electronics

Article

Machine Learning-Based Hand Pose Generation Using a Haptic Controller

Jongin Choi [1], Jaehong Lee [2], Daniel Oh [2] and Eung-Joo Lee [3,*]

[1] Department of Digital Media, Seoul Women's University, Seoul 01797, Republic of Korea; funtech@swu.ac.kr
[2] Department of Computer Science, Boston University, Boston, MA 02215, USA; jhonglee@bu.edu (J.L.); danoh@bu.edu (D.O.)
[3] Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85719, USA
* Correspondence: eungjoolee@arizona.edu

Abstract: In this study, we present a novel approach to derive hand poses from data input via a haptic controller, leveraging machine learning techniques. The input values received from the haptic controller correspond to the movement of five fingers, each assigned a value between 0.0 and 1.0 based on the applied pressure. The wide array of possible finger movements requires a substantial amount of motion capture data, making manual data integration difficult. This challenge is primary due to the need to process and incorporate large volumes of diverse movement information. To tackle this challenge, our proposed method automates the process by utilizing machine learning algorithms to convert haptic controller inputs into hand poses. This involves training a machine learning model using supervised learning, where hand poses are matched with their corresponding input values, and subsequently utilizing this trained model to generate hand poses in response to user input. In our experiments, we assessed the accuracy of the generated hand poses by analyzing the angles and positions of finger joints. As the quantity of training data increased, the margin of error decreased, resulting in generated poses that closely emulated real-world hand movements.

Keywords: hand pose; haptic controller; machine learning

check for updates

Citation: Choi, J.; Lee, J.; Oh, D.; Lee, E.-J. Machine Learning-Based Hand Pose Generation Using a Haptic Controller. *Electronics* **2024**, *13*, 1970. https://doi.org/10.3390/electronics13101970

Academic Editor: Maysam Abbod

Received: 13 March 2024
Revised: 29 April 2024
Accepted: 15 May 2024
Published: 17 May 2024

1. Introduction

In character animation, the generation of dynamic movements, including body and hand poses, is crucial for conveying emotions, actions, and subtle expressions. Smooth and realistic hand movements significantly enhance the immersion of animated characters, thereby elevating storytelling and audience engagement. Character animation is commonly categorized into two main approaches: data-driven and physics-based methods. Data-driven methods utilize existing datasets of motion capture or keyframe animation to inform character movements. These methods typically involve algorithms that analyze and interpolate motion data for animation generation [1,2]. On the other hand, physics-based methods simulate the physical interactions and constraints that govern movement in the real world [3,4]. In recent years, there has been active research into techniques employing deep learning for such a data generation scheme. However, research on specifically generating hand motions remains largely unexplored, with the majority of studies concentrating on character movement and interaction. The field of hand tracking technology has shown significant advancements, particularly with camera-based systems [5]. Such systems excel at capturing the overall shape and position of the hand in real time, offering valuable data for diverse applications. However, camera-based tracking often faces limitations in capturing the fine details of individual finger movements. To address this, haptic controllers provide highly precise data on individual finger positions and bending angles. This information is crucial for tasks requiring delicate manipulation in VR or animation. Furthermore, haptic controllers offer a supplementary control method in situations where camera views might be occluded. Overall, haptic controllers complement camera-based hand tracking by providing more refined finger movement data, enhancing their utility in various applications. Thus, in this study, we propose a novel framework that uses a machine learning model and subsequently utilizes input values from a haptic controller to generate animations of hand poses. In particular, our work introduces a custom-built haptic controller, as illustrated in Figure 1, where the five buttons are manipulated using the fingers. Because each button is attached to a motor, the custom controller is able to provide a haptic feedback to the fingers when grasping a virtual object.

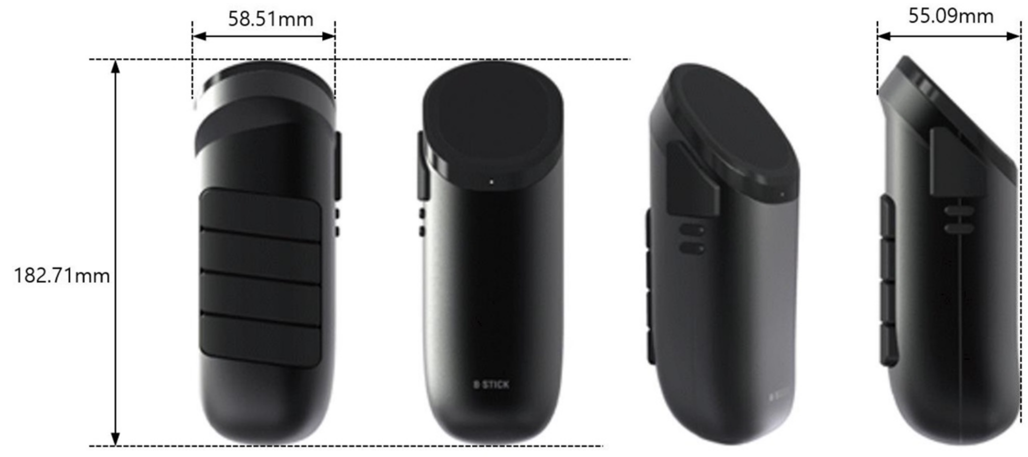


Figure 1. In-house haptic controller.

To facilitate data generation for our machine learning model, we initiated the process by smoothly integrating the Unity game engine with the haptic controller. Through this integration, we were able to use the input values from the controller to create detailed hand poses, which were essential components for training our model. In our setup, the haptic controller serves as the means for user interaction. When no buttons are engaged, the fingers are relaxed, while activating all buttons manually transitions the hand into a clenched fist. This intentional setup ensures precise control over the hand’s state, providing various expressions for data generation. Users are able to easily manipulate the haptic controller with their five fingers, shaping the hand’s configuration according to their preferences. Each manipulation prompts the automatic generation of corresponding data points, effectively translating the user’s intended hand movements into data. These input values, reflecting the movements of the five fingers, formed the foundation of our dataset. Meanwhile, the output values captured the nuanced angles of 15 joints, closely aligned with the poses of the fingers. With each finger having three joints, totaling 45 rotational dimensions, the output data offered a comprehensive representation of hand movement. For every frame of data, we followed a structured format: five input values paired with forty-five output values, ensuring a robust dataset for effective machine learning. Leveraging this carefully created dataset, spanning thousands of frames, we trained an XGBoost model to generate natural hand gestures.

The training methods of machine learning could be classified into three categories: supervised learning, unsupervised learning, and reinforcement learning. In our research, we introduce a method specifically aimed at generating smooth animations through supervised learning, which utilizes labeled data to establish relationships between input features and desired outputs. To effectively train the model, we relied on a labeled dataset containing examples of hand poses paired with their corresponding smooth animations, with various features extracted to capture hand movements such as joint angles and hand positions.

To generate hand gesture data, we captured hand poses using motion capture equipment. The generated motion consists of bending and extending one finger at a time. We used five motions that fully bend and extend the thumb, index finger, middle finger, ring finger, and pinky finger, respectively. Each motion consists of a total of 2000 frames. The label for a fully extended finger is 0, and the label for a fully bent finger is 1. In between, the pose is assigned a real value between 0.0 and 1.0 based on the bending angle. A hand consists of five fingers, and each finger has three joints, for a total of fifteen joints. Each joint has three angles $(x, y,$ and $z)$ in a local coordinate system. This indicates that, in one frame, a hand pose has 45 angle values. The five input buttons of the haptic controller correspond to the poses of each finger. Thus, five real numbers between 0.0 and 1.0 correspond to 45 real numbers between 0 and 360 degrees. The haptic controller was connected to the Unity game engine and automatically generated various data based on user input. Table 1 is an example of data created in this manner. By connecting the Unity game engine and haptic controller, we could automatically generate and label datasets. This allowed us to generate as much training data as needed and rapidly train and evaluate our models, which helped in developing our hand pose mapping solution.

Table 1. Example of training data generated from Unity: each frame includes input values from five fingers, producing three Euler angles per finger joint, for a total of nine output values.

Frame Index	Input Values (Haptic Controller)	Output Values (Euler Angles of Finger Joints)
	0.218	5.857, 283.756, 232.392, 4.937, 359.360, 340.130, 0.309, 358.655, 359.481
1	0.909	5.358, 12.075, 273.488, 9.882, 7.548, 258.746, 1.067, 3.158, 283.456, 343.980
	0.296	9.120, 322.664, 0.135, 354.951, 322.070, 357.975, 2.247, 330.585, 350.555
	0.089	9.931, 338.857, 359.734, 359.633, 342.244, 359.883, 1.348, 341.286
	0.309	353.242, 12.235, 325.342, 0.605, 1.288, 320.620, 1.180, 4.108, 330.486
	0.802	4.733, 293.451, 220.497, 1.697, 4.671, 321.272, 1.791, 359.329, 305.964,
2	0.023	1.462, 5.810, 345.269, 5.207, 359.485, 355.349, 0.314, 2.624, 356.788,
	0.730	347.158, 6.139, 286.962, 355.279, 352.070, 282.921, 355.742, 3.991, 286.708,
	0.786	353.398, 8.984 284.587, 357.526, 357.164, 280.976, 357.948, 1.592, 276.119,
	0.549	354.601, 12.835, 306.178, 1.406, 2.098, 302.382, 0.945, 2.538, 306.932
...	...	...

The generated datasets were of five types: 1000, 3000, 5000, 7000, and 9000. They were trained using the XGBoost algorithm, and each dataset was used as training and validation data in a ratio of 8:2. Throughout the training process, which was evaluated using a validation set to ensure generalizability, the model learned to correlate input hand pose features with the resulting hand movement generation. This approach enabled the generation of smooth animations, even for novel hand poses, as the model can predict the corresponding animation when presented with new input. By leveraging supervised learning techniques, we can train models to produce realistic and seamless animations of the hand movement, ultimately enhancing the quality of animations across diverse applications, including virtual reality (VR), gaming, and motion capture. Particularly, hand pose generation is of significant importance in VR. Realistic hand representation plays a crucial role in enhancing the sense of presence within VR environments. When virtual hands accurately mimic real-world hand movements, it reduces cognitive dissonance and enhances immersion. Thus, our proposed method effectively maps user input from a haptic controller to generate smooth and natural hand pose animations, contributing to both immersive VR experiences and advancements in character animation.

The key contributions of our proposed framework are outlined as follows:

- We utilized an in-house haptic controller with a Unity game engine to expedite motion data collection. A haptic controller and Unity integration improved the speed, accuracy, and user-friendliness of hand pose data collection.
- We employed supervised learning to generate hand pose animation seamlessly, eliminating the need for extensive motion data.
- We systematically assessed the effectiveness of our proposed method by evaluating joint angle and position errors across different amounts of data sample sizes.

2. Related Works

Over the past decade, a variety of methods have been investigated for character animation. Pejsa and Andzic [6] discussed plans for motion synthesis in interactive applications, as well as motion graphs and parametric models. Geijtenbeek and Pronost [7] reviewed literature on physical simulation for interactive character animation. Karg et al. [8] conducted research on the generation and recognition of movements based on emotional expressions. Wang et al. [9] analyzed state-of-the-art techniques in 3D human motion synthesis, with a focus on methods based on motion capture data. Recently, Alemi and Pasquier [10] investigated data-driven movements using machine learning. While their paper includes a review of some deep learning-based methods, there have been numerous advancements in recent years. Recently, various deep learning-based character animation techniques have been investigated, such as motion retargeting using unsupervised learning [11] and motion editing and motion transfer using encoders and decoders [12].

2.1. Character Pose Representation

The methods for representing a character's pose can be divided into two approaches: directly specifying the positions of the joints or specifying the lengths of bones and the angles of joints to determine the positions of the joints. The approach of directly specifying the positions of the joints typically involves defining the local coordinate systems of the character's joints over time [13–15]. The overall motion of the character consists of the local poses of the character's joints and the global movement of the root (pelvis). While various coordinate systems can be used to specify joint positions, most joint positions are described in Cartesian coordinates due to their convenience for interpolation, visualization, and optimization, without discontinuities or singularities. However, there are some limitations related to the structure of human motion. For example, joint positions do not encode the directional information of surrounding bones necessary for representing more natural mesh deformations in animations. Furthermore, since joint position representations do not explicitly constrain bone lengths to remain constant over time, additional constraints are required to ensure that bone lengths do not change.

Euler angles are the most intuitive method for representing joint angles. They describe the position and orientation in a three-dimensional coordinate system around three different axes. If two of the three rotation axes coincide, a well-known problem known as gimbal lock can arise, leading to a reduction in degrees of freedom. Gimbal lock can only be avoided when one or more rotation axes are restricted to less than 180 degrees. Consequently, Euler angles are not suitable for inverse kinematics, dynamics, and space–time optimization. Furthermore, due to the highly non-linear nature of orientation space, they are not suitable for interpolation [16]. Additionally, to avoid ambiguity, various rules exist regarding the order of axes that must be formally defined for each application. For such reasons, this representation is not suitable for diverse applications.

The alternative method for representing joint angles is quaternions. Similar to Euler angles, quaternion space maintains the same local geometry and topology [16]. This angle representation has been popularized in deep learning-based animation through the quaternion-based framework, QuaterNet [17,18] for human motion prediction. In this framework, a penalty term is introduced into the loss function for all predicted quaternions by a network that minimizes divergence from unit length, encouraging the network to predict valid rotations and enhance training stability. Furthermore, the predicted quaternions are normalized after penalty computation to reinforce their validity. The distribution of predicted quaternion norms converges to a Gaussian with a mean of 1 during training, indicating that the model learns to represent valid rotations effectively. Pavllo et al. [17] demonstrated promising results using quaternions, which have since been adopted by various researchers [19–21]. Additionally, several studies have proposed hybrid representations that combine joint translation and rotation. This approach has been shown to improve the learning process, particularly when augmented with additional parameterization information like velocity. For example, hybrid representations have incorporated joint position and orientation [22], the position and velocity of joints [23], or linear and angular velocities of joints [24]. Tang et al. [25] used Recurrent Neural Networks and Variational Autoencoders to represent translation and rotation, while Duan et al. [26] used Transformer Encoders to represent a three-dimensional position.

Since there is no interpolation between poses and since hand poses are generated directly from the input values of the haptic controller, we trained a model using Euler angles to generate the angles of finger joints. Due to their limited range, quaternions are prone to side effects such as jittering. By leveraging Euler angles, we aimed to mitigate potential issues associated with quaternions, thus enhancing the overall reliability and performance of our hand pose generation system.

2.2. Machine Learning

Various machine learning techniques have been investigated. A few studies have explored the application of Random Forest for regression tasks. This method employs decision trees to infer a set of posterior probabilities for the input data. Decision trees are comprised of internal nodes and leaf nodes, and internal nodes direct the data to one of their children. In binary cases, data splitting decisions are made through simple yes/no criteria. The output of multiple decision trees are combined to produce a single result [27–29]. XGBoost is a method that involves creating multiple weak classifiers and integrating them into a robust model [30]. The objective is to correct errors made by preceding learners through a sequential process. XGBoost offers advantages in speed and performance when working with smaller datasets typical of pose estimation tasks [27–29]. These datasets often provide keypoints of the hand (e.g., joint coordinates) in a structured, tabular format, which aligns well with the strengths of XGBoost. Furthermore, XGBoost's training and inference processes are generally faster and less computationally demanding than those of convolutional neural networks (CNNs). This characteristic can be crucial for real-time hand pose estimation systems, especially those operating on resource-constrained devices.

Additionally, convolutional neural networks have been proposed for hand pose estimation, predicting 3D joint locations using depth map information. In particular, this approach incorporates a prior on the 3D pose [31]. Furthermore, by leveraging the physical constraints of a hand, a novel approach has been introduced employing an object-manipulating loss function. This approach considers knowledge of hand–object interaction to enhance performance. Furthermore, the work deployed real-time interactive applications utilizing the prototype, showcasing the potential for intuitive finger-based interaction within VR environments [32]. Moreover, another study introduced a 2.5D representation to apply CNN methods to a capacitive image of a curved surface, building upon recent achievements. Specifically, they presented an innovative system for 3D hand pose estimation capable of accurately predicting human hand poses from vision-based frames. Additionally, the system can effectively estimate hand pose whether the hand is bare or manipulating an external object. Their experimental results across multiple datasets demonstrate the strong performance and robustness of the proposed method [33]. The other study introduces a unique confidence estimation approach using the disparity between predictions from two teacher networks for self-training domain adaptation in hand keypoint regression and segmentation. Their proposed approach integrates the confidence estimation method into self-training via mutual training, employing knowledge distillation with a student network and two teacher networks. They demonstrated superior performance over existing methods across various imaging conditions and enhanced qualitative results in in-the-wild, egocentric videos [34].

3. Methods

We propose a novel machine learning-based technique to generate natural hand poses from a haptic controller. Figure 2 presents the overall pipeline of our work. First, to generate training data, we paired five input values from the haptic controller with corresponding hand poses. Each hand pose consisted of fifteen joint angles, three for each of the five fingers, stored in the Euler (x, y, z) angle format within a local coordinate system. Consequently, each training dataset comprised five real values as inputs and 45 (15×3) angle values as outputs. Using thousands of such generated datasets, we trained a machine learning model to generate hand gestures. In our experiments, we observed variations in the accuracy of results based on the method and quantity of training datasets generated.

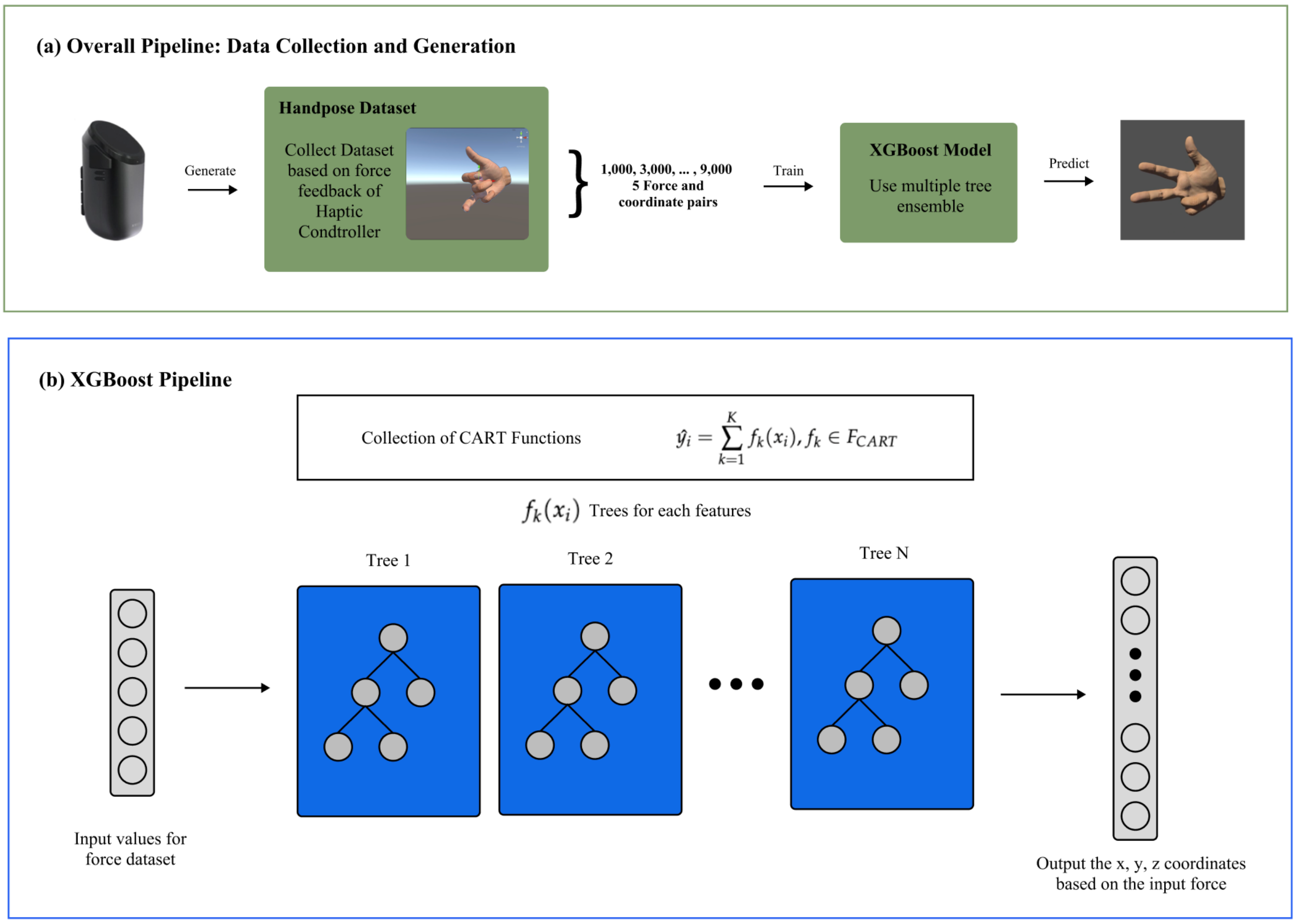
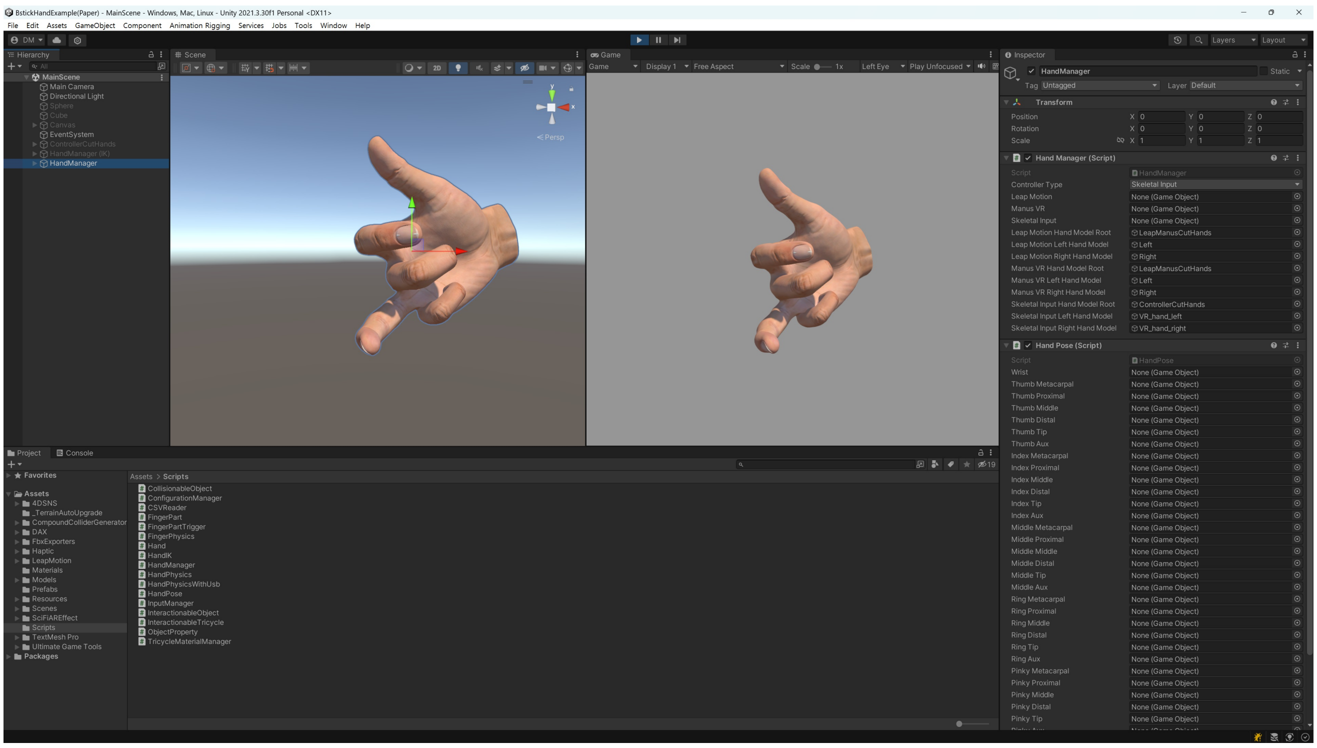

(a) Overall Pipeline: Data Collection and Generation
Generate
Handpose Dataset
Collect Dataset based on force feedback of Haptic Condtroller
1,000, 3,000, ... , 9,000 5 Force and coordinate pairs
Train
XGBoost Model
Use multiple tree ensemble
Predict
(b) XGBoost Pipeline
Collection of CART Functions
$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F_{CART}$
$f_k(x_i)$ Trees for each features
Tree 1
Tree 2
Tree N
Input values for force dataset
Output the x, y, z coordinates based on the input force

Figure 2. The overall pipeline. (**a**) The haptic controller is utilized to generate coordinates from each joint, and (**b**) XGBoost is then employed for estimation. Input data are the force of 0–1 applied to each finger, and output data are the corresponding coordinates (x, y, and z) of each joint. Here, i represents the individual row of data from the force and coordinate pair.

3.1. Data Collection Using Unity Game Engine

In this work, to train the machine learning model for generating hand poses, we used the Unity game engine to create multiple training datasets with varying sizes. As shown in Figure 3, we used the Unity game engine to generate hand pose data and check the results trained with machine learning. Initially, the haptic controller was manually manipulated to generate values ranging from 0.0 to 1.0. From these values, we established a real-time data generation system capable of saving hand poses. The input values from users were collected via the five buttons on the haptic controller. This controller was linked to the Unity game engine using a specially crafted module, facilitating the instantaneous transfer of input data. When the buttons are inactive, the fingers maintain their fully extended position, and, when pressed, they bend accordingly. This setup enabled us to precisely capture the user’s hand movements.

We gathered input data from the haptic controller and calculated the output values, representing the angles of 15 joints in hand poses. These angles were expressed as Euler angles (x, y, and z) and were computed at a rate of 90 frames per second, resulting in the creation of 1000 datasets. Moreover, we conducted experiments to explore the effects of varying data quantities. By incrementally increasing the dataset size in 2000-unit intervals, we produced datasets containing 3000, 5000, 7000, and 9000 samples. Using this approach, we evaluated how the size of various datasets impacted the accuracy of generated hand poses. To streamline the data collection process, the system incorporates an automated saving feature. When users repeatedly press and release the buttons on the haptic controller within a set timeframe, the data are automatically stored. This functionality ensured efficient and hassle-free data generation, enhancing the overall user experience. Table 1 illustrates an example of the dataset generated from Unity. The left column shows the frame number, the middle column shows the input values of the five buttons on the haptic controller, and the right column shows the 45 output values, which are the angles of the 15 hand joints that generate the hand pose. Each joint has three angle values $(x, y$, and $z)$, and each finger has three joints. Thus, nine output values correspond to one input value. We used the generated data for training and validation in a ratio of 8:2.



Inspector

HandManager ✓ Static

Tag Untagged Layer Default

Transform

	X	Y	Z
Position	0	0	0
Rotation	0	0	0
Scale	1	1	1

✓ Hand Manager (Script)

Script	HandManager
Controller Type	Skeletal Input
Leap Motion	None (Game Object)
Manus VR	None (Game Object)
Skeletal Input	None (Game Object)
Leap Motion Hand Model Root	LeapManusCutHands
Leap Motion Left Hand Model	Left
Leap Motion Right Hand Model	Right
Manus VR Hand Model Root	LeapManusCutHands
Manus VR Left Hand Model	Left
Manus VR Right Hand Model	Right
Skeletal Input Hand Model Root	ControllerCutHands
Skeletal Input Left Hand Model	VR_hand_left
Skeletal Input Right Hand Model	VR_hand_right

✓ Hand Pose (Script)

Script	HandPose
Wrist	None (Game Object)
Thumb Metacarpal	None (Game Object)
Thumb Proximal	None (Game Object)
Thumb Middle	None (Game Object)
Thumb Distal	None (Game Object)
Thumb Tip	None (Game Object)
Thumb Aux	None (Game Object)
Index Metacarpal	None (Game Object)
Index Proximal	None (Game Object)
Index Middle	None (Game Object)
Index Distal	None (Game Object)
Index Tip	None (Game Object)
Index Aux	None (Game Object)
Middle Metacarpal	None (Game Object)
Middle Proximal	None (Game Object)
Middle Middle	None (Game Object)
Middle Distal	None (Game Object)
Middle Tip	None (Game Object)
Middle Aux	None (Game Object)
Ring Metacarpal	None (Game Object)
Ring Proximal	None (Game Object)
Ring Middle	None (Game Object)
Ring Distal	None (Game Object)
Ring Tip	None (Game Object)
Ring Aux	None (Game Object)
Pinky Metacarpal	None (Game Object)
Pinky Proximal	None (Game Object)
Pinky Middle	None (Game Object)
Pinky Distal	None (Game Object)
Pinky Tip	None (Game Object)

Figure 3. Unity screen for hand pose data generation.

3.2. Hand Pose Estimation Using Machine Learning

For our application, we employed machine learning techniques to generate realistic and seamless hand poses. We utilized supervised learning, leveraging an annotated dataset obtained from the Unity game engine. Specifically, we selected Extreme Gradient Boosting (XGBoost), since it is known for its performance and speed. Furthermore, XGBoost can handle regression tasks and incorporate L1 and L2 regularization methods to mitigate overfitting. In this work, we utilized the XGBoost algorithm to illustrate the effectiveness of combining a number of weak models to ultimately create a strong model. Multiple decision trees are combined in the XGBoost method to form Classification and Regression Trees (CART). A decision tree categorizes input data based on features at each node. Therefore, the XGBoost method uses multiple decision trees with different weights according to their characteristics. Consequently, it trains using individual trees based on different features.

The equation for the CART model is as follows:

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F_{CART}$$

Each function f_k in the CART model serves as a weighted tree function for specific features. The XGBoost model predicts the output of input x_i as the sum of each output from all weighted functions in the CART model. The training objective function of CART is as follows:

$$obj(\theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{k} \Omega(f_k)$$

Here, the function $l(y_i, \hat{y}_i)$ represents the loss function used for y_i and $\hat{y}_i$, and $\Omega(f_k)$ acts as a regularization function to prevent overfitting.

4. Experiments

4.1. Experimental Setup

In our experimental data training approach, we aimed to optimize results through parameter tuning within the XGBoost algorithm. We systematically adjusted three key parameters: the number of rounds to determine the quantity of trees, the maximum tree depth, and the lambda for regularization. Our experimentation involved thorough exploration, testing different numbers of iterations for the number of rounds, and varying depths for each decision tree layer. To tackle the prevalent concern of overfitting observed in deeper layers, we strategically approached the task by categorizing the lambda parameter, which plays a important role in L2 regularization. For parameter tuning, we used grid search to identify the most effective configuration. In addition, for this work, we generated thousands of diverse hand pose datasets containing non-sequential and dissimilar hand poses to help avoid overfitting. After thorough experimentation and rigorous evaluation, we discerned that the optimal setup entailed the utilization of 200 rounds of estimators, a 3-layer architecture, and a lambda value of 10. Throughout the duration of our experiment, we maintained a constant training-to-testing ratio of 7:3 for each hand pose dataset. Python 3.12.0 with XGBoost package version 2.0.3, and scikit-learn package version 1.4.0 were utilized for conducting our experiments.

4.2. Pose Difference Detection

In this work, we compared the actual pose of the hand with the generated pose to measure the accuracy of the hand movements produced by the XGBoost algorithm. We analyzed the accuracy of 15 finger joints involved in animation, among the numerous joints comprising the hand. In Figure 4, we demonstrate how we assessed the error for a single finger joint within a single frame. In the left image, dotted lines represent the finger bones generated by our proposed method, while solid lines depict those measured from the actual pose. The small circles along the lines indicate the joints. Within the left image, for one of the three bones forming a finger, we choose a consistently colored reference joint and place it at the same position. Subsequently, we calculate both the angle and distance between these two bones. In the right image, the red arc illustrates the angle error between the two bones, while the blue line signifies the distance error between the two joints. Considering that we align the reference joint uniformly, all measured errors are classified as local errors. We exclusively measure angle differences concerning local axes, as referencing them based on the world axis holds no significance. As for position differences, we determine the distance variation relative to the wrist's root position, which serves as the reference for the world position.

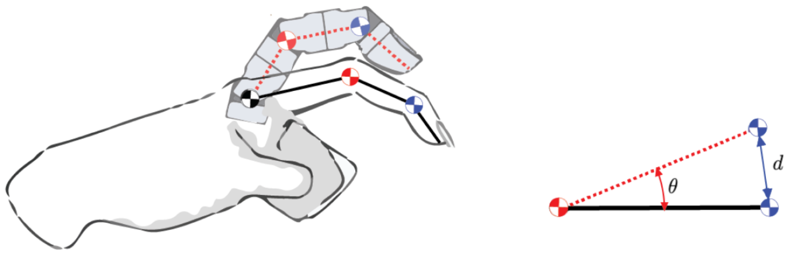


Figure 4. Difference measurement of position and angle: The dotted lines represent the generated pose, while solid lines represent the ground truth pose. The red arc indicates the angle difference, and the blue line represents the position difference.

To calculate the position error (S_d) and angular error (S_a) of hand poses, we used the following formula:

$$S_d = \sum_{j=1}^{n_2} \sum_{i=1}^{n_1} d_i \, , \quad d_i = \left\| p_1^i - p_2^i \right\|$$

$$S_a = \sum_{j=1}^{n_2} \sum_{i=1}^{n_1} \theta_i \, , \quad \theta_i = arccos\left(\frac{\vec{v_1^i} \cdot \vec{v_2^i}}{\left\| \vec{v_1^i} \right\| \left\| \vec{v_2^i} \right\|} \right)$$

S_d represents the sum of all finger joint position errors across all frames in a hand gesture animation. p_1 denotes the position of a finger joint in the original pose, and p_2 denotes its position in the generated pose. Likewise, S_a represents the sum of all finger joint angle errors across all frames in a hand gesture animation. v_1 denotes the vector of a finger bone in the original pose, and v_2 denotes its vector in the generated pose. The direction of the vector is defined from the joint closer to the palm to the joint closer to the fingertip. The angle between the two vectors is calculated using the properties of the vector dot product. n_1 represents the number of finger joints, and n_2 represents the total number of frames in the hand gesture animation, respectively. We set n_1 at 15 and n_2 at 2000.

The values in Table 2 and the graph in Figure 5 represent the angular differences between the original and generated hand poses. These angular differences are calculated using the formula for S_a, which measures the sum of the angle errors across all finger joints and all frames in the animation. The green graph in Figure 6 represents the local position differences of finger joints (p_1, p_2) in the local coordinate system. These local position differences are calculated using the formula for S_d, which measures the sum of the position errors across all finger joints and all frames in the animation. The red graph in Figure 6 represents the world position differences of finger joints (p_1, p_2) in the world coordinate system. These world position differences are also calculated using the formula for S_d.

Table 2. Effects of training dataset size on hand pose generation accuracy.

Number of Data Samples	1000	3000	5000	7000	9000
Angle (°) Difference	93,232	33,992	26,777	23,507	17,757
World Position Difference	524.55	191.87	156.32	132.84	104.23
Local Position Difference	414.89	512.30	122.13	106.84	83.73

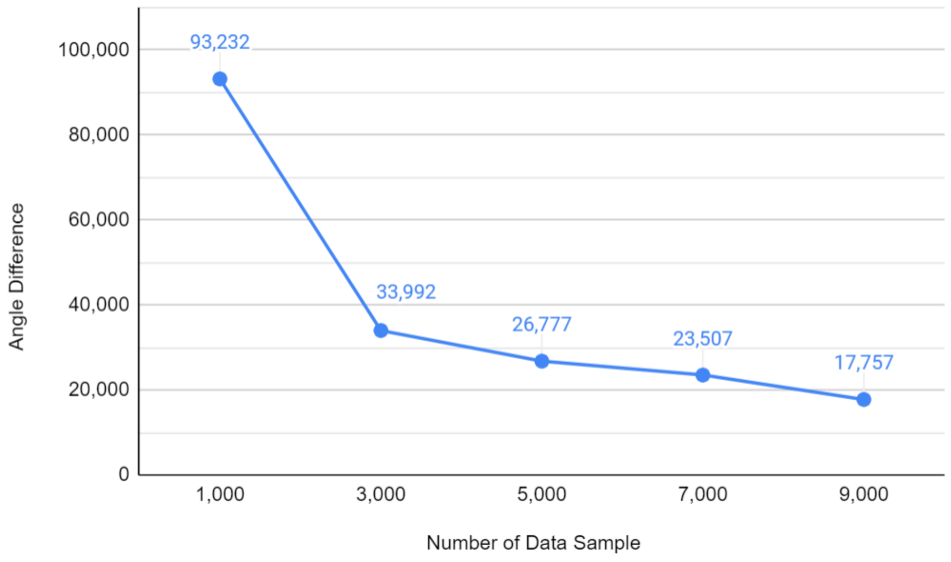


Figure 5. Results of angle differences. The x-axis indicates the number of training data samples and the y-axis indicates the angle error in degrees (°).

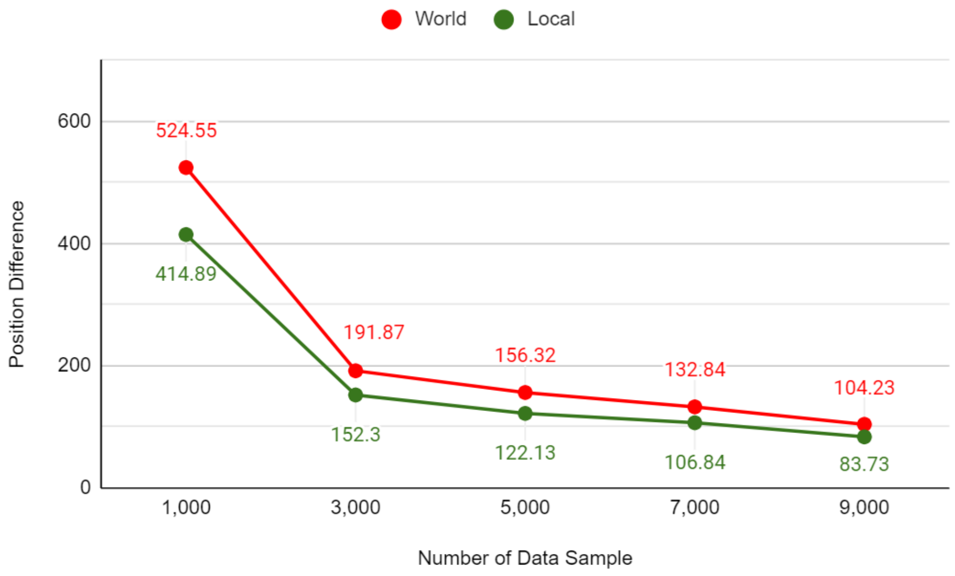


Figure 6. Results of position differences. The x-axis indicates the number of training data samples and the y-axis indicates the position differences.

4.3. Experimental Results

We utilized visually distinguishable differences in the angles and positions of joints among various errors to measure the accuracy of the hand poses generated from the XGBoost algorithm. Table 2 shows the errors of hand poses generated using the training dataset. From left to right, the table presents results obtained using 1000, 3000, 5000, 7000, and 9000 data samples. The angle difference represents the cumulative error of finger joint angles between the original pose and the pose generated by the proposed method. The unit for the angle is degrees (°), and position does not have a specific unit as it corresponds to values measured in the Unity game engine.

To measure the accuracy of hand poses generated by the trained machine learning model, 2000 data samples were created by manipulating the haptic controller. Although generated using the same method as the training data, the only difference lies in their quantity and content. The error values illustrate the cumulative difference in all angles over

2000 frames across 15 joints, compared to the ground truth pose. For instance, using a machine learning model trained with 1000 data samples resulted in a cumulative error of 93,232 across all poses generated. The measurement method for position difference is similar to angle difference, but instead calculates the error using joint positions instead of joint angles. World position refers to the results measured along the world axis, while local position refers to the results measured along each joint's local axis. There was a tendency for the error to decrease as the number of data samples increased.

Figure 5 shows the results of the angle differences illustrated as a graph in Table 2. The vertical axis represents the angle error, while the horizontal axis represents the amount of training data. As shown in Table 2, as the number of data samples increases, the joint angle error decreases. Similarly, Figure 6 illustrates the results of the position differences from Table 2 as a graph. The red line represents the world position difference, while the green line represents the local position difference. Similar to angle differences, the joint position error decreases as the number of data samples increases. The world position error is slightly larger than the local position error due to the cumulative effect of errors from local positions into world positions. The larger discrepancy in errors between the 1000 and over 3000 datasets may be attributed to the test dataset containing twice as many samples, totaling 2000. From approximately 3000 samples, when the amount of test data exceeds that of the training data, we observe a consistent error rate. However, the reduction in error is relatively modest compared to the increase in dataset size.

Figure 7 presents a sequence of images with generated hand poses based on the number of data samples. The numbers at the top represent the number of data points used for training. The column on the far left shows the hand poses generated using real-world data. These are a few representative poses out of 2000 hand poses. From left to right, it illustrates the hand poses generated using 1000, 3000, 7000, and 9000 training data samples, respectively.

Each column of consecutive images captures results at intervals of 150 frames (about 1.67 s), starting from frame 30 (i.e., frames 30, 180, 330, 480, 630, 780, 930, 1080, and 1380). Since poses in sequential frames are similar, we extracted poses with a difference of more than 1 s for acquisition. Hand poses generated with 1000 data samples visually differ from the ground truth pose, especially in the way the second finger bends in the first and last images. However, as the number of data samples increases, the similarity to the ground truth pose becomes more apparent. The hand poses generated by the model trained with 1000 data samples exhibit noticeable visual discrepancies. However, as the amount of training data increases to 3000, subtle differences from the actual poses emerge, though they may not be easily discernible without close observation. With over 9000 samples, the generated poses closely resemble the actual ones, making it very difficult to distinguish between them.

In Figure 7, inconsistencies in the 1000-data point results can be observed in spot poses, but it is difficult to detect differences in the pose when there are 3000 or more data points. This is mainly because the error values specified in Table 2 are cumulative values of all the errors that occur in the 15 finger joints over 2000 frames, thus appearing large. However, the error that occurs in a single finger joint in a single frame is negligible. In other words, in the 1000-data point result, the accumulated angular difference is 93,232 degrees, but the actual error that occurs in a single finger joint, visible to the naked eye, is very small, averaging 3.1 degrees (93,232/(2000 $\times$ 15)). In the 3000-data point results, the angular error averages 1.133 degrees (33,992/(2000 $\times$ 15)), and the error that occurs in a single finger joint is even smaller, around 1 degree. The positional difference can also be calculated in the same way, and the accumulated error is 152.3 in local terms, but the error that occurs in a single finger joint is also negligible, averaging 0.051 (152.3/(2000 $\times$ 15)).

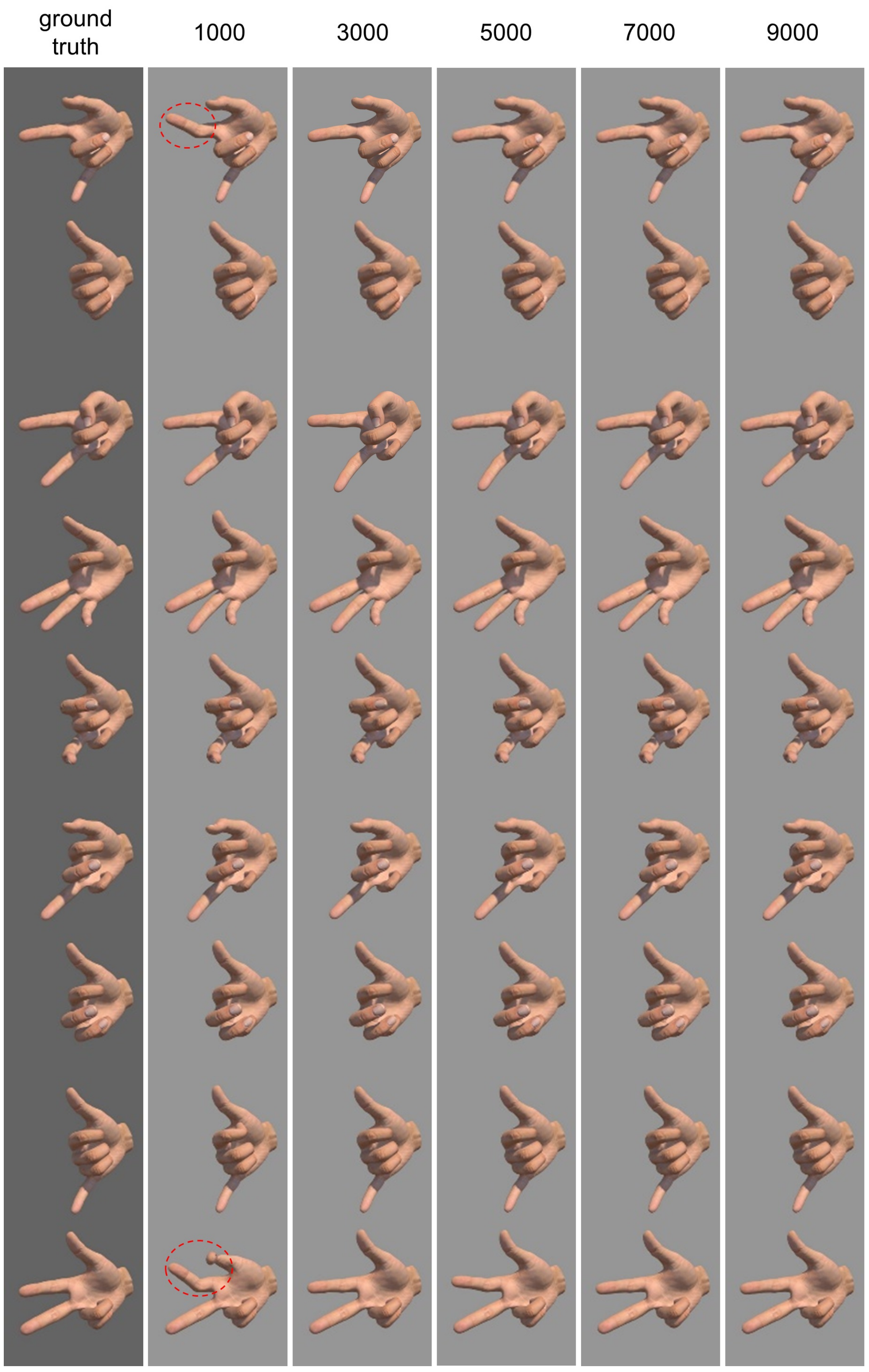


Figure 7. Sequence of images showing hand poses generated from varying numbers of data samples. The dotted red circles indicate the areas where there are errors in the position and angle of the hand joints.

5. Conclusions

In this work, we present a method to create natural hand movements using an XGBoost algorithm based on input values from our custom-built haptic controller. We synchronized the Unity game engine with a haptic controller to generate multiple training datasets consisting of thousands of frames. Moreover, we conducted experiments by adjusting the hyperparameters of the XGBoost algorithm numerous times to identify the most accurate parameters for generating hand poses. Specifically, we generated hand gestures using five different datasets ranging from 1000 to 9000 data points with 2000-point increments. We observed that the error decreased, and the generated poses became more similar to the real poses, as the number of data points increased. To measure the error, we considered not only visual aspects such as the pose but also numerical values such as the angles and positions of the finger joints, which are difficult to observe accurately with the naked eye. For the position error, we measured it using both local coordinates with the parent of the finger joint as the origin and world coordinates with the wrist as the origin. The error was larger in the world coordinate system, which occurred because the overall position of the joints was farther from the origin. However, similar to the local coordinate system, the error showed a decreasing trend as the number of data points increased. The results using the 1000-data point dataset could be distinguished to some extent with the naked eye, but the results trained with 3000 or more data points were difficult to distinguish. Therefore, we measured the positions and angles of the finger joints as a criterion for accuracy. This led to the outcome that the number of training data points and the size of the error showed an inversely proportional relationship.

However, as the amount of training data surpassed 10,000 samples, significant improvements in visual aspects or error reduction were not observed. By employing a trained XGBoost model, we can generate hand poses that appear natural without the need for extensive motion data. Furthermore, we propose a technique to generate hand poses from stick-type haptic controllers, which do not align with hand shape. However, even with extensive training data, minor tremors may still occur during hand pose generation. This highlights the potential benefits of integrating time-series methods or employing other refined approaches to enhance the process further. In the future, it would be interesting to explore making the controller thinner and more ergonomically designed for comfortable handling, or to utilize motion data interpolation to enable interaction with surrounding objects or environments. This study could enhance user experience and offer exciting possibilities for further development.

Author Contributions: Methodology, E.-J.L.; Implementation, J.C. and D.O.; Validation, J.L.; Investigation, J.L. and D.O.; Data curation, J.C.; Writing—review & editing, E.-J.L.; Supervision, E.-J.L.; Funding acquisition, J.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Institute of Information and Communications Technology Planning and Evaluation (IITP): 2021-0-00986, and supported by a research grant from Seoul Women's University: 2024-0028.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Gleicher, M. Retargetting Motion to New Characters. In Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques, Orlando, FL, USA, 19–24 July 1998; Association for Computing Machinery: New York, NY, USA, 1998; pp. 33–42. [CrossRef]
2. Kovar, L.; Gleicher, M.; Pighin, F. Motion Graphs. In Proceedings of the Annual Conference on Computer Graphics and Interactive Techniques, San Antonio, TX, USA, 23–26 July 2002; Association for Computing Machinery: New York, NY, USA, 2002; Volume 21, pp. 473–482. [CrossRef]

3. Faloutsos, P.; Van De Panne, M.; Terzopoulos, D. Composable controllers for physics-based character animation. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '01), Los Angeles, CA, USA, 12–17 August 2001; Association for Computing Machinery: New York, NY, USA; pp. 251–260. [CrossRef]
4. Coros, S.; Beaudoin, P.; Van De Panne, M. Robust task-based control policies for physics-based characters. In *SIGGRAPH Asia '09: ACM SIGGRAPH Asia 2009 Papers*; Association for Computing Machinery: New York, NY, USA; pp. 1–9. [CrossRef]
5. Kapitanov, A.; Kvanchiani, K.; Nagaev, A.; Kraynov, R.; Makhliarchuk, A. HaGRID–HFurthermore, Gesture Recognition Image Dataset. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, Waikoloa, HI, USA, 4–8 January 2024.
6. Pejsa, T.; Pandzic, I.S. State of the Art in Example-Based Motion Synthesis for Virtual Characters in Interactive Applications. In *Computer Graphics Forum*; Blackwell Publishing Ltd.: Oxford, UK, 2020; Volume 29, pp. 202–226. [CrossRef]
7. Geijtenbeek, T.; Pronost, N. Interactive Character Animation Using Simulated Physics: A State-of-the-Art Review. In *Computer Graphics Forum*; Blackwell Publishing Ltd.: Oxford, UK, 2012; Volume 31, pp. 2492–2515. [CrossRef]
8. Karg, M.; Samadani, A.-A.; Gorbet, R.; Kühnlenz, K.; Hoey, J.; Kulić, D. Body Movements for Affective Expression: A Survey of Automatic Recognition and Generation. *IEEE Trans. Affect. Comput.* **2013**, *4*, 341–359. [CrossRef]
9. Wang, X.; Chen, Q.; Wang, W. 3D Human Motion Editing and Synthesis: A Survey. *Comput. Math. Methods Med.* **2014**, *2014*, 104535. [CrossRef] [PubMed]
10. Alemi, O.; Pasquier, P. Machine Learning for Data-Driven Movement Generation: A Review of the State of the Art. *arXiv* **2019**, arXiv:1903.08356.
11. Marsot, M.; Rekik, R.; Wuhrer, S.; Franco, J.S.; Olivier, A.H. Correspondence-free online human motion retargeting. *arXiv* **2023**, arXiv:2302.00556.
12. Victor, L.; Meyer, A.; Bouakaz, S. Pose Metrics: A New Paradigm for Character Motion Edition. *arXiv* **2023**, arXiv:2301.06514.
13. Holden, D.; Saito, J.; Komura, T.; Joyce, T. Learning Motion Manifolds with Convolutional Autoencoders. In *SA '15: SIGGRAPH Asia Technical Briefs*; Association for Computing Machinery: New York, NY, USA, 2015. [CrossRef]
14. Holden, D.; Saito, J.; Komura, T. A Deep Learning Framework for Character Motion Synthesis and Editing. *ACM Trans. Graph.* **2016**, *35*, 138. [CrossRef]
15. Holden, D.; Habibie, I.; Kusajima, I.; Komura, T. Fast Neural Style Transfer for Motion Data. *IEEE Comput. Graph. Appl.* **2017**, *37*, 42–49. [CrossRef]
16. Grassia, F.S. Practical Parameterization of Rotations Using the Exponential Map. *J. Graph. Tools* **1998**, *3*, 29–48. [CrossRef]
17. Pavllo, D.; Grangier, D.; Auli, M. QuaterNet: A Quaternion-based Recurrent Model for Human Motion. In Proceedings of the British Machine Vision Conference, Newcastle, UK, 3–6 September 2018; BMVA Press: London, UK, 2018. Available online: Https://dblp.org/rec/conf/bmvc/PavlloGA18 (accessed on 31 September 2018).
18. Pavllo, D.; Feichtenhofer, C.; Auli, M.; Grangier, D. Modeling Human Motion with Quaternion-Based Neural Networks. *Int. J. Comput. Vis. (IJCV)* **2020**, *128*, 855–872. [CrossRef]
19. Kim, S.; Park, I.; Kwon, S.; Han, J. Motion Retargetting based on Dilated Convolutions and Skeleton-specific Loss Functions. *Comput. Graph. Forum* **2020**, *39*, 497–507. [CrossRef]
20. Aberman, K.; Weng, Y.; Lischinski, D.; Cohen-Or, D.; Chen, B. Unpaired Motion Style Transfer from Video to Animation. *ACM Trans. Graph.* **2020**, *39*, 64. [CrossRef]
21. Aberman, K.; Li, P.; Lischinski, D.; Sorkine-Hornung, O.; Cohen-Or, D.; Chen, B. Skeleton-Aware Networks for Deep Motion Retargeting. *ACM Trans. Graph.* **2020**, *39*, 62. [CrossRef]
22. Lee, K.; Lee, S.; Lee, J. Interactive Character Animation by Learning Multi-Objective Control. *ACM Trans. Graph.* **2018**, *37*, 180. [CrossRef]
23. Starke, S.; Zhao, Y.; Komura, T.; Zaman, K. Local Motion Phases for Learning Multi-Contact Character Movements. *ACM Trans. Graph.* **2020**, *39*, 54. [CrossRef]
24. Holden, D.; Kanoun, O.; Perepichka, M.; Popa, T. Learned Motion Matching. *ACM Trans. Graph.* **2020**, *39*, 53. [CrossRef]
25. Tang, X.; Wang, H.; Hu, B.; Gong, X.; Yi, R.; Kou, Q.; Jin, X. Real-time controllable motion transition for characters. *ACM Trans. Graph.* **2022**, *41*, 1–10. [CrossRef]
26. Duan, Y.; Shi, T.; Zou, Z.; Lin, Y.; Qian, Z.; Zhang, B.; Yuan, Y. Single-shot motion completion with transformer. *arXiv* **2021**, arXiv:2103.00776.
27. Kirac, F.; Kara, Y.E.; Akarun, L. Hierarchically constrained 3D hand pose estimation using regression forests from single frame depth data. *Pattern Recognit. Lett.* **2014**, *50*, 91–100. [CrossRef]
28. Tang, D.; Yu, T.H.; Kim, T.K. Real-time articulated hand pose estimation using semi-supervised transductive regression forests. In Proceedings of the IEEE International Conference on Computer Vision, Sydney, Australia, 1–8 December 2013.
29. Tang, D.; Jin Chang, H.; Tejani, A.; Kim, T.K. Latent regression forest: Structured estimation of 3D articulated hand posture. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014.
30. Chen, T.; Guestrin, C. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, 13–17 August 2016.
31. Oberweger, M.; Wohlhart, P.; Lepetit, V. Hands deep in deep learning for hand pose estimation. *arXiv* **2015**, arXiv:1502.06807.

32. Arimatsu, K.; Mori, H. Evaluation of machine learning techniques for hand pose estimation on handheld device with proximity sensor. In Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems, Honolulu, HI, USA, 25–30 April 2020.
33. Wu, M.Y.; Ting, P.W.; Tang, Y.H.; Chou, E.T.; Fu, L.C. Hand pose estimation in object-interaction based on deep learning for virtual reality applications. *J. Vis. Commun. Image Represent.* **2020**, *70*, 102802. [CrossRef]
34. Ohkawa, T.; Li, Y.-J.; Fu, Q.; Furuta, R.; Kitani, K.M.; Sato, Y. Domain adaptive hand keypoint and pixel localization in the wild. In Proceedings of the European Conference on Computer Vision, Tel Aviv, Israel, 23–27 October 2022; Springer Nature: Cham, Switzerland, 2022.