To appear in 2019 IEEE Global Communications Conference (GLOBECOM'19), Waikoloa, HI, USA, Dec. 2019.

Head and Body Motion Prediction to Enable Mobile VR Experiences with Low Latency

Xueshi Hou[1], Jianzhong Zhang[2], Madhukar Budagavi[2], Sujit Dey[1]
[1]University of California, San Diego [2]Samsung Research America
x7hou@ucsd.edu, {jianzhong.z, m.budagavi}@samsung.com, dey@ece.ucsd.edu

***Abstract*—As virtual reality (VR) applications become popular, the desire to enable high-quality, lightweight and mobile VR leads to various edge/cloud-based techniques. This paper introduces a predictive pre-rendering approach to address the ultra-low latency challenge in edge/cloud-based six Degrees of Freedom (6DoF) VR. 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. In our approach, the predictive view is rendered in advance based on the predicted viewing direction and position, leading to a reduction in latency. The key to achieving this efficient predictive pre-rendering approach is to predict the head and body motion accurately using past head and body motion traces. We develop a deep learning-based model and validate its ability using a dataset of over 840,000 samples for head and body motion.**

***Index Terms*—Virtual reality, video streaming, six Degrees of Freedom (6DoF).**

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/AR experience on the mobile HMDs using bandwidth constrained mobile networks, while satisfying the ultra-low latency requirements.

Currently used HMDs can be divided into several categories [2]: PC VR, standalone VR, 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 lower 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/cloud-based solution. By performing the rendering on edge/cloud servers and streaming videos to users, we can complete the heavy computational tasks on the edge/cloud server and thus enable mobile VR with lightweight VR glasses. The most challenging part of this solution are 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/cloud-based VR system includes following parts: time of transmitting sensor data from HMD to server, time of rendering (and encoding) on server, time of transmitting rendered video from server to HMD, and time of (decoding and) displaying the view on 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 edge/cloud-based VR system, it is extremely challenging to meet this requirement.

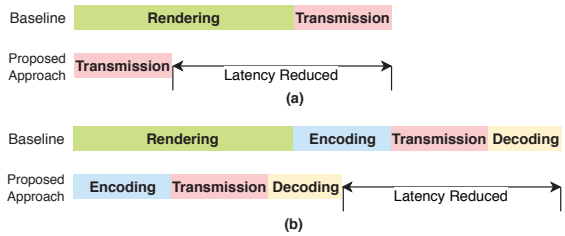


Fig. 1. Illustration of rendering and streaming pipeline: (a) Without encoding and decoding; (b) With encoding and decoding.

Motivated by the latency challenge, in this paper, we propose a novel approach of enabling mobile VR with prediction for head and body motions to satisfy ultra-low latency requirement. If we can predict head and body motion of users in the near future, we can do predictive pre-rendering on the edge device and then stream (even pre-deliver) predicted view to the HMD. Note that both the 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 module on HMD to buffer the predicted view and determine whether the predicted viewing position and direction are correct; (ii) transmits extra content when prediction is inaccurate. 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 server to HMD). The key to achieving this efficient predictive pre-rendering approach is solving the problem of motion prediction stated in Section III.A.

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 are addressing the problem of both head and body motion prediction in six Degrees of Freedom (6DoF) VR applications. Compared to 360-degree videos and 3DoF VR (support only head motion), 6DoF VR supports both head and body motion. 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 larger field of view (FOV) with high quality or rendering larger FOV. However, 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 paper will explore the feasibility of doing motion prediction with high precision in 6DoF VR, and its main contributions can be summarized as follows:

- For 6DoF VR applications, we propose a new *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 deep learning 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 motion traces from VR applications, we show the feasibility of our long short-term memory (LSTM) model for body motion prediction and multi-layer perceptron (MLP) model for head motion prediction with high precision.
- To the best of our knowledge, we are the first to come up with this predictive pre-rendering idea 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.

The rest of the paper is organized as follows. §2 reviews related work. §3 presents the system overview and problem definition. §4 describes our dataset. The methodology for head and body motion prediction is described in §5. We present our experimental results in §6, and conclude our work in §7.

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 [12]–[14] explore solutions to enable lightweight and mobile VR experiences, and improve the performance of current VR system. To provide high-quality VR on mobile device, [12] presents a pre-rendering and caching design called FlashBack, which pre-renders all possible views for different positions as well as orientations, stores them on a local cache, and delivers frames on demand according to current position and orientations. Due to infinite choices for different position, they propose to only render the view at each 3D grid point (grid density is 2-5cm). This may bring high inaccuracy and error to the VR system, leading to negative user experiences. Pre-caching all possible views will also cause overwhelming storage overhead (e.g. 50GB for an app). In contrast, our method aims to do a predictive pre-rendering based on head and body motion predictions, thus eliminating the above error as well as overwhelming storage overhead. [13] introduces a parallel rendering and streaming mechanism to reduce the add-on streaming latency, by pipelining the rendering, encoding, transmission and decoding procedures. Their method focuses on minimizing streaming latency, thus the latency for rendering part remains the same with traditional method. [14] presents a collaborative rendering method to reduce overall rendering latency by offloading costly background rendering to a server and only performing foreground rendering on the mobile device. This method reduces the rendering latency to some extent. In contrast, our method propose to pre-render based on head and body motion predictions, reducing the latency of rendering more drastically.

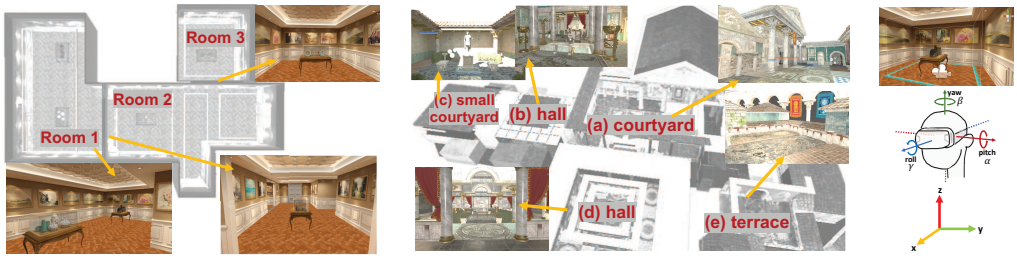
Human Motion Prediction: Learning statistical models of human motion is challenging due to the stochastic nature of human movement to explore the environment, and many work [15]–[19] propose methods to address it. Based on classical mechanics, there are some studies [15]–[17] showing the efficiency of linear acceleration model (Lin-A) by doing motion prediction or estimation with assumption of linear acceleration, especially in a small time interval (e.g. order of tens of milliseconds). [15] describes a good performance of a simple first-order linear motion model for tracking human limb segment orientation, and [16], [17] reveal acceptable results when employing the linear model as a baseline to predict human trajectory. Meanwhile, deep learning approaches [16]–[19] for human body prediction have also achieved remarkable accomplishments. Specifically, [16], [17] propose their LSTM models to predict human future trajectories, but their models aim to learn general human movement from massive number of videos and the corresponding precision of predicted position is much lower than needed for pre-rendering in VR scenarios. [18], [19] propose various recurrent neural network (RNN) models for human motion prediction to learn human kinematics from skeletal data. But these models are complex because of being designed to learn the patterns from a series of skeletal data and predict as long as 80ms ahead.

Moreover, [20]–[23] 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 high-precision human body and head motion prediction in 6DoF VR applications for the purpose of pre-rendering.

III. System Overview

In this section, we describe our system overview. In our proposed system, a user's head motion, body motion as well as other controlling commands will firstly be sent to the edge device, which performs the *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; (ii) do pre-rendering based on the predicted viewing position and direction; (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 server before it is needed; for the latter, latency remains the same with the conventional method of streaming from the edge server. 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.

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



(a) (b) (c)

Fig. 2. 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.

choices has tradeoffs, this paper 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 predictive pre-rendering approach, 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 a video and delivered to users, or directly streamed to users as a raw video. Specifically, we describe the problem formulation for motion prediction below.

A. Problem Formulation

Definition 1: (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 $S(t_w, t_{t+n-1}) = q_{t_w} q_{t_{w+1}} \cdots q_{t_{w+n-1}}$.

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.*, $S(t_w, t_{t+n-1}) = q_{t_w} q_{t_{w+1}} \cdots q_{t_{w+n-1}}$;
- Output: predicted spatiotemporal point $\widehat{q_{t_{w+n}}}$ at time point t_{w+n};

In this paper, 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.

IV. Dataset

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

To investigate head and body prediction in 6DoF VR applications, we conduct our study on a real motion trace dataset we collected from over 20 users using HTC Vive to experience two 6DoF VR applications called Virtual Museum [24] and Virtual Rome [25] in our laboratory. The hardware setup used will be described in Section VI. The trace consists of 840,000 sample points of head and body motion data collected from the users. Fig. 2(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. *Walkable area* is restricted 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. 2(c) uses light blue lines to show the boundary of *walkable area* in the VR. As shown in Table I, we set three sessions respectively for each application: (i) in session 1, users are given a rough guidance of taking a stroll about the room at the beginning of session, without controller in their hand; (ii) in session 2, users walk around freely in the room, without 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 *walkable area* in VR also changes accordingly.

TABLE I
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			✔			✔

TABLE II
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)

Motion files include the user ID, session timestamp, euler angles for 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. 2(c) exhibit the coordinates for head pose using euler angles and for body pose using position. To depict key characteristics of the head motion and viewpoint changes in the dataset quantitatively, we offer the following definitions.

Definition 2: (Head Motion Vector): The corresponding head poses at time points t_1 and t_2 (where $t_1 < t_2$) can 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 $(\triangle\alpha, \triangle\beta, \triangle\gamma)$, *i.e.*, $(\alpha(t_2) - \alpha(t_1), \beta(t_2) - \beta(t_1), \gamma(t_2) - \gamma(t_1))$.

Definition 3: (Head Motion Speed): Head motion speed v_{head} is defined as the distance the head moved divided by time, i.e., $v_{head} = (\frac{\triangle\alpha}{t_2 - t_1}, \frac{\triangle\beta}{t_2 - t_1}, \frac{\triangle\gamma}{t_2 - t_1})$.

Definition 4: (Body Motion Vector): The corresponding body poses at time points t_1 and t_2 (where $t_1 < t_2$) can are denoted by $(x(t_1), y(t_1), z(t_1))$ and $(x(t_2), y(t_2), z(t_2))$ respectively. Head motion vector $(\triangle x, \triangle y, \triangle z)$, *i.e.*, $(x(t_2) - x(t_1), y(t_2) - y(t_1), z(t_2) - z(t_1))$.

Definition 5: (Body Motion Speed): Body motion speed v_{body} is defined as the distance the body moved divided by time, i.e., $v_{body} = (\frac{\triangle x}{t_2 - t_1}, \frac{\triangle y}{t_2 - t_1}, \frac{\triangle z}{t_2 - t_1})$.

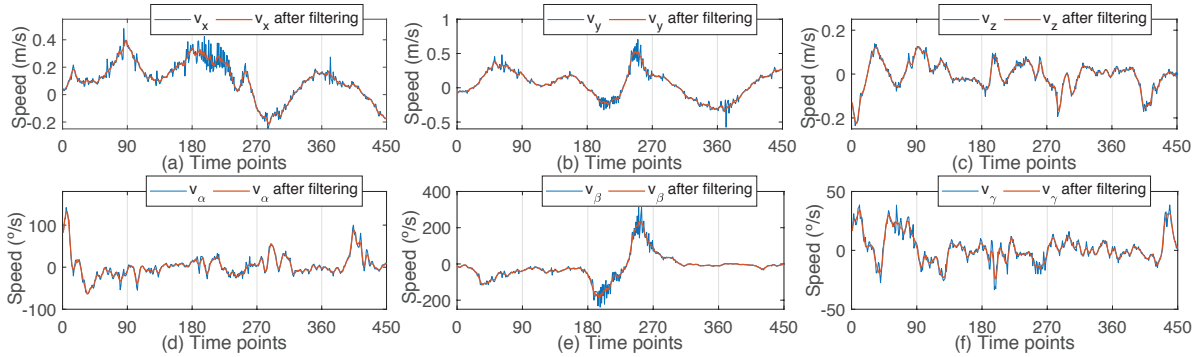


Fig. 3. Motion speed obtained before and after the preprocess: (a)(b)(c) for body motion; (d)(e)(f) for head motion.

Table II 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.

V. Our Approach

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

Preprocess: We aim to remove noise within head and body motion in the preprocessing procedure. We first calculate head motion speed and body motion speed for each time point. Fig. 3 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 Virtual Museum. 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 error from HTC Vive HMD and base stations. This noise is easy to identify 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 [26] because of its efficiency and high speed. This filter approximates (i.e. least-square fitting) the underlying function within the moving window by a polynomial of higher order. The blue and red lines in Fig. 4 show the speed before and after the preprocessing. We can see the noise is eliminated after the filtering.

Predictive Model: For 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. With this representation, we can obtain motion features from previous and current motion speed.

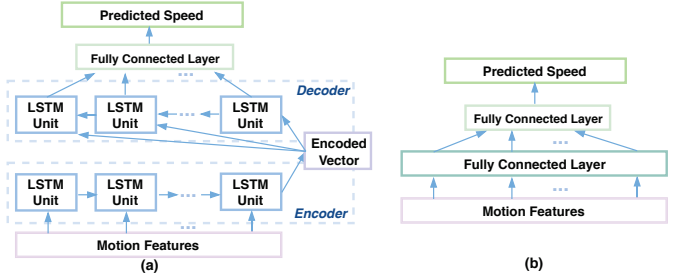


Fig. 4. (a) LSTM model and (b) MLP model used for motion prediction.

- *LSTM Model:* Inspired by the success of the RNN Encoder-Decoder in modeling sequential data [27] and good performance of LSTM to capture transition regularities of human movements since they have memory to learn the temporal dependence between observations [28], [29], 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. 4(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. 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:

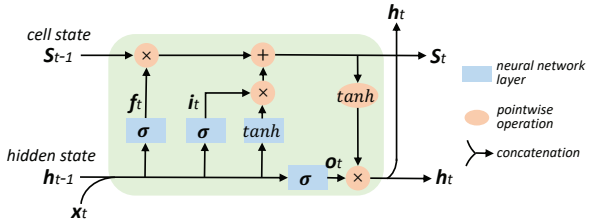


Fig. 5. The structure of LSTM unit.

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

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 mean square error and training is terminated after 50 epochs in our experiments.

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

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

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. 5, 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 [27], [30]:

$$\boldsymbol{f_t} = \sigma(\boldsymbol{W_f}[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}] + \boldsymbol{b_f}),$$

$$\boldsymbol{i_t} = \sigma(\boldsymbol{W_i}[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}] + \boldsymbol{b_i}),$$

$$\boldsymbol{o_t} = \sigma(\boldsymbol{W_o}[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}] + \boldsymbol{b_o}),$$

$$\boldsymbol{s_t} = \boldsymbol{f_t} \odot \boldsymbol{s_{t-1}} + \boldsymbol{i_t} \odot (tanh(\boldsymbol{W_s}[\boldsymbol{h_{t-1}}; \boldsymbol{x_t}] + \boldsymbol{b_s})),$$

$$\boldsymbol{h_t} = \boldsymbol{o_t} \odot tanh(\boldsymbol{s_t}),$$

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

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

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

$$\boldsymbol{f'_t} = \sigma(\boldsymbol{W'_f}[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}] + \boldsymbol{b'_f}),$$

$$\boldsymbol{i'_t} = \sigma(\boldsymbol{W'_i}[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}] + \boldsymbol{b'_i}),$$

$$\boldsymbol{o'_t} = \sigma(\boldsymbol{W'_o}[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}] + \boldsymbol{b'_o}),$$

$$\boldsymbol{s'_t} = \boldsymbol{f'_t} \odot \boldsymbol{s'_{t-1}} + \boldsymbol{i'_t} \odot (tanh(\boldsymbol{W'_s}[\boldsymbol{d_{t-1}}; y_{t-1}; \boldsymbol{c}] + \boldsymbol{b'_s})),$$

$$\boldsymbol{d_t} = \boldsymbol{o'_t} \odot tanh(\boldsymbol{s'_t}),$$

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 [31] model presented in Fig. 4(b) to do motion prediction. Using the same representation and loss function described above, this model also 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. We build up 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).

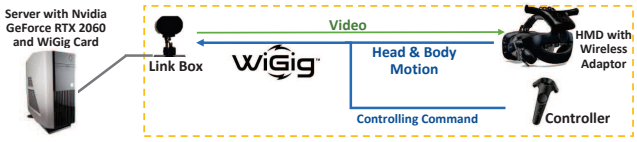


Fig. 6. Hardware setup.

TABLE III
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

VI. EXPERIMENTAL RESULTS

The hardware setup of our experiments is shown in Fig. 6, where the rendering server 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 will be within user's room and transmit rendered frames in a video format from the server to the HMD. On the user side, there are the link box and two HTC lighthouse base stations in the room. User will wear a HTC Vive HMD equipped with Vive wireless adaptor [32], and use 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 3m×3m of free space in our experiments, which cannot exceed 4.5m×4.5m since the maximum distance between base stations is 5m [33]. All head and body motions on HMD can be captured accurately using this HTC Lighthouse tracking system while controller detects user's controlling commands. For software implementation, we implement our proposed techniques based on SteamVR SDK [34], OpenVR SDK [35] as well as the Unity game engine [36] for data collection, and use Keras [37] 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 III 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 I).

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

- *Root Mean Square Error (RMSE)*:

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

- *Mean Absolute Error (MAE)*:

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

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 [15]–[17], we compare against this linear regression model, which extrapolates trajectories with assumption of linear acceleration. The Lin-A model employs the motion speed at 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

TABLE IV
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**

TABLE V
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 VI
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 VII
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

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 at the latest 2 time points to predict the expected motion speed of next time point.

Tables IV, V, VI, and VII exhibit the results of our body motion and head motion prediction for the two applications respectively. Specifically, for results of body motion prediction in Tables IV and VI we give the distance between actual and predicted body position in x, y, z axis (denoted as d_x, d_y, d_z), while for results of head motion prediction in Tables V and VII we 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:

- Tables IV and VI, 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 V and VII, 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.

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. 3(e)).

To evaluate the adverse 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 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 [38]. For each pixel i in the grayscale intensity images, we calculate the difference between the two intensity images, I_{dif}, as follows.

$$I_{dif}(i) = \begin{cases} I_1(i) - I_2(i), \text{if } I_1(i) \geq I_2(i) \\ 0, \qquad \text{otherwise} \end{cases} \tag{1}$$

Note that we set the I_{dif} as 0 in the second case of Equation 1, 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 1) to evaluate the difference between the two views. Fig. 7 presents an example of two views and the corresponding I_{dif}. In Fig. 7(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

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

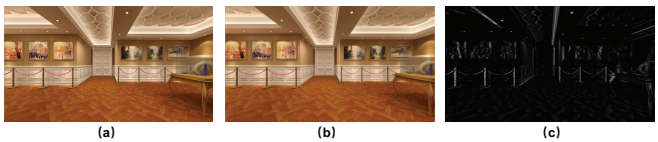


Fig. 7. (a) Actual user's view; (b) Predicted user's view with x-axis error $\Delta x = 0.1m$; (c) I_{dif} obtained from views in (a)(b).

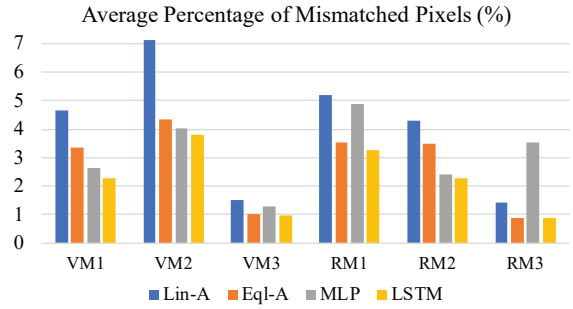


Fig. 8. The average percentage of mismatched pixels for different models during each session.

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. 8 illustrates the average *percentage of mismatched pixels* caused by predicted body motion 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. 8 demonstrates that compared to other models in each session, our LSTM model achieves less adverse effect on user experience caused by the body prediction error (denoted with yellow bar). Using the LSTM model, the average *percentage of mismatched pixels* can be smaller than 1% in both VM3 and RM3 sessions.

VII. Conclusions and Future Work

In this paper, we propose a head and body motion prediction model for 6DoF VR applications, to enable predictive pre-rendering and thus address latency challenge in edge/cloud-based 6DoF VR. We present a multi-layer LSTM model and MLP model which can learn general head and body motion pattern, and predict the future viewing direction and position based on past traces. Our method shows good performance on a real motion trace dataset with high precision.

While this work focused on prediction models, we will next implement the full system and demonstrate the feasibility of edge-based 6DoF VR based on predictive pre-rendering achieving ultra-low latency. Our planned future work includes further development and evaluation of proposed predictive pre-rendering approach from latency perspectives, and performing subjective studies to understand and quantify user experience using our proposed approach.

Acknowledgement

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

References

[1] C. Wiltz, "Five major challenges for vr to overcome," April 2017. [Online]. Available: https://www.designnews.com/electronics-test/5-major-challenges-vr-overcome/187151205656659/
[2] L. Cherdo, "Types of vr headsets," 2019. [Online]. Available: https://www.aniwaa.com/guide/vr-ar/types-of-vr-headsets/
[3] Oculus, "Oculus rift," 2019. [Online]. Available: https://www.oculus.com/rift/
[4] HTC, "Htc vive," 2019. [Online]. Available: https://www.vive.com/us/
[5] ——, "Htc focus," 2019. [Online]. Available: https://www.vive.com/cn/product/vive-focus-en/
[6] Oculus, "Oculus go," 2019. [Online]. Available: https://www.oculus.com/go/
[7] Samsung, "Samsung gear vr," 2019. [Online]. Available: https://www.samsung.com/us/mobile/virtual-reality/
[8] Google, "Google daydream," 2019. [Online]. Available: https://vr.google.com/daydream/
[9] X. Hou, Y. Lu, and S. Dey, "Wireless vr/ar with edge/cloud computing," in *ICCCN*. IEEE, 2017.
[10] Qualcomm, "Whitepaper: Making immersive virtual reality possible in mobile," 2016. [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 *VR/AR Network*. ACM, 2018, pp. 20–26.
[12] K. Boos, D. Chu, and E. Cuervo, "Flashback: Immersive virtual reality on mobile devices via rendering memoization," in *MobiSys*. ACM, 2016, pp. 291–304.
[13] L. Liu, R. Zhong, W. Zhang, Y. Liu, J. Zhang, L. Zhang, and M. Gruteser, "Cutting the cord: Designing a high-quality untethered vr system with low latency remote rendering," in *MobiSys*. ACM, 2018, pp. 68–80.
[14] 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 *MobiCom*. ACM, 2017, pp. 409–421.
[15] X. Yun and E. R. Bachmann, "Design, implementation, and experimental results of a quaternion-based kalman filter for human body motion tracking," *Trans. on Robotics*, vol. 22, no. 6, pp. 1216–1227, 2006.
[16] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *CVPR*. IEEE, 2016, pp. 961–971.
[17] A. Gupta, J. Johnson, L. Fei-Fei, S. Savarese, and A. Alahi, "Social gan: Socially acceptable trajectories with generative adversarial networks," in *CVPR*. IEEE, 2018, pp. 2255–2264.
[18] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *CVPR*. IEEE, 2017, pp. 2891–2900.
[19] J. Butepage, M. J. Black, D. Kragic, and H. Kjellstrom, "Deep representation learning for human motion prediction and classification," in *CVPR*. IEEE, 2017, pp. 6158–6166.
[20] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, "Optimizing 360 video delivery over cellular networks," in *AllThingsCellular*. ACM, 2016, pp. 1–6.
[21] C. Fan, J. Lee, W. Lo, C. Huang, K. Chen, and C.-H. Hsu, "Fixation prediction for 360 video streaming to head-mounted displays," *ACM NOSSDAV*, 2017.
[22] 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 *BigData*. IEEE, 2016, pp. 1161–1170.
[23] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive view generation to enable mobile 360-degree and vr experiences," in *VR/AR Network*. ACM, 2018, pp. 20–26.
[24] Unity, "Virtual museum," 2019. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/museum-117927
[25] ——, "Virtual rome," 2019. [Online]. Available: https://assetstore.unity.com/packages/3d/environments/landscapes/rome-fantasy-pack-ii-111712
[26] R. W. Schafer *et al.*, "What is a savitzky-golay filter," 2011.
[27] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.
[28] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, and S. Savarese, "Social lstm: Human trajectory prediction in crowded spaces," in *CVPR*, 2016, pp. 961–971.
[29] J. Liu, A. Shahroudy, D. Xu, and G. Wang, "Spatio-temporal lstm with trust gates for 3d human action recognition," in *ECCV*. Springer, 2016, pp. 816–833.
[30] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *arXiv preprint arXiv:1409.2329*, 2014.
[31] R. J. Frank, N. Davey, and S. P. Hunt, "Time series prediction and neural networks," *J. Intell. Robotic Syst.*, vol. 31, no. 1-3, pp. 91–103, 2001.
[32] HTC, "Htc vive wireless adaptor," 2019. [Online]. Available: https://www.vive.com/us/wireless-adapter/
[33] Valve, "Steamvr faq," 2019. [Online]. Available: https://support.steampowered.com/kb_article.php?ref=7770-WRUP-5951
[34] ——, "Steamvr sdk," 2018. [Online]. Available: https://valvesoftware.github.io/steamvr_unity_plugin/
[35] ——, "Openvr sdk," 2018. [Online]. Available: https://github.com/ValveSoftware/openvr/
[36] U. Technologies, "Unity," 2019. [Online]. Available: https://unity3d.com/
[37] Keras, "Keras," 2019. [Online]. Available: https://keras.io/
[38] Matlab, "Convert rgb image or colormap to grayscale," 2019. [Online]. Available: https://www.mathworks.com/help/matlab/ref/rgb2gray.html