Motion Retargeting on a Human Hand Model

Project by:
Charlotte Ludovica Primiceri, Serena Trovalusci,
Diana Ioana Bubenek Turconi, Giordano Pagano
M.Sc. Students, AIRO, Sapienza University of Rome

Supervised by:
Prof. Marilena Vendittelli – Full Professor
Dr. Emanuele De Santis – Research Fellow
Department of Computer, Control, and Management Engineering (DIAG),
Sapienza University of Rome

June 13, 2025

1 Introduction

This project addresses the problem of real-time motion retargeting for a human hand model by leveraging a data-driven approach based on a neural network, as an alternative to classical direct or inverse kinematics techniques. The network is trained to predict the complete joint configuration of a virtual hand from a reduced set of input features. Specifically, the available inputs consist of the closure parameters of the thumb, index, and middle fingers, as well as the abduction parameter of the thumb. These signals are acquired in real time from the Weart TouchDIVER G1 haptic glove, which provides continuous hand motion data. The objective is to reconstruct a plausible and consistent full hand pose in a virtual environment using only these partial measurements.

The Weart TouchDIVER G1 is a lightweight, wearable haptic device that integrates lifelike force feedback, texture rendering, and thermal cues. It is equipped with precise sensors for tracking hand and finger movements, allowing for natural and intuitive interaction with virtual environments and objects [1].

The motion retargeting system was developed and tested within a Unity-based simulated environment, using the official Weart SDK and documentation available on the Weart website. This setup enabled the integration of real-time data acquisition, neural network inference, and visual feedback of the virtual hand motion [2].

We developed a complete pipeline consisting of:

- A custom C# logging tool that records synchronized joint rotation angles and sensor inputs, used to build the training dataset.

- Two machine learning models—a Fully Connected Neural Network (FCNN) and a Transformer-based model—trained to map a 4-dimensional input vector to 62 outputs representing processed joint angle information.
- A client-server architecture in which Unity sends real-time sensor data to a Python-based server, which returns the predicted joint angles to drive the animation of the virtual hand in real time.

Human hand movements exhibit synergies, meaning that many joints move in coordinated and highly correlated patterns rather than independently. These natural correlations between finger joints imply that it is not necessary to represent or predict every single joint angle to reconstruct realistic hand motions. Instead, a reduced set of parameters can sufficiently capture the essential variability of hand poses. This topic has been a subject of research in the study by Santello, Flanders and Soechting [3], and their approach will be applied in the present work.

To exploit this property and improve both generalization and efficiency, we apply dimensionality reduction techniques such as Principal Component Analysis (PCA) and Autoencoders. These methods allow us to represent hand configurations in a compact latent space, enabling the network to generate anatomically plausible hand motions from a limited set of input signals, with low latency.

2 Methodology

The goal of this work is to learn a mapping from glove-derived parameters to a complete 3D pose of a human hand. The input consists of four scalar parameters: three closure values (thumb, index, middle), and one abduction value for the thumb. The output targets are the rotational configurations of 15 joints (three per finger), each represented along three rotational axes (XYZ), for a total of 45 joint angles.

To address issues of discontinuity inherent in Euler angle representation, several rotational components were encoded using their sine and cosine values. This increased the output dimensionality to 62, allowing the network to represent circular quantities more robustly.

2.1 Data acquisition

Data collection was carried out in three phases. We initially recorded around 3,000 samples from the thumb, index, and middle fingers, which provided a limited yet sufficient starting point for early experiments. We started with three closure parameters. Successively, since thumb movements are more complex and have a wider range, we decided to consider a more accurate description by including the abduction parameter of the thumb. We followed with an expanded dataset of approximately 11,000 samples, still restricted to the same three fingers. Finally, we acquired a full dataset of 17,000 samples that included all five fingers.

Each data sample in the final dataset consists of:

- 45 joint angle values (3 rotational axes $\times$ 15 joints)
- 4 input parameters (ThumbClosure, IndexClosure, MiddleClosure, ThumbAbduction)
- 1 timestamp indicating when the sample was recorded

All angles were recorded in Euler format with respect to the palm's coordinate frame, and all signals were synchronized and stored in CSV format using a custom Unity script.

Early Post-Processing Attempts

In the first stages, we encountered unrealistic flipping and jumps in the predicted joint motions, especially in some specific joints:

- All joints of the thumb (Thumb01, Thumb02, Thumb03), across all three axes
- The Distal Interphalangeal (DIP) joint of the middle finger (Middle03), mostly around the Y-axis
- The Proximal Interphalangeal (PIP) and DIP joints of the index finger (Index02 and Index03), again around the Y and Z axes

We initially tried to manually clamp the rotation values, based on the finger closure range. For instance, if the closure of the index finger was between two chosen thresholds, we would clip or adjust the output rotation of Index03 around the problematic axis.

Although it partially worked, this method failed for the thumb, where closure and abduction moved simultaneously. It was very difficult to identify a clean interval to apply corrections. Even when thresholds were effective, the resulting motion appeared unnatural in the post-processed intervals.

Sine/Cosine Encoding

To address the discontinuity and ambiguity problems caused by the Euler angle wrapping (e.g. +180 ° = -180 °), we introduced sine and cosine encoding for problematic joint angles. Instead of predicting a raw angle θ, the model predicts $\sin(\theta)$ and $\cos(\theta)$, which ensures continuity (no sharp jumps at $\pm 180°$) and geometric uniqueness.

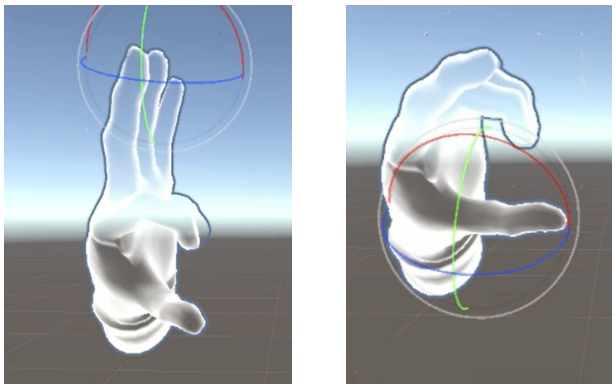
The decoding was performed using: $\theta = \arctan2(\sin(\theta), \cos(\theta))$

This technique was applied selectively to:

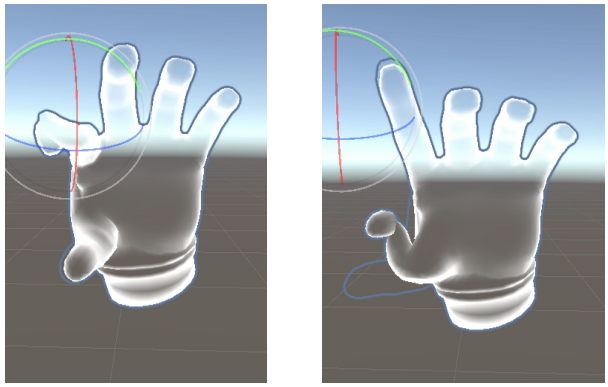
- All thumb joints
- Middle03

- Index02 and Index03

With this fix, the motion of the fingers became visually stable and biomechanically accurate.



(a) Sine/cosine encoding



(b) Without sine/cosine encoding

Figure 1: Finger pose prediction using different strategies.

Z-Score Filtering for Outlier Removal

We also applied z-score filtering to eliminate outliers from the dataset before training. The z-score of a value is defined as: $z = \frac{x-\mu}{\sigma}$ where:

- x is the sample value
- μ is the mean across the dataset
- σ is the standard deviation

We computed the z-score for each joint angle component, and excluded any sample where any value exceeded a z-score of ± 2.5. This threshold was chosen because:

- In a normal distribution, 98.8% of values fall within ± 2.5 standard deviations
- This filters out only the most extreme 1.2% of data, likely to be noisy samples or sensor errors

This step helped clean the dataset while preserving the valid data, ensuring that our models were not influenced by noisy samples.

Standardization

Joint angle values were standardized using a StandardScaler: after the application of sine and cosine encoding, all joint angle values were centered around the mean and scaled to unit variance. This preprocessing step ensures numerical stability and promotes faster convergence during model training.

2.2 Dimensionality Reduction

Building on the initial approach of using regression models to map the four input parameters to hand joint angles expressed via sine and cosine transformations, our focus shifted toward exploring how dimensionality reduction techniques could enhance both reduce complexity of the task and biomechanical realism in motion prediction.

We explored multiple strategies to incorporate postural synergies into the learning process:

1. `Direct PCA Output`

   In an initial approach, we trained both the FCNN and Transformer models to directly predict the principal components (PCA coefficients) of the preprocessed joint angle data. The models were trained to minimize the MSE loss between the predicted and ground-truth PCA components.

2. `PCA in the Loss`

   Also, the models were trained to predict the full dimensional representation of joint angles, but the loss was computed in the PCA space. Specifically, the MSE was applied between the PCA projection of the model's output and that of the ground truth. This formulation allowed us to retain the expressiveness of the original output space while encouraging the network to focus on capturing the most significant motion patterns.

3. `Latent-Space Constraints from Autoencoders`

   To explore nonlinear dimensionality reduction, we trained an autoencoder to learn a compact latent representation of the joint angle space. The

FCNN and Transformer models were then trained to predict the full output, while using a loss computed in the autoencoder's latent space, comparing the encoded output of the prediction to that of the ground truth. This approach allows for richer, learned synergies beyond linear PCA while maintaining compatibility with the existing regression models.

2.3 Models

Model 1: Fully Connected Neural Network

The architecture of this model consists of a series of four Linear Layers with the following dimensions:

```
Input (4) → 512 → 256 → 128 → 64 → Output (62)
```

Between each Linear Layer, there is a **LeakyReLU** activation function, followed by either a **BatchNorm** or a **Dropout** layer.

This model features a simple architecture, enabling very short training times. It is well suited for analyzing direct and statistical relationships between input and output. Due to its low computational demands, it achieves faster inference speeds.

However, it has limited generalization capabilities for temporal dynamics or sequential data, highly non-linear relationships, and noisy input conditions.

Model 2: Transformer-Based Neural Network

Architecture flow:

```
Input → (Input Embedding) → (Positional Encoding) → (Encoder) →
          (Output Heads) → (Synergy Layer) → Output
```

Key components:

- **Input Embedding** [Linear → GELU → Linear → LayerNorm] Adapts the input dimension to the expected transformer size.
- **Positional Encoding** Computed using sine and cosine functions, then added to the input. Helps the model capture temporal dependencies between poses.
- **Transformer Encoder** A stack of 3 Encoder Layers, each containing:
  - Multi-Head Attention (MHA) Layer (128-dim, 4 heads of size 32)
  - Add & Norm (Residual connection + Layer Normalization)
  - Feed-Forward (FF) Layer (512-dim)
  - Add & Norm

- **Output Heads** A dictionary of 5 heads (one per finger). Each head consists of: [Linear → GELU → Linear]. Each head predicts a dynamic number of outputs, depending on how many joint angles are split into sine/cosine components.

- **Synergy Layer (Optional)** [Linear] Activated only when the network has to predict finger synergies instead of joint angles.

Advantages & Trade-offs:

Captures complex nonlinear relationships between closure parameters and joint rotations. Requires a larger dataset for effective training and has a longer training time. Despite higher computational cost during inference, it remains real-time capable (as demonstrated in our use case).

Comparison Summary

Feature	**FCNN**	**Transformer**
Architecture	Simple (Dense Layers)	Complex (Self-Attention + Heads)
Training Speed	Very Fast	Slower (Needs More Data)
Inference Speed	Extremely Fast	Fast (Real-Time Viable)
Generalization Strength	Limited (Linear / Simple Cases)	Strong (Nonlinear / Noisy Data)

Table 1: Comparison of architectural, training, and generalization characteristics between the FCNN and Transformer models.

2.4 Unity

To simulate and visualize real-time human hand motion, the project was developed in the Unity game engine, where a 3D hand model was rendered and animated. The Weart Unity Software Development Kit (SDK) was integrated into the environment, and the Weart Left Hand prefab was used to represent the hand's anatomical structure. Each finger consists of three joints: proximal (01), intermediate (02), and distal (03) phalanges, resulting in 15 joints in total, each with three rotational degrees of freedom (DoF) along the x, y, and z axes. Joint rotations are computed and applied relative to the palm's reference frame to ensure anatomically plausible and consistent articulation and then to maintain consistency regardless of global orientation.

Dataset Generation

To construct the dataset for model training, a custom C# script (HandDataLogger.cs) was developed within Unity. This logger captures data at fixed intervals of 0.05 seconds, recording both the rotational states of the 3D hand model and the corresponding sensor inputs from the Weart haptic glove which are real-time parameters defined in the WeArtThimbleTrackingObject components attached to the thumb, index, and middle fingers.

Real-Time Prediction Framework

Following the training phase, a real-time inference pipeline was implemented using a client-server architecture to connect Unity with the Python-based neural network.

Unity acts as the client, acquiring four input values from the Weart TouchDIVER device: the input parameters of the index, and middle fingers, along with the abduction parameter of the thumb. These float32 values are serialized into a byte stream and transmitted via a TCP socket to the Python server. The Python server, operating independently, receives the incoming data, reshapes it into a tensor, and feeds it to a pre-trained neural model. The model output then is converted, when needed, into 45 values, corresponding to the predicted Euler angles for the 15 joints. The predicted joint rotations are then sent back to Unity via the same TCP connection. Unity deserializes the received data and applies the joint rotations using Quaternion.Euler, ensuring alignment with the local reference frame of the palm. This communication loop is executed in a background thread to guarantee low-latency inference, enabling smooth and responsive hand movement in the virtual environment.

3 Experiments

3.1 Direct PCA Output

In this section, we analyze the initial experiments carried out during the early stages of the project. The objective was to train various models capable of predicting the principal components obtained through PCA. To this end, models were trained to output different numbers of principal components.

After experimentally identifying a list of **17 joint angles** related to the finger phalanges to be fixed, a **reference model** was defined. This model does not directly predict the 45 joint angles but instead outputs a vector of **62 values**, including angles, sines, and cosines. This configuration serves as the baseline for comparison with the other model variants; we will refer to this model as T-62 or F-62 (Transformer or FCNN) and to the model that directly predicts 45 joint angles as T-joints or F-joints.

The alternative models were designed to predict **45, 30, 15, and 10 PCA components** and we will refer to them respectively as T-45/F-45, T-30/F-30,

T-15/F-15, T-10/F-10. The decision to apply PCA starting from the reference model is motivated by the fact that this configuration demonstrated the best overall performance. Therefore, it is particularly relevant to analyze how the use of PCA influences prediction quality.

Since the **hand model requires exactly 45 joint angles** as input, the output of the neural networks must be reconverted into the appropriate format. Specifically:

- The reference model performs a *conversion from sine and cosine to angle* for the fixed joints;
- The PCA-based models perform two sequential transformations: first, a *PCA inverse transformation* to reconstruct the 62 values, followed by a *final conversion to the 45 joint angles* required by the hand model.

3.2 PCA-Based Loss Only

The project also explores a novel loss formulation for regressing the full 62-dimensional hand pose from four high-level closure parameters. Rather than directly minimizing the Mean Squared Error (MSE) between the predicted and ground truth joint representations, we leverage PCA to compute the loss in a lower-dimensional latent space. This approach should encourage the models to focus on the most important motions in the joint data.

The training process is implemented in the script train_losspca.py. It supports two model architectures (FCNN and Transformer) which are detailed in the previous section. In this experiment, we do not train the models to predict the PCA components directly. Instead, we predict the full 62-dimensional joint representation, but compute the **loss in the PCA space**. Specifically:

- A PCA model is first fit to the full output space, capturing almost all of its variance.
- During training, the predictions and ground truth outputs are projected into this PCA space, and the loss is computed as the MSE between the projections.
- This should encourage the model to learn to match the most relevant subspace of the joint pose distribution, while still outputting interpretable joint values.

3.3 Autoencoder-Based Augmented Loss

Autoencoder-Based Loss

In a further development of the PCA-based loss approach, we replaced the linear dimensionality reduction of PCA with a nonlinear autoencoder. Unlike PCA, the autoencoder is capable of learning a latent space that better captures

complex non-linear synergies between finger joints. This latent space is not used as the output target of the regression models but only to define the loss.

In particular, we trained the FCNN and Transformer models to predict the full 62-dimensional joint representation, as in the baseline case. However, during training, the output prediction and the ground truth are both passed through a pretrained and frozen encoder:

$$\mathcal{L}_{\text{latent}} = \|f_{\text{enc}}(\hat{y}) - f_{\text{enc}}(y)\|^2 \tag{1}$$

Basically, during training, we took both the ground truth output and the model's prediction (each a 62D vector), passed them through the encoder, and then computed the MSE between their latent representations. This way, we pushed the prediction to lie close to the correct pose, in terms of how that pose "behaves" in the compressed space of plausible hand movements.

This method encourages the model to generate more natural and consistent hand poses without having to learn or predict the latent vector directly. It's also more efficient: for example, when the latent space is just 10 dimensions, computing the loss is much lighter than in full 62D space. Training is faster, and the model still learns to output realistic results.

During training, the encoder weights are kept frozen, and only the regression model is optimized. The predicted outputs consist of 62 scalar values, representing the sin/cos-encoded joint angles. This format is directly compatible with the real-time Unity pipeline, without requiring any inverse transformation.

Constraint Loss Term

In the initial experiments, the prediction models were trained using only a latent-space loss, where the predicted and ground-truth joint configurations were passed through a pre-trained autoencoder and compared in the compressed latent space. While this approach worked well in terms of global hand pose consistency, we noticed that the model could still produce implausible joint configurations, especially for individual joints with more freedom or weaker correlation to others.

To address this, we added a constraint loss term that penalizes joint predictions that fall outside the observed range in the dataset. For each joint component, we computed the empirical minimum and maximum values from the training set. The constraint loss is defined as:

$$\mathcal{L}_{\text{constraint}} = \frac{1}{n}\sum_{i=1}^{n}\left[\text{ReLU}\left(y_{\text{min}}^{(i)} - \hat{y}^{(i)}\right)^2 + \text{ReLU}\left(\hat{y}^{(i)} - y_{\text{max}}^{(i)}\right)^2\right] \tag{2}$$

where:

- $\hat{y}^{(i)}$ is the predicted value for joint i
- $y_{\text{min}}^{(i)}$ and $y_{\text{max}}^{(i)}$ are the minimum and maximum values for that joint in the dataset

- ReLU is the Rectified Linear Unit, which ensures the penalty is only applied when predictions exceed the limits

This loss was added to the total loss during training:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{latent}} + \mathcal{L}_{\text{constraint}} \tag{3}$$

After introducing this constraint loss, we observed a clear improvement in model behavior. The outputs were more stable, joints stayed within realistic limits, and the overall training process became more reliable. Validation loss consistently improved, and the poses looked more natural during real-time inference in Unity.

4 Results

In this section, we present the main results obtained from comparing the models in the Unity simulator.

4.1 Direct PCA Output

FCNN

Initially we compared F-62 and F-joints models observing the different behaviors, illustrated previously in the Sine/Cosine Encoding section: F-62 model can represent in a better way the hand posture rather than the F-joint model.

Next, we compare the prediction results of the other models to understand how PCA dimensionality influences hand posture prediction. When using PCA to reduce the complexity of our problem, we observed that there is no significant drop in performance between 45, 30 and 15 PCA components, but with the model F-10 the Index finger can't be reconstructed correctly. This suggests that we can highly reduce the complexity of our problem, but at the same time still get some error in reconstruction.

Transformer

For the Transformer architecture we analized the same differences in the T-62 and T-joint models, highlighting the importance of Sine/Cosine Encoding.

As shown in the image 1 in the Sine/Cosine Encoding section, the T-joints model struggles to accurately predict the hand posture in certain configurations, resulting in performance drops. In contrast the T-62 model handles these scenarios more effectively, maintaining reliable prediction quality.

Next, we compare the prediction results of the other models. When using PCA to reduce the complexity of our problem, we observed that there is no significant performance drop, even when using only 10 PCA components, which account for 98% of the total hand posture variance represented in the dataset.

This suggests that we can substantially reduce the complexity of our problem without losing important information.

In Figures 2 and 3 it is possible to analyze the per-joint MAE (in degrees) across all 45 joints between F-45 and F-62 models, and between T-45 and T-62 models.

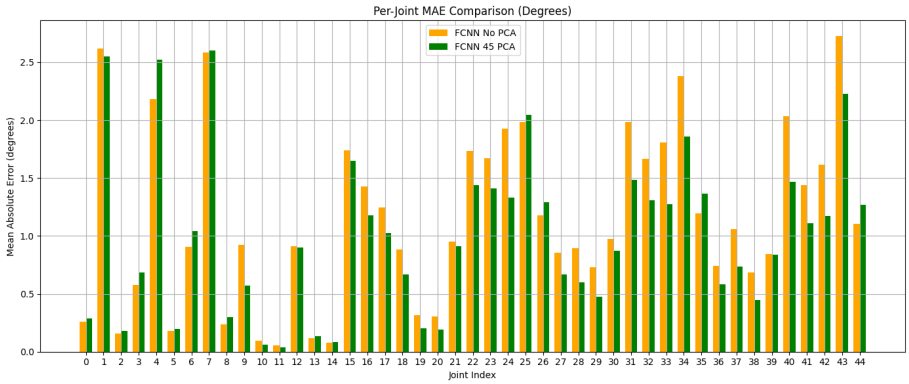


Figure 2: Average degree error for each angle, FCNN with PCA 45 and FCNN without PCA.

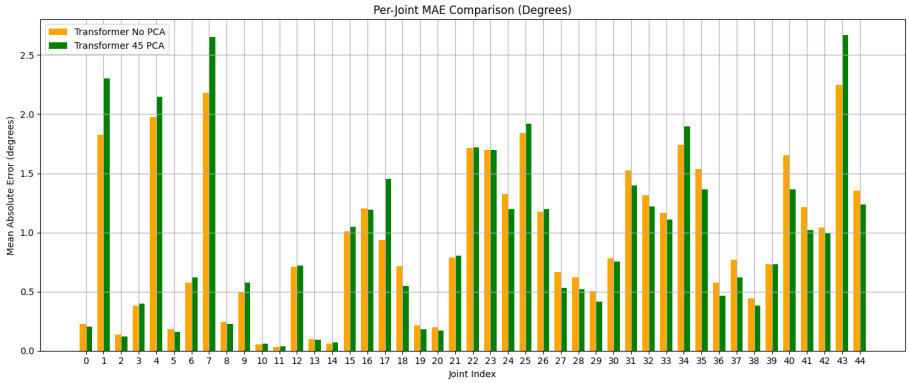


Figure 3: Average degree error for each angle, Transformer with PCA 45 and Transformer without PCA

4.2 PCA-Based Loss

To evaluate the trained models in a real-time scenario, we developed a separate script to run the prediction server and stream the joint angles into a virtual environment such as Unity. While the test loss remains consistently low across all training runs, as reported in Table 3 for both the FCNN and Transformer models, visual inspection reveals significant issues with joint-level accuracy.

N PCA Components	Best Test MSE Loss	
	FCNN	**Transformer**
45	0.0070	0.0059
30	0.0104	0.0089
15	0.0190	0.0134
10	0.0190	0.0158

Table 2: Training results with PCA in the loss domain.

Notably, the overall hand behavior (such as opening and closing of the hand and abduction of thumb) is captured well and feels responsive in the virtual environment. However, individual joints exhibit unnatural or erratic rotations, particularly at intermediate or fine levels of articulation. This issue becomes clearly observable in the demonstration videos provided in the video_results/training_losspca_results folder.

To better understand the impact of dimensionality reduction, we conducted experiments using 45, 30, 15, and 10 PCA components. Although the PCA reconstruction loss remains low, we observed that joint-level prediction quality deteriorates significantly when fewer than 45 components are used. This behavior is illustrated in Figure 4 and 5, which compares the average absolute prediction error (in degrees) for each joint between the models trained with 45 and 15 PCA components. The 15-component model introduces substantial errors in many joints, including dominant ones, despite having similar PCA loss.

When computing MSE in the PCA domain, there is a compression of the output space and the model only learns to minimize error in this compressed space, not directly in the joint angle space. As a result the PCA reconstructed output may look good statistically, but small errors in PCA space can cause big deviations in original joint angles after inverse PCA. So the model is not penalized for inaccurate predictions of specific joints, especially the non-dominant ones.

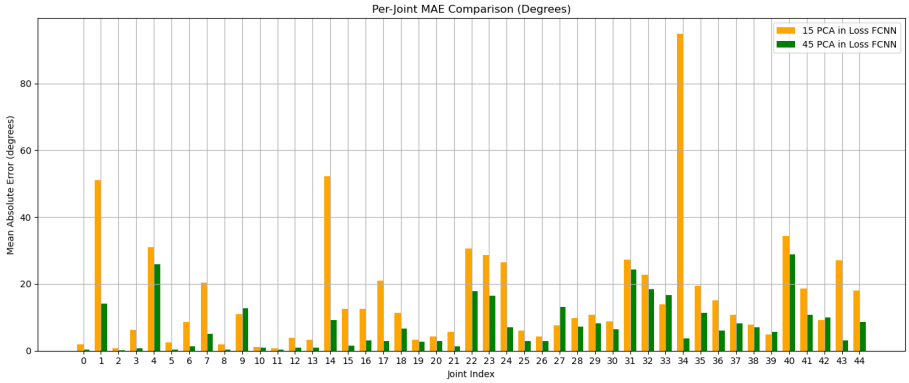


Figure 4: Average degree error for each angle with PCA in the loss of the FCNN model (45 components, 15 components)

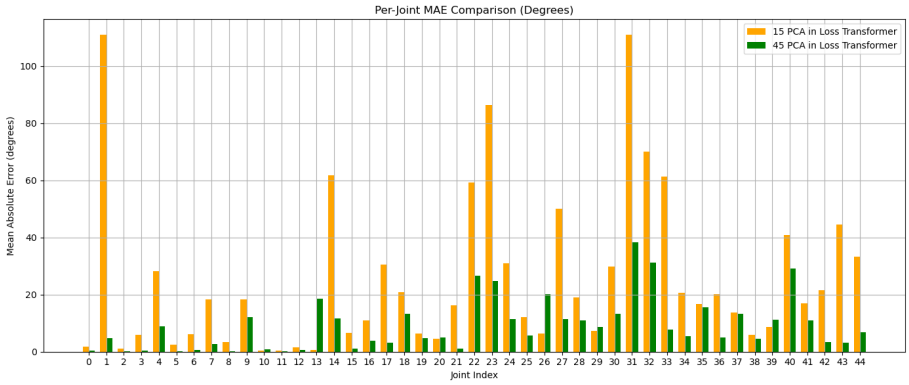


Figure 5: Average degree error for each angle with PCA in the loss of the Transformer model (45 components, 15 components).

4.3 Autoencoder-Based Augmented Loss

This section presents the results obtained from training and evaluating the FCNN and the Transformer model using a constrained latent-space loss.

Performance is evaluated both **quantitatively**, using the MAE computed per joint over the entire test set, and **qualitatively**, through visual inspection of the predicted hand movements in Unity. Additional experiments explore the impact of the latent space dimensionality and the effect of constraint-based regularization on the stability of the model.

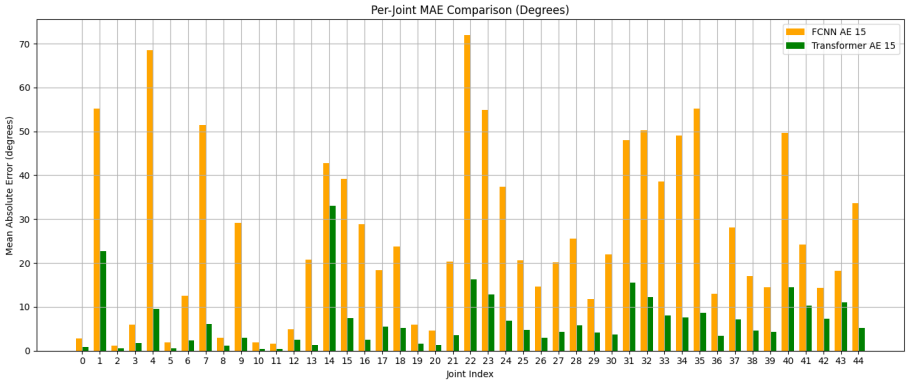


Figure 6: Average degree error for each angle with Autoencoder Based-Loss, latent space 15 (Transformer, FCNN)

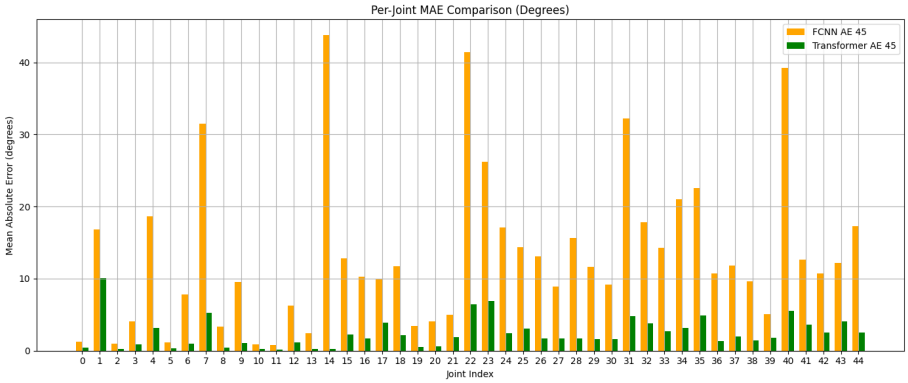


Figure 7: Average degree error for each angle with Autoencoder Based-Loss, latent space of 45 (Transformer, FCNN)

In particular, Figures 6 and 7 compare the per-joint MAE (in degrees) across all 45 joints, for both the FCNN and the Transformer-based models. Figure 6 shows the results using a latent space of 15 dimensions, while Figure 7 uses a latent space of 45 dimensions.

We can observe that the Transformer consistently achieves lower MAE values across the majority of joints, especially in those with higher variability or greater freedom of movement. This trend is even more pronounced in the case of the 45-dimensional latent space (Figure 7), where the performance gap between the FCNN and the Transformer becomes more significant. These results suggest that the Transformer model is better at capturing complex joint dependencies and benefits more from richer latent representations. On the other hand, when the latent space is smaller (15 dimensions), both models see a performance drop, but the Transformer still maintains a noticeable advantage. An important finding is that the visual simulations revealed that using the Transformer trained with a 15-dimensional latent space autoencoder strikes a good balance between

performance and efficiency. It delivers visually consistent and natural hand motion while significantly reducing training complexity and computational cost, making it a lightweight yet effective solution for real-time applications.

Moreover, this observation is supported by the results reported in Table 3, where the best test MSE loss achieved by the Transformer is consistently lower than that of the FCNN across all tested latent space dimensions.

Latent Space Dimension	**Best Test MSE Loss**	
	FCNN	**Transformer**
10	0.0062	0.0053
15	0.0061	0.0051
30	0.0043	0.0036
45	0.0049	0.0040

Table 3: Training results with Encoder in the Loss.

References

[1] "Weart website," https://weart.it/.

[2] "Weart developer guide td g1," https://weart.it/developer-guide-td-g1/.

[3] M. F. Marco Santello and J. F. Soechting, "Postural hand synergies for tool use," *The Journal of Neuroscience*, 1998.