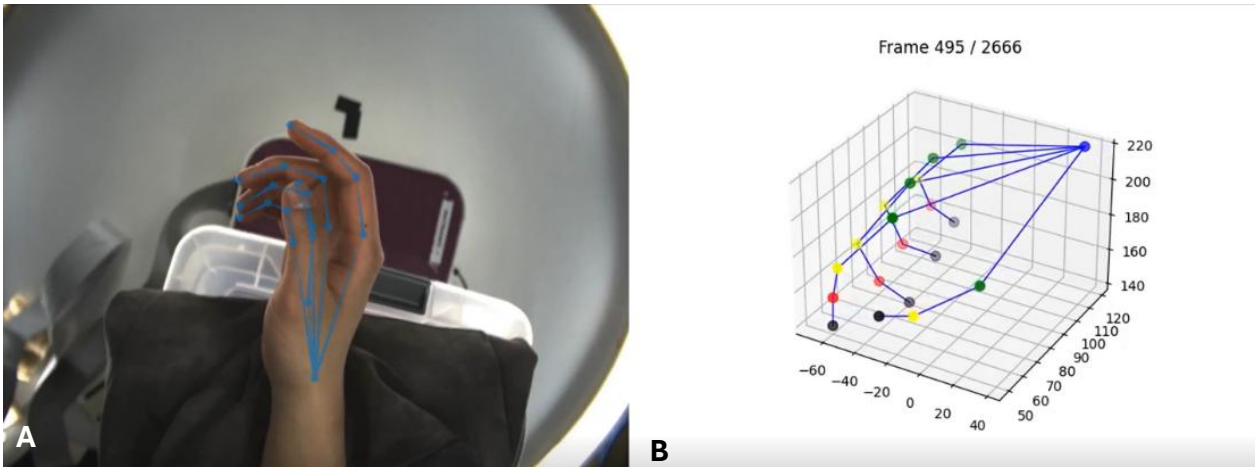


Figure 3: **(A)** Frame labelled with connected skeleton of 20 hand nodes. **(B)** 3D reconstruction of the same hand.

2.5.1. LSTM Neural Networks

LSTM networks have a specialised neural network architecture that incorporates the long-term characteristics of signals. This is crucial as muscle activity for an action occurs over time, not within an instant. It was decided that a neural network would be used to address the project's aims.

With reference to Figure 4, a LSTM block, the fundamental processing unit of a LSTM network starts by choosing 2 random starting values, the initial long-term (C_{t-1}) memory and short-term (h_{t-1}) memory. When an input value (X_t) arrives, weights and biases are applied to X_t, h_{t-1} and C_{t-1} to update long-term memory to C_t [30][31].

Weights allow only a proportion of the value of interested to be taken, while biases change the value by a specific amount. This allows calculation of the weight and bias to update C_{t-1} to C_t. This is repeated with C_t, h_{t-1} and X_t to update h_{t-1} to h_t. h_t and C_t can then be used with the next input value in the next LSTM block.

To train the LSTM network model, the inputs are fed into each block sequentially to undergo linear and non-linear transformations, called forward propagation. The difference (error) between the output (h_t) and reference values is used to compute the loss function to represent the error across all the outputs. The loss function is used to update the weights and biases used in the network to minimise itself. This was termed backpropagation [32].

To reduce the loss function and attain the best model, the cycle of forward propagation and backpropagation was repeated multiple times. This model was then tested on a new dataset to evaluate its performance.

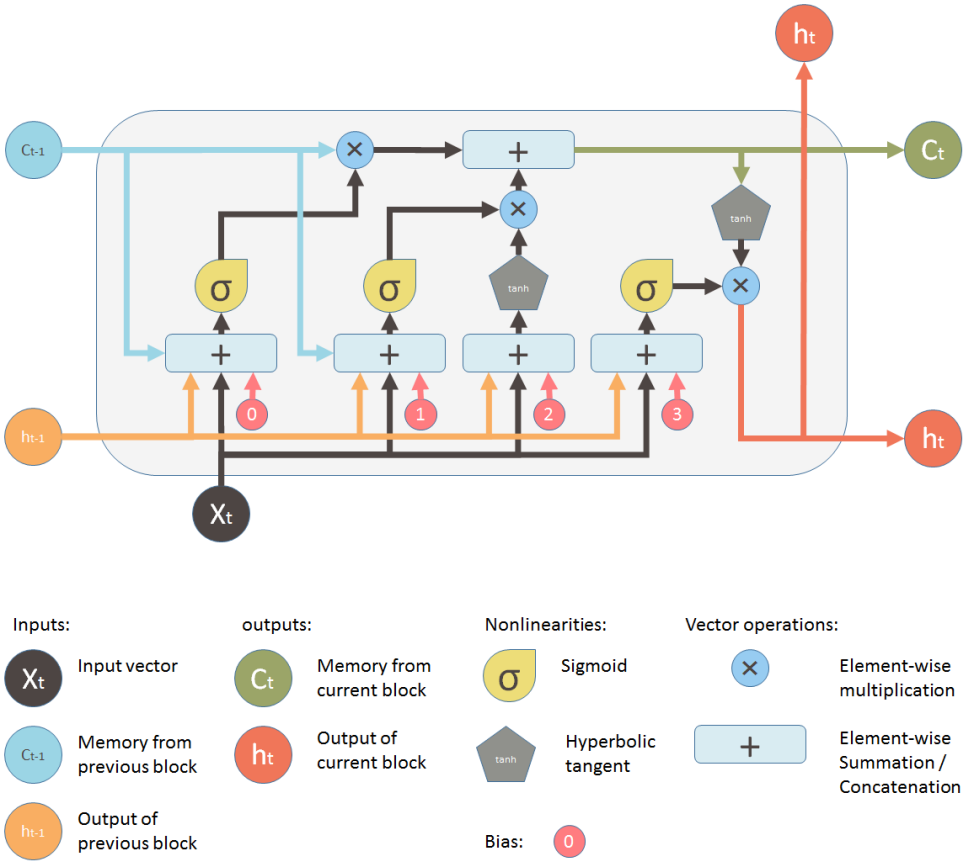


Figure 4: Visualisation of how an LSTM block's internal processes. Weights are applied as element-wise multiplication. Biases are applied as element-wise summation.

2.5.2. Labelling sEMG data with reference labels

The aim was for the model to predict hand motion with low latency. sEMG data was labelled to train the model. First, hand motion was represented between frames by computing the change in 3D coordinates for all 20 triangulated nodes. For hand position data with N frames, N-1 datapoints for hand motion were attained as the difference in positional data between every 2 consecutive frames was taken.

Raw sEMG data was sampled at 2048Hz, while the 3D data was acquired at a much lower frequency of 25Hz. This mismatch was used in order to allow a sequence in the sEMG signal, represented by 3 PCA components, to correspond to a specific hand motion, represented by a numerical change in 3D coordinates [33]. 80% of these labelled short sequences were input into the LSTM network sequentially to train the model. The remaining 20% of sequences were used to test the model on test data.

Mean Square Error (MSE), Root Mean Square Error (RMSE), Mean Absolute Error (MAE) and the coefficient of determination (R^2) were used as metrics to measure the

performance of the model. The ideal outcome was to attain lower MSE, RMSE and MAE error values but higher R^2 values closer to 1. R^2 tells represents how well the model explains the variance in the output with the input. The model with the lowest MSE value from all the training loops was chosen as the best model.

2.5.3. PyTorch

PyTorch [34], a machine learning framework and (python package) was used to construct the LSTM network architecture (comprising input, hidden and output layers) divided the data into training and test sets, and defined key hyperparameters governing data processing and model training (see model results in *Appendix K*).

The LSTM architecture consists of 3 nodes on the input, corresponding to each PCA component, followed by 2 hidden layers of 64 nodes, for increased complexity, and 60 output nodes that correspond to the 3D coordinates for each of the 20 hand nodes of interest. To train the model well, various hyperparameter values were trialled to reduce the loss function, MSE.

Hyperparameters were changed by trial and error to attain better results with a lower loss function value without excessive overfitting [35]. Overfitting occurs when the model trains very well on a training set but fails to perform well on new data. Optimisation includes adjusting the learning rate and batch size hyperparameters to help reduce training time whilst getting optimal metric values (MSE, RMSE, MAE, R^2). This allows the LSTM network to train more efficiently.

To compare the effect PCA has on model training, another control LSTM architecture, with 64 input nodes to take in raw 64 channel sEMG signals, was constructed with the same hyperparameter values.

3. Results

Input	MSE	RMSE	MAE	R^2
(A) Raw sEMG	0.3349	0.5787	0.2145	0.1924
(B) 3 PCA components	0.3756	0.6128	0.2074	0.0391

Table 1: Statistical analysis results of raw sEMG and PCA processed signals

After running a training loop of 300 epochs using raw (A) and processed (B) sEMG data, and the LSTM network architecture, the best model with input A had a lower MSE than that with input B, when evaluated on the test set. This indicates that model A was able to generate values closer to the reference than model B. The RMSE values give the same conclusion. The MAE for model A was higher than model B. The MSE has units which are the square of the reference values, thereby exaggerating large errors more than the RMSE and MAE (which have the same units as the output).

Model A had a higher R^2 value than model B, therefore model A can better explain the variance in the reference data. However, the generally low R^2 values of both models, indicated that neither model explains the variance in the data well.

These results suggest that using 3 PCA components as an input caused worse performance compared to using raw sEMG signals for the same dataset.

4. Discussion

This study demonstrates a comprehensive framework for capturing and analysing hand kinematics in conjunction with sEMG signals to decode their relationships. A markerless motion capture system incorporating synchronised FLIR cameras, combined with labelling and triangulation tools such as SLEAP and sleap-anipose, enabled robust 3D reconstruction of hand movements using only three camera angles. Out of the 8 cameras available, only 5 were utilised due to faulty wiring. Due to time constraints, SLEAP labelling and triangulation was limited to 3 camera angles. However, this provided a sufficiently comprehensive view of the subject's hand for 3D reconstruction.

The approach taken overcomes many of the limitations associated with traditional marker-based systems, such as occlusion and interference, and provides a detailed spatiotemporal mapping of hand kinematics.

Limitations in frame rate to 25 fps, despite the cameras supporting higher frame rates, reduced the amount of temporal data available. This limited the model's ability for fine-motor control. Higher frame rates would result in more training data allowing for potential improvement in the LSTM model.

Which muscle signals correspond to specific hand movements was determined through the integration of sEMG data with the motion capture system. While this synchronisation is critical for correlating muscle activation with kinematics, surface EMG also presents challenges due to crosstalk [36]. The overlapping anatomical structure of the forearm muscles, such as the FCU, flexor carpi radialis, and palmaris longus, could lead to ambiguous readings that overestimate muscle activity.

Although placing electrodes to target specific muscles and using meticulous skin preparation mitigates some of these issues, the lack of muscle-specific resolution remains a notable limitation of sEMG. Alternative techniques like invasive EMG may have yielded EMG data with less noise and fewer artifacts [37][38].

The machine learning approach, particularly the application of LSTM networks to capture the temporal dynamics inherent in sEMG signals for neural decoding, demonstrated promising results. The best model predicted 19.24% of the variance in hand motion data from sEMG signals despite a limited data set.

The low R^2 values indicate that a significant portion of the variance in hand motion remains unaccounted for, likely due to the limited dataset and the complexity of mapping sEMG to 3D kinematics. Increasing the sample size would capture a broader range of inter-subject variations and muscle activation nuances, thereby enhancing model

generalisation. These poor results can be attributed to insufficient data available to train the model.

The SLEAP pipeline, where training a model to assist in labelling requires multiple hours, yet can still result in errors, caused more time to be committed to troubleshooting. This resulted in fewer reference values being produced for the LSTM. Thus, highlighting that further refinement is required to improve the robustness of the model.

Additionally, refining signal processing- such as testing alternative dimensionality reduction techniques, such as Adaptive Dimensionality Reduction [39], or optimising the number of PCA components - may preserve critical information currently lost. Improvements in motion capture calibration, for example by incorporating more cameras or increasing the frame rate, could yield more accurate 3D reconstructions and better temporal alignment with the sEMG signals.

During data preprocessing, limitations in applying the PCA to a full data set, due to memory constraints, were encountered. The large recording files made it impractical to process the entire dataset. To overcome this, PCA was applied to the dataset in smaller portions. A draw-back of implementing the more memory-efficient method resulted in data segments being processed independently.

The network architecture or more hyperparameters like weight decay rate and batch size could also be modified to make a better model with a higher R^2 value and lower MSE, RMSE and MAE values. The number of PCA components used as an input could be refined, provided sufficient data is available for training. Collectively, these enhancements are expected to increase the explained variance and bolster the system's clinical and neuro-prosthetic potential.

Additionally, access to Imperial's High-Performance Computing (HPC) Facility for SLEAP model training and LSTM network training was attained. The laboratory computer (a shared workspace) lacked available storage space and GPU memory for the tasks and was also not readily available as and when it was needed. Leveraging remote computing allowed data to be analysed remotely without the need to use the laboratory computer in person for extended lengths of time.

4.1. Future Works

With more time, more data could be collected, processed and input into the LSTM, enhancing the model's learning capacity. The implementation of both a memory-efficient algorithm and HPC would significantly reduce processing time and allow for a parallel workflow. This would enable both full dataset processing (through avoiding data segmentation) as well as leading to the possibility of real-time data handling. Ultimately, such improvements may allow the system to be viable for real-time neuro-prosthetic control applications.

The network architecture or more hyperparameters, like weight decay rate and batch size, could also be modified to make a better model with a higher R^2 value and lower MSE, RMSE and MAE values. The number of PCA components used as inputs could also be refined, provided sufficient data available

for training. Collectively, these enhancements are expected to increase the explained variance and bolster the system's clinical and neuro-prosthetic potential.

5. Acknowledgements

We would like to express our sincere gratitude to our academic supervisors and friends, Dr Juan Alvaro Gallego, Ciara Gibbs, Natalia M Torres-Consul, Lara C Gouveia Vila, Genji Kawakita and Byron Chan for their invaluable guidance and insightful feedback throughout this project. Their expertise and unwavering support were instrumental in shaping the direction and success of our work.

We gratefully acknowledge the Bioengineering Department at Imperial College London for hosting our research, and we extend our thanks to the Level 4 Sir Michael Uren Hub (Dario Lab) for providing the essential laboratory facilities and resources.

We also wish to thank the teams and individuals responsible for the development and support of the software and hardware tools used in this project. Special appreciation goes to the developers and technical support staff for the Quattrocento system, FLIR cameras, SLEAP and Sleap-anipose software, as well as those behind Python, MATLAB, the Arduino IDE, Arduino UNO, the JARVIS Acquisition Tool, and OT+ BioLab systems. Their contributions have been integral to the successful implementation and operation of our integrated neural interface framework.

Finally, we extend our heartfelt thanks to everyone who assisted with administrative and logistical support, ensuring the smooth execution of this project. Your collective efforts have made this research possible, and we are truly grateful for the support and collaboration provided.

6. Author Contributions

Dario Farina, Juan Alvaro Gallego, and Ciara Gibbs played key roles in the initial conceptualisation and design of the project. Oversight and continuous guidance were provided by Dr Juan Alvaro Gallego and Ciara Gibbs, ensuring that the project remained focused on its core objectives throughout its duration.

The markerless pose estimation implementation using SLEAP was a collaborative effort from Felix Syn, Malaika Wasim, Jasper Lin, Josh Jacob, Jinny Chatananchai, and Neo Perera, with Sleap-anipose utilisation by Felix Syn. MATLAB processing and PCA analysis were conducted by Malaika Wasim. Motion capture setup modified by Jasper Lin. Collection of both sEMG signals and motion capture data was performed by Felix Syn, Malaika Wasim, Jasper Lin, Mahish Thevarajah, and Josh Jacob, with subsequent data analysis on the sEMG side carried out by Malaika Wasim, and the neural network analysis by Felix Syn.

The report was initially drafted by Jasper Lin, Felix Syn, Malaika Wasim, Mahish Thevarajah, and Josh Jacob, with figures and tables prepared by Jasper Lin, Malaika Wasim, and Felix Syn. Revisions, editing, and critical review of the report were undertaken by Jasper Lin, Malaika Wasim, Felix Syn, Josh Jacob, Mahish Thevarajah.

7. References

[1] Cauraugh J, Light K, Kim S, Thigpen M, Behrman A. Chronic Motor Dysfunction After Stroke. Stroke. [Internet]. PubMed. 2000 [cited 2025 Apr 14]. Available from: https://pubmed.ncbi.nlm.nih.gov/10835457/

[2] Stroke Association. Developing new technologies for stroke treatment [Internet]. Stroke.org.uk. 2024 [cited 2025 Apr 14]. Available from: https://www.stroke.org.uk/research/developing-new-technologies-stroke-treatment

[3] Pelc C. How researchers are restoring movement in people with spinal cord injuries [Internet]. Medicalnewstoday.com. Medical News Today; 2022 [cited 2025 Apr 14]. Available from: https://www.medicalnewstoday.com/articles/how-researchers-are-restoring-movement-in-people-with-spinal-cord-injuries

[4] Battraw MA, Fitzgerald J, Winslow EJ, James MA, Bagley AM, Joiner WM, et al. Surface electromyography evaluation for decoding hand motor intent in children with congenital upper limb deficiency. Scientific Reports [Internet]. 2024 Dec 30 [cited 2025 Apr 14]. Available from: https://www.nature.com/articles/s41598-024-82519-z

[5] Coste CA, William L, Fonseca L, Hiairrassary A, Andreu D, Geffrier A, et al. Activating effective functional hand movements in individuals with complete tetraplegia through neural stimulation. Scientific Reports [Internet]. 2022 Oct 6 [cited 2025 Apr 14];12(1). Available from: https://www.nature.com/articles/s41598-022-19906-x

[6] De Luca CJ. The Use of Surface Electromyography in Biomechanics. Journal of Applied Biomechanics [Internet]. 1997 May [cited 2025 Apr 14];13(2):135–63. Available from: https://journals.humankinetics.com/view/journals/jab/13/2/article-p135.xml

[7] Sîmpetru RC, Souza D, Matthias Ponfick, Vecchio AD. Identification of Spared and Proportionally Controllable Hand Motor Dimensions in Motor Complete Spinal Cord Injuries Using Latent Manifold Analysis. IEEE Transactions on Neural Systems and Rehabilitation Engineering [Internet]. 2024 Jan 1 [cited 2025 Apr 14];1–1. Available from: https://ieeexplore.ieee.org/document/10703163

[8] Nath T, Mathis A, Chen AC, Patel A, Bethge M, Mathis MW. Using DeepLabCut for 3D markerless pose estimation across species and behaviors. Nature Protocols [Internet]. 2019 Jun 21 [cited 2025 Apr 14];14(7):2152–76. Available from: https://www.nature.com/articles/s41596-019-0176-0

[9] Pereira TD, Tabris N, Matsliah A, Turner DM, Li J, Ravindranath S, et al. SLEAP: A deep learning system for multi-animal pose tracking. Nature Methods [Internet]. 2022 Apr [cited 2025 Apr 14];19(4):486–95. Available from: https://www.nature.com/articles/s41592-022-01426-1

[10] Wu C, Farokh AS, Ghassemi MM, Alhanai T. An LSTM Feature Imitation Network for Hand Movement Recognition from sEMG Signals [Internet]. arXiv.org. 2024 [cited 2025 Apr 14]. Available from: https://arxiv.org/abs/2405.19356

[11] HDsEMG - OT Bioelettronica [Internet]. OT Bioelettronica. 2025 [cited 2025 Apr 14]. Available from: https://otbioelettronica.it/en/

[12] Zhang Z. A flexible new technique for camera calibration. IEEE Transactions on Pattern Analysis and Machine Intelligence [Internet]. 2000 Jan 1 [cited 2025 Apr 14];22(11):1330–4. Available from: https://ieeexplore.ieee.org/document/888718

[13] McManus L, Vito GD, Lowery MM. Analysis and Biophysics of Surface EMG for Physiotherapists and Kinesiologists: Toward a Common Language with Rehabilitation Engineers. Frontiers in Neurology [Internet]. 2020 Oct 15 [cited 2025 Apr 14];11. Available from: https://www.frontiersin.org/journals/neurology/articles/10.3389/fneur.2020.576729/full

[14] How do I use the Ten20 paste correctly? [Internet]. Pluxbiosignals.com. 2022 [cited 2025 Apr 14]. Available from: https://support.pluxbiosignals.com/knowledge-base/how-do-i-use-the-ten20-paste-correctly/

[15] Lung BE, Siwiec RM. Anatomy, Shoulder and Upper Limb, Forearm Flexor Carpi Ulnaris Muscle [Internet]. Nih.gov. StatPearls Publishing; 2024 [cited 2025 Apr 14]. Available from: https://www.ncbi.nlm.nih.gov/books/NBK526051/

[16] Quattrocento – User Manual [Internet]. OT Bioelettronica. 2024 [cited 2025 Apr 14]. Available from: https://otbioelettronica.it/wpfd_file/quattrocento-user-manual/

[17] Boyer M, Bouyer L, Roy JS, Alexandre Campeau-Lecours. Reducing Noise, Artifacts and Interference in Single-Channel EMG Signals: A Review. Sensors [Internet]. 2023 Mar 8 [cited 2025 Apr 14];23(6):2927–7. Available from: https://www.mdpi.com/1424-8220/23/6/2927

[18] Naik GR, Suviseshamuthu Easter Selvan, Gobbo M, Amit Acharyya, Nguyen HT. Principal Component Analysis Applied to Surface Electromyography: A Comprehensive Review. IEEE Access [Internet]. 2016 Jan 1 [cited 2025 Apr 14]; 4:4025–37. Available from: https://ieeexplore.ieee.org/document/7516567

[19] Maggioni V, Azevedo-Coste C, Durand S, Bailly F. Optimisation and Comparison of Markerless and Marker-Based Motion Capture Methods for Hand and Finger Movement Analysis. Sensors [Internet]. 2025 Feb 11 [cited 2025 Apr 14];25(4):1079. Available from: https://www.mdpi.com/1424-8220/25/4/1079

[20] Cohen EJ, Bravi R, Minciacchi D. 3D reconstruction of human movement in a single projection by dynamic marker scaling. Bensmaia SJ, editor. PLOS ONE [Internet]. 2017 Oct 18 [cited 2025 Apr 14];12(10): e0186443. Available from: https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0186443

[21] Mathis A, Mamidanna P, Cury KM, Abe T, Murthy VN, Mathis MW, et al. DeepLabCut: markerless pose estimation of user-defined body parts with deep learning. Nature Neuroscience [Internet]. 2018 Aug 20 [cited 2025 Apr 14];21(9):1281–9. Available from: https://www.nature.com/articles/s41593-018-0209-y

[22] Nordberg JJ, Sluder G. Practical Aspects of Adjusting Digital Cameras. Methods in cell biology [Internet]. 2013 Jan 1 [cited 2025 Apr 14]. Available from: https://www.sciencedirect.com/science/article/pii/B9780124077614000075?via%3Dihub

[23] Vila LG. Population dynamics of motor neurons across different behaviours. London, England: Imperial College London; 2024 p. 1–18.

[24] JARVIS Mocap - Documentation [Internet]. Github.io. 2022 [cited 2025 Apr 14]. Available from: https://jarvis-mocap.github.io/jarvis-docs/

[25] Karashchuk P, Rupp KL, Dickinson ES, Walling-Bell S, Sanders E, Azim E, et al. Anipose: A toolkit for robust markerless 3D pose estimation. Cell Reports [Internet]. 2021 Sep [cited 2025 Apr 14]. Available from: https://www.cell.com/cell-reports/fulltext/S2211-1247(21)01179-7?_returnURL=https%3A%2F%2Flinkinghub.elsevier.com%2Fretrieve%2Fpii%2FS2211124721011797%3Fshowall%3Dtrue

[26] Oßwald M, Cakici AL, Oliveira DS de, Braun DI, Vecchio AD. Mechanical Hand Synergies during Dynamic Hand Movements are Mostly Controlled in a Non-Synergistic Way by Spinal Motor Neurons. 2023 Jul 26 [cited 2025 Apr 14]; Available from: https://www.biorxiv.org/content/10.1101/2023.07.25.550369v1

[27] Shrivastava R. A hidden Markov model based dynamic hand gesture recognition system using OpenCV. 2013 Feb 1 [cited 2025 Apr 14];947–50. Available from: https://ieeexplore.ieee.org/document/6514354

[28] Bai D, Liu T, Han X, Yi H. Application Research on Optimization Algorithm of sEMG Gesture Recognition Based on Light CNN+LSTM Model. Cyborg and Bionic Systems [Internet]. 2021 Jan [cited 2025 Apr 14];2021. Available from: https://spj.science.org/doi/10.34133/2021/9794610

[29] Swati Shilaskar, Shripad Bhatlawande, Ranveer Chavare, Aditya Ingale, Joshi R, Aditya Vaishale. Human Hand Movement Classification based on EMG Signal using different Feature Extractor. Biomedical and Pharmacology Journal [Internet]. 2024 Mar 20 [cited 2025 Apr 14];17(1):71–82. Available from: https://biomedpharmajournal.org/vol17no1/human-hand-movement-classification-based-on-emg-signal-using-different-feature-extractor/

[30] Youtube.com. 2025 [cited 2025 Apr 14]. Available from: https://www.youtube.com/watch?v=YCzL96nL7j0

[31] Garcia-Vellisca MA, A. Matran-Fernandez, Poli R, L. Citi. Hand-movement Prediction from EMG with LSTM-Recurrent Neural Networks. Open Access at Essex (University of Essex) [Internet]. 2021 Mar 15 [cited 2025 Apr 14]; 6:1–5. Available from: https://ieeexplore.ieee.org/document/9434840

[32] Islam M, Chen G, Jin S. An Overview of Neural Network. American Journal of Neural Networks and Applications [Internet]. 2019 Jan 1 [cited 2025 Apr 14];5(1):7–7. Available from: https://www.sciencepublishinggroup.com/article/10.11648/j.ajnna.20190501.12

[33] Olsson AE, Sager P, Andersson E, Björkman A, Malešević N, Antfolk C. Extraction of Multi-Labelled Movement Information from the Raw HD-sEMG Image with Time-Domain Depth. Scientific Reports [Internet]. 2019 May 10 [cited 2025 Apr 14];9(1). Available from: https://www.nature.com/articles/s41598-019-43676-8

[34] PyTorch (2019). PyTorch documentation — PyTorch master documentation. [online] Pytorch.org. Available at: https://pytorch.org/docs/stable/index.html.

[35] Goodfellow I. Deep Learning [Internet]. Deeplearningbook.org. 2016 [cited 2025 Apr 14]. Available from: https://www.deeplearningbook.org/

[36] Farina. Surface EMG crosstalk evaluated from experimental recordings and simulated signals. Reflections on crosstalk interpretation, quantification and reduction. Methods of information in medicine [Internet]. 2023 [cited 2025 Apr 14]. Available from: https://pubmed.ncbi.nlm.nih.gov/15026832/

[37] Péter, A., Andersson, E., Hegyi, A., Finni, T., Tarassova, O., Cronin, N., Grundström, H. and Arndt, A. (2019). Comparing Surface and Fine-Wire Electromyography Activity of Lower Leg Muscles at Different Walking Speeds. *Frontiers in Physiology*, [online] 10, p.1283. doi: https://doi.org/10.3389/fphys.2019.01283

[38] Martinez-Valdes E, Enoka RM, Aleš Holobar, McGill K, Farina D, Besomi M, et al. Consensus for experimental design in electromyography (CEDE) project: Single motor unit matrix. Journal of Electromyography and Kinesiology [Internet]. 2022 Nov 28 [cited 2025 Apr 14]; 68:102726–6. Available from: https://www.sciencedirect.com/science/article/pii/S1050641122000992

[39] Wang J, Cao D, Li Y, Wang J, Wu Y. Multi-user motion recognition using sEMG via discriminative canonical correlation analysis and adaptive dimensionality reduction. Frontiers in Neurorobotics [Internet]. 2022 Oct 28 [cited 2025 Apr 14];16. Available from: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2022.997134/full

[40] Andreassend, M. (2023). Sign language alphabets from around the world - ai-media. Retrieved from https://www.ai-media.tv/knowledge-hub/insights/sign-language-alphabets/.

[41] sleap-anipose 0.1.8. (2024) https://pypi.org/project/sleap-anipose/ [Accessed 18th Dec 2024].

8. Appendix

Appendix A

Hand Node Labelled Using SLEAP

F1 – Thumb

F2 – Index Finger

F3 – Middle Finger

F4 – Ring Finger

F5 – Fifth Digit

FX (0) - Metacarpophalangeal joints.

X = 1, 2, 3, 4, 5

Fx (1) - Proximal interphalangeal joints.

x = 2, 3, 4, 5

F1(1) - Interphalangeal joint.

Fx (2) - Distal interphalangeal joints.

F1(2) - Tip of thumb.

Fx (3) - Finger Tips.

W - Wrist

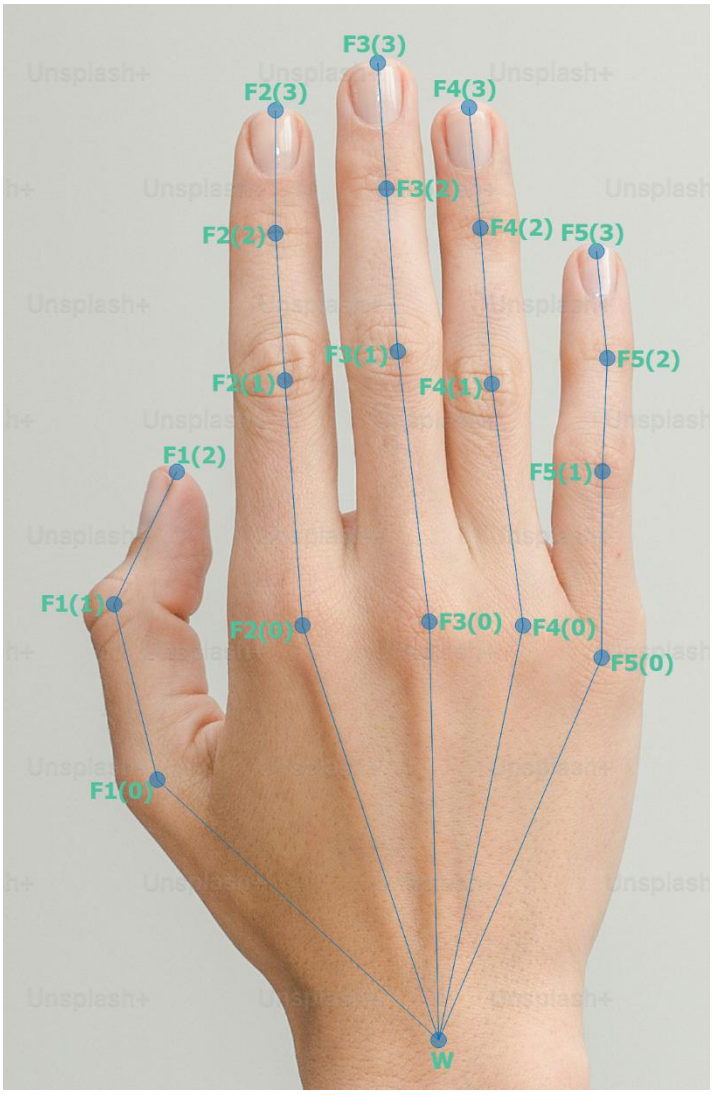


Figure A: Image of hand with labelled nodes using SLEAP

Five principal labels represent the five fingers: F1 corresponds to the thumb, F2 to the index finger, F3 to the middle finger, F4 to the ring finger, and F5 to the Fifth Digit. For each finger, nodes are further categorized by joint regions. The nodes labelled Fx (0), where x = 1, 2, 3, 4, or 5, indicate the metacarpophalangeal joints. For fingers F2 through F5, Fx (1) designate the proximal interphalangeal joints, while F1(1) specifically denotes the interphalangeal joint of the thumb. The nodes marked as Fx (2) correspond to the distal interphalangeal joints, with F1(2) representing the tip of the thumb. Finally, nodes labelled Fx (3) indicate the fingertip for fingers F2 through F5. This structured layout enables precise tracking of hand motion and is critical for accurate 3D reconstruction and neural decoding in our project.

Appendix B

Calibration & Triangulation with sleap-anipose

CamCal_260225.py > ...

```python
import sleap_anipose as slap

slap.draw_board(board_name = 'MyCharUco.jpg',
                board_x = 7,
                board_y = 10,
                square_length = 17,
                marker_length = 12,
                marker_bits = 4,
                dict_size = 50,
                img_width = 200,
                img_height = 150,
                save = 'drawboardFelix260325.toml')

# Calibration with Recordings of ChArUco board

cgroup, metadata = slap.calibrate(session = "Session1",
                                  board = "drawboardFelix260325.toml",
                                  excluded_views = ('C3', 'C7'),
                                  calib_fname = "Session1/CalResult1.toml",
                                  metadata_fname = "Session1/calibration_metadata.h5",
                                  histogram_path = "Session1/reprojection_histogram.png",
                                  reproj_path = "Session1")

# Triangulation with the 2D points exported from SLEAP to generate 3D point

points3d = slap.triangulate(p2d = 'Session1',
                            calib = 'Session1/Cal2Result258_260325.toml',
                            frames = (0, -1),
                            excluded_views = ('C3', 'C7'),
                            fname = 'Session1/points3dTry1_300lab.h5',
                            constraints = [[0,1], [0,4], [0,8], [0,12], [0,16], [1,2], [2,3], [4,5], [5,6], [6,7], [8,9], [9,10], [10,11], [12,13], [13,1
                            # constraints_weak = [[0,1], [0,4], [0,8], [0,12], [0,16], [1,2], [2,3], [4,5], [5,6], [6,7], [8,9], [9,10], [10,11], [12,13]
                            scale_smooth = 4,
                            scale_length = 2,
                            scale_length_weak = 0.5,
                            reproj_error_threshold = 10,
                            reproj_loss = 'soft_l1',
                            n_deriv_smooth = 2)
```

Figure B: Image of code used to calibrate the motion capture cameras using the Sleap-anipose library

This Appendix presents code utilising the Sleap-anipose library [41] to calibrate our cameras in the camera setup and triangulate 3D points from the 2D labelled data exported from SLEAP analysis.

It starts with creating a reference image of our ChArUco board (10x7) with the ‘draw_board’ method. This, along with the calibration recordings of the ChArUco board in motion within the cameras’ views, was used to calibrate the cameras with the ‘calibrate’ method. The output of ‘calibrate’ was a toml file, that contains the intrinsic (camera matrix) and extrinsic parameters (rotation matrix & translational vector) of each camera, which is used to calculate the reprojection matrix that would translate 2D points to 3D points. These 3D points were generated by the ‘triangulate’ method.

Appendix C

MsTimer2 Library Arduino Code for Trigger Synchronisation

```
#include <MsTimer2.h>

#define the trigger pins
const int triggerPin = 2;
const int cameraSyncPin = 7;
boolean flag = LOW; #low = OFF
boolean output = LOW;

void setup() { #initialise arduino
  // put your setup code here, to run once:
  #det ON/OFF camerasync pin at 20 ms intervals
  MsTimer2::set(20, toggle);
  MsTimer2::start();
  pinMode(triggerPin, OUTPUT);
  pinMode(cameraSyncPin, OUTPUT);
  Serial.begin(9600);
}

void loop() {
  // put your main code here, to run repeatedly:
  char command;
  if (Serial.available()) {
    command = Serial.read();
    if (command == 'b') {
      flag = HIGH;#define 'b' to ON trigger
      digitalWrite(triggerPin, HIGH);
      Serial.println("Trigger on");
    }
    if (command == 's') {#define 's' to OFF trigger
      flag = LOW;
      digitalWrite(triggerPin, LOW);
      Serial.println("Trigger off");
    }
  }
}

void toggle() {#at 20ms intervals, when HIGH, toggle
  #between HIGH and LOW
  if (flag == 1) {
    digitalWrite(cameraSyncPin, output);
    output = !output;
  } else {
    digitalWrite(cameraSyncPin, LOW);
  }
}
```

Figure C: Image of Arduino Code for synchronising Arduino code and sEMG data acquisition

This appendix presents the Arduino code used for generating a synchronisation trigger for our motion capture and sEMG data acquisition systems. The code employs the MsTimer2 library, which allows for the execution of a specific function at fixed intervals. In this case, every 20 milliseconds – to produce a periodic square wave signal.

Two digital output pins are defined: one, labelled as triggerPin, controls the overall trigger state for initiating data collection, while the second, cameraSyncPin, outputs the periodic synchronisation signal required by the camera system. During initialisation, the setup routine configures these pins, starts the timer, and initialises serial communication for external command control.

Within the main loop, the code listens for serial commands; upon receiving the character ‘b’, the trigger is activated, this sets the flag high and turns on triggerPin, whereas receiving ‘s’ deactivates the trigger by setting the flag low. The toggle function, invoked by the MsTimer2 timer, checks the flag and, if it is active, alternates the state of cameraSyncPin to create a square wave.

Appendix D

American Sign Language (ASL) Alphabet

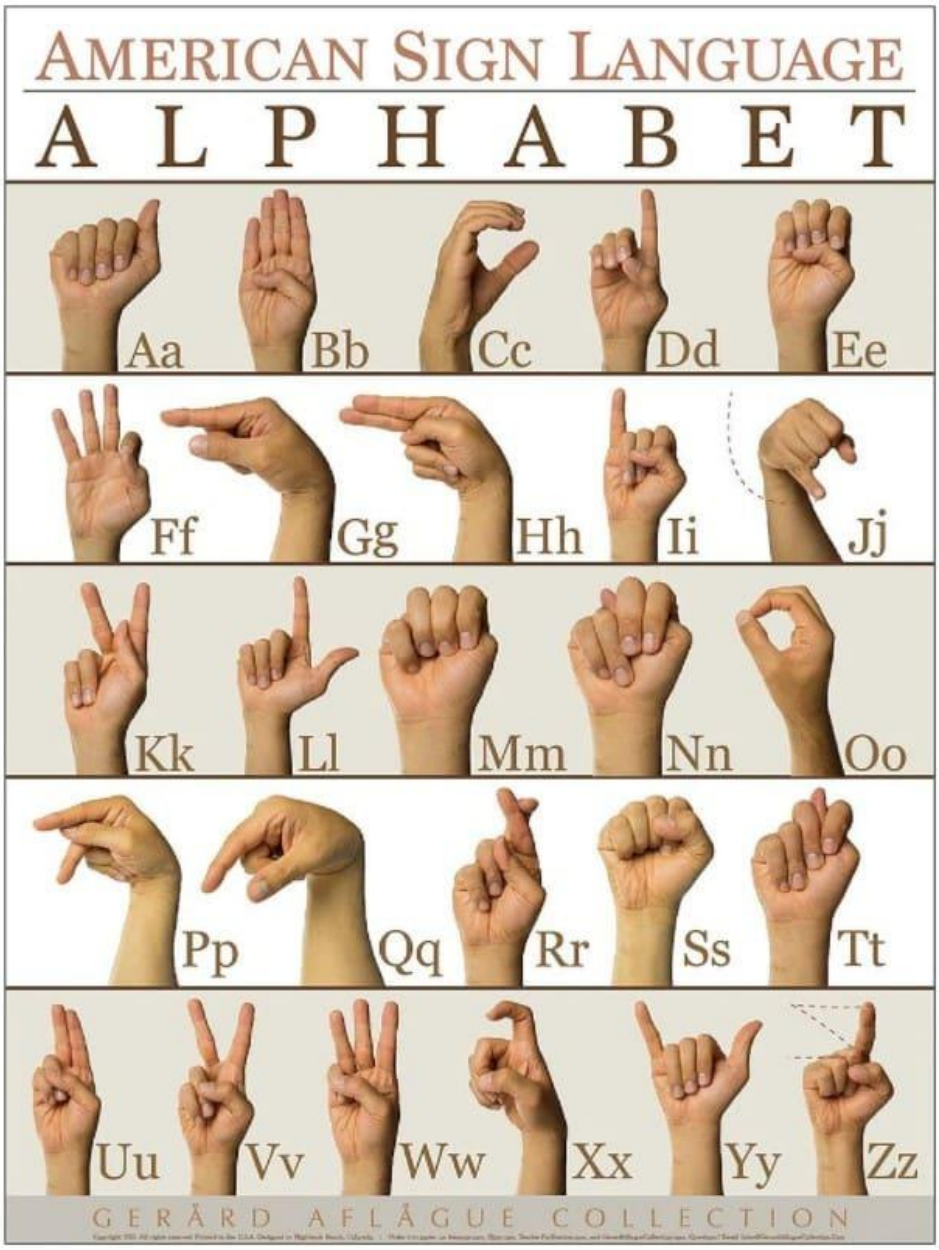


Figure D: Image of ASL Alphabet [40].

Appendix E - Ethics

Ethics Statement

The work under this project is covered under the submission of Dr Dario Farina under the title of "Investigating the input-output relationship of motor units in humans."

Project Specific Ethics Based Risk Assessment

Ethical Risk Type	Specific Risk	Impact	Mitigation
Declaration of Helsinki Violation	Not following ethical guidelines during medical research	Harm to participants	Storage of participant data on encrypted, personal accounts
Data Storage and Protection	Patient data leaks	Stigmatisation of participants	Data storage in password protected cloud storage
Participant Fatigue	Injury to arm, hand or shoulder	Short term inability to perform activities of daily living	Ample rest time between recordings

Table E: Ethical risk assessment table for Virtual-Hand model project. Outlining the potential risks, their respective impacts to the study and their corresponding mitigating strategies.

Appendix F

Appendix F-1: Loading MATLAB file

```
clc;
clear;

% loads the file
dt = load('C:\Users\malai\Desktop\sEMG\sEMG_Signal_Analysis\
JoshRH_FCU_E_130325.mat'); %CHANGE LETTER HERE
dats = dt.Data{1}'; %transposing 64 channels
t = dt.Time{1};
dsp = dt.Description{1};
datdims = size(dats);
fs = 2048;  %defining the sampling frequency

% puts data into array
L = datdims(2);
dominant_freqs = zeros(datdims(1), 1);
filty = zeros(size(dats));
```

Figure F-1: MATLAB code to load the OTBio+ MATLAB file and arrange each sEMG signal to a corresponding time vector

Appendix F-2: Code for Signal Filtering

```
    % Fourier transform of dom. frequency
    N = length(g);
    f = (0:N-1) * (fs/N);
    fft_signal = abs(fft(g)/N);
    half_N = floor(N / 2);
    [~, max_idx] = max(fft_signal(1:half_N));

    dominant_freqs(i) = f(max_idx);

    % low pass filter
    fc = max(10, dominant_freqs(i) + 5);
    wc = 2 * fc / fs * pi;
    filter_len = 2 * floor(L/2) + 1;
    t = -floor(L/2):floor(L/2);
    h = sin(wc * t) ./ (pi * t);
    h(floor(filter_len/2)+1) = wc / pi;
    h = h .* hamming(filter_len)';
    h = h / sum(h);
    filty_temp = conv(g(:), h, 'same');

    %Notch filter
    f0 = 50;
    bw = 0.05;
    w0 = f0 / (fs / 2);
    b_notch = [1, -2*cos(2*pi*w0), 1];
    a_notch = [1, -2*(1-bw)*cos(2*pi*w0), (1-bw)^2];
    filty(i, :) = filter(b_notch, a_notch, filty_temp);
```

Figure F-2: Image of code for sEMG signal filtering. By looping through the channels, the Fourier transform is conducted to determine the dominant frequency. An adaptive low-pass filter is then applied at dominant frequency + 5Hz. A Notch filter is then applied at 50Hz to remove common interference (power line).

By loading the OTBio+ MATLAB file and initialising the data, an adaptive low-pass filter can be applied to the signal at the dominant frequency + 5Hz. This smoothens the signal by removing high frequency noise, without over filtering. The notch filter targets the removal of power line interference at 50Hz.

Appendix G

Appendix G-1: PCA analysis of processed sEMG

```python
#initilise script / enter to bash
import numpy as np
import scipy.io
import joblib
import os
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

# load processed sEMG from MATLAB
file_name = r"C:\Users\malai\Desktop\sEMG\sEMG_Signal_Analysis\JoshRH_FCU_E_130325_filty.mat"
matty_data = scipy.io.loadmat(file_name)
semg_array = matty_data['filty']  # shape: (64, N)

#Transpose to (samples x channels)
semg_array = semg_array.T  # shape: (N, 64)

#Normalise
scaler = StandardScaler()
semg_normalised = scaler.fit_transform(semg_array)

#PCA (at 95% variance)
pca = PCA(n_components=0.95)
semg_pca = pca.fit_transform(semg_normalised)

print(f"PCA reduced from {semg_array.shape[1]} to {semg_pca.shape[1]} components.")
```

Figure G-1: Image of python script that conducts PCA analysis of processes EMG signal. Python script loads the processed signal from Appendix A*, normalising the data. This is followed by a PCA at 95% significance.*

Appendix G-2: - Appending PCA component analysis to sEMG CSV file

```python
for chunk_idx, chunk in enumerate(reader):
    if chunk.shape[1] != 65:
        raise ValueError(f"Chunk {chunk_idx} has {chunk.shape[1]} columns, expected 65.")

    chunk.columns = ['Time'] + [f'Ch{i}' for i in range(1, 65)]
    semg = chunk.iloc[:, 1:].values
    norm = scaler.transform(semg)
    pca_vals = pca.transform(norm)

    # Add available PCA columns
    for i in range(pca_vals.shape[1]):
        chunk[f'PCA{i+1}'] = pca_vals[:, i]

    # Write to output
    chunk.to_csv(output_csv, mode='a', index=False, header=not header_written)
    header_written = True
```

Figure G-2: Image of python script that appends PCA component analysis to sEMG CSV file. Applies the PCA model to the relevant OTBiolab+ file. By processing the data in chunks, the sEMG signals are transformed to the resulting principal components.

Appendix G-3: - Visualisation of PCA % variance

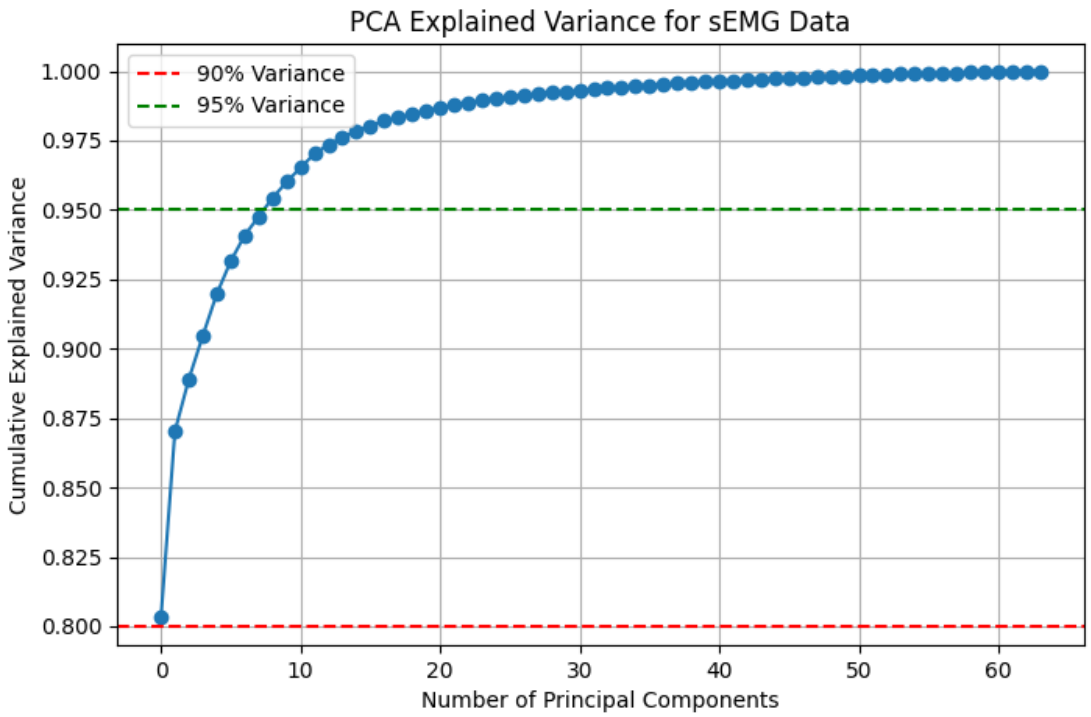


Figure G-3: Visualisation of PCA analysis of test data. Plotting the number of principle components and the associated variance.

Appendix H - FLIR CM3-U3-31S4M-CS cameras

Figure H: Image of the FLIR CM3-U3-31S4M-CS cameras used for the motion capture of hand kinematics

The FLIR CM3-U3-31S4M-CS cameras used in our project are sophisticated imaging devices engineered for high-precision motion capture. These cameras are designed as monochrome sensors, offering the flexibility to operate in colour if required, and are equipped with a global shutter that eliminates motion artifacts by capturing the entire frame simultaneously. With a resolution of 1280 × 1024 pixels, the sensors provide high-definition imaging necessary for capturing fine details in dynamic scenes, which is critical for accurate 3D reconstruction in our markerless motion capture system.

Key to their performance is the global shutter capability, which ensures that rapid movements are captured without the distortions typically associated with rolling shutter systems. This feature is particularly important in our application, where even minor frame-to-frame discrepancies can significantly impact the accuracy of 3D hand kinematics. In our setup, these cameras were mounted in a cylindrical structure measuring 91 cm in height and 114 cm in diameter.

Additional features of the FLIR CM3-U3-31S4M-CS include robust image processing capabilities and excellent sensitivity under various lighting conditions, making them well-suited for controlled laboratory environments. The integration of these cameras into our system, along with precise external triggering, underpins the reliability and accuracy of the motion capture data, forming a critical component of our integrated neural interface framework.

Appendix I - ChArUco Calibration Board

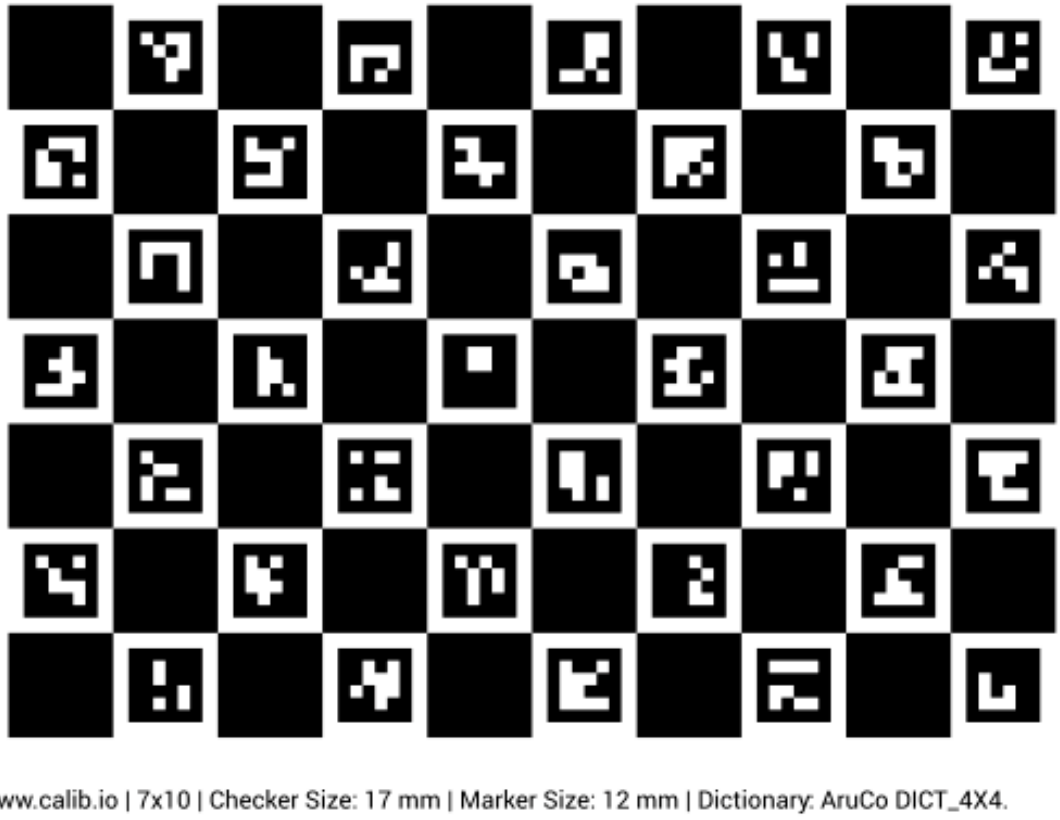


Figure I: Image of ChArUco board used to calibrate the motion capture cameras

In our study, we employed a ChArUco calibration board to perform camera calibration. A ChArUco board is a hybrid calibration target that combines a traditional checkerboard pattern with ArUco markers embedded within the white squares. This design leverages the precise corner detection of checkerboards and the robust identification capabilities of ArUco markers, thus allowing accurate camera calibration to be carried out even when the board is partially occluded or viewed at extreme angles.

The specific ChArUco board used in our project featured a 7 x 10 grid layout, with each square measuring 15 mm, this resulted in an overall board size of 105 x 150 mm. The ArUco markers were generated using a 4x4 dictionary, this ensured unique identification for each marker. The ChArUco board was mounted on a rigid cardboard spine to prevent distortions. During calibration, the board was positioned at various angles and distances within the camera's field of view to capture a diverse set of images. This enabled comprehensive calibration data collection.

Appendix J - 3D Point Visualisation with matplotlib.pyplot

3dpointplotting.py > ...

```python
import h5py
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation

# Load the 3D hand data from the HDF5 file.
filename = "Session1/points3dTry1_300lab.h5"  # update with your actual filename
with h5py.File(filename, 'r') as f:
    # 'tracks' is expected to have shape ([frames], 1, 20, 3)
    tracks = f['tracks'][:]
# Remove the singleton instance dimension to get shape ([frames], 20, 3)
tracks = np.squeeze(tracks, axis=1)
print(np.shape(tracks))
n_frames, n_nodes, _ = tracks.shape
print(n_frames)
print(n_nodes)

# Define the connectivity (edges) based on the suggested node order:

edges = [
    (0, 1), (1,2), (2,3),
    (0, 4), (4,5), (5,6), (6,7),
    (0, 8), (8,9), (9,10), (10,11),
    (0, 12), (12, 13), (13, 14), (14,15),
    (0, 16), (16,17), (17, 18), (18,19)
]

# Create the 3D plot.
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Plot the initial scatter of points.
points_initial = tracks[0]  # shape: (20, 3)
# node_colour = ['blue'] + ['green'] + ['yellow'] +['black'] + ['green'] + ['yellow'] +['red']
node_colour = ['purple'] + ['red']*3 + ['orange']*4 + ['yellow']*4 + ['green']*4 + ['blue']*4
scat = ax.scatter(points_initial[:, 0], points_initial[:, 1], points_initial[:, 2],
                  c=node_colour, marker='o', s=50)

# Plot initial lines for each edge and store the line objects.
line_objs = []
for i, j in edges:
    p1 = points_initial[i]
    p2 = points_initial[j]
    # Draw a line connecting p1 and p2.
    line, = ax.plot([p1[0], p2[0]], [p1[1], p2[1]], [p1[2], p2[2]], 'b-', lw=1)
    line_objs.append(line)

def update(frame):
    points = tracks[frame]  # shape: (20, 3)
```

```python
def update(frame):
    points = tracks[frame]  # shape: (20, 3)
    # Update scatter plot.
    scat._offsets3d = (points[:, 0], points[:, 1], points[:, 2])
    # Update each connecting line.
    for idx, (i, j) in enumerate(edges):
        p1 = points[i]
        p2 = points[j]
        line_objs[idx].set_data([p1[0], p2[0]], [p1[1], p2[1]])
        line_objs[idx].set_3d_properties([p1[2], p2[2]])
    ax.set_title(f"Frame {frame+1} / {n_frames}")
    return scat, *line_objs

# Create the animation.
ani = FuncAnimation(fig, update, frames=n_frames, interval=10, blit=False)
# ani.save('3DplotTry2.mp4', writer='ffmpeg')
plt.show()
```

Figure J: Image of code to visualise the triangulated 3D points across each frame using matplotlib.pyplot. Sample visualisation found in this GitHub link.

Appendix K - Architecture & Hyperparameters for LSTM models.

Model A

Architecture	
Input nodes	64
Hidden layers	2 (128 nodes each)
Output nodes	60
Hyperparameters	
Optimiser	AdamW (weight decay = 1e-4)
Dropout probability	0.5
Sequence Length	39
Batch Size	10000
Learning Rate	0.0002
Number of Epochs	300

Table K-1: Architecture and Hyperparameter values for Model A taking in raw 64 channel sEMG signals

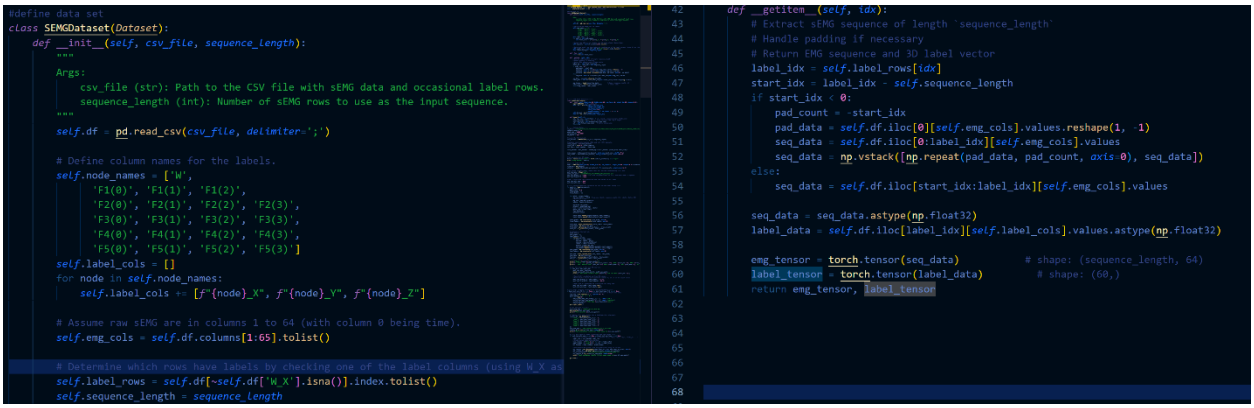
Model B

Architecture	
Input nodes	3
Hidden layers	2 (64 nodes each)
Output nodes	60
Hyperparameters	
Optimiser	AdamW (weight decay = 1e-4)
Dropout probability	0.5
Sequence Length	39
Batch Size	10000
Learning Rate	0.0002
Number of Epochs	300

Table K-2: Architecture and Hyperparameter values for Model B taking in 3 PCA components

Appendix L - LSTM network training and testing code

Appendix L-1: Dataset class



```
#define data set
class SEMGDataset(Dataset):
    def __init__(self, csv_file, sequence_length):
        """
        Args:
            csv_file (str): Path to the CSV file with sEMG data and occasional label rows.
            sequence_length (int): Number of sEMG rows to use as the input sequence.
        """
        self.df = pd.read_csv(csv_file, delimiter=';')

        # Define column names for the labels.
        self.node_names = ["W",
                'F1(0)', 'F1(1)', 'F1(2)',
                'F2(0)', 'F2(1)', 'F2(2)', 'F2(3)',
                'F3(0)', 'F3(1)', 'F3(2)', 'F3(3)',
                'F4(0)', 'F4(1)', 'F4(2)', 'F4(3)',
                'F5(0)', 'F5(1)', 'F5(2)', 'F5(3)']
        self.label_cols = []
        for node in self.node_names:
            self.label_cols += [f"{node}_X", f"{node}_Y", f"{node}_Z"]

        # Assume raw sEMG are in columns 1 to 64 (with column 0 being time).
        self.emg_cols = self.df.columns[1:65].tolist()

        # Determine which rows have labels by checking one of the label columns (using W_X as
        self.label_rows = self.df[~self.df['W_X'].isna()].index.tolist()
        self.sequence_length = sequence_length
```

```
    def __getitem__(self, idx):
        # Extract sEMG sequence of length `sequence_length`
        # Handle padding if necessary
        # Return EMG sequence and 3D label vector
        label_idx = self.label_rows[idx]
        start_idx = label_idx - self.sequence_length
        if start_idx < 0:
            pad_count = -start_idx
            pad_data = self.df.iloc[0][self.emg_cols].values.reshape(1, -1)
            seq_data = self.df.iloc[0:label_idx][self.emg_cols].values
            seq_data = np.vstack([np.repeat(pad_data, pad_count, axis=0), seq_data])
        else:
            seq_data = self.df.iloc[start_idx:label_idx][self.emg_cols].values

        seq_data = seq_data.astype(np.float32)
        label_data = self.df.iloc[label_idx][self.label_cols].values.astype(np.float32)

        emg_tensor = torch.tensor(seq_data)           # shape: (sequence_length, 64)
        label_tensor = torch.tensor(label_data)       # shape: (60,)
        return emg_tensor, label_tensor
```

Figure L-1: Code to load data from a CSV file of sEMG data and returns sEMG sequences paired with their corresponding 3D joint label vectors.

Appendix L.2: Model architecture class

```
class sEMGLSTM(nn.Module):
    def __init__(self, input_dim=64, hidden_dim=128, num_layers=2, output_dim=60, dropout=0.5):
        super(sEMGLSTM, self).__init__()
        self.lstm = nn.LSTM(input_size=input_dim,
                            hidden_size=hidden_dim,
                            num_layers=num_layers,
                            batch_first=True,
                            dropout=dropout if num_layers > 1 else 0)
        self.dropout = nn.Dropout(p=dropout)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, x):
        out, (hn, cn) = self.lstm(x)  # out: (batch, sequence_length, hidden_dim)
        last_time_step = out[:, -1, :]  # Use the last time step's output
        last_time_step = self.dropout(last_time_step)
        out = self.fc(last_time_step)   # (batch, output_dim)
        return out
```

Figure L-2: Code that defines an LSTM-based neural network for mapping sEMG sequences to joint position predictions.

Appendix L-3: Training loop

```
for epoch in range(num_epochs):
    model.train()
    epoch_loss = 0.0
    train_preds = []
    train_labels = []

    for batch in train_loader:
        emg_seq, labels = batch  # emg_seq: (batch, sequence_length, 64), labels: (batch, 60)
        # Move data to GPU.
        emg_seq = emg_seq.to(device)
        labels = labels.to(device)

        optimizer.zero_grad()
        outputs = model(emg_seq)
        loss = criterion(outputs, labels)
        epoch_loss += loss.item()
        loss.backward()
        optimizer.step()

        train_preds.append(outputs.detach().cpu().numpy())
        train_labels.append(labels.detach().cpu().numpy())

    train_preds = np.concatenate(train_preds, axis=0)
    train_labels = np.concatenate(train_labels, axis=0)

    train_mse = mean_squared_error(train_labels, train_preds)
    train_rmse = np.sqrt(train_mse)
    train_mae = np.mean(np.abs(train_labels - train_preds))
    train_r2 = r2_score(train_labels, train_preds)
```

```
    #Evaluate on the test set.
    model.eval()
    test_preds = []
    test_labels = []
    with torch.no_grad():
        for batch in test_loader:
            emg_seq, labels = batch
            emg_seq = emg_seq.to(device)
            labels = labels.to(device)
            outputs = model(emg_seq)
            test_preds.append(outputs.detach().cpu().numpy())
            test_labels.append(labels.detach().cpu().numpy())
    test_preds = np.concatenate(test_preds, axis=0)
    test_labels = np.concatenate(test_labels, axis=0)

    test_mse = mean_squared_error(test_labels, test_preds)
    test_rmse = np.sqrt(test_mse)
    test_mae = np.mean(np.abs(test_labels - test_preds))
    test_r2 = r2_score(test_labels, test_preds)

    print(f"Epoch [{epoch+1}/{num_epochs}]")
    print(f"  Train Loss: {epoch_loss/len(train_loader):.4f} | MSE: {train_mse:.4f}, RMSE: {t
    print(f"  Test  Metrics: MSE: {test_mse:.4f}, RMSE: {test_rmse:.4f}, MAE: {test_mae:.4f},
```

Figure L-3: Trains the LSTM model, evaluates performance on test set each epoch, and saves the best-performing model.

Appendix L-4: Visualisation of LSTM model on F2(3) node in the test set

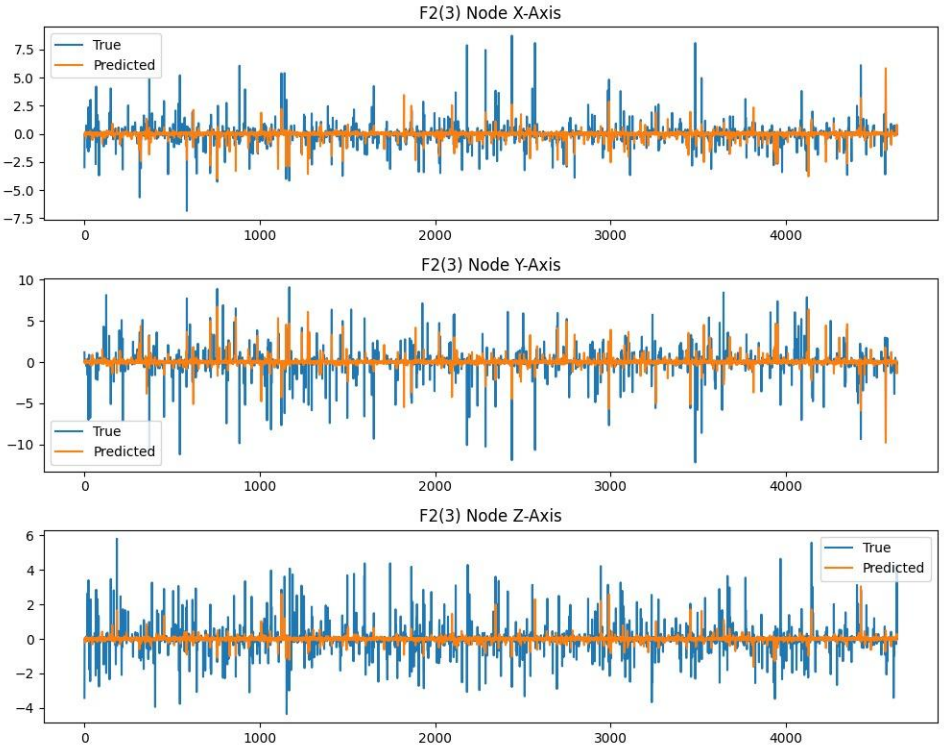


Figure L-4: Plot of prediction accuracy for one joint across the 3D axis.

This appendix shows code that shows how our LSTM network was built using PyTorch. In Appendix L-1, we created a SEMGDataset class that would create our training and test sets based on our imported labelled data (separate csv file). In Appendix M-2, the LSTM architecture was created by specifying the number of inputs, hidden and output layers as well as the associated hyperparameters, AdamW optimiser and dropout layers to improve generalisation. We then run

the training loop on our training set for 300 epochs, printing the statistical metrics (MSE, RMSE, MAE and R^2) every training epoch to evaluate model performance during training. We saved the best model with the lowest test MSE along with the predictions on the test set

Appendix M - Project Management Assessment

Departures and Deviations from Project Pitch

Several amendments were necessary during the project, which resulted in deviation from the initial strategy outlined in the Project Pitch. For example, although we intended to use eight FLIR cameras for 3D reconstruction, only five had functional cables and just three were used for triangulation, reducing spatial resolution and model accuracy. Furthermore, the original plan to use PCA-processed sEMG data was revised when raw data yielded better performance ($R^2 = 0.19$ vs. 0.04), despite increased noise and computational demands. We initially planned to train our LSTM models locally but migrated to Imperial's HPC cluster to handle high computational demands, adding logistical complexity. The data labelling process with SLEAP proved more time-consuming than expected; manual correction and model training delays reduced the number of labelled sequences available for training, limiting model generalisability. In MATLAB, PCA processing of the full dataset was hindered by memory constraints, forcing a chunk-based approach that may have introduced inconsistencies. Lastly, while we initially proposed F1 score as a performance metric, the task was reframed from classification to regression, requiring metrics such as MSE, RMSE, MAE and R^2. These changes required adaptability without compromising the project's core aim of mapping neural intent to 3D hand motion.

Project Management Lesson 1

Data labelling using SLEAP was the most labour-intensive aspect of the project, requiring manual annotation of over 12,000 frames to ensure sufficient training accuracy. To manage this demand, the task was distributed across team members. Internal deadlines were set to maintain a consistent pace and prevent bottlenecks, especially since this task coincided with other major coursework deadlines.

Due to the varying availability among members, tasks had to be reallocated. However, due to the flexible nature of our workload distribution, an uneven workload distribution grew evident, and some members took on more work than others.

This was inevitable given the differences team member's lifestyles. To mitigate this in future projects, project management tools such as Microsoft Planner can be implemented. Regular progress reviews should also be scheduled to ensure that tasks are evenly distributed and any member who is overburdened receives the necessary support or reallocation of work.

Project Management Lesson 2

There were many technical hurdles to overcome to get our system to work. For example, we had a lot of trouble at the start of the project, as were unable to connect the Quattrocento to our laptops. Having mustered up the courage to ask another person working in the laboratory for help, we learnt about the niche way to connect the Quattrocento. Additionally, we also received advice on how to troubleshoot devices (i.e. cameras and associated software) generally. We learnt that it is alright

to ask for help, especially when we have tried our best but have been unsuccessful. It was made clear that there is always a potential learning opportunity even when you are stuck.

Project Management Lesson 3

Throughout the project, meetings and discussions regarding progress were limited to in-person laboratory-based discussions. However, this meant that there was little consistency in who attended the meetings, resulting in discrepancies in project involvement, and a need to catch up with work. Looking forward, having compulsory weekly team meetings, would prevent team members from falling behind, and would force active involvement in the project. This would also allow for progress updates on work and encourage contribution towards each other's allocated tasks.