

Received 9 August 2023, accepted 12 September 2023, date of publication 14 September 2023,  
date of current version 26 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3315656

## RESEARCH ARTICLE

# An IoT-Based Microservice Platform for Virtual Puppetry Performance

HELIN LUO<sup>1,2</sup>, (Member, IEEE), YI-BING LIN<sup>1,3,4,5,6,7</sup>, (Fellow, IEEE),  
CHEN-CHI LIAO<sup>3</sup>, AND YUNG-HUI HUANG<sup>3</sup>

<sup>1</sup>Research Center for Information Technology Innovation, Academia Sinica, Taipei 115, Taiwan

<sup>2</sup>Graduate Institute of Animation and Film Art, Tainan National University of the Arts, Tainan 720, Taiwan

<sup>3</sup>Department of Computer Science and Engineering, National Yang Ming Chiao Tung University, Hsinchu 300, Taiwan

<sup>4</sup>College of Humanities and Sciences, China Medical University, Taichung 406, Taiwan

<sup>5</sup>Miin Wu School of Computing, National Cheng Kung University, Tainan 701, Taiwan

<sup>6</sup>Department of Computer Science and Information Engineering, Asia University, Taichung 413, Taiwan

<sup>7</sup>College of Artificial Intelligence, National Yang Ming Chiao Tung University, Tainan 711, Taiwan

Corresponding author: Yi-Bing Lin (liny@nctu.edu.tw)

This work was supported in part by the National Science and Technology Council (NSTC) under Grant 110-2622-8-A49-022, Grant NSTC112-2221-E-A49-049, Grant 112-2420-H-A49-001, Grant 112-2221-E-369-001, and Grant 111-2221-E-369-001; in part by the Miin Wu School of Computing, National Cheng Kung University (NCKU), AiQ Smart Clothing Inc.; and in part by the Research Center for Information Technology Innovation, Academia Sinica.

**ABSTRACT** This study proposes a web-based real-time remote-controlled virtual puppetry (avatar) performance platform called AvatarTalk to combine traditional Taiwanese puppetry, the Internet of Things (IoT), and 3D avatar models. AvatarTalk enables puppeteers to control avatar puppets on the web using either motion capture gloves or camera-based image recognition. This transformative performance approach not only facilitates remote performances and multi-screen presentations across various devices but also introduces a fresh gesture interpretation technique for dance artists. AvatarTalk supports a mechanism that can accommodate new control and puppet devices from other approaches through the IoT-based microservice concept. We develop a calibration procedure to enable the accurate capture of hand gestures to manipulate the movements of virtual puppets, empowering them to perform fundamental traditional puppetry poses such as nodding, bowing, and synchronized hand movements. We have conducted experiments to show the accuracy of AvatarTalk calibration. Our study indicates that AvatarTalk can almost detect the right gestures (98.75%-100% recall) and very seldom mistake the wrong gestures (90.8%-100% precision). Additionally, we also provide the mechanism to measure the delays of controlling the puppets. An analytic model is proposed to design the delay times of the messages between the control device of a puppeteer and the AvatarTalk server. In the current AvatarTalk implementation, if the message delay does not exceed 0.1 seconds, four puppeteers can synchronize their actions if the elapsed time between two actions of a puppeteer is longer than 0.3 second.

**INDEX TERMS** Arts and humanities, avatar, interaction paradigms, interactive art performance, Internet of Things (IoT), microservice, puppet.

## I. INTRODUCTION

Puppetry is one of Taiwan's cultures, commonly seen in traditional festivals or religious occasions. Puppeteers wear real puppet dolls on their hands and interpret various storylines

The associate editor coordinating the review of this manuscript and approving it for publication was Wenbing Zhao<sup>1</sup>.

through gestures, music, and dialogues. Therefore, it is also known as "palm puppetry." Puppetry has experienced a gradual decline in recent years. To preserve this invaluable cultural heritage, we propose AvatarTalk, an innovative approach that merges information and communications technology with traditional puppetry. Through this integration, we aim to create a new and captivating form of performance while ensuring

the longevity of this art form. Based on an Internet of Things (IoT) application development platform [1], we make non-trivial extension of our previous work PuppetTalk [2], and propose AvatarTalk that utilizes cameras or smart gloves for motion capture. It allows puppeteers to control virtual or real puppets using gestures or body movements. This enables puppeteers to perform using 3D virtual puppets on a web page. This performance method can be used for remote performances, multi-screen presentations across devices, and can also be combined with dance performances, providing a new interpretation of performance for dance artists.

AvatarTalk follows an IoT-based microservice approach to create the puppetry applications. To achieve the microservice goal, there are two requirements:

**Microservice Requirement 1 (MS R1):** The microservices must be capable of independent deployment.

**Microservice Requirement 2 (MS R2):** An integration mechanism exists to facilitate the arbitrary reuse of microservices while maintaining loose coupling, enabling the creation of applications.

With MS R1 and R2, the applications developed in the platform are reusable, scalable and maintainable. AvatarTalk uses the device model concept to accommodate the IoT-based control mechanisms (smart gloves, MediaPipe [3] [4], etc.) and the puppets (either virtual avatars or real robots), and implement them as microservices, which satisfies MS R1. AvatarTalk uses the device feature concept to arbitrarily link multiple microservices to create an application, which satisfies MS R2. Details of device model and device feature will be elaborated in Section III.

The paper is organized as follows: Section II provides an overview of the literature, discussing the digitization of traditional puppets and the relevant applications of avatar models; Section III describes the AvatarTalk architecture; Section IV elaborates on remote motion control including smart gloves and MediaPipe; Section V designs the Avatar device; Section VI showcases the Avatar graphical user interface (GUI) and explain the operational flow for controlling the virtual puppets; Section VII elaborates on the microservice design issues, the calibration performance and the synchronization problem that may occur in remote control of puppets.

## II. RELATED WORK

In addition to traditional temple performances, puppetry has further extended to become an artistic form of television and film, or performances that incorporate technological methods. One example of combining technology with puppetry is the real-time remote mechanical puppet control platform called PuppetTalk [2], developed using the IoT technology. This study uses the smart glove to control the mechanical puppets. When the mechanical puppets receive gesture data through IoT, they will control the rotation angles of the motors according to the corresponding finger positions. In addition, the system also provides various input devices such as mobile phones to trigger pre-set actions for additional

control of the mechanical puppets. Through the microservice approach, we have seamlessly integrated PuppetTalk as part of AvatarTalk.

The Leap Motion device is used in a study [5] to capture users' hand movements to control virtual puppetry created in Unity game engine. Leap Motion is a hardware specifically designed for hand tracking. It uses two infrared cameras to track hand movements at a speed of over 200 frames per second. In this study, the virtual puppet is divided into four parts: body, head, and left and right arms. The palm is used to control the body, the index finger controls the head, and the thumb and little finger are used to control the two arms of the puppet. No measurement data and analysis was provided in this work.

Literature [6] and [7] explore the concept of a puppetry cloud theater called Virtual Reality Peripheral Network (VRPN). VRPN supports multiple users to share the same virtual space over the network from their personal computers and participate in controlling and performing puppetry together. The Leap Motion device [8] is also used to capture hand movements, with finger controls corresponding to the virtual puppet's parts, similar to literature [5]. The study combines VR technology to provide different performance experiences and independent perspectives for each user. No measurement data and analysis was provided in this work.

There have also been related studies on virtual puppets in other traditional puppetry forms. Literature [9] was inspired by traditional wooden puppetry and developed a virtual puppet system applicable to film animation. The system, built on the Unity game engine, utilizes Leap Motion for hand recognition to support hand control of virtual puppets. This system provides options for controlling the expressions and body of virtual puppets. In terms of expression control, users can use their fingers to control the facial expressions of the virtual puppet. The mapping between fingers and facial features is determined through biological analysis and experimental statistics. For example, the thumb can control the mouth shape, the index finger controls the eyelids, the middle finger and ring finger correspond to the eyebrows, and the little finger is used to control the direction of the eyes. AvatarTalk references the finger angle ranges proposed in this study. On the other hand, message delay analyzed by Avatar is not found in this study.

An Indonesian puppetry approach [10] discusses how to track gestures using the Microsoft Kinect depth sensor and applies the density-based spatial clustering of applications with noise method to cluster and classify hand gestures, distinguishing between left and right hand movements and recognizing three predefined gestures. The system triggers animations of virtual puppets based on the gestures, and the performance takes place on a stage with backgrounds. The precision, accuracy and recall are in the ranges of 94%-100%, 91%-100% and 94%-100%. As shown in Table 5, AvatarTalk performs better in some of these measures.

There have also been many related studies on Chinese shadow puppetry. An interactive animation generation framework is proposed [11] to improve the reusability, scalability, and modularity of animation production. The framework generates animations based on the mapping between input/output modes, animation data, and gesture combinations. The gestures describe information such as hand movement speed and angle. In this study, the author uses the Leap Motion device to demonstrate how the animation generation framework assists in controlling virtual shadow puppets. When the system recognizes predefined gestures, it triggers pre-recorded animations of the virtual puppets based on the animation generation framework.

The study in [12] explored various shadow puppetry character types and develops parametric blueprints for human, bipedal, quadrupedal, and serpent-like creatures. These templates enable the system to generate 2D shadow puppets in diverse shapes and styles to suit user preferences. Unlike previous research relying on animation controls, this study employs finger coordinates and bending angles to simulate an upper-body puppet control lever.

Additionally, an interactive art piece influenced by Chinese shadow puppetry is presented [13]. It utilizes Microsoft Kinect to track the audience's body skeleton, directing the shadow puppets' actions projected on a screen for engagement. Unity game engine powers this single-user interactive system, making it a potential new feature for AvatarTalk.

In a study [14] akin to Taiwanese traditional puppetry, a gesture-based multi-user digital storytelling system was created using VR and Leap Motion. It's aimed at aiding children in storytelling and performance. The authors used Photon Cloud and Unity tech, enabling synchronous participation and interaction among multiple players. The study demonstrated this with fables, letting children control animal models in the story through gestures and add narration guided by a teacher.

In a distinct study on non-gesture control of virtual models [15], the idea is presented that humans can convey or mimic emotions through these models. The primary use case is in pediatric hospitals, serving as an adjunct tool for psychologists to assess emotions and states by enabling young patients to interact with virtual models using the pen.

In addition to using specialized hardware such as Leap Motion or Microsoft Kinect for motion capture and recognition, there are also studies that utilize MediaPipe for image recognition and tracking. AvatarTalk uses MediaPipe instead of Leap Motion for three reasons. Firstly, using Leap Motion requires the purchase of additional hardware, whereas MediaPipe can utilize ordinary cameras (such as smartphones) for recognition, resulting in lower costs. Secondly, Leap Motion can only recognize up to two hands, while MediaPipe can recognize more than two hands. Lastly, Leap Motion has a limited capture range, only spanning from 25 millimeters to 600 millimeters above the hardware's top surface and within a field of view of 150 degrees for successful recognition. In contrast, MediaPipe can capture hand movements at a

distance of several meters, but this comes at the cost of lower accuracy. Therefore, for hand image recognition, we choose to use MediaPipe in AvatarTalk.

The study in [16] proposed Puppeteer, a Unity-based system enabling players to control models through a combination of gestures and upper body motions. Dual cameras and MediaPipe technology captured body keypoints, triggering animations based on specific poses and gestures. The system demonstrated its capabilities through three different game scenarios. A similar experience in AvatarTalk employs smart gloves to tackle this challenge.

In reference [17], real-time virtual model animation generation technology based on a single camera was discussed. Regarding the control of virtual models, different models may have varying skeletal structures, which can lead to different operations even when performing the same action. To address this issue, the authors standardized the skeleton using the coordinate system of the VRM model format [18] [19]. In terms of control methods, the authors used MediaPipe's BlazePose to detect body keypoints, converted the results into angles, and mapped them to the model's skeleton for control, generating real-time virtual model animations. AvatarTalk adapts a similar approach using the VRM model format.

The above previous studies "hardwire" puppetry control and cannot transparently add new control/puppet components. To resolve this issue, CATtalk [20] proposed a low-code mechanism in the PuppetTalk architecture [2]. AvatarTalk nontrivially extends CATtalk's low-code mechanism to satisfy the microservice requirements. AvatarTalk is designed as an IoT-based microservice platform for controlling virtual models, providing users with many control modes, and illustrate two in this paper: smart gloves and MediaPipe.

In contrast to the previous studies (including CATtalk), this paper further describes the methods of gesture data calibration and control of virtual puppetry parts. In terms of specialized hardware for capturing, smart gloves were chosen to overcome the limitations and inaccuracies caused by the distance restrictions and obstructions when using Leap Motion. Smart gloves can accurately capture subtle hand movements compared to Microsoft Kinect. Furthermore, unlike most of the systems built using Unity [5] [9] [13] [14] [16] [19] or Unreal [12] in the aforementioned studies, the control platform in this research is web-based, eliminating the need for users to download additional software for control.

The previous studies with remote control do not consider the synchronization of puppets except for PuppetTalk and CATtalk [20]. AvatarTalk extends the two-puppeteer communication delay analysis of PuppetTalk and CATtalk to derive the conditions when the puppets/avatars can be synchronized with more than two remote puppeteers.

### III. AVATARTALK ARCHITECTURE

AvatarTalk enables message exchange between various IoT devices, and use them to manipulate the virtual puppets. Figure 1 illustrates an example to show how IoT devices

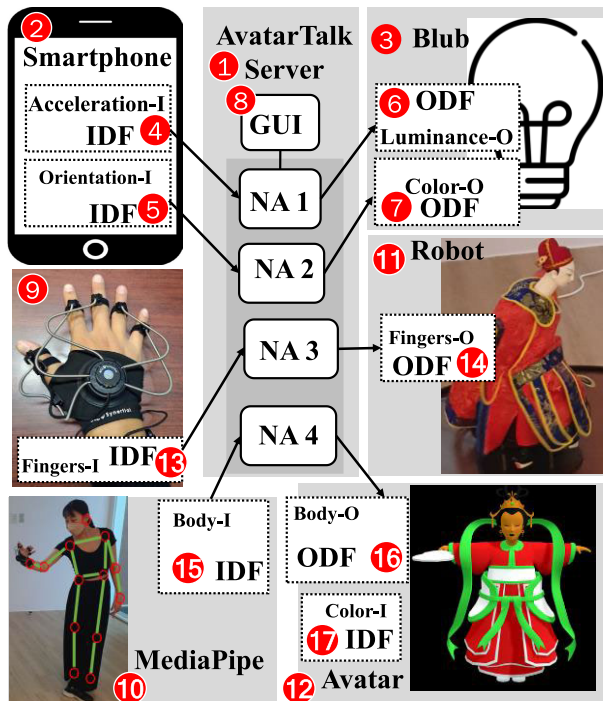


FIGURE 1. The AvatarTalk configuration.

connected to the AvatarTalk server (Figure 1 (1)). AvatarTalk defines IoT-based microservices called Device Models (DM) to describe real-world IoT devices. For example, the IoT devices Smartphone (Figure 1 (2)) and Bulb (Figure 1 (3)) are represented as DMs in the AvatarTalk server.

Each DM includes one or more Device Features (DFs). Based on their functionalities, the DFs are further categorized into Input Device Features (IDF; sensors or controls) with the names appended with “-I” and Output Device Features (ODF; actuators) with the names appended with “-O”. For example, Smartphone has two IDFs: Acceleration-I and Orientation-I (Figure 1 (3) and (5)), while Bulb has two ODFs: Luminance-O and Color-O (Figure 1 (6) and (7)). In AvatarTalk, the DMs are realized as individual IoT devices. Since IoT devices are independent of each other, DMs satisfy the microservice requirement 1 (MS R1).

The configuration in Figure 1 is established using the web-based AvatarTalk Graphical User Interface (GUI; Figure 1 (8)), illustrated in Figure 2. This GUI is a direct extension of the GUI in [20]. It allows users to create custom DMs and DFs, and configure DMs, DFs, their connections, and corresponding functions through the AvatarTalk server.

By connecting an IDF to an ODF through the “join links” in the GUI, The interaction between the IoT devices is established with input and output capabilities. When the Acceleration-I IDF generates a new value, Smartphone notifies the Network Applications NA1 in Figure 1 (corresponding to the “Join 1” link in Figure 2) for processing and sends the result to the Luminance-O ODF of Bulb. Then, shaking the smartphone would make the light bulb brighter.

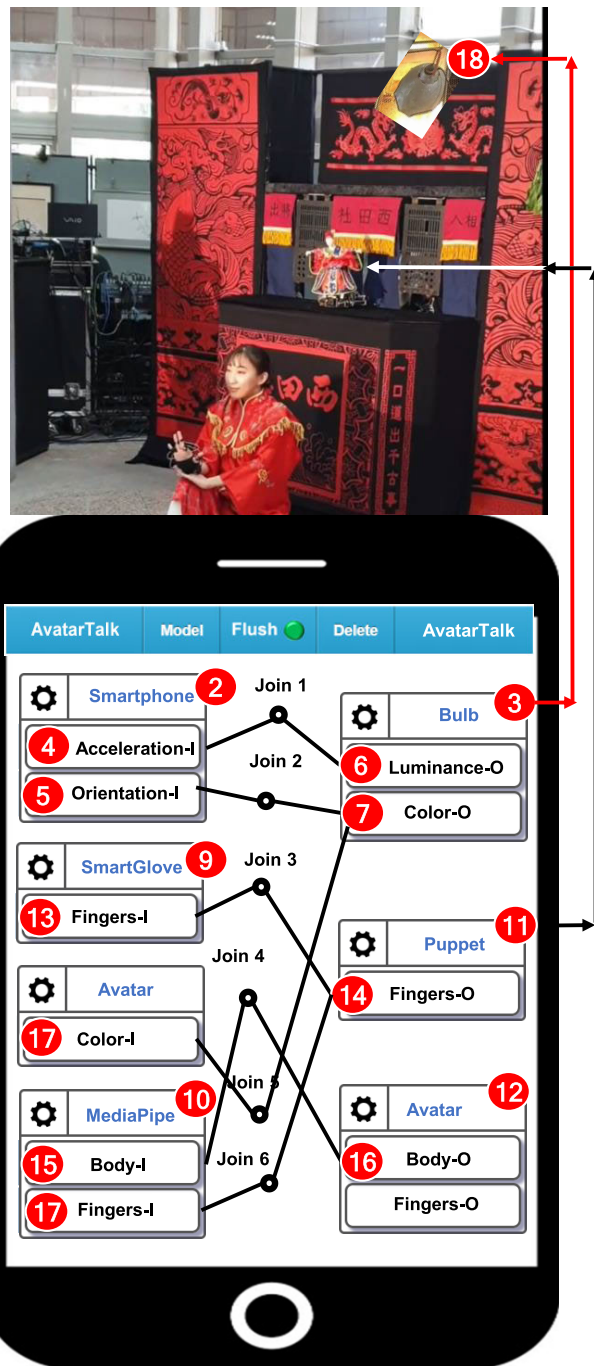


FIGURE 2. The AvatarTalk GUI.

Similarly, by connecting Orientation-I to Color-O, NA2 (“Join 2” in Figure 2) processes the angles generated by rotating the smartphone and sends the results to Bulb to change its color. Since IDF and ODF are independent of each other, predefined modules can be reused to create NAs for different DMs with the same IDFs or ODFs. The network applications are automatically generated by the AvatarTalk server [1] [20]. We note that the above light control application is used in the stage for the puppet show. Multiple

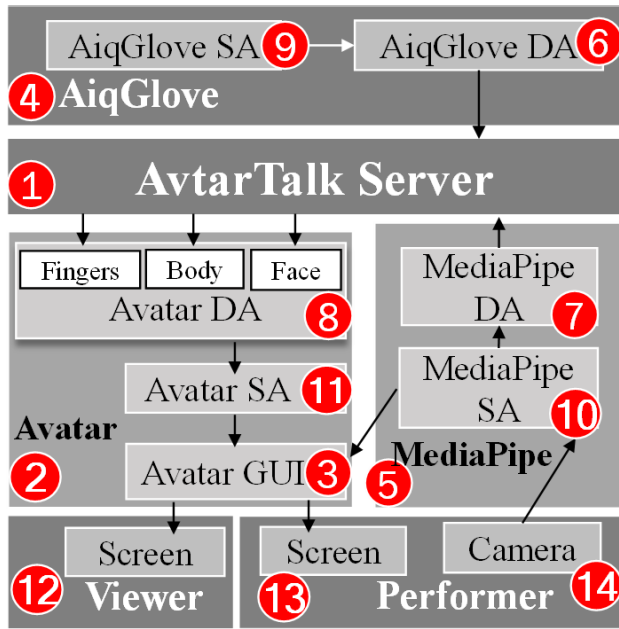


FIGURE 3. AvatarTalk software architecture.

smartphones are allowed to map to Smartphone DM, allowing the audience to control the stage lights (Figure 2 (18)) through “voting” [21].

In the current implementation, AvatarTalk supports AiQ smart gloves (Figure 1 (9)) and MediaPipe (Figure 1 (10)) as input devices and utilizes the AvatarTalk platform for real-time remote control of puppet robots (Figure 1 (11)) and virtual avatar (Figure 1 (12)).

Fig. 3 illustrates the software architecture of AvatarTalk. The AvatarTalk server (Figure 3 (1)) supports interaction between the virtual Avatars (Figure 3 (2)) with the AiQ gloves (Figure 3 (3)) and MediaPipe (Figure 3 (5)) through the interface described below.

The AiqGlove SA (Figure 3 (9)) obtains the sensor data to calculate finger coordinates and wrist angles. The MediaPipe SA (Figure 3 (10)) is responsible for image recognition and data computation. The Avatar SA (Figure 3 (11)) uses the received coordinates to drive the avatar model under control. The behavior of the avatar is shown in the browser of the user who is a viewer (an audience member; Figure 3 (12)) or a puppeteer (Figure 3 (13)).

If the user is a puppeteer, there is a feature to enable MediaPipe recognition in the Avatar GUI (Figure 3 (3)) using the camera (Figure 3 (14)). The recognition results will be processed by MediaPipe SA and then sent to the AvatarTalk Server via MediaPipe DA. Whether it is AiQ Smart Glove, MediaPipe, or any other future body motion capture technologies that may be added, the data will be sent from their IDFs in their DA to the ODFs of the Avatar DA.

In AvatarTalk, we only need to define the IDFs and the ODFs of a device model, and the corresponding DA is automatically generated by AvatarTalk. This design aims

to improve the convenience of subsequent code maintenance and expansion. Details of automatic code generation is described in Section VII-A.

IV. REMOTE MOTION CONTROL

Remote puppet control requires the use of hand motion capture technology to recognize the changes in hand gestures when controlling the puppets. AvatarTalk supports two types of gesture control of puppetry: smart gloves [2] (Figure 3 (4)) and camera-based image tracking [8] (Figure 3 (5)). Camera-based image tracking techniques (e.g., Leap Motion or MediaPipe) capture gestures using the machine learning models. This approach only requires a single camera to achieve hand tracking, pose tracking, and classification functions. Compared to smart glove, this approach has lower costs as it only requires a camera to capture images for recognition, while also avoiding hygiene issues associated with wearing motion capture gloves shared among multiple people. Camera-based image tracking provides various recognition solutions through the Hands, Pose, and Holistic recognition functionalities.

A drawback of the camera-based image tracking is the inability to perceive objects that are obscured by fingers. For instance, when one finger is positioned over another, the device struggles to see through and accurately detect the hidden finger. Additionally, the close proximity of adjacent fingers creates difficulties for the cameras in recognizing and distinguishing each finger as separate entities, possibly leading to recognition issues.

When the puppeteer’s position may frequently change during the performance (for example, a dancer is controlling the puppets), smart gloves for motion capture ensure that the results of moving gesture recognition are not affected by the puppeteer’s position changes or object occlusion. The smart gloves also allow gesture capture while the puppeteer is wearing the puppets on the gloves. Additionally, compared to camera-based image tracking, smart gloves can avoid interference from changes in ambient lighting conditions and expand the puppeteer’s range of movement. They can recognize gestures without being limited by the camera’s field of view.

A. AiQ SMART GLOVE: MOTION CAPTURE GLOVES

We use AiQ Gloves as a smart glove example (Figure 4 (a)). AiQ gloves use Inertial Measurement Units (IMUs), including accelerometers, gyroscopes, and magnetometers, to capture the puppeteer’s gestures. The gesture data is then transmitted wirelessly via Wi-Fi to the AvatarTalk server for processing. The PuppetTalk study [2] also considered an inexpensive mechanical glove with one sensing point per finger (Figure 4 (b)), and concluded that the glove is too simple to capture the required gestures. As we will see in Section IV.D, AvatarTalk can utilize this simple glove to yield good performance as the AiQ glove.

The AiqGlove DA includes the compound Fingers-I IDF (Figure 1 (13)) consisting of six IDFs in the format

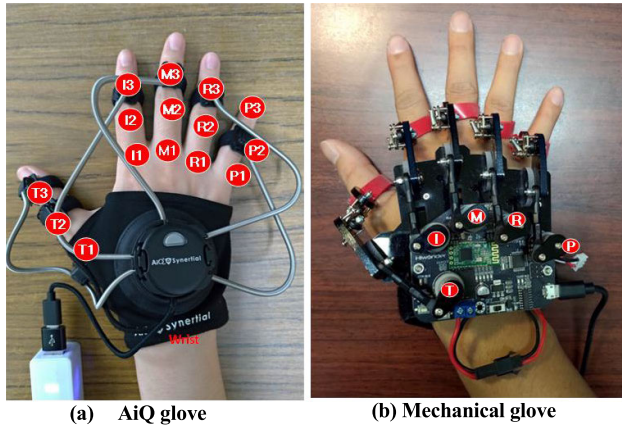


FIGURE 4. Smart gloves and the recognized 3D finger joint coordinates.

$FCoord-I=(F1, F2, F3)$ , where  $F=$  Thumb (T), Index (I), Middle (M), Ring (R) or Pinky (P). For a finger  $F$ , the sensors report the values for three joints: metacarpophalangeal joint (F1), proximal interphalangeal joint (F2), and distal interphalangeal joint (F3). Therefore, Fingers-I includes ThumbCoord-I =  $(a_{T1}, a_{T2}, a_{T3})$ , IndexCoord-I =  $(a_{I1}, a_{I2}, a_{I3})$ , MiddleCoord-I =  $(a_{M1}, a_{M2}, a_{M3})$ , RingCoord-I =  $(a_{R1}, a_{R2}, a_{R3})$ , PinkyCoord-I =  $(a_{P1}, a_{P2}, a_{P3})$ , and WristAngle-I. Note that  $a_F$  is a vector representing the 3-dimension coordinate of a finger joint  $F$ . For example,  $a_{T1} = (a_{T1,x}, a_{T2,y}, a_{T3,z})$ . These six IDFs contain 3D data representing finger coordinates and wrist angles, as well as information about the controlling hand (left or right) and control mode.

The AiQGlove SA obtain 3D coordinates of the finger joints from the motion sensors, represented by the red dots in Figure 4. The AiQGlove SA performs two-stage glove calibration. The first-stage calibration obtains accurate 3D finger joint coordinates to ensure the correct recognition of gestures. Before this calibration, inaccurate 3D finger joint coordinates are observed (Figure 5 (a)). After calibration, we obtain accurate mappings (Figure 5 (b)). The second-stage calibration is performed in the avatar device (Figure 3 (2)) to establish the mapping between the gestures and the avatar models (to be elaborated in Section VI-C).

**B. MediaPipe: CAMERA-BASED IMAGE TRACKING**

We use MediaPipe developed by Google as an example of camera-based image tracking. MediaPipe Hands is a machine learning-based hand tracking technique that can determine 21 3D coordinates of the hands in each frame of the camera-captured image or video [3]. The joint notation of MediaPipe in Figure 6 is different from AiQ glove in Figure 4 (a). Both of them are mapped to the AvatarTalk finger notation  $F = (F1, F2, F3)$ ; see also the big red circles in Figure 6 for the mapping.

In AvatarTalk, the video image is streamed through the camera (Figure 8 (2)) to the MediaPipe SA (Figure 8 (1)) using the streaming() function.

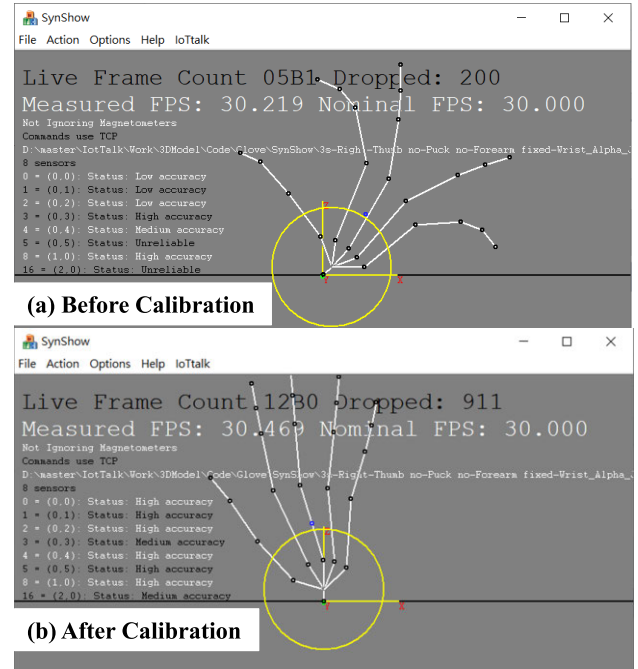


FIGURE 5. First-stage glove calibration.

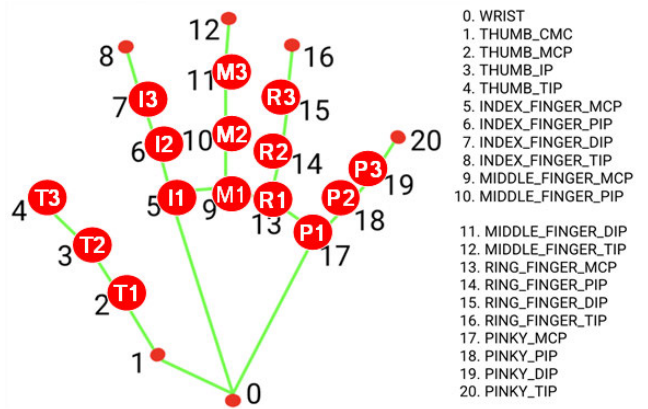


FIGURE 6. MediaPipe hand coordinates and the mappings to the AvatarTalk finger notation.

After capturing the 3D coordinates of the hands, the MediaPipe device (Figure 1 (10)) obtains the finger coordinates and the calculated wrist angle and then extracts the finger joint coordinates for avatar model control. However, since MediaPipe uses a single camera for image recognition, the z-axis data in the 3D coordinates is a predicted result of the model. Therefore, compared to motion capture gloves, the control effectiveness of MediaPipe Hands may be lower.

MediaPipe Pose is also a machine learning-based human body tracking technique that can determine the 33 3D coordinates of the body in each frame of the camera-captured image or video [4]. We use the open-source library Kalidokit to convert the recognized body coordinate data into joint angles for model manipulation. Based on the 3D coordinates obtained from MediaPipe Pose, Kalidokit calculates the

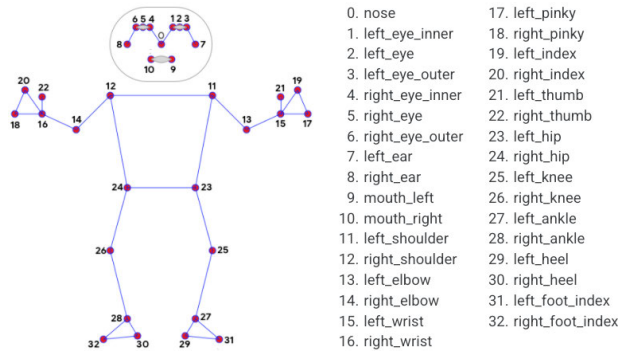


FIGURE 7. MediaPipe body coordinates.

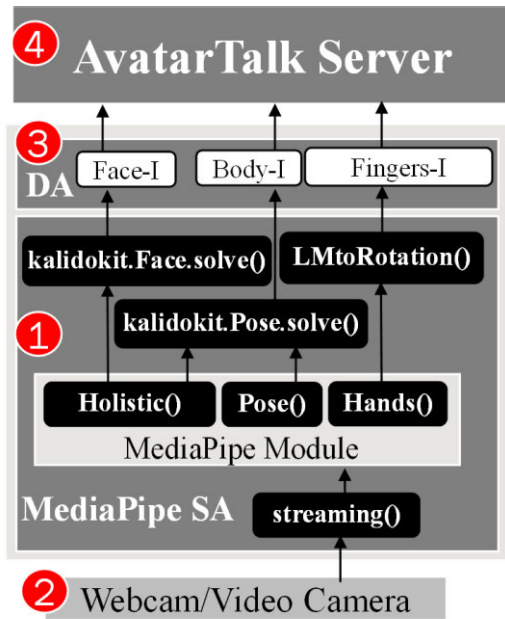


FIGURE 8. The MediaPipe device.

rotation angles of body joints shown in Figure 7. Additionally, we establish a reference coordinate system based on the hip and leg coordinates of a person standing in the camera or video frame. This reference system is used for the relative positioning of the model, and the coordinates of the head, hip, and legs are stored in the Avatar DB.

MediaPipe Holistic also uses machine learning to recognize facial, body, and hand poses from each frame of the camera-captured image or video. The body recognition data from MediaPipe Holistic is the same as the output from MediaPipe Pose, and we use kalidokit to convert the data into body joint angles. As for facial recognition, through kalidokit, we calculate the 478 3D coordinates obtained from facial recognition and convert the data into expression control weights for the VRM format model to simulate facial expressions.

When Hands recognition is selected, LMtoRotation() calculates wrist angles and coordinates, and the finger data is sent to the Finger DA (Figure 8 (3)). When Pose recognition is

selected, kalidokit.Pose.solve() converts the recognized body coordinates into body joint angles and sends them to the Body DA. For Holistic recognition, kalidokit.Pose.solve() and kalidokit.Face.solve() are used separately, and the corresponding results are sent to the Body DA and the Face DA, respectively. MediaPipe DA utilizes Fingers-I along with Face-I and Body-I, to transmit data to the AvatarTalk Server (Figure 8 (4)).

### C. CONSISTENT COORDINATE WITH RESPECT TO THE WRIST

In AvatarTalk, a puppeteer can be a dancer who not only changes hand gestures but also perform dance movements during the performance. In this scenario, the avatar model's actions should remain consistent when the puppeteer rotates the wrist while keeping the same gesture. However, in practice, the finger joint coordinates change with the rotation of the wrist, which affects the model's movements. To ensure that the coordinates and model remain unchanged when the wrist is rotated with the same gesture, we use the inverse rotation method to obtain the consistent finger joint coordinates when the wrist rotates. Consider a finger joint of  $F$ , where  $F$  is  $T$ ,  $M$ , or  $I$  as shown in Figure 4. Assuming the wrist of the right hand is facing the screen at the beginning of the first calibration (see Section VI-C), where the wrist's angle  $\theta$  with respect to the  $x$ ,  $y$ , and  $z$  axes are  $(\theta_x, \theta_y, \theta_z)$ . After the completion of the first calibration, the wrist's angles with respect to the axes are  $(\theta_x^0, \theta_y^0, \theta_z^0)$ , respectively. We define the angle differences after wrist rotation as  $(\Delta\theta_x, \Delta\theta_y, \Delta\theta_z) = (\theta_x - \theta_x^0, \theta_y - \theta_y^0, \theta_z - \theta_z^0)$ .

When the wrist angle is  $(\theta_x^0, \theta_y^0, \theta_z^0)$ , we need to rotate the finger joint coordinates back to the position with respect to the wrist angle  $(\theta_x, \theta_y, \theta_z)$ , we substitute these angle values into the rotation Eq. (1).

$$\begin{aligned}
 R_x(\Delta\theta_x) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \Delta\theta_x & -\sin \Delta\theta_x \\ 0 & \sin \Delta\theta_x & \cos \Delta\theta_x \end{bmatrix}, \\
 R_y(\Delta\theta_y) &= \begin{bmatrix} \cos \Delta\theta_y & 0 & \sin \Delta\theta_y \\ 0 & 1 & 0 \\ -\sin \Delta\theta_y & 0 & \cos \Delta\theta_y \end{bmatrix}, \\
 R_z(\theta_z) &= \begin{bmatrix} \cos \Delta\theta_z & -\sin \Delta\theta_z & 0 \\ \sin \Delta\theta_z & \cos \Delta\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1)
 \end{aligned}$$

From Eq. (1), define

$$R = R_z(\Delta\theta_z) R_x(\Delta\theta_x) R_y(\Delta\theta_y) \quad (2)$$

Let  $a_F^0$  be the coordinate for finger joint  $F$  sent from the smart glove to the avatar device. Eq. (2) is used to derive the consistent coordinate  $a_F$  as

$$a_F = R^{-1} \times a_F^0 \quad (3)$$

### V. THE AVATAR DEVICE

In the Avatar DA, the compound Fingers-O (Figure 9 (1)) includes the ODFs that one-to-one correspond to the IDF





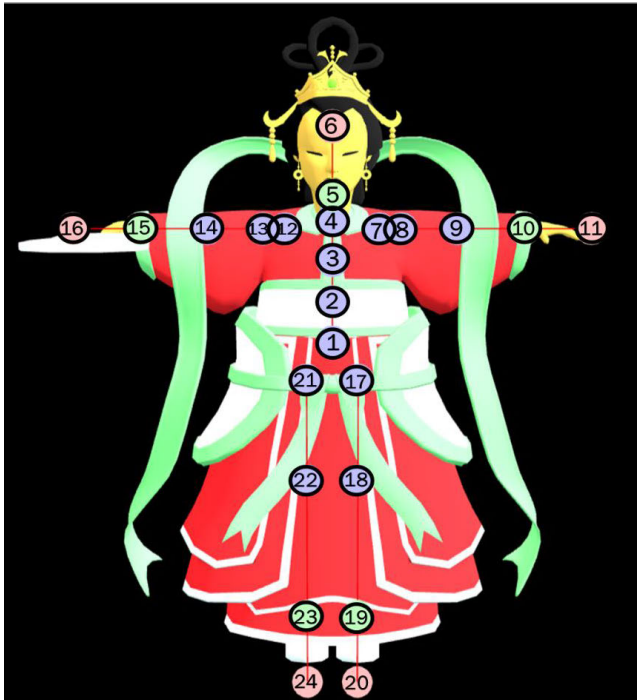


FIGURE 10. The IK chains for an avatar.

is a model element that contains both the mesh and the skeleton. It will be used to control the model in the subsequent IK calculations. The `loadAvatars()` function automatically adds the IK target points (the pink dots in Table 1) to the loaded model's skeleton.

The function `changeAvatar()` is executed during the initialization phase to display the default avatar model on the screen (Figure 9 (6)) and allows users to switch between avatars later. During system initialization, `loadAvatars()` has loaded a list of models from the Avatar DB. If the user selects a new model through the Avatar Selection dropdown menu (Figure 9 (7)), `changeAvatar()` displays the selected model. Then CCDIKSolver for this avatar model is initialized to update the skeleton information of the model in the IK settings.

The `initializeCCDIK()` function initializes CCDIKSolver [23] required for finger control, which solves IK Problem with Cyclic Coordinate Descent (CCD). CCDIKSolver helps calculate the poses that bring the model's limbs to the target positions, enabling more natural control and interaction. When setting up CCDIKSolver, we need to define an IK chain consisting of the model's limb bones. CCDIKSolver utilizes the CCD algorithm to iteratively calculate and improve the poses of the IK chain. The ODFs corresponding to the IK chains in Fingers-O are mapped to the IK target points using Eq. (4) to be elaborated in Section V-B. The accuracy of the algorithm's result improves with a higher number of iterations. To ensure convergence, the calculations stop once the specified iteration count is reached. It then rotates the end effector by a certain angle to move it closer to

TABLE 2. Rotation limits of the joints.

Joints and the IK points in Figure 10	Rotation limit (x-axis)	Rotation limit (y-axis)	Rotation limit (z-axis)
Spine (IK point 1)	$[-45^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$	$[-30^\circ, 25^\circ]$
Chest (IK point 2)	$[-10^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$	$[0^\circ, 10^\circ]$
Upper Chest (IK point 3)	$[-5^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$
Neck (IK point 4)	$[-45^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$
Left Shoulder (IK point 7)	$[0^\circ, 0^\circ]$	$[-20^\circ, 20^\circ]$	$[-20^\circ, 20^\circ]$
Right Shoulder (IK point 12)	$[0^\circ, 0^\circ]$	$[-20^\circ, 20^\circ]$	$[-20^\circ, 20^\circ]$
Left Upper Arm (IK point 8)	$[-60^\circ, 60^\circ]$	$[-150^\circ, 0^\circ]$	$[-70^\circ, 90^\circ]$
Right Upper Arm (IK point 13)	$[-60^\circ, 60^\circ]$	$[0^\circ, 150^\circ]$	$[-90^\circ, 70^\circ]$
Left Lower Arm (IK point 9)	$[0^\circ, 0^\circ]$	$[-160^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$
Right Lower Arm (IK point 14)	$[0^\circ, 0^\circ]$	$[0^\circ, 160^\circ]$	$[0^\circ, 0^\circ]$
Left Upper Leg (IK point 17)	$[-50^\circ, 95^\circ]$	$[0^\circ, 0^\circ]$	$[-65^\circ, 20^\circ]$
Right Upper Leg (IK point 21)	$[-50^\circ, 95^\circ]$	$[0^\circ, 0^\circ]$	$[-20^\circ, 65^\circ]$
Left Lower Leg (IK point 18)	$[-130^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$
Right Lower Leg (IK point 22)	$[-130^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$	$[0^\circ, 0^\circ]$

the target point. This process continues until the end effector reaches the target point or the maximum iteration count is reached. The details are given in Section V-B.

CCDIKSolver avoids unexpected rotations and movements of IK chains. In virtual puppetry, due to the limited ways of expressing hand gestures, we only need to establish five IK chains, each controlled by a finger. The IK chain for the head consists of the spine ((1) in Figure 10), chest (2), upper chest (3), neck (4), head (5) and HeadIK ((6); the head IK target point). The IK chain for the left hand includes the left shoulder (7), left upper arm (8), left lower arm (9), left hand (10), and LeftHandIK ((11); the left hand IK target point). The IK chain for the right hand consists of the right shoulder (12), right upper arm (13), right lower arm (14), right hand (15), and RightHandIK ((16); the right hand IK target point). The IK chain for the left leg includes the left upper leg (17), left lower leg (18), left foot (19), and LeftFootIK ((20); the left foot IK target point). The IK chain for the right leg consists of the right upper leg (21), right lower leg (22), right foot (23), and RightFootIK ((24); the right foot IK target point). In the avatar model, the lower bound and upper bound angles of joint rotation have been adjusted based on the data obtained from references [24] and [25]. They have been modified according to the model's coordinate system as shown in Table 2. When CCDIKSolver moves the end point to the target point, the rotation of the purple dot joints in an IK chain is limited by these angles.

In `loadAvatars()`, we have automatically added the IK target points to the model. In this function, we incorporate these target points into their respective IK chains. For each joint in the avatar model, we have adjusted the maximum and minimum rotation angles based on Table 2.

## B. AVATAR MOVEMENT

When Fingers-O (Figure 9 (3)) receives the raw gesture data  $a_F^0$  from the AvatarTalk server, `updateHandData()` uses Eq. (3) in Section IV-C to reverse rotate the finger joint coordinates

$\mathbf{a}_F^0$  to obtain the consistent coordinate  $\mathbf{a}_F$ , and stores  $\mathbf{a}_F$  in the Avatar DB. Then `fingerControl()` processes  $\mathbf{a}_F$  to manipulate the avatar model using fingers, similar to traditional puppetry. For the IK target point of an avatar's IK chain  $B$ , where  $B$  can be  $H$  (HeadIK),  $R$  (RightHandIK) or  $L$  (LeftHandIK) mapped from Fingers-O, we define the lower bound coordinate  $\mathbf{p}_B^l = (p_{B,x}^l, p_{B,y}^l, p_{B,z}^l)$  and the upper bound coordinate  $\mathbf{p}_B^u = (p_{B,x}^u, p_{B,y}^u, p_{B,z}^u)$  for the range of movement. These bounds are automatically obtained from the skeleton data of the model after loading, and the model's reference pose is set accordingly. There is no need for manual adjustments for different avatar models. For finger  $F$  in the  $x$ -axis (which is mapped to the target point of the avatar's IK chain  $B$ ), `fingerControl()` produces the IK target point  $p_{B,x}$  by mapping  $a_{F,x}$  from the range  $[a_{F,x}^l, a_{F,x}^u]$  to the range  $[p_{B,x}^l, p_{B,x}^u]$ :

$$p_{B,x} = \left( \frac{a_{F,x} - a_{F,x}^l}{a_{F,x}^u - a_{F,x}^l} \right) \times (p_{B,x}^u - p_{B,x}^l) + p_{B,x}^l \quad (4)$$

The results produced by Eq. (4) are sent to `update()` to change the positions of the IK target points in a manner that resembles the motion of objects in reality. We do not directly update the coordinates of the IK target points. Instead, we employ Eq. (5), a linear damping function to gradually change the coordinates of the IK target points along the  $x$ ,  $y$ , and  $z$  axes, dividing the distance between the previous IK target point  $p_{B,x}^*$  and the next point  $p_{B,x}$  obtained from Eq. (4) into  $K$  segments. Then we display  $K$  continuous frames in the screen. At the  $k$ -th frame, we generate the IK target point coordinate  $p_{B,x}[k]$  using Eq. (5). For  $k = 1, \dots, K$ ,

$$p_{B,x}[k] = p_{B,x}^* + (p_{B,x} - p_{B,x}^*) \times \left( \frac{k}{K} \right) \quad (5)$$

This damping function helps achieve a smooth transition as the IK target points move from their current positions to the target positions, following the mapped gesture data. Once the update is complete, the Avatar device uses `render()` from Three.js to render the scene, converting the 3D scene into a 2D image displayed on the screen. When modifications are made to the avatar's movements or expressions, the user needs to wait for the next frame where `update()` and `render()` are automatically executed before you can see the changes reflected on the screen. The time interval between frames primarily depends on the screen refresh rate and is also influenced by factors such as hardware performance, browser limitations, and the complexity of the graphics. If you experience lag in the model's movements, you can observe the FPS indicator (Figure 11 (5)) to determine whether it is necessary to switch to hardware with better performance.

When the Body DA (Figure 9 (2)) receives data from the AvatarTalk server, `bodyControl()` rotates the model's body based on the body joint angles and adjusts the position of the model's hips and feet, achieving the effect of controlling the model's body. When the Face DA (Figure 9 (3)) receives

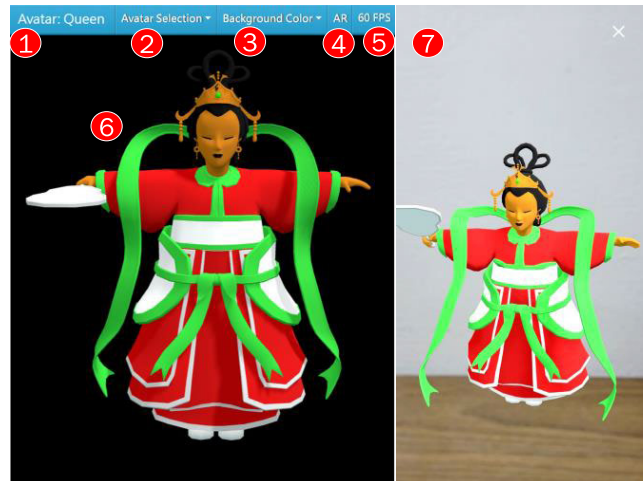


FIGURE 11. The display setup section.

values from AvatarTalk, `faceControl()` sets the model's blinking, eye position, and mouth shape based on the facial control data, enabling facial control of the model. Additionally, users can activate `webXR()` by clicking the AR Button (Figure 9 (8)) to experience webAR effects [26]. Finally, the `update()` and `render()` functions are automatically executed per frame to display the updated model actions on the screen.

## VI. GRAPHICAL USER INTERFACE OF AVATAR

This section describes the GUI components and features of the Avatar device, and explains their respective operational instructions. In AvatarTalk, a user is either a puppeteer or a viewer. We divide the GUI into two sections: Display Setup and Control Setup. Puppeteers can control the avatar model's movements for performances, and therefore can access both sections. The viewers in the audience cannot control the model and can only access the Display Setup section to watch the puppeteer's show. Sections VI-A and VI-B describe Avatar GUI operations. The readers may jump to Section VI-C directly if they are not interested in the details of GUI operation.

### A. THE DISPLAY SETUP SECTION

The Display Setup section of the Avatar GUI (left side of Figure 11) consists of a top navigation bar and a canvas, which includes the following elements: Avatar displays the name of the currently presented avatar model on the screen (Figure 11 (1)); The Avatar Selection (Figure 11 (2)) is a dropdown menu where users can choose the avatar model they want to display on the screen. The switching of models is not time-limited and can be changed at any time during the performance without affecting the model's actions. The Background Color (Figure 11 (3)) is also a dropdown menu that provides commonly used colors and a color picker. Users can freely adjust the background color of the Three.js canvas (Figure 11 (6)) using this feature.

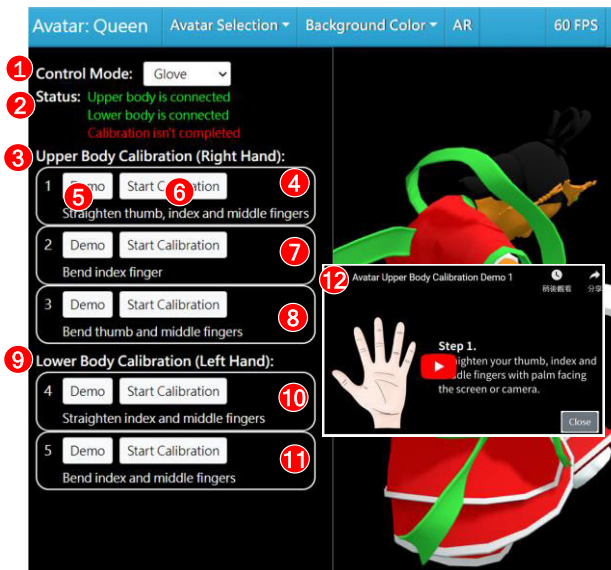


FIGURE 12. Control setup for AiqGlove.

The Augmented Reality (AR) button (Figure 11 (3)) is a toggle button that activates the AR feature. If the user's device meets the requirements of WebXR [26], including support for ARCore and Depth API, clicking on the AR button will overlay the entire device screen with the WebXR Canvas (Figure 11 (7)), displaying the camera's captured image and AR effects.

The Frames Per Second (FPS) indicator (Figure 11 (5)) provides information about the number of frames that are updated per second. It serves as a reference for users to gauge the smoothness of the display. Higher FPS values indicate smoother animation and display, while lower FPS values can result in a lagging or stuttering visual experience.

Besides the navigation bar, the Three.js Canvas (Figure 11 (6)) is a 3D viewport rendered by Three.js [23]. It provides a visual representation of the virtual model, allowing users to see the movements and actions of their chosen avatar in real-time. Both puppeteers and viewers can choose personalized model styles background colors, and the AR feature to be shown in the canvas. This not only provides a variety of performance presentations but also enhances audience interactivity and engagement.

## B. THE CONTROL SETUP SECTION

In the Control Setup section, puppeteers can configure the control settings for the avatar model. Control setup includes Control (Figure 12 (1)) and Status (Figure 12 (2)) subsections. The Status subsection displays the connection status between AvatarTalk and the control devices (gloves, MediaPipe, etc.), as well as the gesture calibration results.

When controlling the avatar model using smart gloves or MediaPipe Hands for hand recognition, the GUI will sequentially display three different status information: upper body, lower body, and calibration. For the upper body, the

status will be displayed in green font as "Upper body is connected" when the model successfully receives data from the glove connected through the AvatarTalk server. Otherwise, the status will be displayed in red font as "Upper body isn't connected." The lower body status follows the same display content and meanings as the upper body. After successfully connecting to the gloves, the GUI will check if the calibration for the respective body part is completed. If so, it will be displayed in green font as "Calibration is completed." These status displays provide feedback to puppeteers regarding the connection and calibration status of the model's upper and lower body, ensuring proper control and interaction with the avatar model.




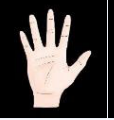

The Control subsection currently provides two options: smart gloves and MediaPipe. By default, the initial control mode is set to smart gloves. In this scenario, the right hand controls the upper body and the left hand controls the lower body of the model. Since hand gestures alone cannot directly control the movements of the avatar model accurately, calibration is required to map the gestures correctly onto the avatar model. For this purpose, the GUI provides calibration buttons of upper body (Figure 12 (3)) and lower body (Figure 12 (9)) for puppeteers to configure.

In the upper body calibration, there are three calibration sets (Figure 12 (3), (7), and (8)). Each set includes a Demo button (Figure 12 (5)) and a Calibration button (Figure 12 (6)). The Demo button contains an instructional video (Figure 12 (12)) that explains the hand gestures the puppeteers need to make during the calibration process, the calibration steps, and the reasons for calibration. Clicking the first calibration button (Figure 12 (6)) records the positions when the thumb, index finger, and middle finger are extended (Gesture 1 in Table 3). This data is used to calibrate the model's head and hand movements. The second calibration button (Figure 12 (7)) records the position when the index finger is bent (Gesture 2 in Table 3). This data is used to calibrate the model's head movement. The third calibration button (Figure 12 (8)) records the positions when the thumb and middle finger are bent (Gesture 3 in Table 3). This data is used to calibrate the model's hand movements. By performing these calibrations, the system can accurately map the gestures performed by the smart gloves to the corresponding movements of the avatar model's upper body, ensuring a more precise and synchronized performance experience. Similarly, in the lower body calibration, there are two calibration sets (Figure 12 (10) and (11)) corresponding to Gestures 4 and 5 in Table 3.

When the control mode is switched to MediaPipe, the GUI will display the control components of MediaPipe, as shown in Figure 13.

The Source Picker (Figure 13 (1)) allows puppeteers to choose between using a camera or a video as the image input source for MediaPipe. If there are multiple available cameras, MediaPipe will automatically select the first camera as the input source. However, puppeteers can manually choose a different available camera if desired.

**TABLE 3.** The calibration buttons and the corresponding gestures.

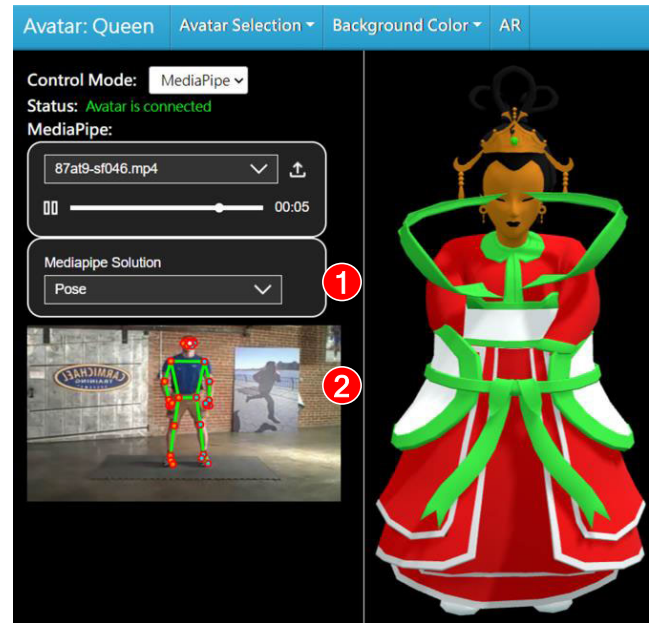
Button	Figure 12 (4)	Figure 12 (7)	Figure 12 (8)	Figure 12 (10)	Figure 12 (11)
Fingers in effect	Gesture 1: Straighten thumb, index, and middle finger	Gesture 2: Bend index finger	Gesture 3: Bend thumb and middle finger	Gesture 4: Straighten index and middle finger	Gesture 5: Bend index and middle finger
Gesture demo					



**FIGURE 13.** The MediaPipe control mode (by Hands).

The MediaPipe solution (Figure 13 (2)) is a dropdown menu that provides puppeteers with options to choose between MediaPipe Hands, Pose, or Holistic recognition solutions. MediaPipe Hands can recognize 3D hand coordinates, allowing puppeteers to control the model using hand gestures. MediaPipe Pose can recognize 3D body coordinates, enabling puppeteers to control the model using body movements. MediaPipe Holistic can recognize 3D body, facial, and hand coordinates. However, in AvatarTalk, it is not possible to use both hand gestures and body control simultaneously. Therefore, when choosing Holistic, only body and facial data can be used to control the model, while hand data will be ignored.

The MediaPipe Canvas (Figure 13 (3)) displays the camera or video feed and overlays the recognized hand, body, or facial information. This allows puppeteers to visually observe and understand the recognition results in real-time,



**FIGURE 14.** The MediaPipe control mode (by Pose and Holistic).

providing an intuitive way to monitor the performance and interaction with the avatar model.

When selecting “Hands” in the MediaPipe solution (Figure 13 (2)), finger gestures are used to control the avatar model, and the GUI provides calibration buttons for the upper body and lower body (Figure 13 (3) and (5)), allowing puppeteers to configure the settings. The calibration process and buttons for finger control follow the same internal structure and principles as those shown in Figure 12. When puppeteers select Pose or Holistic in the MediaPipe solution (Figure 14 (1)), the interface will display as shown in Figure 14 (2).

Since body parts can be directly mapped to the model, there is no need for additional calibration or setup steps. In these modes, the body movements captured by MediaPipe are directly translated into the corresponding movements of the avatar model, eliminating the need for separate calibration or adjustment processes. Puppeteers can start controlling the model using their body movements once the connection to the AvatarTalk server is established.

### C. THE CALIBRATION PROCEDURE

Before controlling the avatar model, users need to perform manual calibration to establish the coordinate mapping between the fingers and the avatar model through calibration. When pressing the calibration buttons in the Avatar GUI (Figure 12 (3)-(11)), the corresponding gestures that need to be performed are listed in Table 3. In traditional puppetry, the upper body of the puppet is usually controlled with right hand.

Typically, the lower body of the puppet is controlled with the left hand, and in AvatarTalk, we use the index finger and middle finger for control. When the index finger and middle

finger are pointing upwards (Table 3 (3)), the puppet’s legs will be extended straight. When the index finger and middle finger are bent downwards (Table 3 (5)), the puppet’s legs will be lifted up.

The gesture calibration is performed by the function `calibrate()` in the avatar SA (Figure 9 (3)) to identify the coordinate range  $[a_F^l, a_F^u]$  of movement for a finger  $F$ .

When Gesture 1 in Table 3 is made and the calibration button (Figure 12 (6)) is pressed, the glove sends the raw coordinate  $a_F^0$  for finger  $F$  to the avatar device. The avatar SA uses Eq. (3) to generate the consistent coordinate  $a_F^l$  (elaborated in Section IV-C). The consistent coordinate  $a_F^l$  serves as the lower bound for the movement of finger  $F$ . That is,  $a_F^l \leftarrow a_F^0$  for  $F = T, I, M$ . The coordinates  $a_F^l$  are saved in the Avatar DB. When Gesture 2 in Table 3 (2) is made and the button in Figure 12 (7) is pressed, the upper bound coordinate  $a_F^u$  for the movement of the index finger is created and stored. Similarly, when Gestures 4 and 5 are made and the corresponding buttons are pressed, the upper bound coordinates  $a_T^u$  and  $a_M^u$  are stored. These coordinate bounds are used for calibration in Section V-B.

**VII. THE IoT-BASED MICROSERVICE ISSUES**

In the IoT-based microservice concept [27], an IoT device (represented by a DM in AvatarTalk) is treated as a microservice. Through this concept, AvatarTalk can easily create arbitrary interaction among the control devices and the puppet devices. For example, to control the robot puppet (Figure 1 (11)) by the AiqGlove (Figure 1 (9)) and to control the virtual avatar (Figure 1 (12)) by MediaPipe (Figure 1 (10)), we only need to drag a “join” link between each of the device pairs ((13)-(join3)-(14) and (15)-(join4)-(16) in Figure 2, respectively), then AvatarTalk automatically generates NA3 and NA4 in Figure 1 to create these two applications.

In the above IoT-based microservice design, the IoT devices (especially the physical ones) are typically independent of each other, and therefore the microservice requirement 1 (MS R1) can be easily satisfied. However, in a heterogeneous IoT system, it is not trivial to satisfy MS R2 because the communication protocols and the data formats for various IoT devices may be different. For example, the puppet in [2] cannot directly talk to the puppet in [12].

To fulfill MS R2 for AvatarTalk, we need to provide two mechanisms. The first mechanism is an application programming interface (API) that allows new IoT devices (especially new artworks) to exchange data with the AvatarTalk server. The second mechanism is the data format translation. In AvatarTalk, format translation includes angle transformation (Sections IV-C and V-B) and coordinate mapping (calibration in Section VI-C). Then through the AvatarTalk GUI in Figure 2, we can conveniently chain the microservices through the “join” links to create the applications. For example, through the IDF and the ODF connections ((13)-(join3)-(14) and (15)-(join4)-(16) in Figure 2, we chain the microservices (9)→(11) and (10)→(12). Figure 15 shows the following microservice chains configured in



**FIGURE 15. Creating AvatarTalk applications through chaining the microservices.**

Figure 2: (9\*)→(10)→(12), (9)→(11) and (2)→(3). We can reconnect the join links to re-chain the microservices as (9)→(11)→(10)→(12). Also, if the Avatar device features Color-I for controlling the color of other devices and Color-O for having its background color controlled, then we can create another microservice chain as (2)→(12)→(3). These examples show that AvatarTalk satisfies MS R2.

In Section VII-A, we first describe how the first MS R2 mechanism is implemented to accommodate new microservices. In Section VII-B, we evaluate the mapping (calibration) performance for the second mechanism. In Section VII-C, we show how the message delays (the data delivery times) affect the synchronization in AvatarTalk when the microservices are chained to create an application.

**A. THE COMMUNICATION API FOR THE IoT DEVICES**

AvatarTalk implements a DA library as the API (based on HTTPS and MQTT) for the first mechanism that satisfies MS R2. In our design, the DA library is generic and can be reused by all IoT devices. Use AiqGlove as an example. Parts of DA library python-like code are shown in Figures 16-19. Figure 16 declares the device’s metadata *profile*, which is derived from the AiqGlove or the MediaPipe devices. Line 1 imports the HTTP protocol library *requests* in Python. Line 5 defines Fingers-I (Figure 1 (13)).

In Figure 17, the *register()* function registers the AiqGlove device to the AvatarTalk server. The AvatarTalk server uses the RESTful API to perform Avatar operations with the same URL. Line 1 specifies the URL of the AvatarTalk server and the device address (typically a MAC address) to identify, e.g., AiqGlove. Lines 4-7 use the HTTP POST method to perform the registration procedure in the AvatarTalk server with the metadata *profile*. Lines 8 and 9 raise an error if it fails, otherwise Line 10 returns the registered device name.

```

1. def push(feature_name, data):
2.     r = requests.put(
3.         endpoint + '/' + feature_name,
4.         json={'data': data}
5.     )
6.     if r.status_code != 200:
7.         raise Error(r.text)
8.     return True

```

**FIGURE 16.** The push function for the avatar device to send data to the AvatarTalk server.

```

1. import requests
2. profile = {
3.     'dm_name': 'SmartGlove',
4.     'd_name': 'AiqGlove',
5.     'df_list': [ThumbCoord-I, IndexCoord-I, MiddleCoord-I,
6.               RingCoord-I, PinkyCoord-I, WristAngle-I],
7. }
8. endpoint = None

```

**FIGURE 17.** The profile of the AiqGlove.

```

1. def register(ServerURL, reg_addr):
2.     global endpoint
3.     endpoint = ServerURL + '/' + reg_addr
4.     r = requests.post(
5.         endpoint,
6.         json={'profile': profile}
7.     )
8.     if r.status_code != 200:
9.         raise Error(r.text)
10.    return r.json().get('d_name')

```

**FIGURE 18.** Registration.

The *push()* function defines in Figure 18 allows AiqGlove to send data of a specific IDF in Fingers-I IDF (Figure 1 (13)) to the AvatarTalk server. In the RESTful API style, the HTTP PUT method is used to update the information. Lines 2-4 pack the data in JSON format and send the IDF data to the AvatarTalk server. Line 8 returns True if the push operation is executed successfully.

The *pull()* function in Figure 19 is used by the Avatar device to obtain the data of a specified ODF in Fingers-O (Figure 9 (1)) from the AvarTalk server. Lines 2-4 invoke an HTTP GET request with the *feature\_name* (e.g., ThumbCoord-O) to query the corresponding data from the AvatarTalk server. Line 7 returns the data.

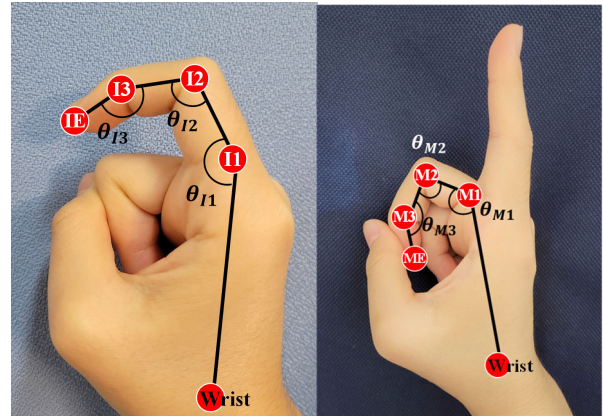
Figures 16-18 are used to create the DA for an IoT device, and the code can be automatically generated by AvatarTalk. In this section, the DA communicates with the AvatarTalk server through HTTPS/RESTful. We have also developed the MQTT version and the details are omitted. The SA of a microservice can be automatically implemented through another mechanism called deviceTalk, which uses the yellow

```

1. def pull(feature_name):
2.     r = requests.get(
3.         endpoint + '/' + feature_name
4.     )
5.     if r.status_code != 200:
6.         raise Error(r.text)
7.     return r.json().get('data')

```

**FIGURE 19.** The pull function for the avatar device to receive data from the AvatarTalk server.



**FIGURE 20.** The joint angle of a finger.

marks in Figures 16-19 to automatically generate the interface code between the SA and the DA. In this way, the microservice requirement 2 (MS R2) is easily satisfied. Details of deviceTalk can be found in [30].

The relationship between communication and microservices is illustrated in Figure 2. In this figure, the microservices are represented as (2), (3), (9), (10), (11), and (12), while the communication protocol is utilized in Joins 1-6.

## B. PERFORMANCE OF CALIBRATION

To make sure the gestures of a puppeteer with a smart glove or MediaPipe correctly drive the avatar, the puppeteer should go through the calibration procedure described in Section VI-C. AvatarTalk movement provides a mechanism to assist calibration procedure. In Section IV-A, the behavior of a finger  $F$  is defined by three joints. For example, when  $F = I$  (index finger), the angle and the coordinate of the proximal interphalangeal joint are  $\theta_{I2}$  (Figure 20) and  $\mathbf{a}_{I2} = (a_{I2,x}, a_{I2,y}, a_{I2,z})$ , respectively. In the calibration, the angle  $\theta_{I2}$  is recorded from the puppeteer's gesture and then translated into the coordinate  $\mathbf{a}_{I2}$ .

In AvatarTalk, we have stored the standard angles of gestures in the Avatar DB. When a puppeteer (with glove/MediaPipe) makes gestures, the angles of his/her gestures are checked with the pre-stored angles to determine how to drive the avatar. However, we found that it is not trivial to produce the "standard" angles for two reasons. First, when we repeat the same standard gesture, the angle  $\theta_{I2}$  may vary.

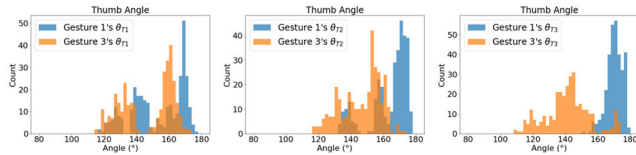


FIGURE 21. Histograms of method 1: joint angles of the thumb (Gestures 1 and 3).

TABLE 4. The percentages of overlay for the angles and the angle sums.

Angles	Method	Overlaid measures	Overlaid Angles
$\theta_{T1}$	1	45.6%	86.2%
$\theta_{T2}$	1	33.8%	85.7%
$\theta_{T3}$	1	11.3%	71.4%
$(\theta_{T1} + \theta_{T2})/2$	2	32.5%	81.8%
$(\theta_{T2} + \theta_{T3})/2$	2	15.9%	50.0%
$(\theta_{T1} + \theta_{T3})/2$	2	24.4%	50.0%
$(\theta_{T1} + \theta_{T2} + \theta_{T3})/3$	2	25.9%	52.9%

Therefore, we must record a range of standard  $\theta_{I2}$  angles of the gesture. Second, different gestures may have the same  $\theta_{I2}$  angle value. Therefore, we also need to consider the relationship between the three joint angles  $\theta_{I1}$ ,  $\theta_{I2}$  and  $\theta_{I3}$  to distinguish different gestures.

We have conducted 2400 experiments. For each finger, there are 480 experiments, where 320 of them are used for training and 160 of them are used for testing. Use the thumb as an example to show how our calibration method works. Figure 21 shows the histograms of the measured angles for  $\theta_{T1}$ ,  $\theta_{T2}$  and  $\theta_{T3}$  (called method 1), where the blue area represents the angles when the thumb is not bending (Gesture 1 in Table 3) and the orange area represents the angles when it is bending (Gesture 3 in Table 3). For three joints, there are overlay areas in the figure that we cannot tell where the thumb bends or not.

On the other hand, when considering the sums of the joints (called method 2), the overlay of the two histograms reduces as illustrated in Figure 22.

Based on the above two histogram figures, Table 4 lists the percentages of overlay for various sums of joint angles of the thumb. In this table, “overlaid measure” means the percentage of the gesture samples that can not be distinguished. The table indicates that in both “overlaid measures” and “overlaid angles” output indicators, method 2 is better than method 1.

The range of the standard angles is stored in the Avatar DB and is then used to determine the types of gestures. Table 5 shows accuracy, precision, and recall using the original joint

TABLE 5. The performance of the calibration.

Method	Gesture	Accuracy	Precision	Recall
1	1	100.00%	100.00%	100.00%
	2	98.13%	96.39%	100.00%
	3	91.25%	85.11%	100.00%
	4	100.00%	100.00%	100.00%
	5	100.00%	100.00%	100.00%
2	1	100.00%	100.00%	100.00%
	2	98.75%	97.56%	100.00%
	3	94.38%	90.80%	98.75%
	4	100.00%	100.00%	100.00%
	5	100.00%	100.00%	100.00%

angles (method 1), and the results using the angle sums (method 2).

Our experiments indicate that method 2 can achieve accurate calibration where AvatarTalk can almost detect the right gestures (98.75%-100% recall) and very seldom mistake the wrong gestures (90.8%-100% precision).

### C. DELAY TIME PERFORMANCE

The IoT-based microservice approach allows AvatarTalk to arbitrary combine heterogeneous control and puppet microservices to create an application. Since the remote data delivery delays may not be the same for different microservice combinations, we need to address the synchronization problem. Consider a cyber-physical puppet performance scenario where the robot puppet (Figure 1 (11)) and the avatar (Figure 1 (12)) are acting a scene together. Puppeteer A controls the robot puppet through (13)-(join3)-(14) in Figure 2 and Puppeteer B controls the avatar through (15)-(join4)-(16). Consider the timing diagram in Figure 23. Suppose that Puppeteer A makes a gesture at time  $T_A^0$  and the robot receives the data and performs at time  $T_A$ . Similarly, Puppeteer B makes a gesture at time  $T_B^0$  (assuming  $T_B^0 \geq T_A^0$ ) and the avatar performs at time  $T_B$ . Both puppeteers will wait to see how the robot and the avatar respond and then take the next actions after  $\max(T_A, T_B)$ . In Figure 23, after Puppeteer B takes action at  $T_B^0$ , he/she should wait for a period  $t^* \geq \max(0, \tau_A - t)$ . Unfortunately, Puppeteer B does not know the time point  $T_A$ . Our previous work [20] imposes a strong restriction to ensure first-come-first-serve data delivery for two puppeteers. In this way, the synchronization is partially resolved. In this subsection, we predict the time point  $T_A$  for Puppeteer B so that she/he can estimate when to take next action.

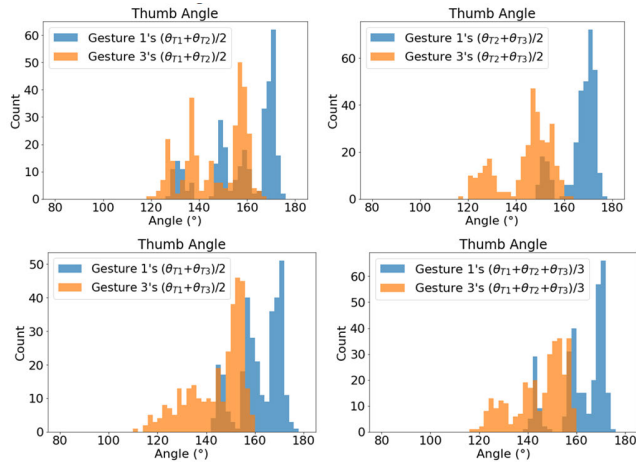


FIGURE 22. Histograms of method 2: the sums of the joint angles of the thumb (Gestures 1 and 3).

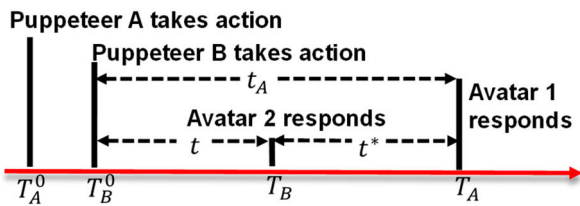


FIGURE 23. The timing diagram.

To derive a quick and primary estimation for  $t^*$ , we conduct the mean value analysis [28], where both  $T_A - T_A^0$  and  $T_B - T_B^0$  have the means  $1/\lambda_A$  and  $1/\lambda_B$ , and the Exponential density functions

$$f_A(t) = \lambda_A e^{-\lambda_A t} \text{ and } f_B(t) = \lambda_B e^{-\lambda_B t} \quad (6)$$

In Figure 23,  $t_A = T_A - T_B^0$  is the residual life of  $T_A - T_A^0$ . From the mean value analysis,  $T_B^0$  is the random observer of  $T_A - T_A^0$ , and from the residual life theorem of the Exponential distribution,  $t_A$  has the same density function as  $T_A - T_A^0$ .

We consider a more general scenario where Puppeteer A is replaced by  $I \geq 1$  puppeteers, and Puppeteer B needs to wait for the other  $I$  puppeteers' actions before taking the next action. Suppose that the  $i$ -th puppeteer issues the action at time  $T_i^0$  and the  $i$ -th puppet/avatar performs at  $T_i$ . Puppeteer B will not make out-of-order action if she/he makes next gesture at time  $T_B + t^*$  after  $T_i$ . The period  $t^*$  is called the think time of Puppeteer B.

Let  $T_i - T_i^0$  be i.i.d. random variables with the Exponential density function  $f_A(t_i)$  and the cumulative distribution function  $F_A(\tau_i)$ . Then  $t_i = T_A - T_B^0$  also has the same distribution. Consider the order statistics  $t_{(i)}$  of  $t_i$ , where  $t_{(1)} \leq \dots \leq t_{(i)} \leq \dots \leq t_{(I)}$  ( $1 \leq i \leq I$ ). From the order statistics [29], the density function  $f_i(\tau_i)$  of  $\tau_i = t_{(i)}$  is

$$f_i(\tau_i) = \frac{I! [F_A(\tau_i)]^{i-1} [1 - F_A(\tau_i)]^{I-i} f_A(\tau_i)}{(i-1)! (I-i)!} \quad (7)$$

For  $i = I$  and from Eq. (6), Eq. (7) is rewritten as

$$\begin{aligned} f_I(\tau_I) &= I [F_A(\tau_I)]^{I-1} f_A(\tau_I) = I \left(1 - e^{-\lambda_A \tau_I}\right)^{I-1} \lambda_A e^{-\lambda_A \tau_I} \\ &= \sum_{i=1}^I (-1)^{i-1} I \lambda_A \binom{I-1}{i-1} e^{-i \lambda_A \tau_I} \end{aligned} \quad (8)$$

Suppose that  $t^*$  has a general density function  $g(t^*)$ , then from Eq. (8) we have

$$\begin{aligned} \Pr[t^* \geq \max(0, \tau_I - t)] &= \Pr[\tau_I < t + t^*] \\ &= \int_{t=0}^{\infty} f_B(t) \int_{t^*=0}^{\infty} g(t^*) \int_{\tau_I=0}^{t+t^*} f_I(\tau_I) d\tau_I dt^* dt \\ &= \sum_{i=1}^I \left[ \frac{(-1)^{i-1} I}{i} \right] \binom{I-1}{i-1} \\ &\quad \times \left[ 1 - \int_{t=0}^{\infty} f_B(t) \int_{t^*=0}^{\infty} g(t^*) e^{-i \lambda_A (t+t^*)} dt^* dt \right] \\ &= \sum_{i=1}^I (-1)^{i-1} \binom{I}{i} \\ &\quad \times \left[ 1 - \left( \frac{\lambda_B}{i \lambda_A + \lambda_B} \right) \int_{t^*=0}^{\infty} g(t^*) e^{-i \lambda_A t^*} dt^* \right] \end{aligned} \quad (9)$$

Let the Laplace transform of  $g(t^*)$  be

$$g^*(s) = \int_{t^*=0}^{\infty} f(t^*) e^{-st^*} dt^*$$

Then Eq. (9) is rewritten as

$$\begin{aligned} \Pr[\tau_I < t + t^*] &= \sum_{i=1}^I (-1)^{i-1} \binom{I}{i} \\ &\quad \times \left\{ 1 - \left( \frac{\lambda_B}{i \lambda_A + \lambda_B} \right) g^*(s) \Big|_{s=i \lambda_A} \right\} \end{aligned} \quad (10)$$

If  $g(t^*)$  is a Gamma density function with the shape parameter  $\alpha$  and the scale parameter  $\beta$  (a generalization of the Exponential density), then its mean value is  $\mu = \frac{\alpha}{\beta}$  and the variance is  $V = \gamma \mu^2 = \frac{\mu}{\beta}$ , where  $\gamma$  is the variance normalized by  $\mu^2$ . Therefore, we have  $\alpha = \frac{1}{\gamma}$  and  $\beta = \frac{1}{\gamma \mu}$ , and the Laplace transform is

$$g^*(s) = \frac{\beta^\alpha}{(s + \beta)^\alpha} = \left( \frac{1}{\gamma \mu s + 1} \right)^{\frac{1}{\gamma}} \quad (11)$$

From Eq. (11), Eq. (10) is rewritten as

$$\begin{aligned} \Pr[\tau_I < t + t^*] &= \sum_{i=1}^I (-1)^{i-1} \binom{I}{i} \\ &\quad \times \left\{ 1 - \left( \frac{\lambda_B}{i \lambda_A + \lambda_B} \right) \left( \frac{1}{i \lambda_A \gamma \mu + 1} \right)^{\frac{1}{\gamma}} \right\} \end{aligned} \quad (12)$$



For  $I = 1$ , Eq. (12) is rewritten as

$$\Pr[\tau_I < t + t^* | I = 1] = 1 - \left(\frac{\lambda_B}{\lambda_A + \lambda_B}\right) \left(\frac{1}{\lambda_A \gamma \mu + 1}\right)^{\frac{1}{\gamma}} \quad (13)$$

For  $I = 2$ , Eq. (12) is rewritten as

$$\Pr[\tau_I < t + t^* | I = 2] = 1 - \left(\frac{2\lambda_B}{\lambda_A + \lambda_B}\right) \left(\frac{1}{\lambda_A \gamma \mu + 1}\right)^{\frac{1}{\gamma}} + \left(\frac{\lambda_B}{2\lambda_A + \lambda_B}\right) \left(\frac{1}{2\lambda_A \gamma \mu + 1}\right)^{\frac{1}{\gamma}} \quad (14)$$

For  $I = 3$ , Eq. (12) is rewritten as

$$\begin{aligned} \Pr[\tau_I < t + t^* | I = 3] &= 1 - \left(\frac{3\lambda_B}{\lambda_A + \lambda_B}\right) \left(\frac{1}{\lambda_A \gamma \mu + 1}\right)^{\frac{1}{\gamma}} \\ &+ \left(\frac{3\lambda_B}{2\lambda_A + \lambda_B}\right) \left(\frac{1}{2\lambda_A \gamma \mu + 1}\right)^{\frac{1}{\gamma}} \\ &- \left(\frac{\lambda_B}{3\lambda_A + \lambda_B}\right) \left(\frac{1}{3\lambda_A \gamma \mu + 1}\right)^{\frac{1}{\gamma}} \end{aligned} \quad (15)$$

The analytic equations are validated against the simulation similar to the one in [20]. The errors are within 1%. We have also conducted 1000 measurements to measure the delays  $t_A$  for (13)-(join3)-(14) in Figure 2 and the delays for  $t_B$  (15)-(join4)-(16), and obtain  $\lambda_A = 32.87203$  ( $E[t_A] = 0.030421$  seconds), and  $\lambda_B = 9.241973$  ( $E[t_B] = 0.108202$  seconds). Figure 23 plot  $\Pr[\tau_I < t + t^*]$  against  $\mu, \gamma (= \frac{V}{\mu^2})$  and  $I$  using Eqs. (13)-(15). Figure 24 (a) indicates that if the variance  $V$  is smaller than  $0.1\mu^2$ , then the puppets can be better synchronized. For the think time of a professional puppeteer,  $V \leq 0.01\mu^2$  is observed.

We also observed that if the think time  $E[t^*] = \mu \leq 0.3$  seconds then AvatarTalk can well synchronize four puppeteers ( $I = 3$ ) with probability higher than 99.99%.

Figure 24 (b) reverses the roles of puppeteers A and B (i.e.,  $\lambda_A = 9.241973$  and  $\lambda_B = 32.87203$ ). The figure shows that if the think time  $E[t^*] = \mu \leq 0.3$  seconds then AvatarTalk can well synchronize four puppeteers ( $I = 3$ ) with probability higher than 95.02%. If the think time  $E[t^*] \leq 0.5$  seconds then AvatarTalk can well synchronize four puppeteers with probability higher than 99.66%.

### VIII. CONCLUSION

This paper proposed AvatarTalk that empowers puppeteers to manipulate avatar puppets online, utilizing either motion capture gloves or camera-based image recognition. AvatarTalk not only enables seamless remote performances and multi-screen presentations on diverse devices but also

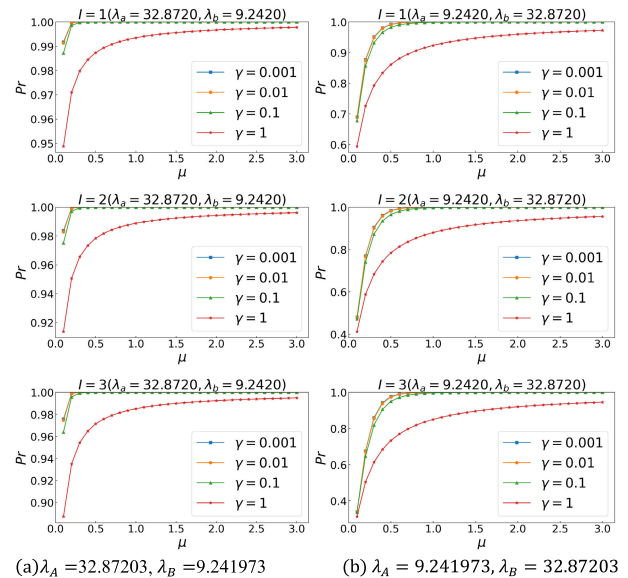


FIGURE 24.  $\Pr[\tau_I < t + t^*]$  against  $\mu, \gamma = V/\mu^2$  and  $I$ .

offers a novel gesture interpretation technique for dance artists. To ensure flexibility, AvatarTalk supports the integration of new control and puppet devices from other approaches through an IoT-based microservice concept.

A meticulous calibration procedure has been developed to precisely capture hand gestures and translate them into corresponding movements for virtual puppets. The study's experiments demonstrate high accuracy of AvatarTalk's gesture recognition, with the platform consistently detecting the correct gestures (recall rates between 98.75% and 100%) and rarely misinterpreting erroneous ones (precision rates between 90.8% and 100%). Both the smart gloves and MediaPipe allow the puppeteer to freely generate gestures with distinct angles. However, for complicated gestures, we strongly suggest that the puppeteer follows calibration steps similar to what we described in Section VI-B.

Additionally, we have presented a mechanism for evaluating the delays in puppet control. An analytic model has been proposed to determine the delay times of messages exchanged between a puppeteer's control device and the AvatarTalk server. For example, in the current AvatarTalk implementation, if the message delay does not exceed 0.1 seconds, four puppeteers can synchronize their actions if the elapsed time between two actions of a puppeteer is longer than 0.3 second. This feature enhances the overall performance experience by minimizing latency issues.

In summary, AvatarTalk seamlessly combines tradition with cutting-edge technology. Its web-based nature and integration of IoT principles provide a powerful and versatile platform for remote performances and open up new possibilities for gesture interpretation in the realm of dance artistry.

In the future, we will use the proposed microservice mechanism to integrate other Puppetry solutions to extend Avatar

applications. For example, features described in [14] and [15] can be accommodated in AvatarTalk as new applications. Also, we will introduce the AI mechanism that interprets the puppet script to fine-tune the IK chain movements to enhance the avatar gestures in the performance.

## REFERENCES

- [1] Y.-B. Lin, Y.-W. Lin, C.-M. Huang, C.-Y. Chih, and P. Lin, "IoTtalk: A management platform for reconfigurable sensor devices," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1552–1562, Oct. 2017.
- [2] Y.-B. Lin, H. Luo, C.-C. Liao, and Y.-F. Huang, "PuppetTalk: Conversation between glove puppetry and Internet of Things," *IEEE Access*, vol. 9, pp. 6786–6797, 2021.
- [3] Google. *MediaPipe Hands*. Accessed: Apr. 15, 2023. [Online]. Available: <https://github.com/google/mediapipe/blob/master/docs/solutions/hands.md>
- [4] Google. *MediaPipe Pose*. Accessed: Mar. 1, 2023. [Online]. Available: <https://github.com/google/mediapipe/blob/master/docs/solutions/pose.md>
- [5] C.-Y. Lin, Z.-H. Yang, H.-W. Zhou, T.-N. Yang, H.-N. Chen, and T. K. Shih, "Combining leap motion with unity for virtual glove puppets," in *Proc. IEEE Int. Conf. Artif. Intell. Virtual Reality (AIVR)*, Taiwan, Dec. 2018, pp. 251–255.
- [6] D.-L. Way, W.-K. Lau, and T. Y. Huang, "Glove puppetry cloud theater through a virtual reality network," in *Proc. ACM SIGGRAPH Posters*. New York, NY, USA: Association for Computing Machinery, Jul. 2019, pp. 1–2.
- [7] D.-L. Way and Y.-H. Wei, "Use of cloud-based virtual reality in Chinese glove puppetry to preserve intangible cultural heritage," *Appl. Sci.*, vol. 13, no. 9, p. 5699, May 2023.
- [8] E. Theodoridou, L. Cinque, F. Mignosi, G. Placidi, M. Polsinelli, J. M. R. S. Tavares, and M. Spezialetti, "Hand tracking and gesture recognition by multiple contactless sensors: A survey," *IEEE Trans. Human-Mach. Syst.*, vol. 53, no. 1, pp. 35–43, Feb. 2023.
- [9] L. LEite and V. Orvalho, "Mani-pull-action: Hand-based digital puppetry," in *Proc. ACM Hum.-Comput. Interact.*, 2017, pp. 1–16.
- [10] E. Yohannes, T. K. Shih, and F. Utamingrum, "Virtual reality in puppet game using depth sensor of gesture recognition and tracking," *J. Comput.*, vol. 31, no. 5, pp. 89–98, 2020.
- [11] H. Liang, S. Deng, J. Chang, J. J. Zhang, C. Chen, and R. Tong, "Semantic framework for interactive animation generation and its application in virtual shadow play performance," *Virtual Reality*, vol. 22, no. 2, pp. 149–165, Jun. 2018.
- [12] T. Li and W. Cao, "Research on a method of creating digital shadow puppets based on parameterized templates," *Multimedia Tools Appl.*, vol. 80, no. 13, pp. 20403–20422, May 2021.
- [13] W. Jiang and C. Cao, "Reconstruction: A motion driven interactive artwork inspired by Chinese shadow puppet," in *Proc. 29th ACM Int. Conf. Multimedia*, New York, NY, USA, Oct. 2021, pp. 1441–1442.
- [14] H. Liang, J. Chang, S. Deng, C. Chen, R. Tong, and J. J. Zhang, "Exploitation of multiplayer interaction and development of virtual puppetry storytelling using gesture control and stereoscopic devices," *Comput. Animation Virtual Worlds*, vol. 28, no. 5, 2017, Art. no. e1727.
- [15] J. Vanderdonck and R.-D. Vatavu, "A pen user interface for controlling a virtual puppet," in *Proc. Companion 12th ACM SIGCHI Symp. Eng. Interact. Comput. Syst.* New York, NY, USA: Association for Computing Machinery, Jun. 2020, pp. 1–6.
- [16] C.-W. Hung, R.-C. Chang, H.-S. Chen, C. H. Liang, L. Chan, and B.-Y. Chen, "Puppeteer: Exploring intuitive hand gestures and upper-body postures for manipulating human avatar actions," in *Proc. 28th ACM Symp. Virtual Reality Softw. Technol.* New York, NY, USA: Association for Computing Machinery, Nov. 2022, pp. 1–11.
- [17] W. Song, X. Wang, Y. Gao, A. Hao, and X. Hou, "Real-time expressive avatar animation generation based on monocular videos," in *Proc. IEEE Int. Symp. Mixed Augmented Reality Adjunct (ISMAR-Adjunct)*, Singapore, Oct. 2022, pp. 429–434.
- [18] VRM. *VRM Documentation*. Accessed: 2023. [Online]. Available: <https://vrm.dev/en/>
- [19] C.-M. Wang and S.-M. Tseng, "Design and assessment of an interactive role-play system for learning and sustaining traditional glove puppetry by digital technology," *Appl. Sci.*, vol. 13, no. 8, p. 5206, 2023, doi: 10.3390/app13085206.
- [20] Y.-B. Lin, H. Luo, and C.-C. Liao, "CATtalk: An IoT-based interactive art development platform," *IEEE Access*, vol. 10, pp. 127754–127769, 2022, doi: 10.1109/ACCESS.2022.3227093.
- [21] Y.-B. Lin, Y.-W. Lin, and K. Hui, "ParadeTalk: Innovative interactions between parade and audiences using IoT," *IEEE Internet Things Mag.*, vol. 3, no. 2, pp. 2–6, Jun. 2020.
- [22] F. C. Park and K. M. Lynch, *Modern Robotics*. Cambridge, U.K.: Cambridge Univ. Press, 2017.
- [23] *Three.js—JavaScript 3D Library*. Accessed: 2023. [Online]. Available: <https://threejs.org/>
- [24] V. Maik, D. T. Paik, J. Lim, K. Park, and J. Paik, "Hierarchical pose classification based on human physiology for behaviour analysis," *IET Comput. Vis.*, vol. 4, no. 1, pp. 12–24, Mar. 2010.
- [25] Ö. Terlemez, S. Ulbrich, C. Mandry, M. Do, N. Vahrenkamp, and T. Asfour, "Master motor map (MMM)—Framework and toolkit for capturing, representing, and reproducing human motion on humanoid robots," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots*, Madrid, Spain, Nov. 2014, pp. 894–901.
- [26] *WebXR*. Accessed: 2023. [Online]. Available: <https://immersiveweb.dev/>
- [27] X. Larrucea, I. Santamaria, R. Colomo-Palacios, and C. Ebert, "Microservices," *IEEE Softw.*, vol. 35, no. 3, pp. 96–100, May/Jun. 2018, doi: 10.1109/MS.2018.2141030.
- [28] Y.-B. Lin, C.-Y. Liu, W.-L. Chen, C.-H. Chang, F.-L. Ng, K. Yang, and J. Hsung, "IoT-based strawberry disease detection with wall-mounted monitoring cameras," *IEEE Internet Things J.*, early access, Jun. 22, 2023, doi: 10.1109/JIOT.2023.3288603.
- [29] Y.-B. Lin and Y.-W. Lin, "SensorTalk: Extending the life for redundant electrical conductivity sensor," *IEEE Internet Things J.*, vol. 9, no. 17, pp. 16619–16630, Sep. 2022, doi: 10.1109/JIOT.2022.3151854.
- [30] W.-E. Chen, Y.-B. Lin, T.-H. Yen, S.-R. Peng, and Y.-W. Lin, "DeviceTalk: A no-code low-code IoT device code generation," *Sensors*, vol. 22, no. 13, p. 4942, Jun. 2022.



**HELIN LUO** (Member, IEEE) received the master's degree from the Graduate School of Arts and Technology, Taipei National University of the Arts, and the Ph.D. degree from the Graduate Institute of Networking and Multimedia, National Taiwan University. He specializes in creating interdisciplinary works using art and technology. For his creations, he draws from his personal experience of being extremely addicted to online games during middle and upper school to explore "the power of virtual worlds," "the thrill of speed," and other variations during this era of technology. Furthermore, his works are centered around the concept of "immigrant illness" amidst this generation of digital immigrants. His works have been recognized at many electronic and contemporary art festivals both domestically and internationally. These include the Digital Arts Award Taipei (2008, 2010, 2011, and 2015), FILE—Electronic Language International Festival (2009, 2010, and 2013), and other competitions. He is the first Taiwanese artist to create works using a four-axis drone. His works have won first prize for the performance award at Digital Art Festival Taipei. He was also specially invited to Ars Electronica Festival to present his inter-disciplinary works made through drones.



**YI-BING LIN** (Fellow, IEEE) received the Ph.D. degree in computer science from the University of Washington, Seattle, USA, in 1990. He is currently a Winbond Chair Professor with National Yang Ming Chiao Tung University (NYCU), a Chair Professor with National Cheng Kung University and China Medical University, and an Adjunct Research Fellow of the Research Center for Information Technology Innovation, Academia Sinica. From 1990 to 1995, he was a Research Scientist with Bellcore. Then, he joined National Chiao Tung University (NCTU), where he became the Senior Vice President, in 2011. From 2014 to 2016, he was the Deputy Minister of the Ministry of Science and Technology, Taiwan. He is the coauthor of the books *Wireless and Mobile Network Architecture* (Wiley, 2001), *Wireless and Mobile All-IP Networks* (John Wiley, 2005), and *Charging for Mobile All-IP Telecommunications* (Wiley, 2008). He is a fellow of AAAS, ACM, and IET.



**YUNG-HUI HUANG** received the B.S. degree in computer science and information engineering from National Cheng Kung University, Tainan, Taiwan, in 2021. She is currently pursuing the M.S. degree in computer science and engineering with National Yang Ming Chiao Tung University, Hsinchu, Taiwan.

...



**CHEN-CHI LIAO** received the B.S. and M.S. degrees in computer science and information engineering from National Ilan University, Ilan, Taiwan, in 2016 and 2018, respectively. He is currently pursuing the Ph.D. degree in computer science and engineering with National Yang Ming Chiao Tung University, Hsinchu, Taiwan.