Received 15 August 2020; revised 26 September 2020; accepted 12 October 2020. Date of publication 21 October 2020; date of current version 10 November 2020.

Digital Object Identifier 10.1109/OJCOMS.2020.3032608

Motion Prediction and Pre-Rendering at the Edge to Enable Ultra-Low Latency Mobile 6DoF Experiences

XUESHI HOU (Student Member, IEEE), AND SUJIT DEY (Fellow, IEEE)

Mobile Systems Design Lab, Department of Electrical and Computer Engineering, University of California at San Diego, La Jolla, CA 92093, USA

CORRESPONDING AUTHOR: X. HOU (e-mail: x7hou@ucsd.edu)

This work was supported in part by the Center for Wireless Communications at University of California at San Diego.

ABSTRACT As virtual reality (VR) applications become popular, the desire to enable high-quality, lightweight, and mobile VR can potentially be achieved by performing the VR rendering and encoding computations at the edge and streaming the rendered video to the VR glasses. However, if the rendering has to be performed after the edge gets to know of the user's new head and body position, the ultra-low latency requirements of VR will not be met by the roundtrip delay. In this article, we introduce edge intelligence, wherein the edge can predict, pre-render and cache the VR video in advance, to be streamed to the user VR glasses as soon as needed. The edge-based predictive pre-rendering approach can address the challenging six Degrees of Freedom (6DoF) VR content. Compared to 360-degree videos and 3DoF (head motion only) VR, 6DoF VR supports both head and body motion, thus not only viewing direction but also viewing position can change. Hence, our proposed VR edge intelligence comprises of predicting both the head and body motions of a user accurately using past head and body motion traces. In this article, we develop a multi-task long short-term memory (LSTM) model for body motion prediction and a multi-layer perceptron (MLP) model for head motion prediction. We implement the deep learning-based motion prediction models and validate their accuracy and effectiveness using a dataset of over 840,000 samples for head and body motion.

INDEX TERMS Virtual reality, video streaming, six degrees of freedom (6DoF), edge computing, edge intelligence, motion prediction.

I. INTRODUCTION

VIRTUAL reality (VR) systems have triggered enormous interest over the last few years in various fields including entertainment, enterprise, education, manufacturing, transportation, etc. However, several key hurdles need to be overcome for businesses and consumers to get fully on board with VR technology [1]: cheaper price and compelling content, and, most importantly, a truly mobile VR experience. Of particular interest is how to develop mobile (wireless and lightweight) head-mounted displays (HMDs), and how to enable VR experience on the mobile HMDs using bandwidth-constrained mobile networks, while satisfying the ultra-low latency requirements.

Currently, there are several categories of HMDs [2]: PC VR, standalone VR, and mobile VR. Specifically, PC VR has high visual quality with rich graphics contents as well as high frame rate, but the HMD is usually tethered with PC [3], [4]; standalone VR HMD has a built-in processor and is mobile, but may have relative low-quality graphics and low refresh rate [5], [6]; mobile VR is with a smartphone inside, leading to a heavy HMD to wear [7], [8]. Therefore, current HMDs still cannot offer us a lightweight, mobile, and high-quality VR experience. To solve this problem, we propose an edge computing based solution. By performing the rendering on an edge computing node and streaming videos to users, we can complete the heavy computational tasks on the edge computing node and thus enable mobile VR with lightweight VR glasses. The most challenging part of this solution is ultra-high bandwidth and ultra-low latency requirements, since streaming 360-degree video causes tremendous bandwidth consumption and good VR user experiences require ultra-low latency (<20ms) [9], [10].

Specifically, the total end-to-end latency of edge computing based VR system includes the following parts: time to transmit sensor data from HMD to edge computing node, time to render (and encode) on the edge node, time to

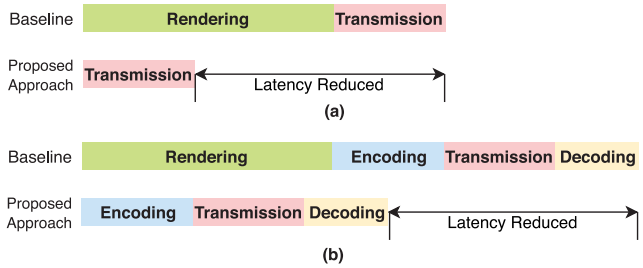

FIGURE 1. Illustration of rendering and streaming pipeline to show how our *predictive pre-rendering* approach reduces latency: (a) Without encoding and decoding; (b) With encoding and decoding.

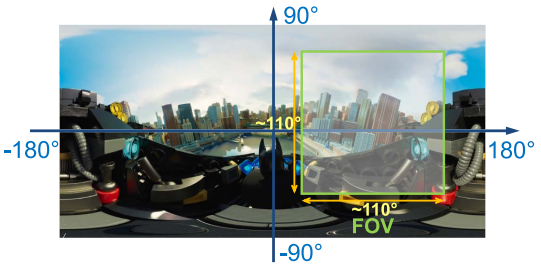

FIGURE 2. Field of view (FOV) in a 360-degree view.

transmit rendered video from the edge computing node to HMD, and time to (decode and) display the view on the HMD. The encoding and decoding are optional according to the specific application design. Once the user moves his/her head or body position, high-quality VR requires this end-to-end latency as less than 20ms [9], [10] to avoid motion sickness. For the edge computing based VR system, it is extremely challenging to meet this requirement.

Motivated by the ultra-low latency requirement challenge, in this article, we introduce edge intelligence for mobile VR, wherein the edge can predict, pre-render and cache the VR video in advance, to be streamed to the user VR glasses as soon as needed. Specifically, we consider six Degrees of Freedom (6DoF) VR experiences, which support both the head and body motions, thus both the viewing direction and viewing position can change. Hence, in order to pre-render the view, edge intelligence is needed to predict both the head and body motions of a user accurately. By predicting head and body motion of users in the near future with edge intelligence, we can do a predictive pre-rendering on the edge computing node and then stream (even pre-deliver) the predicted view to the HMD. The difference between *stream* and *pre-deliver* is that *stream* means holding the pre-rendered frame until determining whether prediction is 'correct' or not using the actual motion, while *pre-deliver* refers to sending the pre-rendered frame immediately to the user without this determination. Note that both *stream* and *pre-deliver* choices can significantly reduce latency: one does pre-rendering and the other does both pre-rendering and pre-delivery. The latter reduces more latency than the former but (i) needs a technique on HMD to buffer the predicted view and determine whether the predicted viewing position and direction are correct; (ii) transmits extra content when the prediction is inaccurate, leading to more bandwidth consumption. Hence, we adopt the former method, where the latency can be significantly reduced since the pre-rendered view will be transmitted if the predicted viewing position and direction are 'correct' (i.e., the error is less than a given ultra-low value); otherwise, latency remains the same with traditional streaming method because the actual view will be rendered and transmitted to the HMD. Fig. 1 illustrates the latency reduced by our pre-rendering approach compared to the traditional approach, in terms of rendering and streaming pipeline (from edge computing node to HMD). The key to achieving this efficient edge-based predictive pre-rendering approach is predicting body and head motion in advance accurately, and then pre-rendering the predicted view accordingly.

In our earlier work [11], we proposed techniques for head motion prediction in 360-degree videos and three Degrees of Freedom (3DoF) VR applications. In this work, we address the more challenging 6DoF VR content. Compared to 360-degree videos and 3DoF (head motion only) VR, 6DoF VR supports both head and body motions, thus not only viewing direction but also viewing position changes. Hence, our proposed VR edge intelligence has to comprise of predicting both the head and body motions of a user accurately using past head and body motion traces. Specifically, for head motion prediction in 360-degree videos and 3DoF VR, a certain prediction error is allowed, because the error can be handled by delivering a larger field of view (FOV) with high quality or rendering larger FOV. Note that FOV is around $90° \times 90°$ for Samsung Gear VR and $110° \times 110°$ for HTC Vive while the 360-degree view is $360° \times 180°$ in size (as is shown in Fig. 2). Compared to 360-degree videos and 3DoF VR, the motion prediction in 6DoF VR is much more challenging, where the body motion prediction needs high precision to pre-render the user's view (otherwise may cause dizzy feeling). For 360-degree videos and 3DoF VR, the 360-degree view at a time point is known and unchanged by any head motion, but for 6DoF VR it can be totally different due to the body motion. Therefore, this article will explore the feasibility of doing motion prediction with high precision in 6DoF VR using edge intelligence, and its main contributions can be summarized as follows:

- For 6DoF VR applications, we propose a new *edge-based predictive pre-rendering* approach involving both body and head motion prediction, in order to enable high-quality, lightweight, and mobile VR with low latency.
- We develop a prediction method using edge intelligence to predict where a user will be standing (i.e., viewing position) and looking into (i.e., viewing direction) in the 360-degree view based on their past behavior. Using a dataset of real head and body motion traces from VR applications, we show the feasibility of our multi-task long short-term memory (LSTM) model for body

motion prediction and multi-layer perceptron (MLP) model for head motion prediction with high precision.
- We propose a FOV selection technique for pre-rendering a larger FOV to further reduce head motion prediction error, and a motion error determination technique as the system mechanism of our edge-based predictive pre-rendering approach.
- To the best of our knowledge, we are the first to come up with this edge-based predictive pre-rendering idea using edge intelligence for 6DoF VR applications and show good results on a real motion trace dataset in the VR applications. We demonstrate the potential of our approach with high accuracy of head and body motion prediction.

Note that a preliminary version of our work has been published in [12], where we reported on edge-based predictive single-task models for head (MLP model) and body (LSTM model) motions, and some preliminary results. In this article, we develop (i) a new multi-task LSTM model for body motion prediction to reduce body motion prediction error, (ii) head and body motion prediction based FOV selection for pre-rendering, such that the selected FOV minimizes the effects of motion prediction error while also minimizing the selected FOV size, and (iii) motion error determination as the system mechanism of our edge-based predictive pre-rendering approach. Note that the methodology proposed in this article applies to single-user scenarios, and we plan to further study more complex multi-user scenarios as part of future work.

The rest of this article is organized as follows. Section II reviews related work. Section III presents a system overview and problem definition. Section IV describes our dataset. The methodology for head and body motion prediction is described in Section V. We present our experimental results in Section VI and conclude our work in Section VII.

II. RELATED WORK

In this section, we review current work in the following topics related to our research.

Enable High-Quality Mobile VR: Some recent studies [13]–[17] explore solutions to enable lightweight and mobile VR experiences, and improve the performance of the current VR system. To provide high-quality VR on a mobile device, [13] presents a pre-rendering and caching design called FlashBack, which pre-renders all possible views for different positions as well as orientations at each 3D grid point with a density of 2-5cm, stores them on a local cache, and delivers frames on demand according to current position and orientations. This method may lead to high inaccuracy and overwhelming storage overhead of pre-caching all possible views (e.g., 50GB for an app). Reference [14] introduces a parallel rendering and streaming mechanism to reduce the add-on streaming latency, by pipelining the rendering, encoding, transmission, and decoding procedures. This method focuses on minimizing streaming latency, thus the latency for rendering part remains the same as the traditional rendering method. Reference [15] presents a collaborative rendering method to reduce overall rendering latency by offloading costly background rendering to an edge computing node and only performing foreground rendering on the mobile device. In contrast, our method proposes to pre-render based on head and body motion predictions, reducing the latency of rendering more drastically. To reduce latency needed, [16] proposes to stream VR scenes containing only the user's FOV and a latency-adaptive margin area around the FOV. Reference [17] aims to address the ultra-high bandwidth challenge in high-quality mobile VR by adaptively reusing the redundant VR pixels across multiple VR frames. The reason these two methods cannot be applied to our scenario is that [16] cannot address 6DoF VR content and [17] reduces network transmission latency to some extent but also brings the larger rendering latency.

Human Motion Prediction: Learning statistical models of human motion are challenging due to the stochastic nature of human movement to explore the environment, and many works [18]–[22] propose methods to address it. Based on classical mechanics, there are some studies [18]–[20] showing the efficiency of linear acceleration model (Lin-A) by doing motion prediction or estimation with an assumption of linear acceleration, especially in a small time interval (e.g., order of tens of milliseconds). Reference [18] describes a good performance of a simple first-order linear motion model for tracking human limb segment orientation, and [19], [20] reveal acceptable results when employing the linear model as a baseline to predict human trajectory. Meanwhile, deep learning approaches [19]–[22] for human body prediction have also achieved remarkable accomplishments. Specifically, [19], [20] propose their LSTM models to predict human future trajectories, but their models aim to learn general human movement from a massive number of videos and the corresponding precision of predicted position does not achieve the requirement of pre-rendering in VR scenarios. References [21], [22] propose various recurrent neural network (RNN) models for human motion prediction to learn human kinematics from skeletal data. But these models are designed to learn the patterns from a series of skeletal data and cannot be applied to our VR scenarios directly.

Moreover, [11], [23]–[25] also explore the feasibility of doing head motion prediction, however, head motion prediction in 6DoF is quite different than 360-degree video (3DoF), since in the latter, for each time point, the whole 360-degree view displayed for viewers is fixed and more regularity and pattern exist in their viewing directions. By learning viewers' traces, for 3DoF applications, the models can well predict the viewing position since at a certain time point, there are always some areas attracting most attention and viewers are more likely to look at them. Head motion in 6DoF is more difficult to predict because both position and viewing direction may continuously change, and there is a much larger virtual space to explore for users. Therefore, the above approaches cannot be used to address our scenario: we aim to explore the high-precision human body

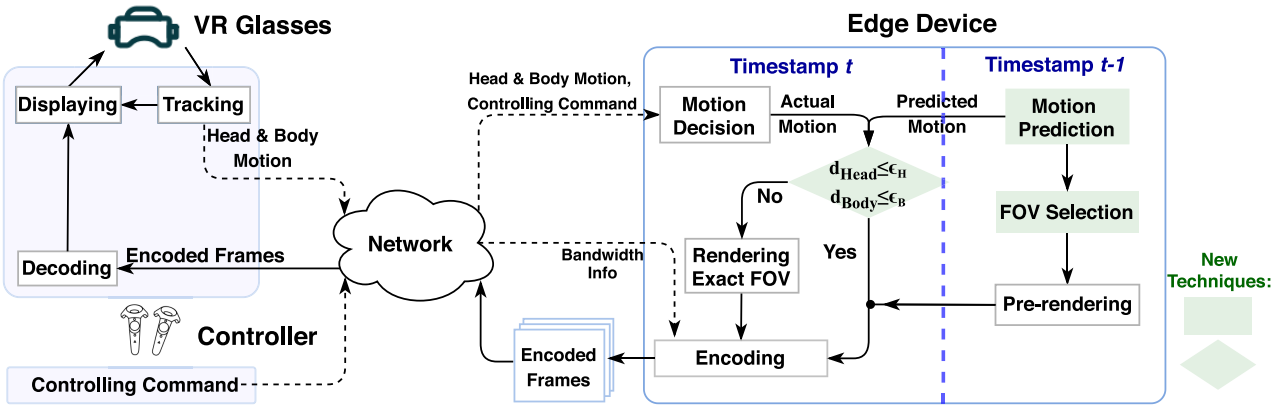

FIGURE 3. System overview.

and head motion prediction in 6DoF VR applications for pre-rendering.

Multi-Task Learning: Multi-task learning aims to improve learning efficiency and prediction accuracy for each task, compared to training a separate model for each task. Some recent studies [26]–[28] explore solutions to improve prediction accuracy by learning multiple tasks from a shared representation, and formulate the multi-task learning problems which involve joint learning of various regression and classification tasks with different units and scales. Reference [26] shows that a shared representation with multi-task learning can improve accuracy on depth regression and instance segmentation over separately trained single tasks because of cues from other tasks. Reference [27] presents that multi-task learning benefits and achieves better results compared with single-task models on event detection in social media by doing text analysis with Twitter datasets. Reference [28] proposes a multi-task RNN for simultaneous recognition of surgical gestures with kinematic signals, and demonstrates that the recognition performance improves with the multi-task learning model compared with single-task models. The reason why we cannot use above methods for body motion is that most of these methods [26], [28] address computer vision recognition problem instead of predicting variables ahead of time and [27] considers event detection based on texts in social media which also cannot be applied to body motion prediction scenario. Our proposed multi-task model distinguishes from the above methods by addressing the real-time body motion prediction problem using real motion traces in the VR scenario and aiming for an ultra-low prediction error.

III. SYSTEM OVERVIEW

In this section, we describe our system overview. In Fig. 3, a user's head motion, body motion as well as other controlling commands will firstly be sent to the edge, which performs the *edge-based predictive pre-rendering* approach. Based on the past few seconds of head motion, body motion and control data received from the user, the edge device will do three things: (i) perform motion prediction (*motion prediction*); (ii) do pre-render based on the predicted viewing position and direction (*motion decision* and *pre-rendering*); (iii) cache the predicted frames in advance. Later, if the predicted viewing position and direction are 'correct' (i.e., the error is less than a given ultra-low value), the cached predicted frames can be streamed from the edge device to the HMD and displayed on HMD immediately; otherwise, the actual view will be rendered by the edge device and transmitted to the HMD. For the former case, latency needed will be significantly reduced since the view is pre-rendered and cached on the edge computing node before it is needed; for the latter, latency remains the same with the conventional method of streaming from the edge computing node. Note that although the controller can affect the rendered frame by pointing at a certain place to teleport in virtual space, we do not need to predict for the new location triggered by the controller, as in this case, users will expect much larger latency than 20ms. We will describe motion prediction, FOV selection, and motion error determination (highlighted in green in Fig. 3) with more details in Section V.

Note that the edge device can be either a Mobile Edge Computing node (MEC) in the mobile radio access or core network, or a Local Edge Computing node (LEC) located in the user premises or even his/her mobile device, connecting to the HMD through WiFi or WiGig. While each of the above choices has tradeoffs, this article will not specifically address these tradeoffs and select either MEC or LEC. Instead, we focus on developing accurate head and body motion prediction techniques, which can be used for the edge-based predictive pre-rendering approach shown in Fig. 3, and will apply to either of the edge device options.

Problem Statement: In each time point, the user can have a specific viewing position and viewing direction, corresponding to the body and head motion. Given previous and current viewing directions and viewing positions, our goal is to predict viewing direction and position for the next time point. After rendering pixels based on predicted viewing position and direction, frames can be further encoded to

TABLE 1. Notations used.

Notation	Meaning
t	Timestamp (time counted since application launches)
RTT	Round-trip latency
(α, β, γ)	Euler angles for head pose (pitch α, yaw β, roll γ)
(x, y, z)	Position for body pose
$\vec{v}_{head}$	Head motion speed $(v_\alpha, v_\beta, v_\gamma)$
$\vec{v}_{body}$	Body motion speed (v_x, v_y, v_z)
d_{head}	Angular distance between actual and predicted head poses
d_{body}	Distance between actual and predicted body positions
$d_\alpha, d_\beta, d_\gamma$	d_{head} in α, β, γ-axis
d_x, d_y, d_z	d_{body} in x, y, z-axis
ϵ_1, ϵ_2	Thresholds of acceptable head and body prediction errors
L_i	Objective loss function for individual task i
w_i	Weight for individual task i
L_{total}	Loss function for multi-task learning model
θ_h, θ_v	Horizontal FOV and vertical FOV
θ'_h, θ'_v	Selected new horizontal FOV and vertical FOV
n_w	Number of frames in a sliding window in FOV selection
$\hat{d_\alpha}, \hat{d_\beta}, \hat{d_\gamma}$	Estimated value of $d_\alpha, d_\beta, d_\gamma$ in FOV selection
I_1, I_2	Two grayscale intensity images
$I_{dif}(i)$	Difference between two intensity images for pixel i
R_{dif}	Percentage of mismatched pixels
N_{dif}	Number of pixels having difference in grayscale intensity
N_{frame}	Total number of pixels per frame

a video and delivered to users. Specifically, we describe the problem formulation for motion prediction below. The notations used in our approach are described in Table 1.

A. PROBLEM FORMULATION

Trajectory Sequence: Spatiotemporal point q_t is a tuple of time stamp t, viewing position b, and viewing direction h, i.e., $q_t = (t, b, h)$. The trajectory sequence from time point t_w to time point t_{w+n-1} is a spatiotemporal point sequence, which can be denoted as $S(t_w, t_{t+n-1}) = q_{t_w} q_{t_{w+1}} \dots q_{t_{w+n-1}}$.

Thus, the problem can be formulated as follows:

- *Input:* a trajectory sequence from time point t_w to time point t_{w+n-1}, *i.e.*, $S(t_w, t_{t+n-1}) = q_{t_w} q_{t_{w+1}} \dots q_{t_{w+n-1}}$;
- *Output:* predicted spatiotemporal point $\widehat{q_{t_{w+n}}}$ at time point t_{w+n};

In this article, we aim to predict the viewing position b and viewing direction h for the next time point using current and previous viewing positions and directions.

B. TIME ANALYSIS

In this subsection, we give an analysis of the time taken for the various tasks of our proposed edge-based predictive pre-rendering method, as shown in Table 2. Specifically, we can see that the latency for transmission from HMD to the edge and from edge to HMD depends on the distance between them. Since we predict the user view 11ms in advance (1 frame ahead, assuming 90 frames/second), we have adequate time to (i) predict motion and do FOV selection (i.e., $< 1ms$, which is described in details in Section VI-D) and (ii) pre-render the predicted view (i.e., $5ms - 10ms$) in advance with

TABLE 2. Time needed for different procedures.

Procedure	*Time Needed*
Transmission from HMD to edge	*Depends on distance*
Rendering	$5ms - 10ms$
Encoding	$3ms - 8ms$
Transmission from edge to HMD	*Depends on distance*
Decoding	$\approx 3ms$
Motion Prediction & FOV Selection	$< 1ms$

no additional latency, hence satisfying the ultra-low latency requirement of 6DoF VR immersive experiences. The round-trip transmission latency, latency of rendering, latency of encoding, and latency of decoding can be denoted as RTT, $T_{rendering}$, $T_{encoding}$, and $T_{decoding}$ respectively.

As for the conventional method, the latency without motion prediction and pre-rendering is

$$RTT + T_{rendering} + T_{encoding} + T_{decoding},$$

where the lower boundary and upper boundary of latency are $RTT + 11ms$ and $RTT + 21ms$ respectively. Thus, given added round-trip transmission latency of around 9ms, the end-to-end latency for conventional method is $20ms - 30ms$.

For our proposed edge-based predictive pre-rendering approach, the latency with 'correct' motion prediction is

$$RTT + T_{encoding} + T_{decoding},$$

where the lower boundary and upper boundary of latency are $RTT + 6ms$ and $RTT + 11ms$ respectively. Otherwise, when the motion prediction is not 'correct', the latency is the same with conventional method. Thus, given added round-trip transmission latency of around 9ms, the end-to-end latency for the proposed edge-based predictive pre-rendering approach is $15ms - 20ms$ with 'correct' motion prediction and $20ms - 30ms$ with 'incorrect' motion prediction. We present experimental results in Section VI which shows high accuracy of our proposed motion prediction techniques, achieving 'correct' motion predictions in most of the time points during 6DoF VR applications.

IV. DATASET AND ITS CHARACTERISTICS

In this section, we first describe the dataset we use and then show characteristics of the dataset using certain metrics we define.

A. DATASET

To investigate head and body prediction in 6DoF VR applications, we conduct our study on a real motion trace dataset we collected from 20 users using HTC Vive to experience two 6DoF VR applications called Virtual Museum [29] and Virtual Rome [30] in our laboratory. The system setup will be described in Section VI-A. The trace consists of 840,000 sample points of head and body motion data collected from the users. Fig. 4(a)(b) show the illustration of the two virtual applications, where Virtual Museum has three exhibition rooms and Virtual Rome contains larger space including different courtyards and halls. The *walkable area* is restricted

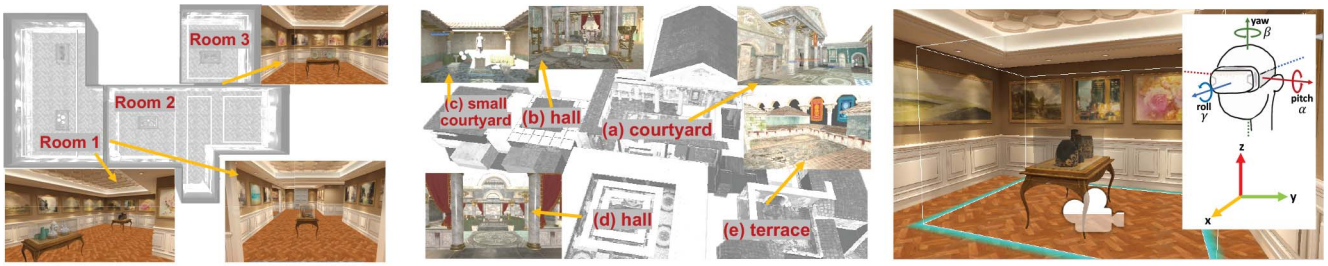

FIGURE 4. Illustration of two virtual applications and other settings: (a) Virtual Museum and (b) Virtual Rome; (c) Boundary of *walkable area*, and coordinates for head and body motions.

TABLE 3. Experimental settings for different sessions in the Virtual Museum and Virtual Rome.

Session	*Virtual Museum (VM)*			*Virtual Rome (RM)*		
	VM1	*VM2*	*VM3*	*RM1*	*RM2*	*RM3*
With Guidance	✔			✔		
Use Controller			✔			✔

by the size of the tracked space in the room and constrained to a fixed regular shape. Users can explore each virtual space by walking in the *walkable area* or teleporting by pointing at a place with a controller. The top subplot in Fig. 4(c) uses light blue lines to show the boundary of the *walkable area* in the VR. As shown in Table 3, we set three sessions respectively for each application: (i) in session 1, users are given rough guidance of taking a stroll about the room at the beginning of the session, without a controller in their hand; (ii) in session 2, users walk around freely in the room, without a controller in their hand; (iii) in session 3, users walk around freely in the room and have a controller in their hand; the controller allows them to teleport to any position in virtual space by pointing at that place, and the position of the *walkable area* in VR also changes accordingly.

Motion traces include the user ID, session timestamp, euler angles for the head pose (pitch α, yaw β, roll γ), and position for body pose (x, y, z). The session timestamp refers to the time counted since application launches in milliseconds, and timestamps appear each 11ms (corresponding to 90Hz, which is the refresh rate of HTC Vive). The middle and bottom subplots of Fig. 4(c) exhibit the coordinates for head pose using euler angles and for body pose using position.

B. DATASET CHARACTERISTICS

To depict key characteristics of the head motion and viewpoint changes in the dataset quantitatively, we offer the following definitions.

Head Motion Vector: The corresponding head poses at time points t_1 and t_2 (where $t_1 < t_2$) are denoted by $(\alpha(t_1), \beta(t_1), \gamma(t_1))$ and $(\alpha(t_2), \beta(t_2), \gamma(t_2))$ respectively. Head motion vector $(\Delta\alpha, \Delta\beta, \Delta\gamma) = (\alpha(t_2) - \alpha(t_1), \beta(t_2) - \beta(t_1), \gamma(t_2) - \gamma(t_1))$.

Head Motion Speed: Head motion speed $\vec{v}_{head}$ is defined as the angular distance the head moved divided by time, i.e., $\vec{v}_{head} = (v_\alpha, v_\beta, v_\gamma) = (\frac{\Delta\alpha}{t_2-t_1}, \frac{\Delta\beta}{t_2-t_1}, \frac{\Delta\gamma}{t_2-t_1})$.

TABLE 4. Description of variables.

	Variable	Seq.	Unit
Measured	Timestamp	✔	Millisecond (ms)
	Euler angles	✔	Degree ($^\circ$)
	Position	✔	Meter (m)
Derived	Head Motion Speed	✔	Degree per Millisecond ($^\circ/ms$)
	Body Motion Speed	✔	Centimeter per Millisecond (cm/ms)

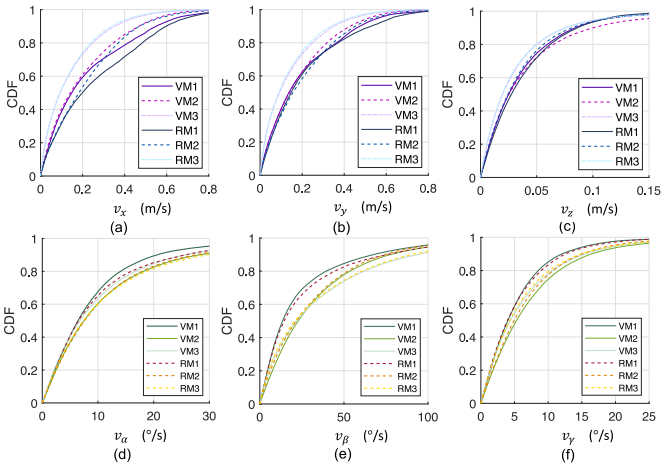

FIGURE 5. CDF of motion speed for different sessions: (a)(b)(c) for body motion; (d)(e)(f) for head motion.

Body Motion Vector: The corresponding body poses at time points t_1 and t_2 (where $t_1 < t_2$) are denoted by $(x(t_1), y(t_1), z(t_1))$ and $(x(t_2), y(t_2), z(t_2))$ respectively. Body motion vector $(\Delta x, \Delta y, \Delta z) = (x(t_2) - x(t_1), y(t_2) - y(t_1), z(t_2) - z(t_1))$.

Body Motion Speed: Body motion speed $\vec{v}_{body}$ is defined as the distance the body moved divided by time, i.e., $\vec{v}_{body} = (v_x, v_y, v_z) = (\frac{\Delta x}{t_2-t_1}, \frac{\Delta y}{t_2-t_1}, \frac{\Delta z}{t_2-t_1})$, and the value of it is

$$v_{body} = |\vec{v}_{body}| = \sqrt{{v_x}^2 + {v_y}^2 + {v_z}^2}. \quad (1)$$

Table 4 presents the description of variables. Apart from measured variables in the dataset, for each sample point, we can obtain the derived variables including head motion speed and body motion speed using definitions above. In Fig. 5, we plot the cumulative distribution function (CDF)

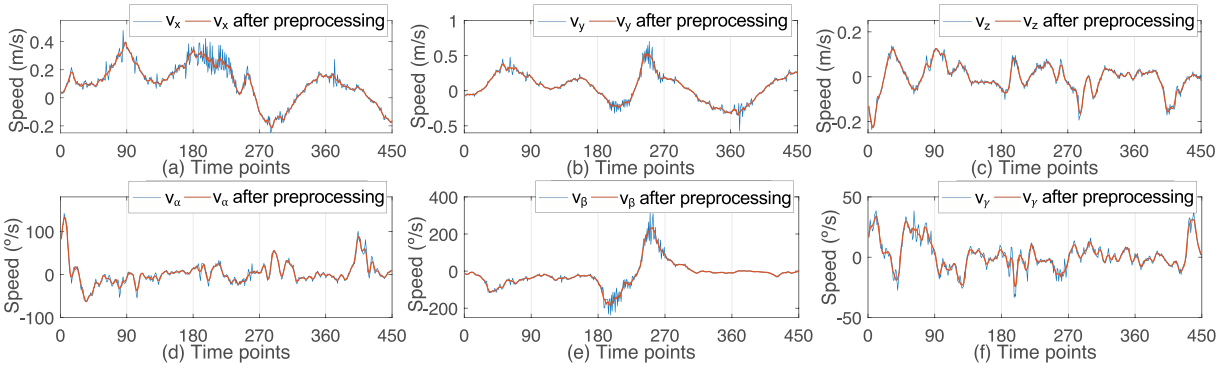

FIGURE 6. Motion speed obtained before and after the preprocessing step: (a)(b)(c) for body motion; (d)(e)(f) for head motion.

of body motion speed in each axis (i.e., v_x, v_y, v_z) and head motion speed in each axis (i.e., $v_\alpha, v_\beta, v_\gamma$) for different sessions. We can see that (i) over 95% of v_x, v_y, v_z are less than 0.8m/s, 0.8 m/s, and 0.15 m/s respectively, and around 90% of $v_\alpha, v_\beta, v_\gamma$ are less than 30°/s, 100°/s, and 25°/s respectively; (ii) for the body motion speed distribution, the speed in each session is as follow from high to low: RM1>VM1>RM2>VM2>VM3>RM3; and (iii) for the head motion speed distribution, the speed in each session is as follow from low to high: VM1<RM1<RM2&VM2<RM3&VM3. Thus among six sessions of two applications, there are more body motion and less head motion in Session 1 (i.e., RM1, VM1) while less body motion and more head motion in Session 3 (i.e., RM3, VM3).

V. OUR APPROACH

In this section, we describe our proposed approach of preprocessing and modeling for head and body motion predictions.

A. PREPROCESSING

We aim to remove noise within head and body motion in the preprocessing step. We first calculate head motion speed and body motion speed for each time point. Fig. 6 presents the body motion and head motion speed in $x, y, z, \alpha, \beta, \gamma$-axis respectively for a sample in the motion trace of one user in the Virtual Museum application. The blue line in each subplot shows there can be at times significant noise in each of motion speed, due to sensor noise and other measuring errors from HTC Vive HMD and base stations. This noise is identifiable since the speed cannot change so rapidly and intensively within several milliseconds. To remove the noise in body motion and head motion, we propose to use the Savitzky-Golay filter method [31] because of its high accuracy and efficiency. This filter approximates (using least-square fitting) the underlying function within the moving window by a polynomial of a higher order. The blue and red lines in Fig. 6 show the speed before and after the preprocessing step. We can see the noise is significantly reduced after preprocessing step.

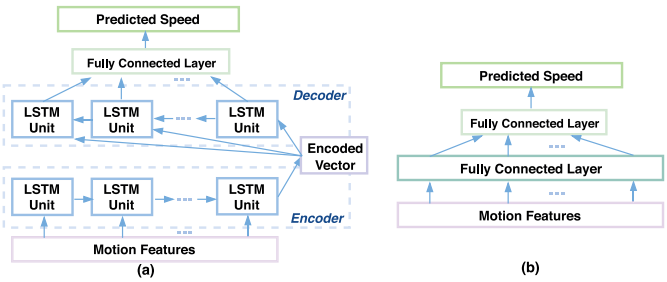

FIGURE 7. (a) LSTM model and (b) MLP model used for motion prediction.

B. PREDICTIVE MODELING

To represent motion features, we select 60 time points as the prediction time window (i.e., predict head and body speed according to speed traces in the latest 60 time points), since it achieves better performance than 40, 50, 70, 80, 90 time points based on our experiments. For training the model, we choose a simple representation for motion as a 1×60 vector, where each element equals to i when the speed is i at that time point, and the dimension of 60 corresponds to 60 time points.

1) SINGLE-TASK MODEL

We investigate a LSTM model as well as an MLP model to be trained for single task separately, where the single task refers to prediction for body motion speed in each axis (i.e., x, y, or z-axis) or head motion speed in each axis (i.e., α, β, or γ-axis).

LSTM Model: Inspired by the success of the RNN Encoder-Decoder in modeling sequential data [32] and good performance of LSTM to capture transition regularities of human movements since they have memory to learn the temporal dependence between observations [19], [33], we implement an Encoder-Decoder LSTM model which can learn general body motion as well as head motion patterns, and predict the future viewing direction and position based on the past traces. Fig. 7(a) shows the LSTM model we designed and used in our training, where first and second LSTM layers both consist of 60 LSTM units, and the fully connected layer contains 1 interconnected node. Note the

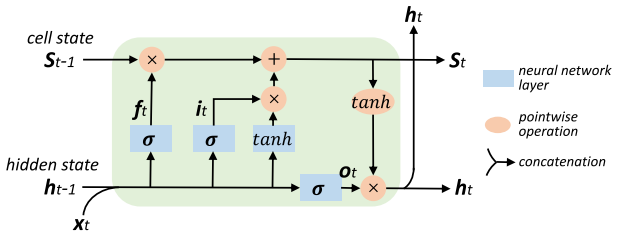

FIGURE 8. The structure of LSTM unit.

interconnected node refers to the general neuron-like processing unit $a = \phi(\sum_j w_j x_j + b)$, where x_j are the inputs to the unit, w_j are the weights, b is the bias, ϕ is the nonlinear activation function, and a is the unit's activation [34].

Our Encoder-Decoder LSTM model predicts what the motion speed will be for next time point, given the previous sequence of motion speed. The outputs are the values of predicted speed for next time point. Note that the settings including 60 LSTM units and 60 time points as window length are selected during experiments and proved to be good by empirical results. For the head and body motion prediction, we use the mean square error (MSE) as our loss function:

$$Loss = \frac{1}{|N_{train}|} \sum_{y \in S_{train}} \sum_{t=1}^{L} \left(y_t - \hat{y}_t\right)^2, \tag{2}$$

where $|N_{train}|$ is the number of total time steps of all trajectories on the train set S_{train}, and L is the total length of each corresponding trajectories. The proposed LSTM model learns parameters by minimizing the mean square error.

Specifically, encoder and decoder sections work as follows. Given the input sequence $\boldsymbol{X} = (\boldsymbol{x_1}, \ldots, \boldsymbol{x_t}, \ldots, \boldsymbol{x_T})$ with $\boldsymbol{x_t} \in \mathbb{R}^n$, where n is the number of driving series (e.g., dimension of feature representation), the encoder learns a mapping from $\boldsymbol{x_t}$ to $\boldsymbol{h_t}$ with

$$\boldsymbol{h_t} = f(\boldsymbol{h_{t-1}}, \boldsymbol{x_t}), \tag{3}$$

where $\boldsymbol{h_t} \in \mathbb{R}^m$ is the hidden state of the encoder at time t, m is the size of the hidden state, and f is a non-linear activation function of LSTM unit. As shown in Fig. 8, each LSTM unit has (i) a memory cell with the cell state $\boldsymbol{s_t}$, and (ii) three sigmoid gates to control the access to memory cell (forget gate $\boldsymbol{f_t}$, input gate $\boldsymbol{i_t}$ and output gate $\boldsymbol{o_t}$). We follow the LSTM structure from [32], [35]:

$$\boldsymbol{f_t} = \sigma\left(\boldsymbol{W_f}\left[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}\right] + \boldsymbol{b_f}\right), \tag{4}$$
$$\boldsymbol{i_t} = \sigma\left(\boldsymbol{W_i}\left[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}\right] + \boldsymbol{b_i}\right), \tag{5}$$
$$\boldsymbol{o_t} = \sigma\left(\boldsymbol{W_o}\left[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}\right] + \boldsymbol{b_o}\right), \tag{6}$$
$$\boldsymbol{s_t} = \boldsymbol{f_t} \odot \boldsymbol{s_{t-1}} + \boldsymbol{i_t} \odot \left(tanh\left(\boldsymbol{W_s}\left[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}\right] + \boldsymbol{b_s}\right)\right), \tag{7}$$
$$\boldsymbol{h_t} = \boldsymbol{o_t} \odot tanh(\boldsymbol{s_t}), \tag{8}$$

where $[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}] \in \mathbb{R}^{m+n}$ is a concatenation of the previous hidden state $\boldsymbol{h_{t-1}}$ and current input $\boldsymbol{x_t}$. $\boldsymbol{W_f}$, $\boldsymbol{W_i}$, $\boldsymbol{W_o}$, $\boldsymbol{W_s} \in \mathbb{R}^{m \times (m+n)}$ as well as $\boldsymbol{b_f}$, $\boldsymbol{b_i}$, $\boldsymbol{b_o}$, $\boldsymbol{b_s} \in \mathbb{R}^m$ are parameters to learn. Notations of σ and $\odot$ are the logistic sigmoid function and element-wise multiplication. After reading the end of input sequence sequentially and updating the hidden state as above, the hidden state of LSTM is a summary (i.e., encoded vector $\boldsymbol{c}$) of the whole input sequence. Subsequently, the decoder is trained to generate the target sequence $(y_1, \ldots, y_t, \ldots, y_T)$ by predicting y_t given hidden state $\boldsymbol{d_t}$ of LSTM units in decoder at timestep t. Note that $y_t \in \mathbb{R}$, and $\boldsymbol{d_t} \in \mathbb{R}^p$, where p is the size of the hidden state in decoder. The update of hidden state is denoted by

$$\boldsymbol{d_t} = f(\boldsymbol{d_{t-1}}, y_{t-1}, \boldsymbol{c}). \tag{9}$$

Since the nonlinear function is the LSTM unit function, similarly, $\boldsymbol{d_t}$ can be updated as:

$$\boldsymbol{f'_t} = \sigma\left(\boldsymbol{W'_f}\left[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}\right] + \boldsymbol{b'_f}\right), \tag{10}$$
$$\boldsymbol{i'_t} = \sigma\left(\boldsymbol{W'_i}\left[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}\right] + \boldsymbol{b'_i}\right), \tag{11}$$
$$\boldsymbol{o'_t} = \sigma\left(\boldsymbol{W'_o}\left[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}\right] + \boldsymbol{b'_o}\right), \tag{12}$$
$$\boldsymbol{s'_t} = \boldsymbol{f'_t} \odot \boldsymbol{s'_{t-1}} + \boldsymbol{i'_t} \odot \left(tanh\left(\boldsymbol{W'_s}\left[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}\right] + \boldsymbol{b'_s}\right)\right), \tag{13}$$
$$\boldsymbol{d_t} = \boldsymbol{o'_t} \odot tanh\left(\boldsymbol{s'_t}\right), \tag{14}$$

where $[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}] \in \mathbb{R}^{p+m+1}$ is a concatenation of the previous hidden state $\boldsymbol{d_{t-1}}$, decoder input y_{t-1}, and encoded vector $\boldsymbol{c}$. $\boldsymbol{W'_f}$, $\boldsymbol{W'_i}$, $\boldsymbol{W'_o}$, $\boldsymbol{W'_s} \in \mathbb{R}^{p \times (p+m+1)}$ as well as $\boldsymbol{b'_f}$, $\boldsymbol{b'_i}$, $\boldsymbol{b'_o}$, $\boldsymbol{b'_s} \in \mathbb{R}^p$ are parameters to learn. Subsequently, the output of the decoder is further fed to the fully connected layer.

MLP Model: Apart from the LSTM model, we propose to use an MLP [34] model presented in Fig. 7(b) to do motion prediction. Using the same representation and loss function described above, this model takes the motion speed during the latest 60 time points as input to predict the motion speed for next time point. The MLP model contains two fully-connected layers with 60 and 1 interconnected nodes respectively for training. The MLP model also learns parameters by minimizing the mean square error.

We build up single-task models including *LSTM* and *MLP* models for body motion and head motion speed in $x, y, z, \alpha, \beta, \gamma$-axis respectively. Given the current and previous speed traces, our predictive models can predict the speed for next time point and thus predict the viewing position b and viewing direction h for next time point (described in Section III-A).

2) MULTI-TASK MODEL

Motivated by achieving better body motion prediction to reduce the potential adverse effect on user experience caused by prediction error, we explore more models to predict body motion more accurately. Since single-task models in Section V-B1 predict body motion speed of each axis separately (using the information in one axis), we explore multi-task models to take advantage of body motion speed in all three axes to predict the body motion for each axis. We investigate a multi-task LSTM model as well as a multi-task MLP model, sharing some layers to determine common

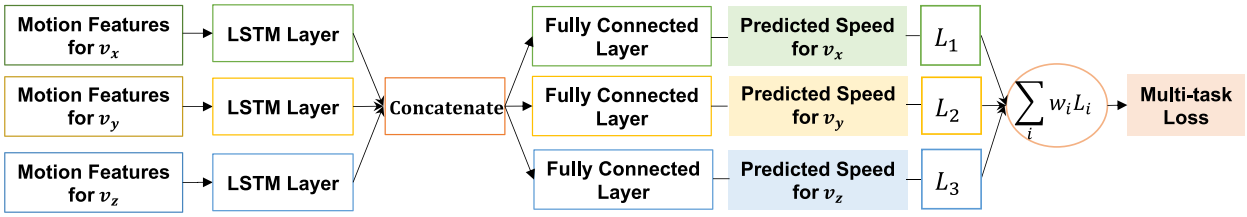

FIGURE 9. Multi-task LSTM model for body motion prediction.

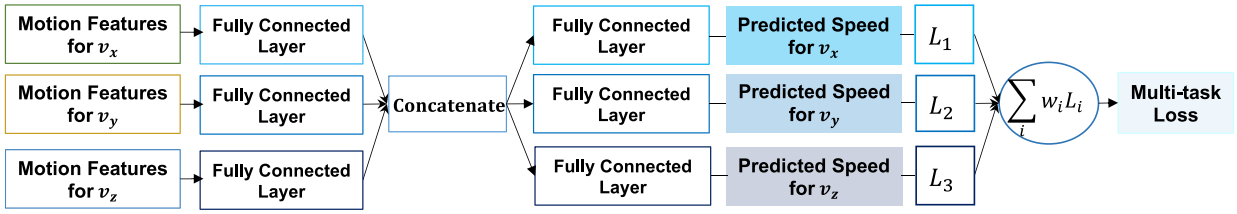

FIGURE 10. Multi-task MLP model for body motion prediction.

features between multiple tasks, where each task refers to the prediction for body motion speed in each axis (i.e., x, y, or z-axis).

Multi-Task LSTM Model: We implement a multi-task LSTM model that can learn a shared representation for body motion pattern, and predict the body motion speed (corresponding to viewing position) for the next time point based on the past traces. Fig. 9 shows our proposed multi-task LSTM model that we have designed and used for training, where the first three LSTM layers after the three motion features layers consist of 60, 60, and 60 LSTM units (Fig. 8) respectively, and the three fully-connected layers after a concatenate layer contain 1, 1, and 1 interconnected node. Our multi-task LSTM model predicts what the body motion speed in x, y and z-axis will be for the next time point, given the previous sequence of the body motion speed. The outputs are the values of predicted speed (i.e., v_x, v_y, v_z) for the next time point. Note that the settings including 60 LSTM units and 60 time points as window length are selected during experiments and proved to be good by empirical results. For the body motion prediction, we define the multi-task loss function as the weighted linear sum of the losses for each individual task:

$$L_{total} = \sum_i w_i L_i, \tag{15}$$

where w_i is the weight for individual task i and L_i is the single task loss function for individual task i (defined as the MSE, which is described before in the LSTM model Section V-B1). Specifically, as shown in Fig. 9, tasks 1, 2, and 3 refer to the prediction for body motion speed in x, y, and z-axis respectively. In our training, we use $w_1 = w_2 = w_3 = 0.333$ as the task weight setting based on good empirical performance and following theoretical observation. Specifically, for body motion prediction, the distance between actual body motion and predicted motion can be defined as

$$d = \sqrt{{d_x}^2 + {d_y}^2 + {d_z}^2}, \tag{16}$$

where d_x, d_y, d_z are the distance between actual body motion and predicted body motion in the x, y, z-axis respectively. Thus, the theoretical observation is that with the setting of $w_1 = w_2 = w_3$, minimizing the multi-task learning loss function for body motion prediction is equivalent to minimizing the square of distance d between the actual viewing position (obtained from body motion) and predicted viewing position. Note that the proposed multi-task LSTM and MLP models learn parameters by minimizing the multi-task loss function. Note that we have considered and conducted experiments to models of sharing LSTM layer and fully-connected layer between x, y, and z, but their performances are worse than the performance of our proposed model (Fig. 9).

Multi-Task MLP Model: Apart from the multi-task LSTM model, we also implement a multi-task MLP model for comparison to do body motion prediction. Using the same representation and multi-task loss function described above, this model also takes the body motion speed during the latest 60 time points in x, y, z-axis as input to predict the body motion speed in the next time point. Our proposed multi-task MLP model has a similar structure like the multi-task LSTM model, shown in Fig. 10, where the first three fully-connected layers after the three motion feature layers consist of 60, 60, and 60 interconnected nodes respectively, and the three fully-connected layers after a concatenate layer contain 1, 1, and 1 interconnected node. The multi-task MLP model predicts what the body motion speed in x, y, and z-axis will be for the next time point, given the previous sequence of the body motion. The outputs are values of predicted speed (i.e., v_x, v_y, v_z) for the next time point.

Given the current and previous speed traces, we build up our multi-task models including *multi-task LSTM* and *multi-task MLP* models to predict the body motion speed in three

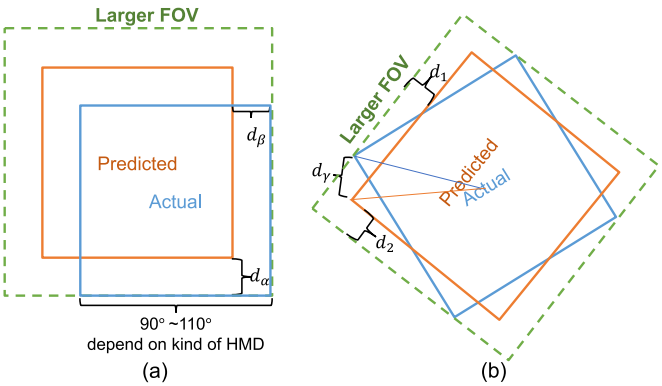

FIGURE 11. Selected FOV for two different types of relative positions between predicted FOV and actual FOV: (a) to address d_α and d_β, (b) to address d_γ.

axes for the next time point and thus predict the viewing position b for the next time point (described in Section III-A).

C. FOV SELECTION

After predicting body and head motion, we propose a *sliding window based FOV selection* method for pre-rendering, such that the selected FOV minimizes the effects of motion prediction error while also minimizing the selected FOV size. This method is also head motion prediction based since it selects the FOV size according to the estimated prediction error calculated by recent head motion prediction errors. Note that the method can only be applied to address head motion error since body motion prediction error can only be reduced by exploring better prediction models with higher precision (e.g., multi-task models presented in Section V-B2). Fig. 11 shows several different types of relative positions between predicted FOV and actual FOV, where blue square, orange square, and dashed green rectangle represent the actual FOV, predicted FOV, and pre-rendered larger FOV. The size of FOV is determined by the kind of HMD device, represented as the horizontal FOV of θ_h times vertical FOV of θ_v (e.g., $90° \times 90°$ for Samsung Gear VR, $110° \times 110°$ for HTC Vive).

Fig. 11(a) exhibits the angular distance between the actual and predicted FOVs in α and β-axis as d_α and d_β, with no angular distance in the γ-axis. We can see that the actual FOV can be covered by the pre-rendered larger FOV via increasing the horizontal FOV to $\theta_h + 2d_\beta$ and the vertical FOV to $\theta_v + 2d_\alpha$. Fig. 11(b) demonstrates the angular distance between the actual and predicted FOVs in the γ-axis as d_γ without any angular distance in α and β-axis. The actual FOV can be covered by the pre-rendered larger FOV via increasing the horizontal FOV to $\theta_h + 2d_2$ and the vertical FOV to $\theta_v + 2d_1$. Since $d_1 \leq d_\gamma$ and $d_2 \leq d_\gamma$ (due to Pythagoras theorem [36]) shown in Fig. 11(b), in this case, we can select a larger FOV by increasing the horizontal FOV to $\theta_h + 2d_\gamma$ and the vertical FOV to $\theta_v + 2d_\gamma$. This is the minimal increase in FOV size compared to predicted FOV such that it minimizes adverse effects due to head motion prediction error. Therefore, by selecting a larger FOV of $\theta_h + 2d_\beta + 2d_\gamma$ as horizontal FOV and $\theta_v + 2d_\alpha + 2d_\gamma$ as vertical FOV, the actual FOV can be completely covered, eliminating the adverse effect of head motion prediction error. The new selected horizontal FOV θ'_h and vertical FOV θ'_v can be represented as follows in Equations (17) and (18):

$$\theta'_h = \theta_h + 2d_\beta + 2d_\gamma, \tag{17}$$

$$\theta'_v = \theta_v + 2d_\alpha + 2d_\gamma. \tag{18}$$

Note that when performing the FOV selection task before pre-rendering the view, the exact head motion prediction error for the next frame is unknown. Hence, in our *sliding window based FOV selection* method, we propose to use a sliding window of n_w frames and n_w denotes the sliding window size. Then we define the estimated value of d_α, d_β, d_γ (i.e., $\hat{d_\alpha}$, $\hat{d_\beta}$, $\hat{d_\gamma}$) as the average head motion prediction error $\overline{d_\alpha}$, $\overline{d_\beta}$, $\overline{d_\gamma}$ of the past n_w frames (i.e., frames in the sliding window) so as to calculate the new selected horizontal FOV θ'_h and vertical FOV θ'_v.

D. PREDICTION ERROR DETERMINATION

In Fig. 3, when the head and body motion, as well as the controlling command, arrive at the edge device, the actual motion can be obtained immediately after the motion decision and there will be a prediction error determination comparing the actual motion with the prediction motion. We will see whether the head motion and body motion prediction error is within the thresholds using the following steps. For head motion, since we pre-render a larger FOV than actual FOV to reduce the effect of head motion prediction error. The determination of $d_{Head} \leq \epsilon_H$ will be achieved by checking whether $|\hat{d_\alpha} - d_\alpha|$, $|\hat{d_\beta} - d_\beta|$, $|\hat{d_\gamma} - d_\gamma|$ are all within a given threshold ϵ_1. For body motion, the determination of $d_{Body} \leq \epsilon_B$ will be achieved by checking whether d_x, d_y, d_z are all within a given threshold ϵ_2. To be sure that the actual view always within the pre-rendered view, the thresholds should be selected as low as possible. However, this will increase the probability of error determination, and hence doing the rendering and encoding again live, thereby increasing latency. On the other hand, setting this threshold too large may cause that the extreme case (e.g., having large head motion prediction error) cannot be efficiently identified. We empirically discuss different choices of given thresholds ϵ_1, ϵ_2 in Sections VI-C, VI-D, and VI-E.

VI. EXPERIMENTAL RESULTS

In this section, we describe our system setup, evaluation metrics, and experimental results.

A. SYSTEM SETUP AND DATASET

The system setup of our experiments is shown in Fig. 12, where the rendering edge device is an Intel Core i7 Quad-Core processor with GeForce RTX 2060. It is equipped with a WiGig card connecting with the HTC Vive's link box using a cable. This link box is within the user's room and transmits rendered frames in a video format from the rendering edge device to the HMD. On the user side, there are

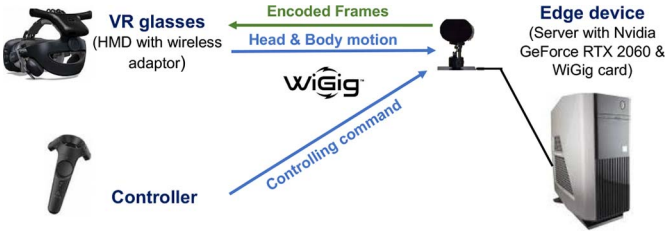

FIGURE 12. System setup.

TABLE 5. Dataset statistics

Virtual Application	*Session*	*#Samples for Training*	*#Samples for Testing*
Museum	*VM1*	41,600	10,354
	VM2	80,484	20,076
	VM3	195,197	48,754
Rome	*RM1*	24,912	6,183
	RM2	48,586	12,103
	RM3	280,540	70,091

the link box and two HTC lighthouse base stations in the room. Users were wearing an HTC Vive HMD equipped with Vive wireless adaptor [37] and using a controller if needed. Note the wireless adaptor and link box aim to transmit and receive the rendered frames using WiGig communications, while the HTC lighthouse base stations are set for capturing 6DoF motions (e.g., including head and body motion). The *walkable area* is around 3 m × 3 m of free space in our experiments, which cannot exceed 4.5 m × 4.5 m since the maximum distance between base stations is 5m [38]. All head and body motions on HMD were captured accurately using this HTC Lighthouse tracking system while the controller detected the user's controlling commands. For a software implementation, we implement our proposed techniques based on SteamVR SDK [39], OpenVR SDK [40] as well as the Unity game engine [41] for data collection, and use Keras [42] in Python for motion prediction.

We use 80% of the dataset for training the prediction model, and 20% for testing, ensuring the test data is from viewers which are different than those in training data. Table 5 presents the number of samples used as training data and testing data for each type of session of the two applications Virtual Museum and Virtual Rome (described in Section IV and listed in Table 3). Moreover, in our experiments, proposed *single-task LSTM* and *single-task MLP* models learn parameters by minimizing mean square error, and training is terminated after 50 epochs in our experiments, while proposed *multi-task LSTM* and *multi-task MLP* models learn parameters by minimizing multi-task loss function and training is terminated after 20 and 50 epochs respectively with a batch size of 32.

B. EVALUATION METRICS AND BASELINES

Evaluation Metrics: We choose several popular metrics in sequential modeling to evaluate the performance on our prediction task.

TABLE 6. Body motion prediction for Virtual Museum.

Session	*Model*	d_x (mm)		d_y (mm)		d_z (mm)	
		RMSE	***MAE***	***RMSE***	***MAE***	***RMSE***	***MAE***
VM1 (w/ Guidance; w/o Controller)	*Lin-A*	0.139	0.068	0.167	0.061	0.030	0.018
	Eql-A	0.079	0.037	0.096	0.033	0.021	**0.013**
	MLP	0.083	0.051	0.080	0.037	0.025	0.018
	LSTM	**0.061**	**0.035**	**0.074**	**0.030**	**0.019**	**0.013**
VM2 (w/o Guidance; w/o Controller)	*Lin-A*	0.094	0.045	0.099	0.041	0.048	0.021
	Eql-A	0.053	0.025	0.056	**0.023**	0.029	**0.013**
	MLP	0.044	0.029	0.047	0.030	0.032	0.015
	LSTM	**0.039**	**0.021**	**0.046**	0.029	**0.026**	**0.013**
VM3 (w/o Guidance; w/ Controller)	*Lin-A*	0.063	0.035	0.074	0.037	0.024	0.015
	Eql-A	0.036	**0.020**	0.042	0.022	0.017	0.011
	MLP	**0.032**	0.021	0.034	0.021	0.017	0.012
	LSTM	**0.032**	0.021	**0.033**	**0.019**	**0.015**	**0.010**

- *Root Mean Square Error (RMSE):*

$$RMSE = \sqrt{\frac{1}{|N_{test}|}\sum_{y\in S_{test}}\sum_{t=1}^{L}\left(y_t - \hat{y}_t\right)^2}, \tag{19}$$

- *Mean Absolute Error (MAE):*

$$MAE = \frac{1}{|N_{test}|}\sum_{y\in S_{test}}\sum_{t=1}^{L}\left(y_t - \hat{y}_t\right), \tag{20}$$

where $|N_{test}|$ is the number of total time steps of all trajectories on the test set S_{test}.

Baselines: We consider the following baselines to compare against the performance of our proposed model:

- *Linear Acceleration Model (Lin-A):* Following the work of [18]–[20], we compare against this linear regression model, which extrapolates trajectories with an assumption of linear acceleration. The Lin-A model employs the motion speed of the latest 3 time points to predict the expected motion speed.
- *Equal Acceleration Model (Eql-A):* The Eql-A model is our modified version of Lin-A, where we assume the acceleration is approximately equal during a small time interval (e.g., 22ms). The advantage of this modification is as follows: by employing a smaller number of time points, the acceleration estimated may approach more the actual value for the following 11ms, than is achieved by the Lin-A model. We implement the Eql-A model using motion speed of the latest 2 time points to predict the expected motion speed of the next time point.

C. PREDICTION ACCURACY

1) SINGLE-TASK MODEL

Tables 6, 7, 8, and 9 exhibit the results of our body motion and head motion prediction for the two applications respectively. Specifically, Tables 6 and 8 show the distance between actual and predicted body position in x, y, z-axis (denoted as d_x, d_y, d_z), while Tables 7 and 9 present the angular distance between actual and predicted head pose in α, β, γ-axis (denoted as $d_\alpha, d_\beta, d_\gamma$). Note that we use MSE as the loss function when doing training. In each table, we compare four models and can make the following observations:

TABLE 7. Head motion prediction for Virtual Museum.

Session	Model	d_α (′)		d_β (′)		d_γ (′)	
		RMSE	*MAE*	*RMSE*	*MAE*	*RMSE*	*MAE*
VM1 (w/ Guidance; w/o Controller)	*Lin-A*	0.64	0.34	0.96	0.43	0.48	0.21
	Eql-A	0.47	0.29	0.57	**0.27**	0.33	0.18
	MLP	0.51	0.35	0.77	0.48	0.40	0.27
	LSTM	**0.44**	**0.28**	**0.54**	0.30	**0.30**	**0.17**
VM2 (w/o Guidance; w/o Controller)	*Lin-A*	0.80	0.35	1.31	0.52	0.41	0.23
	Eql-A	0.49	**0.27**	0.78	**0.34**	0.32	0.19
	MLP	**0.47**	0.30	**0.64**	0.41	**0.31**	**0.18**
	LSTM	0.66	0.34	0.72	0.42	0.55	0.28
VM3 (w/o Guidance; w/ Controller)	*Lin-A*	0.61	0.35	1.38	0.61	0.33	0.21
	Eql-A	0.45	0.29	0.82	0.39	0.26	0.17
	MLP	**0.41**	**0.27**	**0.66**	**0.37**	**0.22**	**0.15**
	LSTM	0.48	0.30	0.99	0.55	0.28	0.17

TABLE 8. Body motion prediction for Virtual Rome.

Session	Model	d_x (mm)		d_y (mm)		d_z (mm)	
		RMSE	*MAE*	*RMSE*	*MAE*	*RMSE*	*MAE*
RM1 (w/ Guidance; w/o Controller)	*Lin-A*	0.174	0.086	0.299	0.084	0.046	0.022
	Eql-A	0.100	0.051	0.172	0.047	0.031	**0.017**
	MLP	0.118	0.075	0.098	0.062	**0.024**	0.018
	LSTM	**0.032**	**0.021**	**0.073**	**0.044**	**0.024**	0.019
RM2 (w/o Guidance; w/o Controller)	*Lin-A*	0.125	0.053	0.145	0.048	0.036	0.020
	Eql-A	0.074	0.032	0.085	0.030	**0.025**	**0.015**
	MLP	0.066	0.037	0.064	0.030	0.064	0.021
	LSTM	**0.058**	**0.030**	**0.065**	**0.032**	**0.025**	**0.015**
RM3 (w/o Guidance; w/ Controller)	*Lin-A*	0.074	0.041	0.077	0.041	0.034	0.019
	Eql-A	0.044	0.025	0.046	0.025	**0.023**	**0.013**
	MLP	0.040	0.025	0.041	0.026	0.077	0.040
	LSTM	**0.040**	**0.024**	**0.040**	**0.024**	**0.023**	**0.013**

TABLE 9. Head motion prediction for Virtual Rome.

Session	Model	d_α (′)		d_β (′)		d_γ (′)	
		RMSE	*MAE*	*RMSE*	*MAE*	*RMSE*	*MAE*
RM1 (w/ Guidance; w/o Controller)	*Lin-A*	0.71	0.47	1.32	0.61	0.40	0.27
	Eql-A	0.55	0.38	0.79	**0.39**	0.30	**0.21**
	MLP	0.55	0.38	0.80	0.49	0.30	**0.21**
	LSTM	**0.53**	**0.36**	**0.73**	0.47	**0.29**	0.22
RM2 (w/o Guidance; w/o Controller)	*Lin-A*	0.92	0.57	2.53	0.66	0.56	0.34
	Eql-A	0.66	0.43	1.48	**0.44**	0.39	0.26
	MLP	**0.63**	**0.42**	**1.34**	0.46	**0.37**	**0.25**
	LSTM	0.64	0.43	1.52	0.55	1.23	0.30
RM3 (w/o Guidance; w/ Controller)	*Lin-A*	0.88	0.50	1.57	0.72	0.44	0.27
	Eql-A	0.63	0.38	0.98	0.49	0.33	0.21
	MLP	**0.57**	**0.36**	**0.82**	**0.43**	**0.28**	**0.18**
	LSTM	0.60	0.39	0.89	0.52	0.35	0.25

- Tables 6 and 8, which report on the accuracy of body motion prediction, show that our LSTM model achieves smallest RMSE in each session and smallest MAE in most sessions except VM2 compared to Lin-A, Eql-A, and MLP models. It demonstrates the effectiveness of using our proposed LSTM model to predict body motion positions.
- Tables 7 and 9, which report on the accuracy of head motion prediction, show that while the LSTM model has smallest RMSE for session 1, the MLP model performs better (results in smaller RMSE) than other three models in sessions 2 and 3 for both the applications. Compared to session 1 (where users take a stroll about the room and have a relatively fixed trajectory), sessions 2 and 3 are more general and closer to normal 6DoF VR scenario. Thus, we can see that MLP is a more feasible model to do head motion prediction in general cases.

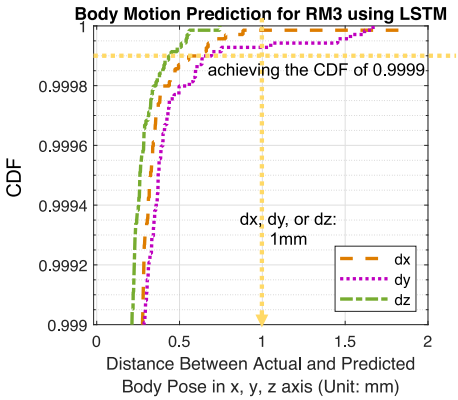

FIGURE 13. Body motion prediction error using the LSTM model in the RM3 session.

We can observe that (i) LSTM model achieves a better performance in every session of body motion prediction and session 1 of head motion prediction. These sessions have a relatively small range (e.g., body motion speed is mostly smaller than $\pm$1m/s), gradual variation and more regularity. (ii) MLP model performs better in sessions 2 and 3 of head motion prediction. These two sessions have a large value range (e.g., head motion can be up to $\pm$300°/s), quicker variation and more frequent fluctuations (e.g., head motion speed v_β has a large and abrupt change from $-$180°/s to 200°/s within 1s, shown in Fig. 6(e)). Note that although RMSE of head motion prediction achieved by MLP model is quite small, we still need to use proposed *FOV selection* method to address the possible challenging case (the extreme case where head motion prediction error is large) in head motion prediction, and minimize effects of motion prediction error while also minimizing selected FOV size.

Next, we study what the values of ϵ_2 should be in the prediction error determination technique (Section V-D), where the prediction motion is compared with actual motion when the head and body motion, as well as controlling command, arrive at the edge device. The determination of body motion prediction error is checking whether d_x, d_y, d_z are all within a given threshold ϵ_2. Fig. 13 presents the body motion prediction error using the LSTM model in the RM3 session. In Fig. 13, body motion prediction using the LSTM model achieves that around 99.99% (i.e., 0.9999) of time points satisfy the $d_x < 0.6$ mm, $d_y < 0.7$ mm, $d_z < 0.45$ mm. Thus, we can observe that if we set the given threshold ϵ_2 as 1 mm, less than 99.99% of time points can be determined as 'correct' for body motion prediction in the proposed system, meaning that there is less than 1 frame on average among 10,000 pre-rendered frames will be 'incorrect' while the rest of more than 9,999 pre-rendered frames will pass the body motion error determination successfully.

2) MULTI-TASK MODEL

Fig. 14(a) and (b) exhibit the results of our body motion prediction for two application sessions VM1 and VM3

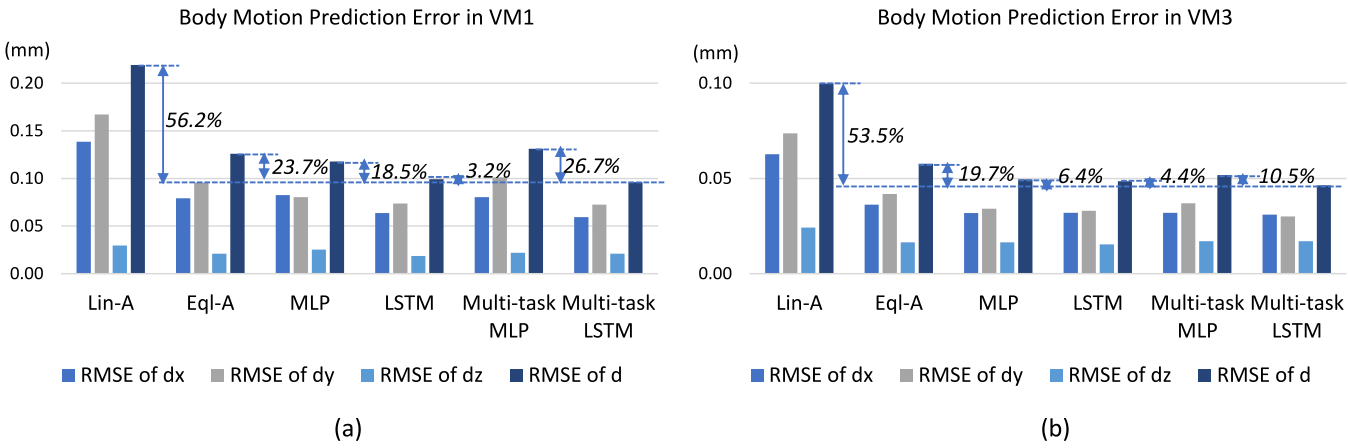

FIGURE 14. Body motion prediction error in VM1 and VM3 sessions comparing the multi-task LSTM model with other predictive models.

respectively. In each figure, we compare six models (single-task and multi-task models) and can make the following observations.

- Fig. 14(a) shows the RMSE of d for body motion prediction error in VM1, where we can see the *multi-task LSTM* model achieves 56.2%, 23.7%, 18.5%, 3.2%, 26.7% improvement compared to *Lin-A, Eql-A, MLP, LSTM, multi-task MLP* models. Our proposed *multi-task LSTM* model achieves the smallest RMSE of d (i.e., 0.096 mm) compared to other models.
- Fig. 14(b) presents the RMSE of d for body motion prediction error in VM3, where we can see the *multi-task LSTM* model achieves 53.5%, 19.7%, 6.4%, 4.4%, 10.5% improvement compared to *Lin-A, Eql-A, MLP, LSTM, multi-task MLP* models respectively. Our proposed *multi-task LSTM* model achieves the smallest RMSE of d (i.e., 0.046 mm) compared to other models. In Fig. 14(a)(b), the reason that the prediction error for body motion in VM1 is larger than VM3 is that users continuously walk without stopping by any place in VM1 while they tend to have less body motion and teleport to other place using the controller in VM3.
- Similarly, in other sessions such as RM3, for RMSE of d for body motion prediction error, the *multi-task LSTM* model achieves 46.6%, 11.9%, 37.5%, 1.3%, 5.9% improvement compared to *Lin-A, Eql-A, MLP, LSTM, multi-task MLP* models. The *multi-task LSTM* model still works better than other five models. Moreover, in some cases like VM2, the *multi-task LSTM* model achieves the same RMSE of d for body motion prediction error with *LSTM* model (i.e., *multi-task LSTM* model has 54.1%, 19.3%, 8.0%, 0%, 6.2% improvement in RMSE of d compared to *Lin-A, Eql-A, MLP, LSTM, multi-task MLP* models). The *multi-task LSTM* model has a smaller RMSE of d_y (i.e., 0.042 mm) compared to the *LSTM* model (i.e., 0.046 mm) as well as a larger RMSE of d_x and d_z. To achieve the smallest RMSE of d for body motion prediction error, in this case, we can consider using *multi-task LSTM* model to predict body motion in the y-axis and two single-task *LSTM* models to predict body motion in the x and z-axis, so that the RMSE of d for this combined models choice is 0.063 mm, smaller than 0.066 mm obtained by single-task *LSTM* models as well as *multi-task LSTM* model. Thus, we can always improve the performance by combing three trained models (*multi-task LSTM* with single-task *LSTM* models) if each of them has the smallest RMSE of d_x, d_y, and d_z to achieve the smallest RMSE of d.

D. RUNTIMES

Training and Prediction Times: Next, we briefly discuss the training times and inference times taken by our proposed prediction models on the edge device selected (Intel Core i7 Quad-Core processor with GeForce RTX 2060). Note that the training for a proposed model is done offline only once for a session with the training samples for that session, and the prediction (testing) is done online for new users, however one frame ahead to predict motion in advance. Hence, the training times do not affect the end-to-end latency of the system. Since the prediction is done for the user's head and body motion one frame ahead in advance, the prediction time as well as FOV selection time have to be less than 1ms (presented in Table 2).

The training time can be different for each session depending on the number of training samples used. In our experiments, for RM3 (which has the largest training data size among all sessions), the training times of one epoch for each *single-task LSTM* and *single-task MLP* models are around 150 seconds and 40 seconds respectively, while the training times of one epoch for the *multi-task LSTM* and *multi-task MLP* models are around 270 seconds and 45 seconds respectively. Thus for RM3, the training times for each *single-task LSTM* model and *multi-task LSTM* model are around 2 hours and 1.5 hours respectively, while the training times for each *single-task MLP* model and *multi-task MLP* model are around 0.55 hours and 0.6 hours respectively. The training times for all the other applications/sessions are lower than RM3 (e.g., for VM1, the training time for

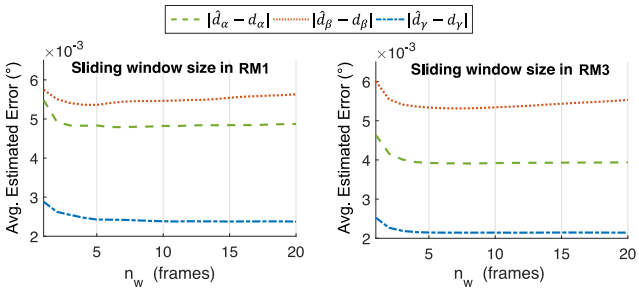

FIGURE 15. Average estimated error in α, β, γ-axis caused by different choices of sliding window size in RM1 and RM3 sessions.

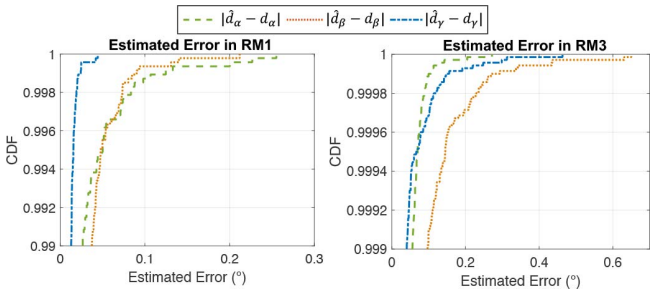

FIGURE 16. CDF of estimated error in α, β, γ-axis when the sliding window size $n_w = 5$ in RM1 and RM3 sessions.

each *single/multi-task LSTM/MLP* model is lower than 18 minutes).

The prediction (testing) times, on the other hand, only marginally varies between different applications and sessions. The average prediction times over all the application sessions considered in our experiments are the following: 0.09ms for each *single-task LSTM* model, 0.04ms for each *single-task MLP* model, 0.38ms for the *multi-task LSTM* model, and 0.04 ms for the *multi-task MLP* model. The above shows that our proposed head and body motion prediction models can execute in real-time on the edge node, and since they are well below the time of 1ms (described in the next paragraph), the predictions are feasible to be performed in advance for the user's head and body motion of next time point.

Total Times of Prediction and FOV Selection: Prediction time consists of head and body motion predictions: head motion prediction using three *single-task MLP* (i.e., 0.12 ms) and body motion prediction using either *option (a) multi-task LSTM* (i.e., 0.38 ms) or *option (b) multi-task LSTM* combined with one or two *single-task LSTM* models (i.e., $0.38 + 0.09$ ms or $0.38 + 0.18$ ms). Thus prediction time for head and body motions is 0.5 ms – 0.68 ms. FOV selection includes two parts: two simple addition operations to calculate horizontal FOV and vertical FOV in Equations (17) and (18), and three averaging operations to calculate the estimated value of d_α, d_β, d_γ (i.e., $\hat{d}_\alpha$, $\hat{d}_\beta$, $\hat{d}_\gamma$) as the average head motion prediction error $\overline{d_\alpha}, \overline{d_\beta}, \overline{d_\gamma}$ of the past n_w frames. FOV selection can be achieved in 0.00016ms when $n_w = 5$ (proved to be a good choice in Section VI-E). Thus, the total times of *prediction* and *FOV selection* is within 1 ms.

E. FOV SELECTION

Next, we evaluate the performance of our proposed *sliding window based FOV selection* method, described in Section V-C. As mentioned before, a sliding window of n_w frames (n_w denotes sliding window size) is used to estimate and obtain the new $d_\alpha, d_\beta, d_\gamma$, the new selected horizontal FOV θ'_h, and vertical FOV θ'_v before pre-rendering. We have described how to calculate the estimated value of d_α, d_β, d_γ (i.e., $\hat{d}_\alpha$, $\hat{d}_\beta$, $\hat{d}_\gamma$) in Section V-C. Fig. 15 shows the absolute value of the average estimated error in each axis (i.e., average $|\hat{d}_\alpha - d_\alpha|, |\hat{d}_\beta - d_\beta|, |\hat{d}_\gamma - d_\gamma|$) caused by different choices of sliding window size n_w (ranging from 1 to 20 frames) in RM1 and RM3 sessions. We can see that the smallest average estimated error can be achieved when $n_w = 5$ in both the sessions. The average estimated error can be as low as less than 5.5×10^{-3} degree in each axis, showing the efficiency of our approach. By achieving the low average estimated error, we can have a better estimation of $d_\alpha, d_\beta, d_\gamma$, so that can finally reduce the adverse effect of head motion prediction error.

We also study what the values of ϵ_1 should be in the prediction error determination technique (Section V-D), where the prediction motion is compared with actual motion when the head and body motion, as well as controlling command, arrive at the edge device. The determination of head motion prediction error is checking whether $|\hat{d}_\alpha - d_\alpha|, |\hat{d}_\beta - d_\beta|, |\hat{d}_\gamma - d_\gamma|$ are all within a given threshold ϵ_1. Fig. 16 shows the CDF of estimated error in α, β, γ-axis when the sliding window size $n_w = 5$ in RM1 and RM3 sessions respectively. Thus, we can observe that if a given threshold ϵ_1 is set as 1°, the estimated errors in each axis (i.e., $|\hat{d}_\alpha - d_\alpha|, |\hat{d}_\beta - d_\beta|, |\hat{d}_\gamma - d_\gamma|$) are smaller than ϵ_1 all the time for both RM1 and RM3 sessions, meaning that the head motion prediction is always 'correct' in this case.

For further performance evaluation, we compare our proposed *sliding window based FOV selection method* (Section V-D) with ***method (a)*** selected FOV is a fixed larger FOV, and ***method (b)*** using predicted FOV as the selected FOV. Specifically, for our proposed *sliding window based FOV selection* method, we use experimental results that average estimated in α, β-axis are 4.8×10^{-3} and 5.5×10^{-3} degrees respectively, shown in Fig. 15. For ***method (a)***, the fixed larger FOV has the size of $(110 + 60)° \times (110 + 60)°$ to cover potential prediction error within 30°. For ***method (b)***, the selected FOV is predicted FOV in size of $110° \times 110°$. As for evaluation metrics, we calculate (i) *overlap* of pre-rendered predicted view with actual FOV (e.g., $110° \times 110°$ for HTC Vive), and (ii) *overhead ratio* for pre-rendering computation and network bandwidth needed to transmit rendered FOV from edge device to VR glasses, defined as the *pre-rendering view size* divided by the *actual FOV size*. For instance, when the pre-rendering view size is $120° \times 120°$, the *overhead ratio* for pre-rendering computation and network

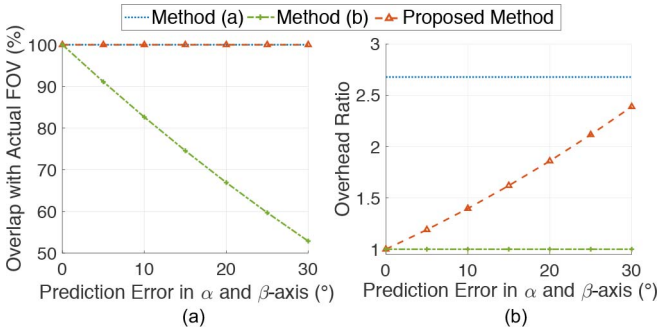

FIGURE 17. (a) Overlap of pre-rendered predicted view with actual FOV versus head motion prediction error in α and β-axis, (b) Overhead ratio versus head motion prediction error in α and β-axis.

bandwidth needed can be calculated as 1.19 according to our definition.

As described in Section VI-C, our proposed *sliding window based FOV selection method* can address the extreme cases where head motion prediction error is large. We compare the above three methods when dealing with head motion prediction error ranging from 0 to 30° in α and β-axis. Fig. 17(a)(b) show the overlap of pre-rendered predicted view with actual FOV and overhead ratio versus head motion prediction error in the α and β-axis (the coordinate of head motion shown in Fig. 4(c)). Note that when the x-axis of Fig. 17(a)(b) equals to 10°, we consider the situation of head motion prediction error in α and β-axis (i.e., d_α, d_β) are both 10° and no head motion prediction error in the γ-axis (i.e., d_γ). In Fig. 17, we can see that our proposed sliding window based FOV selection method achieves (i) the overlap with actual FOV as high as 99.991% (which is close to 100% achieved by *method (a)* and better than *method (b)*), and (ii) the corresponding overhead ratio is always smaller than *method (a)*.

For example, in our experiments, we observe that when the prediction errors in α and β-axis equal to 5°, our proposed *sliding window based FOV selection* method achieves an *overhead ratio* of 1.19, compared to an *overhead ratio* of 2.39 for *method (a)*, which corresponds to around 50% reduction of *overhead ratio* and 47% saving of bitrates (bandwidth needed) compared to *method (a)*. Thus, the high overlap with actual FOV and small overhead ratio illustrate that our proposed *sliding window based FOV selection* method has a good user experience (almost no miss of actual FOV) and low *overhead ratio* for pre-rendering computation as well as network bandwidth needed to transmit rendered FOV from edge device to VR glasses, compared to *methods (a)* and *(b)*.

F. EFFECT ON USER EXPERIENCE

To evaluate the effect on user experience caused by the prediction error between the actual view and the predicted view which will be pre-rendered and delivered to the user, we propose following metric. Assume that we have two views V_1 and V_2 in the RGB format. Firstly, we convert the RGB

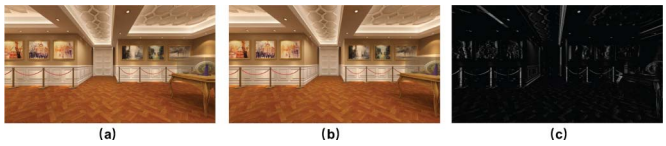

FIGURE 18. (a) Actual user's view; (b) Predicted user's view with x-axis error $\Delta x = 0.1m$; (c) I_{dif} obtained from views in (a) (b).

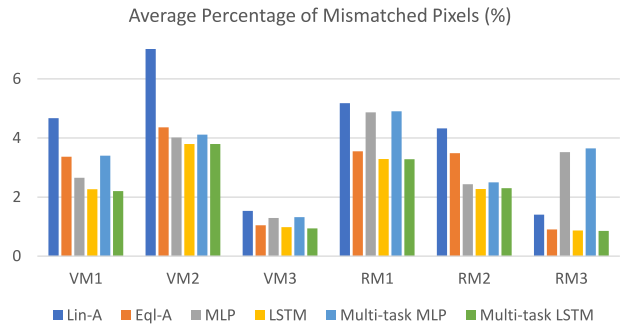

FIGURE 19. The average *percentage of mismatched pixels* for different models during each session.

images (V_1 and V_2) to grayscale intensity images I_1 and I_2 by eliminating the hue and saturation information while retaining the luminance [43]. For each pixel i in the grayscale intensity images, we calculate the difference between the two intensity images, I_{dif}, as follows.

$$I_{dif}(i) = \begin{cases} I_1(i) - I_2(i), & \text{if } I_1(i) \geq I_2(i) \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

Note that we set the I_{dif} as 0 in the second case of Equation (21), because otherwise the motion change of the same object will be presented in I_{dif} twice: positive and negative respectively. Thus we only keep the positive one (i.e., the first case in Equation (21)) to evaluate the difference between the two views. Fig. 18 presents an example of two views and the corresponding I_{dif}. In Fig. 18(c), we can see that most of pixels in the view have the intensity value of 0 while the residual pixels have intensity values larger than or equal to 1. We define the *percentage of mismatched pixels* as

$$R_{dif} = \frac{N_{dif}}{N_{frame}}, \tag{22}$$

where N_{dif} represents the number of pixels which have difference in grayscale intensity and N_{frame} is the total number of pixels per frame.

Fig. 19 illustrates the average *percentage of mismatched pixels* caused by body motion prediction error. Due to the large number for each session in the test dataset, we calculate this value by doing body motion prediction and rendering corresponding predicted as well as actual views for 300 randomly selected samples from test data for every session. Fig. 19 demonstrates that compared to other models in each session, our proposed *multi-task LSTM* and *LSTM* models achieves less adverse effect on user experience caused by the body prediction error (denoted with green and yellow bars). Using the *multi-task LSTM* model, the average *percentage*

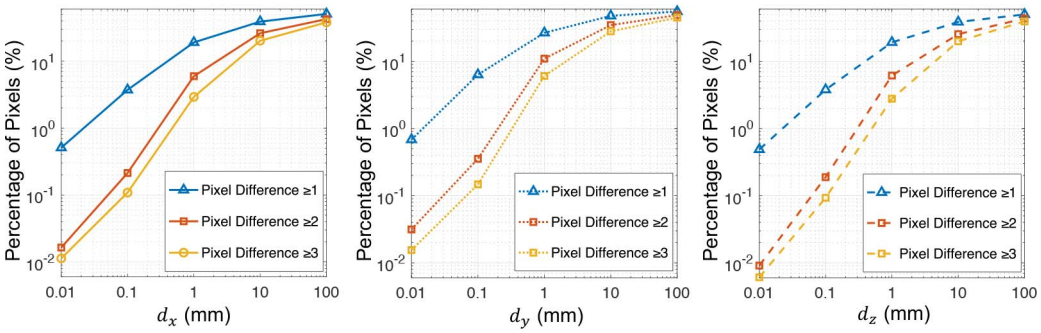

FIGURE 20. The percentage of pixels having pixel difference for versus d_x, d_y, and d_z.

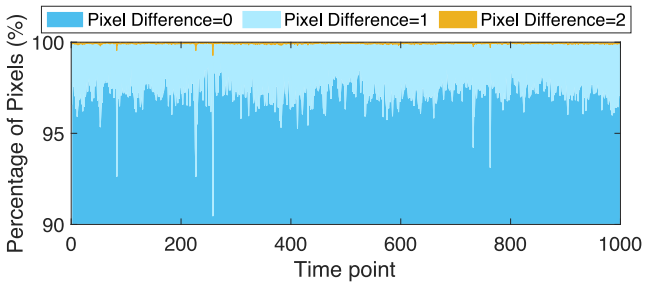

FIGURE 21. The percentage of pixels versus time points achieved by the multi-task LSTM model during VM2.

of mismatched pixels can be smaller than 1% in both VM3 and RM3 sessions.

Next, we define the *percentage of pixels* as

$$R_p = \frac{N_p}{N_{frame}}, \tag{23}$$

where N_p represents the number of pixels and N_{frame} is the total number of pixels per frame. For each pixel, it can have a value of grayscale intensity difference in I_{dif} (which equals to a integer between 0 to 255). Apart from discussion in Section VI-C1, by using this metric of the *percentage of pixels*, we further study what the values of ϵ_2 should be in the prediction error determination technique (Section V-D), where the prediction motion is compared with actual motion. The determination of body motion prediction error is checking whether d_x, d_y, d_z are all within a given threshold ϵ_2. Fig. 20 shows an example of the percentage of pixel versus different d_x, d_y, and d_z in Virtual Museum application. We can observe that when $d_x = 1$ mm, $d_y = 1$ mm, $d_z = 1$ mm, the *percentage of pixels* is more than 97%, 95%, 97% respectively corresponding to *pixel difference* less than 3 (pixel difference $= 0$, 1, or 2), which means $\epsilon_2 = 1$ mm can be a good choice for the body prediction error determination.

Furthermore, for our proposed multi-task LSTM model, we evaluate the adverse effect caused by body motion prediction error using metric of the *percentage of pixels*. Fig. 21 presents the *percentage of pixels* versus the time points achieved by the multi-task LSTM model during the VM2 session. We can see that most of the time, the *percentage of pixels* for pixel difference $= 0$ is larger than 96% (equivalent to the average *percentage of mismatched pixels* smaller than 4%). The average *percentage of pixels* for pixel difference $= 0, 1, 2, 3, 4, 5$ equals to 97.43%, 2.49%, 0.03%, 0.009%, 0.006%, 0.002% respectively, illustrating that the difference between actual view and predicted view is very small. Thus, our proposed multi-task LSTM model performs well in terms of small adverse effect caused by body motion prediction error.

VII. CONCLUSION AND FUTURE WORK

In this article, we propose a head and body motion prediction model for 6DoF VR applications, to enable predictive pre-rendering using edge intelligence and thus address latency challenge in edge computing-based 6DoF VR. We present a multi-task LSTM model and an MLP model to learn general head and body motion patterns and predict the future viewing direction and position based on past traces. We also develop a FOV selection technique for pre-rendering a larger FOV to reduce head motion prediction error and the motion error determination technique as part of the system mechanism. Our method shows good performance on a real motion trace dataset with high precision.

Our planned future work includes (i) further development and evaluation of the proposed edge-based predictive pre-rendering approach from latency perspectives, (ii) performing subjective studies to understand and quantify user experience using our proposed approach, (iii) further experiments predicting more time points and pre-delivering from edge to HMD, and (iv) considering multiple users and more possible gaming effects of the controller in applications, so as to address more challenging latency scenario. We consider applying our approach to more VR applications to show the feasibility of our approach. We also plan to study and develop predictive models for hand motion obtained from controllers to enable more complete 6DoF immersive experiences.

REFERENCES

[1] C. Wiltz. (Apr. 2017). *Five Major Challenges for VR to Overcome*. [Online]. Available: https://www.designnews.com/electronics-test/5-major-challenges-vr-overcome/187151205656659/
[2] L. Cherdo. (2019). *Types of VR Headsets*. [Online]. Available: https://www.aniwaa.com/guide/vr-ar/types-of-vr-headsets/
[3] Oculus. (2019). *Oculus Rift*. [Online]. Available: https://www.oculus.com/rift/
[4] HTC. (2019). *HTC Vive*. [Online]. Available: https://www.vive.com/us/

[5] HTC. (2019). *HTC Focus*. [Online]. Available: https://www.vive.com/cn/product/vive-focus-en/
[6] Oculus. 2019. *Oculus Go*. [Online]. Available: https://www.oculus.com/go/
[7] Samsung. (2019). *Samsung Gear VR*. [Online]. Available: https://www.samsung.com/us/mobile/virtual-reality/
[8] Google. (2019). *Google Daydream*. [Online]. Available: https://vr.google.com/daydream/
[9] X. Hou, Y. Lu, and S. Dey, "Wireless VR/AR with edge/cloud computing," in *Proc. Int. Conf. Comput. Commun. Netw.*, 2017, pp. 1–8.
[10] Qualcomm. (2016). *Whitepaper: Making Immersive Virtual Reality Possible in Mobile*. [Online]. Available: https://www.qualcomm.com/media/documents/files/whitepaper-making-immersive-virtual-reality-possible-in-mobile.pdf
[11] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive view generation to enable mobile 360-degree and VR experiences," in *Proc. Virtual Reality Augmented Reality Netw.*, 2018, pp. 20–26.
[12] X. Hou, J. Zhang, M. Budagavi, and S. Dey, "Head and body motion prediction to enable mobile VR experiences with low latency," in *Proc. Global Commun. Conf.*, 2019, pp. 1–7.
[13] K. Boos, D. Chu, and E. Cuervo, "Flashback: Immersive virtual reality on mobile devices via rendering memoization," in *Proc. Int. Conf. Mobile Syst. Appl. Services*, 2016, pp. 291–304.
[14] L. Liu *et al.*, "Cutting the cord: Designing a high-quality untethered VR system with low latency remote rendering," in *Proc. Int. Conf. Mobile Syst. Appl. Services*, 2018, pp. 68–80.
[15] Z. Lai, Y. C. Hu, Y. Cui, L. Sun, and N. Dai, "Furion: Engineering high-quality immersive virtual reality on today's mobile devices," in *Proc. Int. Conf. Mobile Comput. Netw.*, 2017, pp. 409–421.
[16] S. Shi, V. Gupta, M. Hwang, and R. Jana, "Mobile VR on edge cloud: A latency-driven design," in *Proc. Multimedia Syst. Conf.*, 2019, pp. 222–231.
[17] Y. Li and W. Gao, "DeltaVR: Achieving high-performance mobile VR dynamics through pixel reuse," in *Proc. Int. Conf. Inf. Process. Sensor Netw.*, 2019, pp. 13–24.
[18] X. Yun and E. R. Bachmann, "Design, implementation, and experimental results of a quaternion-based Kalman filter for human body motion tracking," *IEEE Trans. Robot.*, vol. 22, no. 6, pp. 1216–1227, Dec. 2006.
[19] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social LSTM: Human trajectory prediction in crowded spaces," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 961–971.
[20] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social GAN: Socially acceptable trajectories with generative adversarial networks," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2255–2264.
[21] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 4674-4683.
[22] J. Butepage, M. J. Black, D. Kragic, and H. Kjellstrom, "Deep representation learning for human motion prediction and classification," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 1591–1599.
[23] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *Proc. AllThingsCellular*, 2016, pp. 1–6.
[24] C. Fan, J. Lee, W. Lo, C. Huang, K. Chen, and C.-H. Hsu, "Fixation prediction for 360 video streaming to head-mounted displays," in *Proc. ACM SIGMM Workshop Netw. Oper. Syst. Support Digit. Audio Video*, 2017, pp. 67–72.
[25] Y. Bao, H. Wu, T. Zhang, A. A. Ramli, and X. Liu, "Shooting a moving target: Motion-prediction-based transmission for 360-degree videos," in *Proc. BigData*, 2016, pp. 1161–1170.
[26] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proc. Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 7482–7491.
[27] L. Zhao, Q. Sun, J. Ye, F. Chen, C.-T. Lu, and N. Ramakrishnan, "Multi-task learning for spatio-temporal event forecasting," in *Proc. SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2015, pp. 1503–1512.
[28] B. van Amsterdam, M. J. Clarkson, and D. Stoyanov, "Multi-task recurrent neural network for surgical gesture recognition and progress prediction," 2020. [Online]. Available: arXiv:2003.04772.
[29] Unity. (2019). *Virtual Museum*. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/museum-117927
[30] Unity. (2019). *Virtual Rome*. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/landscapes/rome-fantasy-pack-ii-111712
[31] R. W. Schafer, "What is a savitzky-golay filter? [Lecture notes]," *IEEE Signal Process. Mag.*, vol. 28, no. 4, pp. 111–117, 2011.
[32] K. Cho *et al.*, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," 2014. [Online]. Available: arXiv:1406.1078.
[33] J. Liu, A. Shahroudy, D. Xu, and G. Wang, "Spatio-temporal LSTM with trust gates for 3D human action recognition," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 816–833.
[34] R. Grosse. (2019). *Lecture 5: Multilayer Perceptrons*. [Online]. Available: https://www.cs.toronto.edu/mren/teach/csc411_19s/lec/lec10_notes1.pdf
[35] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," 2014. [Online]. Available: arXiv:1409.2329.
[36] Wikipedia. (2020). *Pythagoras Theorem*. [Online]. Available: https://en.wikipedia.org/wiki/Pythagorean_theorem
[37] HTC. (2019). *HTC Vive Wireless Adaptor*. [Online]. Available: https://www.vive.com/us/wireless-adapter/
[38] Valve. (2019). *SteamVR FAQ*. [Online]. Available: https://support.steampowered.com/kb/_article.php?ref=7770-WRUP-5951
[39] Valve. (2018). *SteamVR SDK*. [Online]. Available: https://valvesoftware.github.io/steamvr_unity_plugin/
[40] Valve. (2018). *OpenVR SDK*. [Online]. Available: https://github.com/ValveSoftware/openvr/
[41] U. Technologies. (2019). *Unity*. [Online]. Available: https://unity3d.com/
[42] (2019). *Keras*. [Online]. Available: https://keras.io/
[43] MATLAB. (2019). *Convert RGB Image or Colormap to Grayscale*. [Online]. Available: https://www.mathworks.com/help/matlab/ref/rgb2gray.html/

XUESHI HOU (Student Member, IEEE) received the B.Eng. degree in electronic engineering from Tsinghua University, Beijing, China, in 2015. She is currently pursuing the Ph.D. degree with the University of California at San Diego, La Jolla, CA, USA.

Her research interests include multimedia, virtual reality, mobile computing, and wireless communications.

SUJIT DEY (Fellow, IEEE) received the Ph.D. degree in computer science from Duke University in 1991.

He is currently a Professor with the Department of Electrical and Computer Engineering, the Director of the Center for Wireless Communications, and the Director of the Institute for the Global Entrepreneur with the University of California at San Diego, San Diego. He is the Head of the Mobile Systems Design Laboratory, developing innovative and sustainable edge computing, networking and communications, multimodal sensor fusion, and deep learning algorithms and architectures to enable predictive personalized health, immersive multimedia, and smart transportation applications. He has created inter-disciplinary programs involving multiple UCSD schools as well as community, city and industry partners; notably the Connected Health Program in 2016 and the Smart Transportation Innovation Program in 2018. In 2017, he was appointed as an Adjunct Professor with the Rady School of Management, and the Jacobs Family Endowed Chair in Engineering Management Leadership. He served as the Faculty Director of the von Liebig Entrepreneurism Center from 2013 to 2015, and as the Chief Scientist with Mobile Networks, Allot Communications from 2012 to 2013. In 2015, he co-founded igrenEnergi, providing intelligent battery technology and solutions for EV mobility services. He founded Ortiva Wireless in 2004, where he served as its founding CEO and later as CTO and Chief Technologist till its acquisition by Allot Communications in 2012. Prior to Ortiva, he served as the Chair of the Advisory Board of Zyray Wireless till its acquisition by Broadcom in 2004, and as an advisor to multiple companies, including ST Microelectronics and NEC. Prior to joining UCSD in 1997, he was a Senior Research Staff Member with NEC C&C Research Laboratories, Princeton, NJ. He has coauthored more than 250 publications, and a book on low-power design. He holds 18 U.S. and two international patents, resulting in multiple technology licensing and commercialization.

Prof. Dey has been a recipient of the nine IEEE/ACM Best Paper Awards, and has Chaired multiple IEEE conferences and workshops. For his list of publications, please see http://esdat.ucsd.edu/publications.