Freie Universität Berlin
Fachbereich Mathematik und Informatik
Takustraße 9, 14195 Berlin

MASTER THESIS

USER POSITION PREDICTION IN 6-DOF MIXED REALITY APPLICATIONS USING RECURRENT NEURAL NETWORKS

Oleksandra Baga

Freie Universität Berlin
Matrikelnummer 5480722
Master Computer Science
E-Mail: oleksandra.baga@gmail.com

Prof. Dr. Daniel Göhring
Fachbereich Mathematik und Informatik
Freie Universität Berlin

Prof. Dr. Tim Landgraf
Fachbereich Mathematik und Informatik
Freie Universität Berlin

Statutory Declaration

I herewith formally declare that I have written the submitted master thesis independently. I did not use any outside support except for the quoted literature and other sources mentioned in the paper.

I clearly marked and separately listed all of the literature and all of the other sources which I employed when producing this academic work, either literally or in content.

I am aware that the violation of this regulation will lead to failure of the thesis.

10.10.2022..... Oleksandra Baga

Acknowledgments

This thesis was created in cooperation with the Fraunhofer Heinrich Hertz Institute.

I would like to thank Prof. Dr. Daniel Göhring, who consulted me during the work on a thesis, and Prof. Dr. Tim Landgraf, who taught me machine learning so that I chosen my master thesis topic in the field of ML research.

A special thanks goes to the Serhan Gül, researcher of Fraunhofer Heinrich Hertz Institute, who suggested an exciting topic for a research, which I was allowed to choose for my master thesis.

Contents

List of Figures I

Listings III

List of Abbreviations IV

1 Introduction 1
- 1.1 Problem statement . . . 1
- 1.2 Motivation for the research . . . 2
- 1.3 Structure of the thesis . . . 2

2 Fundamentals 4
- 2.1 Mixed reality with HMD . . . 4
- 2.2 Six degrees of freedom . . . 5
- 2.3 Motion-to-photon latency . . . 6
- 2.4 Cloud-based volumetric video streaming . . . 7
- 2.5 Challenges of head motion prediction . . . 9
- 2.6 Related works . . . 10
  - 2.6.1 Traditional prediction algorithms . . . 10
  - 2.6.2 Recurrent Neuronal Networks . . . 11

3 Implementation 13
- 3.1 6-DoF Dataset . . . 13
  - 3.1.1 Data collection from HMD . . . 13
  - 3.1.2 Data Exploration . . . 15
  - 3.1.3 Data preprocessing . . . 17
- 3.2 Model . . . 21
  - 3.2.1 Inputs and outputs . . . 21
  - 3.2.2 LSTM Model . . . 22
  - 3.2.3 GRU Model . . . 27
  - 3.2.4 Bidirectional GRU Model . . . 30
  - 3.2.5 Development . . . 32
    - Unity application . . . 32
    - Training and evaluation . . . 34
    - Hyperparameter search . . . 36

4 Evaluation **37**
4.1 Goal of evaluation . . . 37
4.2 Evaluation metrics . . . 37
4.3 Baseline model . . . 38
4.4 Experiments . . . 42
4.4.1 First experiments . . . 42
Datasets . . . 42
Batch size . . . 45
Learning rate . . . 45
Weight decay . . . 46
4.4.2 Prediction with LSTM . . . 47
4.4.3 Prediction with GRU . . . 50
4.4.4 Prediction with Bidirectional GRU . . . 50

5 Conclusion **51**
5.1 Analysis . . . 51
5.2 Limitations . . . I
5.3 Suggestions for future work . . . I

Bibliography **IV**

List of Figures

Fig. 1 HoloLens 2 maps itself with a mesh. 5
Fig. 2 Viewing paradigm in 3- and 6-DoF VR. Source: [21] 6
Fig. 3 M2P latency for a remote rendering system. 7
Fig. 4 High level operation of a cloud-based volumetric streaming system. 8

Fig. 5 User position plots from obtained datasets (a, b, c). 15
Fig. 6 Changes in the user's position along the Y axis in the range from 400ms to 500ms. 16
Fig. 7 Interpolated 6-DoF dataset's user position and orientation in quaternions. 18
Fig. 8 Quaternions from 6-DoF dataset's flipped if their real part is negative. 19
Fig. 9 Enlarged quaternion plot with breaks omitted. 20
Fig. 10 Long Short-Term Memory. 23
Fig. 11 Gated Recurrent Unit. 27
Fig. 12 Unity 3D UserPrediction6DOF Project with placed obj-animation, underlying code and scene. 33
Fig. 13 Photo made by Hololens 2 camera during the Unity Application running on HMD with the volumetric animated object. 33

Fig. 14 Outputs of Baseline Model for x-axis. 39
Fig. 15 Outputs of Baseline Model for x, y and z axes. 40
Fig. 16 Outputs of Baseline Model for quaternions components qx, qy. qz and qw. 41
Fig. 17 Outputs of Baseline Model for rotation data represented as Euler angles. 41
Fig. 18 Outputs of LSTM1 model on interpolated dataset for x, y and z axes. 43

Fig. 19 Outputs of LSTM1 model on interpolated dataset for qx, qy, qz and qw axes. 43
Fig. 20 Outputs of LSTM1 model on dataset with flipped negative quaternions for x, y and z axes. 43
Fig. 21 Outputs of LSTM1 model on dataset with flipped negative quaternions for qx, qy, qz and qw axes. 44
Fig. 22 Plot of training and validation loss with small learning rate of Adam optimizer. 46
Fig. 23 Plot of training and validation loss with large weigh decay of Adam optimizer. 47
Fig. 24 Outputs of LSTM2 model with ReLU activation function for x, y and z axes. 48
Fig. 25 Outputs of LSTM2 model with ReLU activation function for qx, qy, qz and qw axes. 48
Fig. 26 Outputs of LSTM3 model with Mish activation function for x, y and z axes. 49
Fig. 27 Outputs of LSTM3 model with Mish activation function for qx, qy, qz and qw axes. 49

Listings

3.1 One-layered LSTM with sliding window 25
3.2 LSTM3 with Mish activation function 26
3.3 GRU1 with Sliding Window 29
3.4 Bidirectional GRU Model 31
3.5 Bidirectional GRU h0 Layer 32

List of Abbreviations

ANN	Artificial Neural Networks
AR	Augmented Reality
CNN	Convolutional Neural Network
CPU	Central processing unit
DoF	Degree of freedome
DL	Deep Learning
FFN	Feed-forward Neural Network
GRU	Gated Recurrent Unit
HMD	Head-Mounted-Display
IEEE	Institute of Electrical and Electronics Engineers
KF	Kalman Filter
LAT	Look ahead time
LSTM	Long-Short-Term Memory
M2P	Motion-to-Photon
MAE	Mean Absolut Error
MEC	Mobile Edge Computing
ML	Machine Learning
MR	Mixed Reality
NLP	Natural Language Processing
ReLu	Rectified Linear Unit
RNN	Recurrent Neural Network
RTT	Round-trip time
SDG	Stochastic Gradient Descent
VR	Virtual Reality
3-DoF	Three degree of freedom
6-DoF	Six degree of freedom

1 Introduction

This thesis is focusing on designing and evaluation of the approach for the prediction of human head position in a 6-dimensional degree of freedom (6-DoF) of Extended Reality (XR) applications for a given look-ahead time (LAT) in order to reduce the Motion-to-Photon (M2P) latency of the network and computational delays. At the beginning of the work the existing head motion prediction methods were analysed, and their similarities differences will be taken into account when a proposed Recurrent Neural Network-based predictor will be developed. Main goal is the systematic analysis of the potential of recurrent neural networks for head motion prediction. The proposed approach was evaluated on a real head motion dataset collected from Microsoft HoloLens. Based on a discussion of the obtained results, suggestions for future work are provided.

1.1 Problem statement

The correct and fast head movement prediction is a key to provide a smooth and comfortable user experience in VR environment during head-mounted display (HDM) usage. The recent improvements in computer graphics, connectivity and the computational power of mobile devices simplified the progress in Virtual Reality (VR) technology. The way users can interact with their devices changed dramatically. With new technologies of VR environment user becomes the main driving force in deciding which portion of media content is being displayed to them at any time of interaction with VR Applications [20]. Until recently the high-quality experiences with modern Augmented Reality (AR) and VR systems were not widely presented in home usage and were mainly used in research labs or commercial setups. The hardware for displaying the VR environment was once extremely expensive but recent years became more broadly accessible and the 6-DoF VR headset designed for the end-user were released[1]. It is possible now to experience virtual reality scenes and watch new type of volumetric media at home and the market interest for development VR and AR applications expected to be huge next years.
In fact, the existing on this moment virtual environments can be divided into two main groups depending on position of the user and their ability to move inside the VR environment. The user motion and prediction within a 3-DoF environment

[1] https://medium.com/@DAQRI/motion-to-photon-latency-in-mobile-ar-and-vr-99f82c480926

has been intensely researched for years. Extending such approaches to a 6-DoF environment is not straightforward, due to the change of the user's viewing point from inward to outward and additional three degrees of freedom [21].
Although all mentioned above improvements, rendering of volumetric content remains very demanding task for existing devices. Thus the improvement of a performance of existing methods, design and implementation of new approaches specially for the 6-DoF environment could be a promising research topic.

1.2 Motivation for the research

Research efforts to reduce the computational load are being already wide attempted. However, these approaches designed for the client side. Recently presented technique of the rendering on a cloud server makes possible to decrease the computational load on the client device by offloading of the task to a server infrastructure and than by sending the rendered 2D content instead of volumetric data [13]. The calculated 2D view must correspond the current position and orientation of a user. However, cloud-based streaming approach adds network latency and processing delays due uploading to a server the user position, rendering a new 2D picture from the 3D data and sending it back to a device. Thus, a rendered 2D image can appear even later on a display than with usage of local rendering system.
The promising research topic is a reducing the Motion-to-Photon (M2P) latency by predicting the future user position and orientation for a look-ahead time (LAT) and sending the corresponding rendered view to a client. The LAT in this approach must be equal or larger to the M2P latency of the network including round-trip time (RTT) and time need for calculation and rendering of a future picture at remote server.

1.3 Structure of the thesis

The organization of this thesis is as follows. The thesis starts from introduction and problem statement, followed by theoretical background related to the research topic. Literature review chapter introduces different approaches and technologies of motion prediction algorithms. The chapters **??**nd 4 show the implementation ff the presented models and evaluation of the results that were obtained during experiments. Last, the discussion regarding method limitations and suggestions for the future work are done.

Chapter 1 - Introduction.
The current chapter shortly introduces a state of development on scientific field achieved at a time of master thesis creation in the context on XR applications. The

necessity of timely action to improve the situation with increasing computational and network latency is shown in problem statement section 1.1. Due to the breadth of the research topic, the section 1.2 focuses and motivates the research topic.

Chapter 2 - Background.
The next chapter includes a review of the area being researched. It starts with a short introduction of the concept of MR applications and presents a 6-DoF environment. The presence and influence of a computational and network latency is covered. In section 2.5 the challenges faced in predicting of the viewer's position are discussed. Last section contains an overview of previous research in the field of prediction of user's head position and orientation and places a master thesis's topic in the context of the existing literature.

Chapter 3 - Implementation.
Chapter describes practical implementation of the approach. The dataset including data collection from head mounted display (HDM) and data understanding and preprocessing are described in section 3.1. Model Inputs, Model architecture and the development steps are covered in section 3.2. The implementation of Unity Application, training and evaluation loop with PyTorch and hyperparameter search described in the section 3.2.5.

Chapter 5 - Evaluation.
A Baseline model, used for comparing the obtained results and tuning the hyper-parameters is described. The goal of evaluation and metrics used in this research are covered. The conducted experiments with a data obtained from HMD for each analysed RNN Model can be found in section 4.4.

Chapter 5 - Conclusion.
The last chapter presents a discussion about the limitation of proposed method and provides a conclusion about the received results including suggestions for potential types of future research.

2 Fundamentals

This chapter introduces theoretical background of the presented research problem. First, the concept of mixed reality (MR) followed by an introduction of six degree of freedom (6-DoF) environment and the difference to the three degree of freedom (3-DoF) are described. The term motion-to-photon latency (M2P) is covered, followed by a short discussion about an influence of M2P latency on the decreasing of user experience. The new developed cloud-based rendering and streaming approach is shortly discussed in this chapter. The last section of this chapter highlights challenges with the prediction of viewer's head pose that arises in modern XR applications in connection especially with the added network latency due the using of remote cloud server for computational offload. The section 2.6 overviews the existing works done in the research field using traditional algorithms and recurrent neural networks.

2.1 Mixed reality with HMD

Mixed reality makes possible to break down the border between the virtual and real world and provides today an experience that just a short-time ago we could only imagine when watching the sci-fi movies. Terms Virtual Reality (VR), Augmented reality (AR) and Mixed reality (MR) are often used interchangeably. VR creates the virtual environment around user and tricks human's senses into thinking one is in a different environment. AR adds a virtual object to the real world that we can see through the lenses of special developed Head Mounted Display (HMD). Thus realistic images, sounds, and other sensations can be generated by a powerful HMD and projected on transparent holographic lenses giving a user the feeling that virtual objects have size and density. However, AR does not allow interactions between users and the virtual objects added to the real-world scene. MR combines the advantages of the VR and AR and adds an interaction between real and artificial elements. Thus users can directly interact with virtual objects (with operations such as scaling, rotation, or translation) in the real environment using their hands. For example, in MR Application virtual objects can be placed on the real table in the user's room, picked up with a hand and moved to another place.

Volumetric video (VV) is a new content creation approach to be used within AR and MR applications [26]. Volumetric video allows to view recorded information

from a range of different angles, as if an observer was physically presented in the room when video was captured by cameras and could move around the object. This thesis uses volumetric video object placed in the real environment when running developed MR Application for collection the user's position and rotation data. Refer sections 2.4 and 3.2.5 for more details about VV and how it was used in thesis.

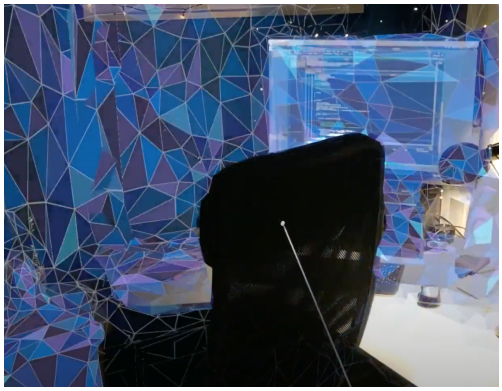


Figure 1: HoloLens 2 maps itself with a mesh.

Nowadays different HMD with varying performance levels and prices are available on the market. In this thesis, Microsoft HoloLens 2 was used for MR experience and data collection. It is an updated version of the previous generation HoloLens 1 headset from Microsoft with such improved feature as display resolution, field of view, weight, battery lifetime. By using the AHAT (Articulated HAnd Tracking) depth camera, the HoloLens 2 can capture hand movements to obtain hand tracking data. The build-in tracking systems allows HoloLens to understand the environment around the user and to place stable and accurate holograms on the correct places where they intended to be by the developer of MR Application. The data used to track users is represented in the spatial map[1]. When VR Application is starting on HoloLens, HoloLens uses unique environmental landmarks to locate itself in a space. The mesh graphic spreading over the space is seen, as illustrated in Fig. 1, during the Application launch and this means a device is mapping to surroundings. As user moves with HMD on their head, built-in cameras continuously scan the environment and construct virtual world geometry for real-world objects. The primary stereo rendering component attached to HMD can be accessed from Unity and thus the position and orientation can obtained for thesis purposes.

2.2 Six degrees of freedom

Term *degrees of freedom* describes how users interact with a virtual environment and how they can move inside it. Within 3-DoF space user has only three possibilities: look left and right, look up and down and pivot left and right. 3-DoF space does not allow to move throughout the virtual space. Thus only rotational movement can be tracked. In 3-DoF VR Application multimedia content is the omnidirectional or spherical video, which represents an entire 360° environment on a virtual sphere

[1] **https://docs.microsoft.com/en-us/hololens/hololens-environment-considerations**.

[21]. In 3-DoF space HMD enables to display only a portion of the environment around a user. User is virtually positioned at the centre of a sphere as shown in Fig. 2, media is displayed from an inward position and user can only change the viewing direction (i.e., by looking up/down or left/right or tilting the head side to side) [21] but can not interact with a media by moving closer/further. Wherever user moves with a HMD on their head, they will remain placed in the at the centre of a sphere and distance to a content can not be changed.

The new VR concept 6-DoF means tracking both position and rotation and refers to the freedom of movement of a rigid body in three-dimensional space. In 6-DoF VR Application user can also change viewing perspective by moving (e.g., walking, jumping) inside the virtual space [21]. Thus the scene is observed from an outward position in 6-DoF environment and extra degree of freedom transforms the virtual experience to be more natural and reflects to human movement in a three-dimensional space. Thus the VV and other volumetric objects such meshes or point clouds are used in MR Applications for 6-DoF scene population. User can freely walks inside the 6-DoF environment with a HMD on a head and observe the placed on scene volumetric objects from all points of view, and if the settings in Unity application allow physical interaction with objects, pick and move them on the new place.

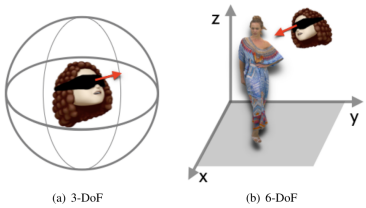


Figure 2: Viewing paradigm in 3- and 6-DoF VR. Source: [21]

2.3 Motion-to-photon latency

VR Application are deployed to the end-user with a goal to create an immersion of a physical presence in a non-physical world. In the real world there is no time delay between action taken and reaction observed. However, in AR/VR/MR Applications the difference between the user's head movement (action) and its corresponding display output reflections (reaction) is defined as motion-to-photon (M2P) latency. The presence of a delay between the physical movement and the display output worsens HMD user experience. In worst case even sense of physical presence in a virtual world would be lost. MTP latencies of more than 20 ms are experienced and cause spatial disorientation and dizziness, referred to as VR sickness or motion sickness [2, 12]. Display lag can produce a range of other perceptual effects include degraded vision, compromised visuo-motor performance and motion sickness [2]. Different components of the HMD, such as the sensors, SOC, display and software can affect M2P latency. Reducing the M2P latency is the key to proving the best

VR experience. Not only improving the device parameters, such a usage of more powerful HMD processor, need to be taken in account. VR Application developers must consider how to deploy more light-weighted applications. If the VR Application need to pull some data from the network or remote server, the network round-trip time and the added processing delays will increase the M2P latency compared to a system that only performs the processing locally [12].

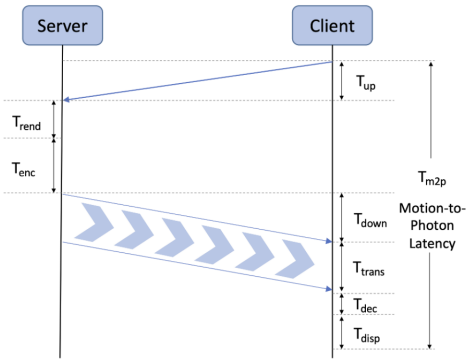


Figure 3: M2P latency for a remote rendering system.

As this thesis evaluates the reducing the M2P latency for VV streaming from remote cloud server, the Fig. 3 from [13] illustrates the different components of the M2P latency for a remote rendering system. Total M2P latency is equal to sum of the time taken by a bit of data to travel across the network from HMD to a server, server delay involved in computing the future user position and render a 2D view and a HMD delay during sensor measurements. If the user's future head pose for a look-ahead time (LAT) equal to or larger than the M2P latency of the system could be predicted, it can eliminate M2P latency and improve the quality of the VR Applications. Studies showed that display lags of greater than 40 ms cause errors in tracking and following a target with the head [2]. This thesis evaluates the performance of RNN Models for LAT 100 ms that is higher than the measured M2P latency of a cloud-based volumetric streaming system described in the next section.

2.4 Cloud-based volumetric video streaming

Volumetric video (VV) is a young technology and is used to build a content for AR and VR Applications. Real-life video from cameras surrounding the 3D object is stored as point clouds or 3D textured mesh sequences and builds a dynamic 3D scene of a real 3D object. In VR Application user can walk through VR environment with a HMD and thus VV can be looked at from any viewpoint. In almost all cases today, the VVs objects stored and rendered locally on a users device. Photo-realistic modelling, real-time rendering and animation of VVs is still computationally difficult. The long sequence VV can even exceed the HMD memory capacity and could not be deployed as a VR Application even on high-cost VR HMD as HoloLens 2. There are still no efficient hardware decoders for point clouds or meshes and software

decoding can be prohibitively expensive in terms of battery usage [12]. Thus in the research field there is a growing interest in VV compression and adaptive streaming, as real-time streaming is necessary for some applications, e.g., telepresence and remote collaboration [26]. The processing and memory load on the user's HMD can be decreased by sending a 2D precomputed rendered view instead of the volumetric 3D content. Some previous studies reveal that participants preferred to stay in front of static point clouds and 1 metre away from them and spent more time looking at the frontal view and faces of human models [26]. VVs are not transparent and provide a feeling of a real 3D object with a mass and a weight thus as a real 3D object they can be looked at only from one viewpoint at on time step. Thus if the sending unneeded information (for example, a back view of a human model when the user looks at model's front view) can be avoided, it decrease the computational load on the user's HMD.

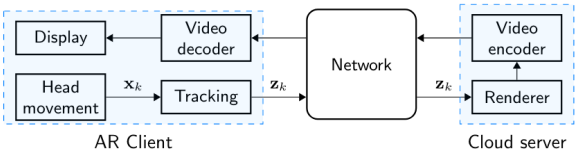


Figure 4: High level operation of a cloud-based volumetric streaming system.

A remote rendering system takes complex graphics computational and rendering tasks and delivers the result over a network to a less-powerful client device. Fig. 4 shows an overview of a cloud-based volumetric streaming system proposed by *Gül et al., 2020*. This thesis evaluates RNN Models and the trained model with the best performance is intended to be used as a part of prediction system of a remote system. A detailed software architecture of a system is described in [13]. In this system a compressed volumetric video is stored as a single MP4 file containing video and mesh tracks [12]. The game engine (Unity) runs at a server and decodes the compressed mesh and texture data. The tracking system of the HMD measures the user position and orientation and sends over a network to the cloud server. Based on the actual user's spatial attributes, cloud server calculates the future position and orientation and renders the corresponding view from the volumetric content. The rendered view is encoded as a video stream and sent to the client over the network. The time period between the head movement and display of the decoded video frame to the viewer is the M2P latency of the system which can be compensated by applying prediction algorithm [12].

2.5 Challenges of head motion prediction

All modern HMD has a position tracker, a device or a system of devices, that is responsible for reporting the position and orientation of HMD to the computational unit that generates the virtual environment images displayed in the HMD. These images represent the view that a wearer of HMD would have seen if user was present in VR at the position and orientation reported by position tracker [6]. While the task of position tracking is performed by HMD hardware, the task of position prediction of the movement of human body in the virtual reality remains challenging, and it is still complicate to achieve high-precision estimation.

Understanding how users interact and behave in AR or VR is important when working with HMD's sensors. The experiment done by *Zerman et al., 2021* found out that users preferred to stay in front of static point clouds and 1-1.5 meter away from them and spent more time looking at the faces of human models [26]. The navigation trajectories of users within a 6-Degrees-of-Freedom (DoF) should be additionally investigated since an extra level of interaction between user and content is available in 6-DoF environment. The user has now the freedom to change the viewing direction (rotating and translating the head as in 3DoF) but also to change position inside the VR environment [20]. In a 6-DoF environment, users are not being positioned at the centre of the spherical content any longer and the distance changes over time when user moves due to the added degrees of freedom. Thus viewport's center position is not sufficient for tracking the trajectories, the additional metrics such the spatial coordinates and user orientation are needed to obtain the point of origin.

Following [20] *Rossi et al, 2021* authored same year another work [21] dedicated 6-DoF metrics. Researchers experimented with different metrics to perform clustering in order to detect group of users with similar behavior in VR. The most promising metric seems to be based on the user position on the virtual floor. Metrics based only on viewport center, as it was used in 3-DoF, and distance failed in detecting the group of similar users [21]. For the trajectory detection best performed a metrics based on user position, orientation on the virtual floor and distance [21]. The analysis above leads to the conclusions that prediction of the user's position and orientation on 6-DoF requires new metrics and approaches to be investigated and implemented.

2.6 Related works

This section presents the overview of previous research in the field of the prediction of user position and focuses on time series methods using different RNN architectures such as LSTM and GRU.

2.6.1 Traditional prediction algorithms

A lot of previous approaches uses basic processing of head movement history to predict the future movement, such as simple average, linear regression, and weighted linear regression [18]. Work of *Corbillon et al., 2017* determines the distance to the center of viewpoint with simple average and calculates the region that receives from server the video data with a better quality than the remaining of part the video [9]. The work of *Duanmu et al., 2017* proposes prediction of the viewing direction for segment n + 1 through linear regression based on the past view segments [11]. Approach of *Xie et al., 2017* uses user's orientation in Euler angle and leverage Linear Regression model to apply Least Square Method and to calculate the trends of head movements [25]. The work [22] proposes to receive from a server only the data of covered by user's viewport. At each point of time, the client requests data which would be played in the future. *Taghavi et al., 2017* use Weighted Linear Regression to predict the next viewport based on window with the latest viewport samples. Researchers mentioned that a client can continue playback of at least a low-quality version of the video when the download time of next video portion varies [22].

Analysis done by *Qian et al., 2016* indicates that at least in the short term, viewers' head movement can be predicted with accuracy > 90% by even using simple methods such as linear regression [19]. The different approaches were compared such as computing the average value, using the linear regression with all samples and with weighted linear regression with recent samples. With weighted linear regression the average prediction accuracy for short-term values was higher than 90% across all users. However in the longer term it is more difficult to achieve the good result and the average accuracy drops to about 70% [19].

A method to apply saliency algorithms to VR video viewings was presented by *Aladagli et al., 2017* in work [1]. Cross-correlation analysis used for measuring the relationship between the predicted fixation sequences and the recorded head movements [1]. Based on works mentioned above, *Nguyen et al., 2018* proposed panoramic saliency algorithm in order to learn the dependence of head tracking logs and saliency maps from the past video frames.

2.6.2 Recurrent Neuronal Networks

As was explained above, traditional prediction algorithms can not be used straightforward on a new media content in 6-DoF VR Applications. The user position and rotation data is coming as time series with a sequential order that is crucial to be followed in order to predict correctly the next future step for a look-ahead time. A sequence of inputs can be processed with Artificial Neural Network (ANN) called Recurrent Neural Network (RNN). Moreover, RNN can processes input with remembering its state while processing the next sequence of inputs. It is known that standard RNN has difficulties to learn long-term dependencies with gradient descent [5]. Though RNN can robustly store information, it yields a problem of vanishing gradient that make leaning difficult [5]. In the last decade, RNN algorithms have been adopted for motion prediction of 3D sequences with long-term dependencies taken into account. For example, the work of *Crivellari et al., 2020* targets traces of tourists in a foreign country and tries to predict the motion of people in the environment they never seen before. LSTM-based model is used thus for analyzing the tourists' mobility patterns [10].

The authors *Aykut et al., 2018* claims their research to be first work that applies deep learning for head motion prediction. The authors experimentally confirmed that Feed-forward Neural Network (FFN) indeed had difficulties to learn for different delays. The decision to use LSTM-based architectures *Aykut et al., 2018* reasoned with feedback loop and ability to establish a way of memory and share weights over time [3]. Conducted by researchers experiments showed that the LSTM-based architecture leads to a significant improvement of the MAE and RMSE metrics [3]. The LSTM-based methods were compared also to widely used approaches like the Linear Regression and a Kalman Filter based optimal state estimate. Thus *Aykut et al., 2018* demonstrated a substantial improvement of the deep predictor for latencies in the range of 0.1–0.9 s [3].

Next year *Aykut et al., 2019* experimented in their work [4] with GRU model that belongs to the group of recurrent neural networks (RNN). Authors considered GRU usage because it is computationally more efficient, as it has fewer parameters and states than LSTM units [4]. Proposed in the research GRU-based network is able to improve the MAE and RMSE compared to mentioned above LSTM model, especially for larger delays [4].

Researchers *Karim et al., 2018* developed long short term memory fully convolutional network (LSTM-FCN). In the proposed models, LSTM block is augmented by an fully convolutional block [15] identical to the convolution block in the CNN architecture proposed by *Wang et al., 2018* in their work [24]. *Karim et al., 2018* tried to reduce

the rapid model's overfitting by transformation of input to have N variables with a single time step [15].

In work of *Chang et al., 2020* used in addition to standard LSTM networks also bidirectional LSTM (Bi-LSTM) networks, which is stacked two LSTM networks in forward and backward directions. Standard LSTM networks can only consider the past information and Bi-LSTM networks can capture both past and future information by two opposite temporal order in hidden layers [7]. Experimentally, authors found that the basic LSTM performs the best comparing to Bi-LSTM and Temporal Convolutional Network [7].

GRU Model and additionally a bidirectional LSTM (Bi-LSTM) network are used for action recognition based on sensor signals from HMD in work [16]. Similar as in work [7] the LSTM model performed better compared to Bi-LSTM and a GRU outperformed both models. Authors said that the possible reason could be the short-term correlation of human actions in their dataset and that Bi-LSTM with its complicated model structure is rather suitable for long-term actions [16]. The experiments provided in these works clearly indicate that GRU unit can outperform LSTM unit. However, researches suggested that the choice of the RNN model can depend heavily on the dataset and corresponding task [8].

3 Implementation

This chapter presents the steps of development and implementation of the proposed approach. The Unity Application for HoloLens was deployed on HMD and a raw data with measures and dimension columns was obtained. This data than was analysed and preprocessed to ensure that the captured data can be used in corresponding machine learning models. The model architecture was implemented and experimentally improved during training and evaluation steps.

3.1 6-DoF Dataset

This section describes how the dataset was obtained, analysed and presents the visualization of user's head position and rotation. Almost all machine learning approaches require not only row data collection but also data exploration and preprocessing steps to be done before training begins.

3.1.1 Data collection from HMD

The real 6-DoF dataset must be used as training data from which the model can learn the spacial and time dependences. In this master thesis HoloLens 2, the second iteration of Microsoft's head-mounted mixed reality device, was used for data collection. The user position and orientation were obtained with Unity application developed for this purpose. Main Camera in Unity is automatically configured to track head movements. More details about Unity application can be found in section 3.2.5. Using the Main Camera, a user position (x, y, z) and orientation in quaternion (qx, qy, qz, qw) were logged in a csv-file. Quaternions obtained from HMD will be used to define a rotation by four numbers. Quaternions representations are very convenient for operations such as composition or rotations and coordinate transformation [23]. For these reasons quaternions are chosen for the representation of user head's rotation in three dimensions. Comparing to dataset in [12], the additional parameters were recorded from the Main Camera in order to add more information during training processes. Thus the world-space speed of the camera in meters per second was recorded. Unity velocity has the speed in (x, y, z) defining the direction. The obtained 6-DoF dataset has 10 features used in training process: position (x, y, z), orientation (qx, qy, qz, qw) and velocity (x, y, z).

The datasets were recorded in the laboratory space. HMD was presented to users and the basic functions were explained. During data recording, users freely walked wearing HMD in laboratory space. The Unity application, running on HMD, not only recorded the mentioned before parameters but also had a volumetric animated object placed 3 meters ahead of the user in the Mixed Reality environment. No personal data was recorded during these sessions and all traces are obtained anonymously. Thus, after an Unity application was launched, user could immediately see the animated object. The several traces were recorded at least for 10 minutes each. It allows to have enough data after splitting the dataset into training, test and validation partitions. Table 3.1 show the first 20 rows from raw dataset obtained from HoloLens 2 and used in training. Although dataset has 10 columns, the table 3.1 presents only $timestamp$ and position (x, y, z) columns.

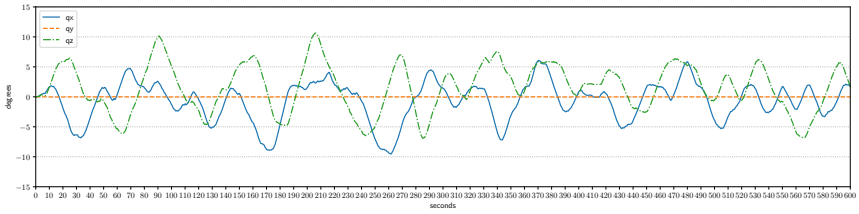
timestamp	x	y	z
2.649431	0.004954389	0.003402365	0.01010712
2.66943	0.00459053	0.003120769	0.01130438
2.698009	0.003960807	0.002990472	0.01276976
2.719285	0.003730714	0.003037783	0.01305151
2.746641	0.003252693	0.003489003	0.01368421
2.764094	0.003153284	0.003518121	0.01400959
2.780033	0.003087142	0.003409061	0.01435899
2.802086	0.003021815	0.00314023	0.01473305
2.815575	0.002789935	0.003551113	0.01506916
2.832602	0.002527435	0.003542757	0.01534094
2.848514	0.002212256	0.003605011	0.01565307
2.863769	0.001921757	0.003369405	0.01590317
2.879648	0.001668522	0.00348538	0.01607716
2.89686	0.001501704	0.003624826	0.01627397
2.913541	0.001487849	0.00359472	0.01643924
2.930006	0.001501501	0.003769569	0.01669565
2.948201	0.001617525	0.004252479	0.01697758
2.964302	0.001755987	0.004224311	0.01721937
2.97978	0.001838901	0.004487753	0.01747578
2.997117	0.002005509	0.005007531	0.01782864

Table 3.1: Raw data from HoloLens 2.

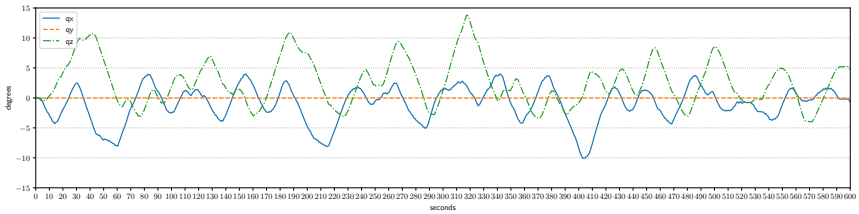
The first column in dataset is $timestamp$. It is obviously, that timestamp appears in row dataset not linearly and comes with different pauses. Even the high-cost HMD, like used in this research HoloLens 2, is sometimes unstable in frame rate during collecting data. Due to signal processing and propagation delays, distance in time between two consecutive samples was either increased or decreased. In the Unity Application, the frame rate is 60 Hz which means that data is expected to be collected every 0.016(6) seconds. Data on some expected timestamps seemed to be unavailable in HMS for recording. Between two sequences with bigger time gap, some records may be considered to be missed. To deal with above situation, the preprocessing steps must be done. They are described in a section 3.1.3 below.

3.1.2 Data Exploration

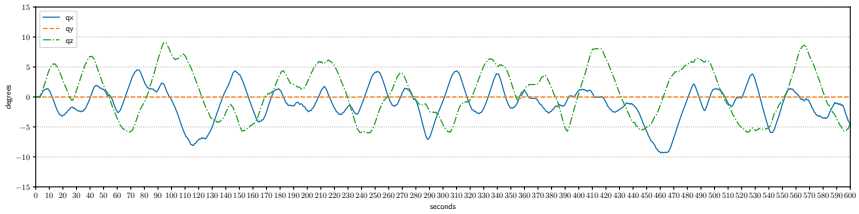
The next step after looking at raw data, gathered for machine learning, is a data exploration. The goal of this initial step is, firstly, a data visualization for understanding of dataset characterizations. As already stated in section 3.1.1, a user position (x, y, z), orientation in quaternion (qx, qy, qz, qw) and the world-space speed of the camera for each direction in (x, y, z) was obtained from Main Camera in Unity application launched on HMD.



(a) Dataset 1556



(b) Dataset 1623



(c) Dataset 1703

Figure 5: User position plots from obtained datasets (a, b, c).

First, let's start with the analysis of user position data. The figures 5a, 5b, 5c show dataset named $1556, 1623$ and 1703 correspondently. As a matter of fact, plotted dataset were already interpolated on the preprocessing step. Although interpolation was done before data exploration, the details about interpolation can be found in section 3.1.3. The names of datasets means only a $timestamp$ in form $HH:MM$ when a dataset was obtained from HMD in laboratory space. Thus the unique name of csv-files on HMD system was guaranteed for the day of experiment. In this thesis the names will be used to identify each of all three datasets. Fig. 5 shows only 3 chosen datasets from those obtained in laboratory space. All datasets indicates the

same behaviour of VR users with HMD looking on the VV projected in VR space as was found out in works [12, 26]. The MAE and RMSE metric results are tend to be similar for every dataset during training and testing.

All traces were recorded over 10 minutes long on average 12 minutes. All traces were then shortened to a precise length of 10 minutes to ensure equal data length for the purpose of visualization and analysis. The observations based on the sample traces can be made similar as it done by *Gül et al., 2020* in their work [12]. The user rarely moves along the y-axis. The y-axis shows the vertical movement that the users could make if they sit down or stand up. Based on the data obtained, users walked around a volumetric object in virtual reality and did not make particularly noticeable and prolonged attempts to examine the object at the lower point of the projection on a laboratory's floor since vertical movement requires more effort to crouch down and stand up. The laboratory space where the dataset was obtained was not cluttered with furniture thus users could walk around the volumetric object projected into their HMD. The figure 6 shows an enlarged y-axis in the range from 400ms to 500ms and thus proves there is no significant change in the vertical position of the user.

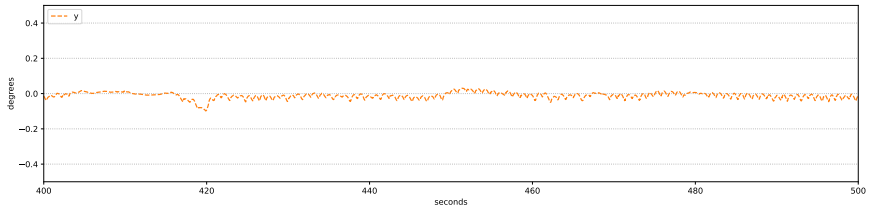


Figure 6: Changes in the user's position along the Y axis in the range from 400ms to 500ms.

Spatial coordinate systems on Windows (HoloLens runs on the Windows Holographic OS) must be right-handed according to the Microsoft documentation. However Unity documentation points that Unity uses a left-handed convention for its coordinate system and experiments performed in Unity with HoloLens 2 during implementation step proved that spacial coordinates in dataset recorded from left-handed system. In both kinds of coordinate systems, the positive X-axis points to the right and the positive Y-axis points up (aligned to gravity). In recorded dataset positive Z-axis points away from a user. Spatial coordinate systems of HoloLens expresses coordinate values in meters. The mean of position for axes are $X = -0.71, Y = 0.01, Z = 1.58$ for dataset 1556. This statistical indicator helps to judge the movement pattern in VR environment projected inside the particular laboratory space. The user's movement along X axis are shifted 0,7 m to the left side in the direction of negative axis. This can be explained by the position of origin of the coordinates when the Unity application was launched. If the application was not launched strictly in the center of the room, but rather closer to the window or wall on one side of the room, then the user had less room space from the side of the window or wall. Y-axis shows no

significant change in the movement and thus rather reflects the difference of the HMD position on the head when user made steps walking in the room. Mean of Y-axis of dataset 1556 shows in average user was about 1,58 m back from the origin of coordinate system. The VV object of a real animated human was placed 3 meters ahead of the user. It seem that users required to step back 1-2 meters to be able to see the whole height of placed VV object respecting the limited Field of view (FoV) of HoloLens 2. Microsoft website[1] states the headset's aspect ratio is 3:2, horizontal Field of view (FoV) of 43° and a vertical of 29°. Indeed the standard deviation for axes $X = 3.23, Z = 3.92$ shows that user circled the hologram (VV of a human) with a average distance 3-4 meters looking on the volumetric object from all sides. For Y-axis $Y_{std} = 0.015$ corresponds to a distance deviation from the measured mean when user was walking in the room without significant movement up (like jumping) or down (sitting down on the floor).

3.1.3 Data preprocessing

As was mentioned in section 3.1.1, the raw sensor data obtained from the HoloLens was unevenly sampled at 60 Hz and had different temporal distances between consecutive samples. The data preprocessing step transforms the data into a format that is more easily and effectively can be processed and visualised. Table 3.2 shows 20 first rows from the resulting dataset after upsampling the positional data with linear interpolation.

timestamp	x	y	z
0.0	0.004954389	0.003402365	0.01010712
5000000.0	0.0048331026667	0.003308667	0.0105062067
10000000.0	0.00471181633333	0.003214633332	0.01090523
15000000.0	0.00459053	0.003120769	0.01130438
20000000.0	0.004485576166	0.00309903333	0.0115486078
25000000.0	0.00438062233	0.00307733667	0.011792899
30000000.0	0.0042756685	0.0030556205	0.01203707
35000000.0	0.004170714667	0.0030339033	0.0122813
40000000.0	0.00406576084	0.00301218867	0.01252553
45000000.0	0.003960807	0.002990472	0.01276976
50000000.0	0.00390328375	0.00300229975	0.0128401975
55000000.0	0.0038457605	0.0030141275	0.012910635
60000000.0	0.00378823725	0.00302595525	0.0129810725
65000000.0	0.003730714	0.003037783	0.01305151
70000000.0	0.003651043834	0.00311298635	0.01315696
75000000.0	0.003571373667	0.00318818967	0.01326241
80000000.0	0.003491703003	0.003263393	0.01336786
85000000.0	0.003412033332	0.0033385963	0.01347331
90000000.0	0.003332363167	0.003413799667	0.01357876
95000000.0	0.003252693	0.003489003	0.01368421

Table 3.2: Interpolated positional data from HoloLens 2.

[1] https://www.microsoft.com/en-us/hololens/hardware

Gül et al., 2020 obtained the similar raw dataset from same HMD and interpolated it to obtain temporally equidistant samples. Same as it was done in work [12], the position and velocity data were upsampled using linear interpolation. Spherical linear interpolation was used to interpolate between rotations represented by quaternions and table 3.3 lists 20 first rows from the resulting dataset after upsampling the rotational data.

qx	qy	qz	qw
0.05225104	-0.0092471	-0.01470939	0.998482825
0.052829134	-0.0094018	-0.01476541	0.9984501
0.053407194	-0.0095559	-0.0148214108	0.99841708
0.053985240	-0.00971031	-0.0148774088	0.9983836
0.054563231	-0.0098646	-0.0149333515	0.99834990
0.054967034	-0.0099280	-0.0147404755	0.99832999
0.055370826	-0.0099914	-0.014547596	0.998309876
0.055774607	-0.0100548	-0.0143547143	0.998289554
0.056178376	-0.0101182	-0.0141618293	0.9982690
0.0565821344	0.0101816	-0.01396894	0.998248298
0.0568467445	-0.0102414	-0.01378002	0.99823527
0.0567612581	-0.01029217	-0.01360108	0.998242075
0.0566757694	-0.01034289	-0.013422148	0.998248830
0.0565902782	-0.01039361	-0.013243212	0.9982555436
0.0565037268	-0.01044992	-0.0130706110	0.998262133
0.0563820024	-0.010691806	-0.01310865	0.998265955
0.0562602738	-0.010933688	-0.01314669	0.998269703
0.0561385409	-0.011175569	-0.013184732	0.9982733762
0.0560168039	-0.011417450	-0.013222770	0.99827697

Table 3.3: Interpolated rotational data from HoloLens 2.

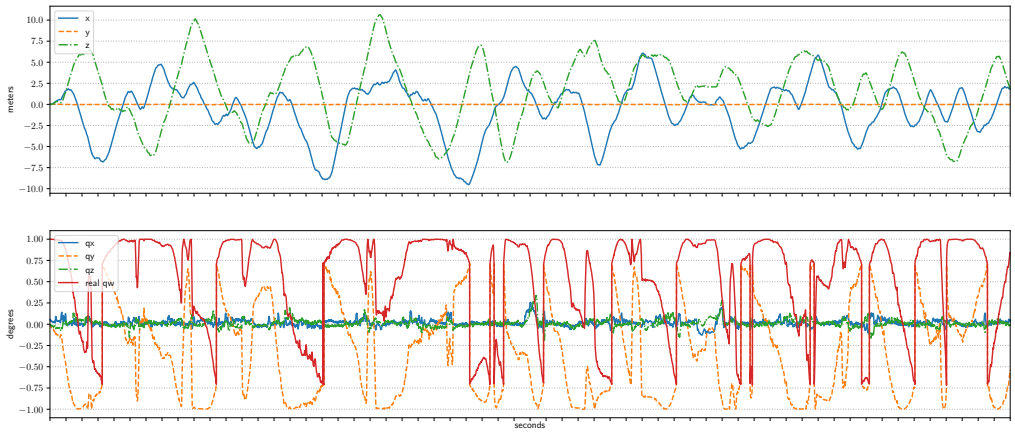


Figure 7: Interpolated 6-DoF dataset's user position and orientation in quaternions.

After the interpolated dataset was plotted as figure 7, the important observations based on the sample trace could be done. While the user position data plots look appropriate for machine learning algorithms, the graph with orientation shows data that is not the perfect case for usage with machine learning technologies and could decrease the prediction rate. The real part qw and the component qy of quaternion have obviously discontinuous (sharp change of sign) making it hard for a predictor to

learn. A orientation on quaternions is used in training, thus this data requires a few additionally preprocessing steps. Usually, when doing calculation with quaternions, quaternions must be normalized to a unit length in order to represent valid rotations [23]. The normalized quaternion can be calculated using formula:

$$U_g = \frac{q}{||q||} = \frac{w}{||q||} + i \cdot \frac{x}{||q||} + j \cdot \frac{y}{||q||} + k \cdot \frac{z}{||q||} \tag{3.1}$$

where $||q||$ is a magnitude and can be found with formula:

$$||q|| = \sqrt{w^2 + x^2 + y^2 + z^2} \tag{3.2}$$

During experiments with quaternions in dataset obtained from HoloLens 2, was detected that quaternion magnitudes $||q||$ in HoloLens dataset are equal to 1. Thus data came from HMD already normalized, so that a quaternion in dataset kept the orientation as it was during user's movement with a magnitude equal to 1.0.

Next, quaternions between neighboring points in obtained dataset represent the very similar orientation made by user wearing HMD step by step. The orientation plot on figure 8 has discontinuities that can be seen on qw line. As a consequence of the discontinuity (sharp change of line from negative to positive area with the same amplitude) the two neighboring quaternions with similar rotation have significant 4D vector space between them. It makes prediction worse what can be proved by RMSE and MAE rotation metrics. Flipping the sign will not affect the rotation, but it will ensure that there are no large jumps in 4D vector space when the rotation difference in rotation space (SO(3)) is small. If negative component of quaternions will be flipped into positive then the dataset, representing same rotation without creating an artificial discontinuity in the space, will be available for model training.

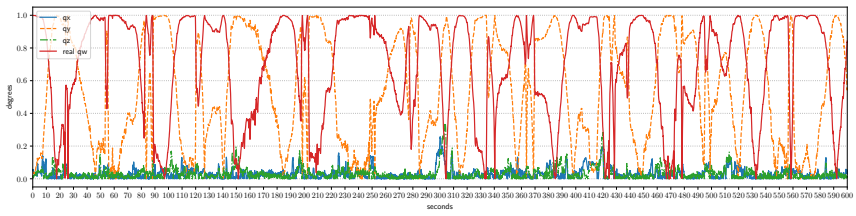


Figure 8: Quaternions from 6-DoF dataset's flipped if their real part is negative.

The figure 9 represents quaternions of the original interpolated dataset on the upper part of the plot and the normalized flipped quaternions on the lower part of the plot. The quaternion's components were flipped only if the if their real part became negative. Different to figure 8 the limit of y-axis is set to [-1, 1] on figure 9 so that

the result of inverting of quaternion is easy co compare to original data. Figure 8 shows plotted data with length of 20 seconds in range 162 - 182 s from both datasets.

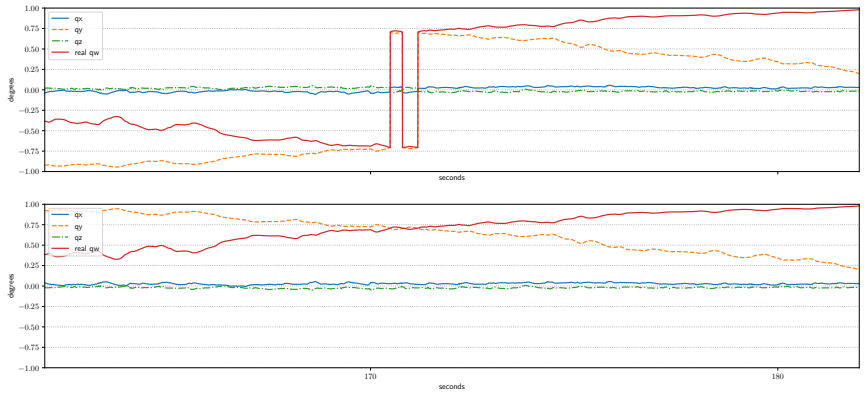


Figure 9: Enlarged quaternion plot with breaks omitted.

Thus the two representations of quaternions were blended into one data set, omitting to discontinuities in the time series as can be seen presented on Figure 8. Indeed, the RMSE and MAE rotation metrics were improved when model was trained with dataset with quaternions without sharp sign changes. More information can be found in section 4.4.

3.2 Model

The section describes the inputs and architecture of evaluated LSTM, GRU and Bidirectional GRU models. A model itself is a mathematical representation that produces expected output based on a given input.

3.2.1 Inputs and outputs

The correct chosen model can discover and learn the patterns in the input dataset while being trained in order to predict a new data after training. In the sections 3.1.2 and 3.1.3 the obtaining and structure of 6-DoF is covered. After data preprocessing step, the interpolated dataset with flipped negative quaternion was used for model training. The dataset itself can not be used with model directly and a sequence data must be prepared to feed as an input into an RNN model. The 6-DoF dataset is a two-dimensional array. The first dimension of 6-DoF dataset represents the number of timesteps of the recorded dataset. The second dimension represents the number of features of input sequence. During data collection a 6-DoF dataset with 10 features was created. Recorded in dataset features are position (x, y, z), orientation (qx, qy, qz, qw) and velocity (x, y, z) data.

However PyTorch's LSTM and GRU models expect all of their inputs to be 3D tensors. Thus 2D data must be converted into 3D data. The meaning of the axes of these tensors is important. The first axis by default is the sequence itself, the second indexes instances in the batches, and the third indexes features of the input. By specifying PyTorch's parameter $batchfirst = true$ the input and output tensors were provided as $(batch, seq, feature)$ instead of $(seq, batch, feature)$. This change does not apply to hidden or cell states and thus tensor containing the initial hidden for each element in the batch must be initialized as $(D * layers, batch, hidden)$. D is 1 for LSTM and GRU Models and equal to 2 for their bidirectional variant.

Usually in machine learning dataset will be split randomly, as there's no dependence from one observation to the other. With time series representing user position and rotation the is not a case and data have to split with respect to time dependencies. The 6-DoF contains a positional data without any seasonal characteristic and there is no obvious way to split data in groups. It is decided to split dataset into three datasets: training, validation and test dataset. The split ratio is 60:20:20 for each corresponding dataset. Thus first 60% of data used for training, next 20% for validation and the rest 20% for testing. No dimensional shuffle was applied to keep the original time dependencies. During training loop on each epoch model trained on training dataset in $train()$ mode that allows the learning process with updating

of model weights. In the end of each epoch model was explicitly set into evaluation mode by calling the $eval()$ function mode to turn off gradients computation and validated on the validation dataset. When model repeatedly trained for 500 epochs, model predicted a new data on never seen before test dataset.

The last step to prepare 6-DoF dataset to be used as model input is using time steps as features. Having a historical data of user position and orientation, the next value, $X(t+n)$ must be predicted by a model from the previous n observations $Xt, X+1$, ..., and $X(t+n-1)$. Since the future values that must be predicted already recorded in the dataset, a sliding-window approach can use prior time steps to predict the next time step and thus turn a time series dataset into a supervised learning problem. With a simple for-loop lagged observations can be created from input by shifting the values in a column by n times and removing the first n columns. The original dataset was interpolated using linear interpolation for position data and SLERP is used for quaternions. Thus, interpolated dataset is an evenly-sampled dataset with a sampling rate of 200 Hz (5 ms). The LAT of 100 ms is used for evaluation. Thus 20 future values correspond the $LAT = 100ms$ must be predicted by LSTM and compared with real data to evaluate the prediction.

Finally, the spit datasets with added sliding window were exported in standard binary npy-files format of NumPy. The format stores all of the shape and dtype information necessary to reconstruct the array correctly even on another machine with a different architecture. Thus the spitting of dataset is not required every time when model trains with different hyperparametes on GPU cluster.

3.2.2 LSTM Model

Recurrent neural networks have recently shown promising results in many machine learning tasks, especially when input and/or output are of variable length and are coming as time series with a sequential order. Unfortunately, the known problem of RNN that was observed many years ago by e.g., *Bengio et al., 1994* that it is difficult to train RNNs to capture long-term dependencies because the gradients tend to either vanish (most of the time) or explode (rarely, but with severe effects) [5]. New approaches are needed to be implemented to reduce the negative impacts of this issue. Since traditional recurrent unit overwrites its content at each time-step, a LSTM unit is able to decide whether to keep the existing memory via the introduced gates. The Long Short-Term Memory (LSTM) has a number of minor modifications [8] since it was initially proposed in work [14]. Analysis done by *Qian et al., 2016* indicates that in the short term users' head movement can be predicted with accuracy > 90% by even using simple methods such as linear regression [19]. However in the longer term it is more difficult to achieve the good result and the average accuracy

drops to about 70% [19]. Thus LSTM model was chosen to evaluate with the 6-DoF dataset based on long term dependencies of the data.

Since LSTM is a special kind of RNN, the RNN architecture will be briefly introduced first. RNN block consist of single computation layer with $tanh$ activation function that is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1. RNN has h_t function of the previous cell state h_{t-1} and current inputX_t. The architecture of LSTM is complexer and consists of several computational blocks that control information flow of information through the cell. The key building block behind LSTM is a structure known as $gates$. They allow LSTM to avoid the weight conflict when making decision which information from the past and current timestamp is important for correct mapping inputs to outputs. In other words, network can decide how to use gates when it is needed to keep or override the information in memory cell or access the current memory cell [14].

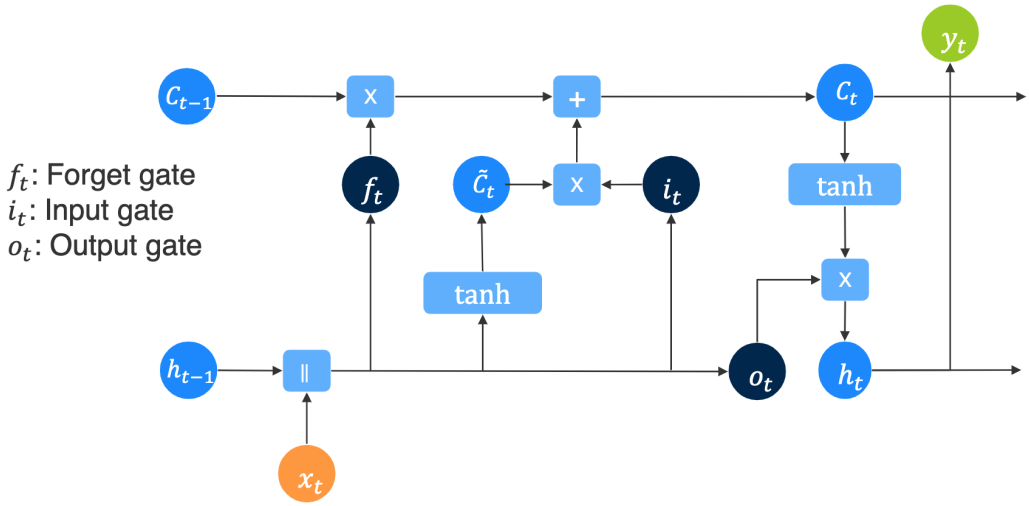


Figure 10: Long Short-Term Memory.

The LSTM architecture is illustrated[2] on Fig. 10. Using first gate f_t model decides which information should be omitted from the cell in that particular time step. The sigmoid function uses the previous state (ht-1) along with the current input xt and computes the cell state using formula:

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \tag{3.3}$$

where f_t is the forger gate, h_t is the hidden state at time t, h_{t-1} is the hidden state of the layer at time $t-1$ or the initial hidden state at time 0, σ is the sigmoid function. All LSTM gates have $sigmoid$ activations that is similar to the $tanh$

[2]Source: Prof. Dr. Tim Landgraf, Lecture 13: Recurrent Neural Networks, WS 20/21: Machine Learning

activation but squishes values between 0 and 1. This function is useful for forgetting the information since any number getting multiplied by 0 is 0 and thus disappears from cell state.

With i_t cell state c_t will be updated. First, the previous hidden state and current input are passed into a sigmoid function on input gate:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \tag{3.4}$$

The transformed values between 0 and 1 meaning 0 means not important, and 1 means important will be multiplied on cell gate $\tilde{c}_t$ with the tanh output of the hidden state and current input:

$$\tilde{c}_t = tahn(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \tag{3.5}$$

New cell state first gets pointwise multiplied by the forget vector anf if these values near 0 they will be dropped from the cell state. The result from the input gate is pointwise added und thus new cell state is created with values that the neural network finds relevant.

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \tag{3.6}$$

The output gate is the last in LSTM calculations and decides what the next hidden state should be. Hidden state is also used to make a prediction because it contains information of previous inputs and thus helps to learn long term dependencies. The sigmoid function gets previous hidden state and the current input and tanh function gets the newly calculated cell state. And similar to previous step tanh output with the sigmoid output are multiplied to decide what information the hidden state should carry. The new cell state and the new hidden is then carried over to the next time step.

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \tag{3.7}$$

$$h_t = o_t \odot tahn(c_t) \tag{3.8}$$

Model implementation, training loop and evaluation are done in Python using PyTorch. Model has input $[batch, sequence, features]$ with sequence equal to 20 last values what corresponds to 100 ms of historical data and 10 features that contain 3 positional, 4 rotation and 3 velocity columns. The batch size was set to 2^7. The hidden dimension set experimentally after parameters grid search on GPU Cluster to be equal 512. Adam optimization algorithm is used, the maximum number of epochs was set to 500, early stopping technique (patience = 7, min. delta = 0,05) was used to avoid overfitting. Additionally, the learning rate was decreased by 50% from initial value of 0.0001 every 30 epochs. The learning rate is a parameter that

determines how much an updating step influences the current value of the weights. Adjustable learning rate was proposed in works [3, 4] and implementing this option had improved prediction and allowed model to learn patterns correctly without overfitting. Weight decay of Adam optimiser experimentally is set to a small value of $1e-12$. Thus this additional term in the weight update rule less causes the weights to exponentially decay to zero, large weights were less penalizes and model could successfully learn the long term dependencies and thus constantly decrease both training loss and validation loss and stabilize them at a specific point.

The high performance was achieved event without additional activation functions with simple one-layered architecture represented below:

```
LSTMModel1(
(lstm): LSTM(10, 512, batch_first=True)
(fc): Linear(in_features=512, out_features=7, bias=True)
==========================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================
LSTMModel1                               [128, 20, 7]              --
        --- LSTM: 1-1                    [128, 20, 512]        1,073,152
        --- Linear: 1-2                  [128, 20, 7]              3,591
==========================================================================
Total params: 1,076,743
Trainable params: 1,076,743
Non-trainable params: 0
==========================================================================
```

Listing 3.1: One-layered LSTM with sliding window

Since only position and rotation data of similar range is used in 6-DoF dataset, no scaler for the features is applied. Experiments showed that normalization of values between [0..1] and between [-1..1] results to higher MSE and RMSE. Already preprocessed interpolated datasets with flipped negative quaternions is loaded in with respect to sequential order as training, validation and test dataset from npy-files. Additional extended LSTM architectures were implemented and tried with HoloLens 2 6-DoF dataset. For example, the non-linear activation functions $ReLU$ and $Mish$ were tried in order to get more sensitivity to the activation sum input and avoid easy saturation. Thus the nodes in model should be only deactivated if the output of is less than 0. Model variant with $ReLU$ experimentally resulted to produce higher MAE and RMSE compared to LSTM1 and therefore was rejected for a final deployment. Since LSTM2 Model has the similar architecture as following afterwards LSTM3, it is not listed separately

LSTM3 uses a new activation function $Mish$ that was presented in machine learning scene in 2019 in work [17] and is a self-regularized non-monotonic activation function which can be mathematically defined as $f(x) = xtanh(softplus(x))$. Mish

is a smooth, continuous activation function and allows to have better gradient flow compared to ReLU that tends to have a lot of sharp transitions [17]. The LSTM3 model improved MAE and RMSE mertics. Model's batch size changed to 2^{10}. Additional linear layer with additional $Mish$-function is added in order to double the hidden size of LSTM. Weight decay of Adam optimiser is experimentally set to $3e-14$ with LSTM3 model.

```
LSTMModel3(
(lstm): LSTM(10, 512, batch_first=True)
(mish_1): Mish()
(fc_1): Linear(in_features=512, out_features=1024, bias=True)
(mish_2): Mish()
(fc_2): Linear(in_features=1024, out_features=7, bias=True)

==========================================================================
Layer (type:depth-idx)                   Output Shape
    Param #
==========================================================================
LSTMModel2                    [1024, 20, 7]              --
        --- LSTM: 1-1         [1024, 20, 512]            1,073,152
        --- Mish: 1-2         [1024, 20, 512]            --
        --- Linear: 1-3       [1024, 20, 1024]           525,312
        --- Mish: 1-4         [1024, 20, 1024]           --
        --- Linear: 1-5       [1024, 20, 7]              7,175
==========================================================================
Total params: 1,605,639
Trainable params: 1,605,639
Non-trainable params: 0
==========================================================================
```

Listing 3.2: LSTM3 with Mish activation function

LSTM4 Model is a three-layered stacked LSTM with introduced additional dropout added after all but last recurrent layer and Mish activation function. This design decision is made in order to try whether adding more components to the neural network could mean the improvement upon simpler model on 6-DoF dataset. By adding more LSTM layers the model parameters that have to be trained were increased in 5 times from 1,605,639 to 7,901,191 trainable parameters. When the model parameters are getting large in count, the model gets more complex, having hard time fitting on the training instances as it needs to optimize parameters in a way that can optimally fit the training instances. The time need for training increased noticeable even on GPU Cluster. Finally, LSTM4 resulted in significant higher MAE and RMSE metrics and thus this architecture is rejected.

This models LSTM1 and LSTM3 are considered to be the best evaluated LSTM models that can predict the future data based on past 20 values (100 ms) in 6-DoF VR environment using sensor data from HMD for LAT of 100 ms. Although sufficient

results for prediction and low MAE and RMSE metrics are already obtained with LSTM model, the GRU and bidirectional models will be implemented in order to evaluate their performance and potentially to find the better model architecture.

3.2.3 GRU Model

Another approach called a gated recurrent unit (GRU) can adaptively capture dependencies of different time scales without having a separate memory cells [8]. It is similar to an LSTM, but only has two gates - a reset gate and an update gate. Although architecture does not provide an output gate, with fewer parameters it can generally easier and faster be trained than LSTM. GRU Model can catch the long-term dependencies in the data obtained from HMD that are otherwise are hidden by the effect of short-term dependencies from the standard RNN models. This chapter decribes GRU model that implemented to predict future values with 6-DoF dataset obtained from HoloLens 2.

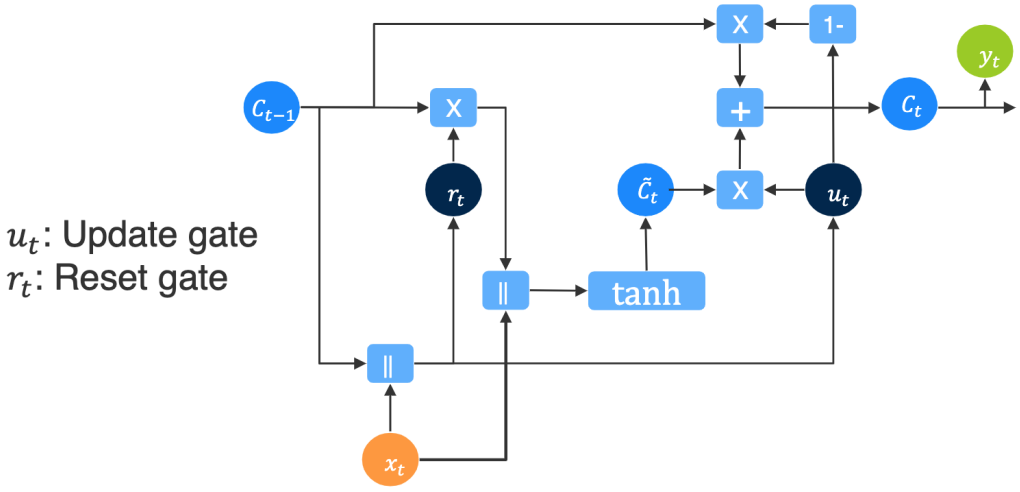


Figure 11: Gated Recurrent Unit.

The GRU architecture is illustrated[3] on Fig. 11. GRU avoids the vanishing gradient problem of a standard RNN with only two gates that decide what information should be passed to the final output. The update gate plugs x_t into the network unit and multiplied with its own weight W_{iz}. The same multiplication is done with previous hidden state $h_(t-1)$ that has its own weight W_{hz}. Both results are added together

[3]Source: Prof. Dr. Tim Landgraf, Lecture 13: Recurrent Neural Networks, WS 20/21: Machine Learning

and a sigmoid activation function is applied to squash the result between 0 and 1. The mathematical expression of this calculation is as following:

$$z_t = \sigma(W_{iz}x_t + b_{iz} + W_{hz}h_{t-1} + b_{hz}) \tag{3.9}$$

where z_t is an update gate, h_t is the hidden state at time t, h_{t-1} is the hidden state of the layer at time $t-1$ or the initial hidden state at time 0, σ is the sigmoid function. With these matrix multiplications model can determine how much of the past information from previous time steps needs to be passed to the next to predict future values.

The reset gates does what its name means - gate can reset the state of the model and thus model decides how much of the past information to forget. It is an useful option when context changes in the historical data and previous values are not more relevant to produce future values. With powerful update gate the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient. The formula of the reset gate is:

$$r_t = \sigma(W_{ir}x_t + b_{ir} + W_{hr}h_{t-1} + b_{hr}) \tag{3.10}$$

Both gates will affect the final output. A new memory content will use the reset gate to store the relevant information from the past. It is calculated as follows:

$$n_t = tahn(W_{in}x_t + b_{in} + r_t * W_{hn}h_{t-1} + b_{hn}) \tag{3.11}$$

As the last step, the network calculates vector which holds information for the current unit and passes h_t to the network. The formula shows that update gates is used to determine what information from previous steps will be passed into the memory. Additionally, calculated recently new memory gate n_t controls the amount from current data to be added into long memory. That is done as follows:

$$h_t = (1 - z_t) * n_t + z_t * h_{t-1} \tag{3.12}$$

GRU model implementation, training loop and evaluation are done similar to LSTM in Python using PyTorch. Model has same input $[batch, sequence, features]$ with sequence equal to 20 last values and 10 features (3 positional, 4 rotation and 3 velocity columns). The batch size was increased to 2^9. The hidden dimension is 512 nodes. Same as with LSTM, Adam optimization, the extended version of stochastic gradient descent and nowadays common algorithm for ML tasks, used for GRU Model. Adjustable learning rate is modified to decrease by 60% from initial value of 0.0001 every 50 epochs. Weight decay kept the same value of $1e-12$.

The maximum number of epochs was set to 500, early stopping technique with same patience and delta as in LSTM was used to avoid overfitting. Model requires less epochs to learn and can predict better than LSTM. Although the error decreases very slowly after 150-200 epochs, the model converged to a smallest achievable error after 500 epochs. The model is overtrained with 1000 epochs if trained without early stopping technique.

Different to LSTM Model, the best performance was achieved with pure GRU model without additional activation functions with simple one-layered architecture represented below:

```
GRUModel1(
(gru): GRU(10, 512, batch_first=True)
(fc): Linear(in_features=512, out_features=7, bias=True)
)
==========================================================================
Layer (type:depth-idx)                   Output Shape
    Param #
==========================================================================
GRUModel1                        [512, 20, 7]              --
        --- GRU: 1-1             [512, 20, 1024]             804,864
        --- Linear: 1-2          [512, 20, 7]               3,591
==========================================================================
Total params: 808,455
Trainable params: 808,455
Non-trainable params: 0
==========================================================================
```

Listing 3.3: GRU1 with Sliding Window

From listing above is clear, that pure GRU1 has 25% less trainable parameters as pure LSTM1 and twice less trainable parameters as LSTM3 that uses additional activation and linear functions. Moreover the result of prediction of future values are preciser than those obtained with LSTM1 and LSTM3 and has significant smaller the MAE and RMSE metrics (more information about results of prediction with LSTM and GRU models can be found in chapter 4.4.2 and 4.4.3).

Additionally to GRU1 several different architectures were implemented and tested. The GRU2 model has $ReLU$-activation function and GRU3 has $Mish$-activation function. Similar to LSTM, using of $ReLU$ did not improved the predictions. Different to LSTM, using of $Mish$ also worsened the results.

Different quite similar architectures using $Mish$ activation function were implemented and tested with parameters grid search on GPU Cluster. GRU31 using only one $Mish$ activation compared to LSTM3 and GRU3 with two activation layer accompanied with linear layers. GRU32 and GRU35 use additionally dropout layer(s) with

different parameters and combinations with linear layer(s). In GRU33 adaptive max pooling over an input signal is done to in part to help over-fitting by providing an abstracted form of the representation. Both models performed worse then those one without dropout techniques even though it was tried to train the model longer for 1000 or 2000 epochs. Every architecture including GRU1 was also tried as 3-layered and 8-layered stacked GRU and all results similar to stacked LSTM were significant worse and took noticeable more computational time as 1-layered variants.

Thus model GRU1 is considered as best evaluated model that predicts accurately the future values for LAT of 100 ms based on past 20 values (100 ms) in 6-DoF VR environment using sensor data from HMD.

3.2.4 Bidirectional GRU Model

The last RNN variant evaluated in this master thesis is a bidirectional GRU model. From related works is known that despite the complex architecture and more historical data available for analysis and training, the prediction results do not exceed the results of a simple model. This section aims the goal to check whether the results of prediction on 6-DoF dataset are similat to those from related works.

A typical state in an RNN (simple RNN, GRU, or LSTM) relies on the past and the present events. A state at time t depends on the states $x_1, x_2, ..., x_{t-1}, x_t$. However, there can be situations where a prediction depends on the past, present, and future events. Theoretically if the future information will be available during the prediction, it can increase the prediction accuracy. For example, predicting a user position in the future to be included in a sequence of movements might require us to look into the future, i.e., particular future position at time t could depend on a future event like turning right, left, accelerating or stopping. Bidirectional RNNs enables straight (past) and reverse traversal of input (future). Computationally Bi-RNNs are a combination of two simple RNNs for moving data beginning from the start of the data sequence, and parallel to that a backward moving occurs beginning from the end of the data sequence. Based on the type of used RNN blocks, Bi-RNN can either be simple RNNs, GRUs, or LSTMs. This section describes the usage if Bi-GRU Model because the best results were achieved with GRU1 Model and therefore no Bi-LSTM implementation was undertaken.

A Bi-GRU has an additional hidden layer to accommodate the backward training process. At any given time t, the forward and backward hidden states are updated as follows:

$$A_t(Forward) = \sigma(W_{xa}^{forward} x_t + b_a^{forward} + W_{aa}^{forward} A_{t-1}) \tag{3.13}$$

$$A_t(Backward) = \sigma(W_{xa}^{backward} x_t + b_a^{backward} + W_{aa}^{backward} A_{t+1}) \tag{3.14}$$

where σ is the activation function, W is the weight matrix, and b is the bias. The hidden state at time t is given by a combination of $A_t(Forward)$ and $A_t(Backward)$. The output at any given hidden state is:

$$O_t = H_t * W_{ay} + b_y \tag{3.15}$$

During the training of Bi-GRU forward and backward passes happening simultaneously and thus updating the weights for the two processes could happen at the same point of time. If standard RNN Backpropagation Through Time algorithm would be applied to Bi-RNN, it could lead to erroneous results. Thus, modified algorithm is used for training a Bi-RNN to accommodate forward and backward passes separately. Thereby both the forward and backward passes together train bidirectional model.

Model implementation done similar to GRU in Python with PyTorch. When parameter $bidirectional = True$ is specified in $nn.GRU$, PyTorch takes care about correctness of forward and backward passes and combines two GRU model by doubling the hidden dimension size. The output will be $(batch, seqeunce, hidden_size * 2)$ where the $hidden_size * 2$ features are the forward features concatenated with the backward features. The model is listed below:

```
GRUModelBiDir1(
(init_linear): Linear(in_features=10, out_features=10, bias=True)
(bi_gru): GRU(10, 512, batch_first=True, bidirectional=True)
(out_linear): Linear(in_features=1024, out_features=7, bias=True)
)
==========================================================================
Layer (type:depth-idx)                        Output Shape
    Param #
==========================================================================
GRUModelBiDir1                          [512, 20, 7]              --
        --- Linear: 1-1                 [512, 20, 10]             110
        --- GRU: 1-2                    [512, 20, 1024]           1,609,728
        --- Linear: 1-3                 [512, 20, 7]              7,175
==========================================================================
Total params: 1,617,013
Trainable params: 1,617,013
Non-trainable params: 0
```

Listing 3.4: Bidirectional GRU Model

The interesting thing that PyTorch requires the initialisation of h_0 initial state to use doubled amount of layer dimension. This one layered GRU will be initialized in PyTorch as 2-layered. It was already proved by experiments on GPU Cluster that any combinations of stacked RNN Models leads to lower prediction accuracy. Anyway it

is worth to evaluate whether the accommodated separately forward and backward passes could improve the model performance.

```
# Initializing hidden state for first input with zeros
h0 = torch.zeros(self.layer_dim * 2, x.size(0), self.hidden_dim).
    requires_grad_()
```

Listing 3.5: Bidirectional GRU h0 Layer

Listing above shows GRUModelBiDir1 that was implemented and evaluated additionally with GRUModelBiDir2 model with $Mish$ activation function. Same as in work of *Chang et al., 2020*, the Bi-GRU performs the worst among all models. The details of evaluation can be found the section 4.4.4.

3.2.5 Development

This section presents as well the development of the Unity application for obtaining the dataset, as the training loop in PyTorch used for all models and shortly describes a process of developing a GPU application.

Unity application

An application was developed in Unity with the Mixed Reality Toolkit (MRTK) and deployed on HoloLens 2. The goal of the application is to obtain the user position and orientation during the time when a user wears a HMD. MRTK provides a cross-platform input system, components, and common building blocks for spatial interactions. The Unity Application was finally built using Visual Studio and deployed using Wi-Fi connection. To enable the deployment on HoloLens 2, Windows 10 has Developer Mode to be turned on and HoloLens must be paired with a Visual Studio using a PIN displayed on HoloLens. Later in final App version, that was used for data collection, the volumetric animated object was added in scene. It consist of 359 frames each with its own mesh. It drastically increased the size of Unity App and the deployment using USB was used instead of Wi-Fi connection.

In Unity, the Main Camera is always the primary stereo rendering component attached to HMD and it is rendering everything the user sees [4]. The starting position of the user is set to $(0, 0, 0)$ during the application launch and the Main Camera tracks movement of the user's head. Although HoloLens allows to build a world-scale application, the room-scale experience was selected for spatial coordinate system. This lets users to walk around within the 10-meter boundary what is quite enough

[4] https://docs.microsoft.com/en-us/windows/mixed-reality/develop/unity/camera-in-unity

for user's movements inside the laboratory space and simultaneously watching the volumetric video object. Fig. 13 illustrates implemented Unity 3D Application window with placed obj-animation, underlying code and scene.

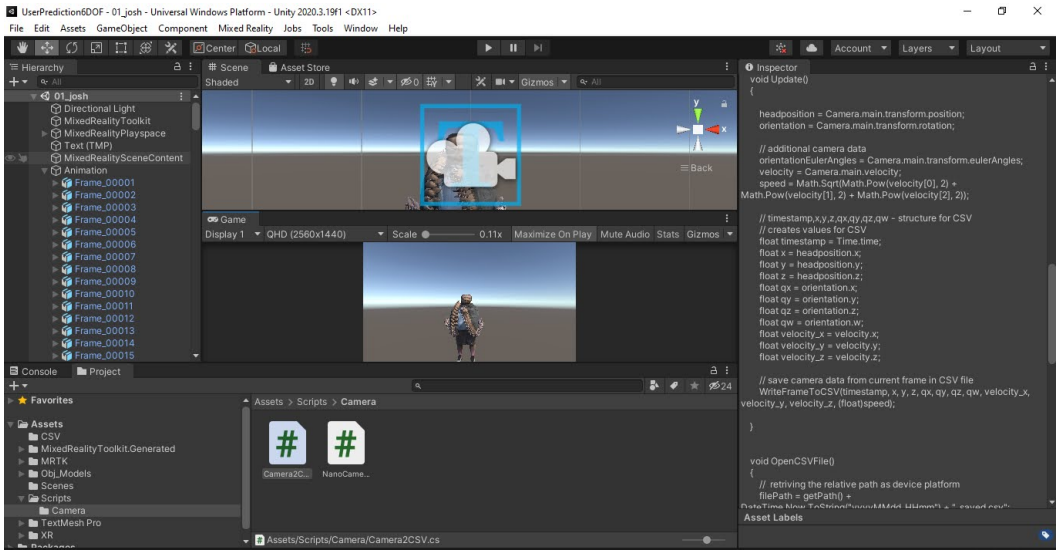


Figure 12: Unity 3D UserPrediction6DOF Project with placed obj-animation, underlying code and scene.

Figure 13: Photo made by Hololens 2 camera during the Unity Application running on HMD with the volumetric animated object.

As this research aims to find an approach to reduce the M2P latency during rendering and delivering the volumetric content to end-user device, the volumetric animated object was placed three meters ahead of the user in Unity application. Fig. 13 presents the photo made in mirror by HoloLens camera with placed HMD on master thesis author's head. The volumetric object placed in the room between author and a mirror in the room and Fig. 13 shows what author saw at that time on the HoloLens screen. Users wearing HMD thus were asked to look on animated volumetric object and to move freely inside the laboratory space. Animation frames of VV object are saved as an obj-files and contain information about the geometry of 3D objects. obj file allows color and texture information to store in an associated file format called the Material Template Library (MTL). Multi-color geometric models render using these two files together. obj-files are not supporting animation. The animation of the VV object to in order to create

an imitation of placed VV in VR environment was quite challenging during programming in Unity. Finally, volumetric animation object created programmatically with a loop that cycles through each individual frame with the corresponding mesh and calculated speed to give the user the experience of a real volumetric video.

User position and rotation data are logged in csv-file. This raw data has been converted into datasets on the preprocessing step described in section 3.1.3 and thus original interpolated dataset, the transformed with flipped negative quaternions and several normalised datasets were used in experiments during model development and hyperparameters search.

Training and evaluation

All evaluated models are implemented in Python with Python-native framework PyTorch. After model with the inner layered structure was initialized using tensor input and output size, the training loop need to be explicit implemented in PyTorch.

The main components of training loop are the model itself, the loss function, and the optimiser. Models creation are described in section 3.2. PyTorch's loss function $MSELoss()$ is used for creation of criteria that help to measure the error of mean squared format that is squared L2 normalization. It computes the average value of the squared differences that lie between the predicted and actual values. The value of $MSELoss()$ will always be a positive number, irrespective of whatever signs the predicted and actual values have. During training loop on every epoch when working with batched input, PyTorch will adjust model weight parameter in such way that value of $MSELoss()$ tends to be 0.00

PyTorch's optimizer is an algorithm used to change the attributes of the neural network such as weights and learning rate in order to reduce the losses. The standard optimiser is Gradient Descent used heavily in linear regression and classification algorithms. The know problem of the vanishing gradient was first discovered by Hochreiter, a German computer scientist, back in 1991 [14]. Bengio, a professor at the University of Montreal, also discovered the vanishing descent problem but a bit later – he wrote about it in 1994 [5]. Network weights are assigned at the start of the neural network with the random values, which are close to zero, and from there the network trains them up. Briefly say, the vanishing descent problem rises from the starting weight value close to zero and repetitive multiplication of inputs $x_t, x_{t-1}, x_{t-2}, ..., x_{t-n}$ by this value so that mathematically gradient becomes less and less with each multiplication. The lower the gradient is, the harder it is for the network to update the weights and the longer it takes to get to the final result.

Additionally, in RNN the training of current cell is based on inputs that are coming from previous untrained layers. So, because of the vanishing gradient, the whole network is not being trained properly.

Adam optimizer is the extended version of stochastic gradient descent and was first introduced in 2014. The name is derived from adaptive moment estimation and uses estimations of the first and second moments of the gradient to adapt the learning rate for each weight of the neural network. The first moment is mean, and the second moment is uncentered variance with calculated with no mean subtracted. The moments are not treated as network's parameters nor constants, they can be thought of as some intermediate results of computing the output of the layer. PyTorch's optimizer $Adam$ is used in evaluated models to reduce the training and validation losses in batches and epochs.

Training of any kind of neural network is a repetitive process, looping back and forth between forward-prop and back-prop. During each epoch in training, there are two stages: training and validation. After each training step, the network's weights are tweaked a bit to minimize the loss function. On the validation step the current state of the model is evaluated to check if there has been any improvement after the most recent update. When two for loops are created, the $train()$ mode must be activated during training and the $eval()$ mode during the validation. With the $train()$ mode the network's weights will to be updated in order to reduce the training loss on the next epoch, the $eval()$ mode signals the model that there is no need to calculate the gradients. The model parameter, such batch size, learning rate and weigh decay must be set correctly so that the training and validation can decreasing parallel on each epoch. Normally, on the very first epochs, the validation loss is greater than the training loss and model is unable to accurately model the predict a validation data, and hence generates higher error. During model training on every epoch the training loss and validation loss both decrease and after some amount of epochs stabilize at a specific point. The model converges upon a final solution and it is a good fit when the loss is low and stable and in this moment model's training must be stopped. It is possible to train model for a very long period so that the validation loss stops to decrease and starts to increase again. It means model is overtrained and cannot generalize on new data. Regarding the very low training loss, model produces higher error on validation test and high error on test dataset. The early stopping technique was used to prevent overfitting with stopping the training when both losses are low and stable. The early stopping algorithm is implemented manually in this thesis using Python and build in final Python application $UserPrediction6DOF$.

Hyperparameter search

Model's training and validation loss are calculated on every epoch and in best case must decrease parallel every epoch and stabilize on very low point. The model hyperparameters influences the training process and can drastically change the final result, leading to overfitted, underfitted or successfully learned model. Hyperparameters are the variables which determines the network structure or how the network is trained. In this master thesis the following hyperparameters must be tuned: the number of hidden layers, optimizer's learning rate, weight decay, batch size, and number of epochs. For example, the learning rate of Adam optimiser is a hyperparameter because it is set before model see the the training data. On the other hand, the weights of a neural network and the first and second moments of the Adam's gradient are not its hyperparameters because they are trained and modified during training loop.

For the hyperparameter tune the grid search is applied. Grid search is a process that searches exhaustively through a manually specified subset of the hyperparameter space of the evaluated model. Every training of the model during 500 epochs on 6-DoF dataset takes on CPU up to 2 hours with up to 95% of CPU usage. The grid for the exhaustive search is created with Bash-script that sets $hidden - dim$ and $batch - size$ in the loop to be equal power of two; $lr - adam$ and $lr - multiplicator$ are float numbers with a floating-point step.

The hyperparameters search is done using VCA GPU cluster which is installed with the $SLURM$ resource manager/scheduler for GPU based HPC (High Performance Computing). $Singularity$ container is similar to a light-weight Virtual Machine is used to containerize the application with the required environment and software stack and submit the container to run as a job in the cluster. Using VCA GPU the computation time for every job (one model training with one set of hyperparameters) is reduced from 2 hours to 15 minutes. For every evaluated model in average over hundred jobs were launched for the initial hyperparameters search and few dozens for parameter tuning. With $export$ command hyperparameters can be set with Bash as environment variables and with $nohup$ all of jobs can be scheduled for computation with preventing the jobs from being aborted automatically if the connection to the remote machine is closed. The $UserPrediction6DOF$ application detects whether a parallel computing platform $cuda$ is available and if it is a case then reads the model parameters from environment variables. The result of every job arrives as tar-archive with all written out-files. Therefore Bash script is created for automatically opening of hundreds archives, finding the evaluation metrics and merging the result in one log-file for future analyse and visualisation purposes.

4 Evaluation

Chapter describes the evaluation metrics and performed experiments, visualises prediction results.

4.1 Goal of evaluation

The goal of model evaluation is am estimation of the generalization accuracy of a model on future unseen data. This master thesis aims to evaluate whether RNN neural networks modification as LSTM, GRU and bidirectional variant are able to reduce the positional and rotation error for given look ahead time of 100 ms. This LAT is higher than normal acceptable latency in VR application and even higher that measured M2P latency in cloud streaming platform presented in work [13]. Since the successfully trained best evaluated model is intended to be build in the server infrastructure, the goal of evaluation is to find out how using of RNN model can reduce the positional and rotational error and thus improve the quality of delivered from cloud server VV content by calculating the proper future 2D image from volumetric data.

To evaluate all described above models, a Python-based application used for training and processing of the recorded via HoloLens 6-DoF datasets. Below, first Baseline model, the experimental setup and the evaluation metrics are discussed, before presenting the obtained results and discussing the limitations of evaluated models.

4.2 Evaluation metrics

Evaluation metrics are used to measure the quality of the predictions made by machine learning model. This thesis similar to work [12] uses two of the most common metrics Mean Absolute Error (MAE) and root mean squared error ($RMSE$).

Mean Absolute Error (MAE): MAE measures the average error over the test sample of the absolute differences between prediction and actual observation without considering their direction.

$$MAE = \frac{1}{n}\sum_{j=1}^{n}|y_j - y_{mean}| \tag{4.1}$$

Root mean squared error ($RMSE$): RMSE measures the average magnitude of the error with square root of the average of squared differences between prediction and actual observation.

$$RMSE = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - y_{mean})^2} \tag{4.2}$$

Both MAE and RMSE express average model prediction error in units of the variable of interest, both metrics can range from 0 to $\inf$ and are indifferent to the direction of errors. They are negatively-oriented scores, which means lower values are better[1]. Since the errors in $RMSE$ are squared before they are averaged, the RMSE gives a relatively high weight to large errors. Normally, The $RMSE$ result will always be larger or equal to the MAE.

For calculation the metrics for positional data the Euclidean distance (L2 norm) is used. Rotational metrics calculated with a distance between quaternions represented as an angle between their 3D orientations using a formula:

$$angularDistance = 2 * arccos(real(p * conj(q))) \tag{4.3}$$

where p and q are unit quaternions representing two rotations in the same basis and $q*$ denote the quaternion conjugate.

4.3 Baseline model

To understand, how actually good evaluated model predicts and helps to reduce the M2P latency, some essentially a simple model that acts as a reference in a machine learning project must be implemented first. Baseline model can lack complexity and may have little predictive power. The LSTM and GRU model should predict much better than a Baseline model and thus comparing the metrics it can be understood how reasonable is the implementing and using of chosen approach. It is intended to use a Baseline model as benchmarks for trained models. There is no rule for what is

[1] https://medium.com/human-in-a-machine-world/mae-and-rmse-which-metric-is-better

good or bad model's prediction. The criteria of model evaluation depends on the dataset and use case. A mean square error gives a values in units of the original dataset. For example, if model predicts the prices of apartments in Berlin, then MAE of 1000 is a very good result and market's players would desire to have a trained model to work on the real estate market. However, it is abysmal for a model that predicts the price of average lunch in Berlin's restaurant. Thus a simple predictor is needed to predict a future data so that a predicted values can be compared with the real values using same evaluation metrics as is used for RNN models.

The Baseline in this thesis is similar to the reference model used in the work [12] and represents the operation of the system without prediction. It is assumed that the prediction time is set equal to the M2P latency of 100 ms such that the prediction completely eliminates the latency. Implemented Baseline is a deterministic model, meaning that it produces an expected output given the same input. For LAT of 100 ms a prediction time N is equal to 20 samples. The position and rotation data x_t is simply propagated N samples ahead in the Baseline prediction and set as the user position and rotation at time $x_t + N$, i.e. $Baseline(x_t) = x_{t+N}$.

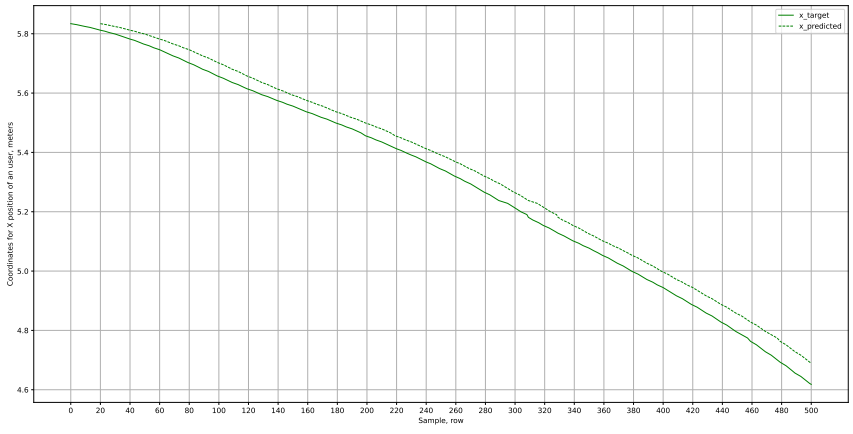


Figure 14: Outputs of Baseline Model for x-axis.

The Fig. 14 shows the first 500 real values and the corresponding output of the baseline for positional axis x. From the plot of the Baseline model outputs, it is clear that the model is 20-step behind reality. It copies with a given delay the falling trend and all fluctuations of the given axis.

The Fig. 15 shows the 500 real values and the corresponding output of the baseline for all three positional axes x, y, z. The plot samples 500 elements starting from the 2500 row and thus no missing data is seen on Baseline output for the first 20

elements as it plotted in Fig. 14. However, the Fig. 15 highlights the limitations of the usage of naive predictor that will deliver 2D image created from volumetric video content with a delay of 100 ms that is unappropriated delay for a human to experience in VR application without physical consequences like motion sickness [2]. The mean square error for the all three positional axes $MAE_{pos} = 0.067$m and root mean square error $RMSE_{pos} = 0.068$m meaning the average distance between predicted position and the real position is equal to almost 7 cm. It is not crucial when an user looks on the big VV object, like a volumetric humans hologram, from the distance of 3-4 meters. But if an user interacts with the small VV objects presented in the VR environment, then this distance became a significant difference between what user will see with a delay and where the VR object would be placed for delayed position.

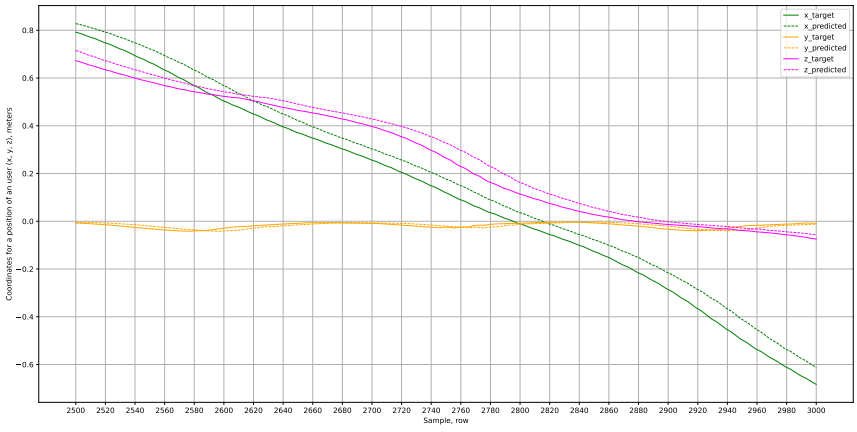


Figure 15: Outputs of Baseline Model for x, y and z axes.

It is worth to mention that in the case if real data is neither increasing nor decreasing in the given interval and fluctuates near the constant value, as it seen at y-axis, than the delay of the Baseline output is more not obvious when visualized due the overlapping of two graphs. In fact, the Baseline outputs have the same delay as those one with good visualized delay as, for example, x-axis.

Fig. 16 shows the Baseline outputs for quaternions components qx, qy, qz and qw. Same as with positional data, the same delay of 20 samples is present on rotational data. For all four quaternion components calculated metrics are: $MAE_{rot} = 14.61°$ and $RMSE_{rot} = 21.24°$.

Because metrics of rotational data is measured in degrees, for visualization purposes, orientations are given in the Fig. 17 as Euler angles (yaw, pitch, roll), although prediction is performed in the quaternion domain.

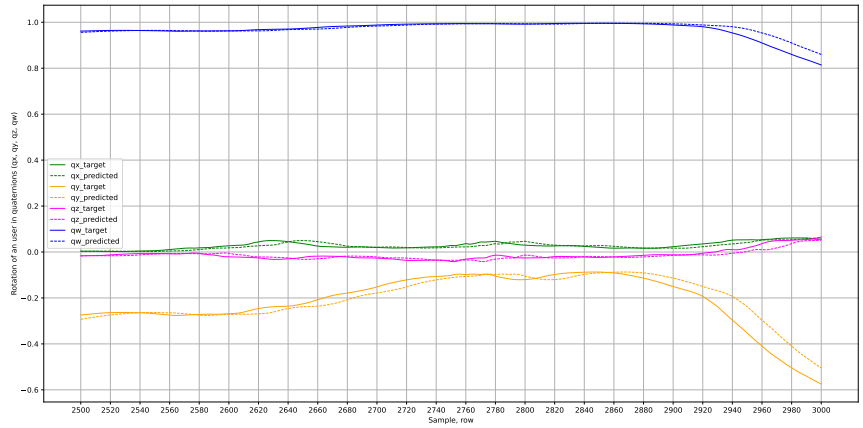


Figure 16: Outputs of Baseline Model for quaternions components qx, qy. qz and qw.

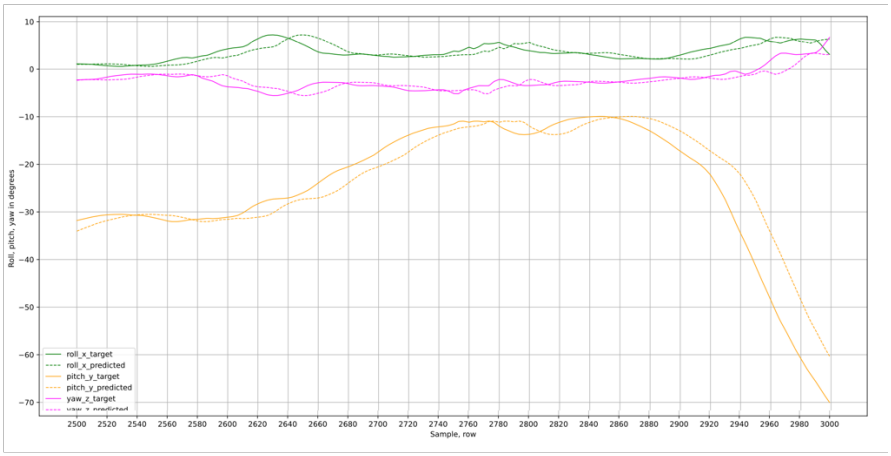


Figure 17: Outputs of Baseline Model for rotation data represented as Euler angles.

The evaluated RNN model is considered to be successfully trained if the MAE and $RMSE$ metrics at least for positional or rotational data can show significant improvement comparing to a Baseline output.

4.4 Experiments

In the experiments implemented models were trained with different hyperparameters and evaluated using MSE and $RMSE$ metrics.

4.4.1 First experiments

Datasets

This section describes the difference of prediction results done by a model LSTM1 on different types of 6-DoF datasets. As already stated in section **??**, original row dataset is not used in model training and was preprocessed before training was started. The following dataset were created and tried:

- **Interpolated dataset:** Row dataset was interpolated so that missing values were inserted with linear interpolation for positional data and spherical linear interpolation for rotational data.
- **Flipped dataset:** Interpolated dataset with flipped negative quaternions so that neighboring quaternions representing same rotation without significant 4D vector space between them
- **Normalized dataset:** Feature scaling (Min-max normalization) is applied on dataset's positional values to scale data in the range [0..1].
- **Position:** Only positional data (x, y, z) as separate dataset to predict future position.
- **Rotation:** Only rotational data (qx, qy, qz, qw) as separate dataset to predict future rotation.

Figures 18 and 19 shows the predictions of LSTM1 model on interpolated dataset. The mean square error for the all three positional axes $MAE_{pos} = 0.019$m and root mean square error $RMSE_{pos} = 0.028$m meaning the average distance between predicted position and the real position reduced from 7 cm to 2 cm if LSTM1 on interpolated dataset is used instead of Baseline. For all four quaternion components calculated metrics are: $MAE_{rot} = 16.92°$ and $RMSE_{rot} = 23.28°$. LSTM1 predicts the rotation on interpolated dataset worse than the Baseline model. The goal of

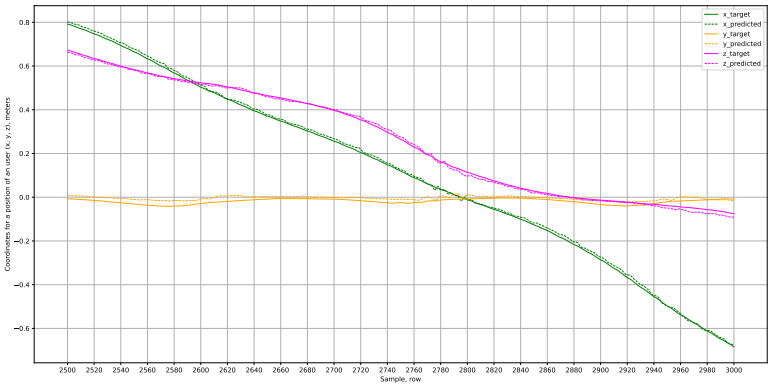


Figure 18: Outputs of LSTM1 model on interpolated dataset for x, y and z axes.

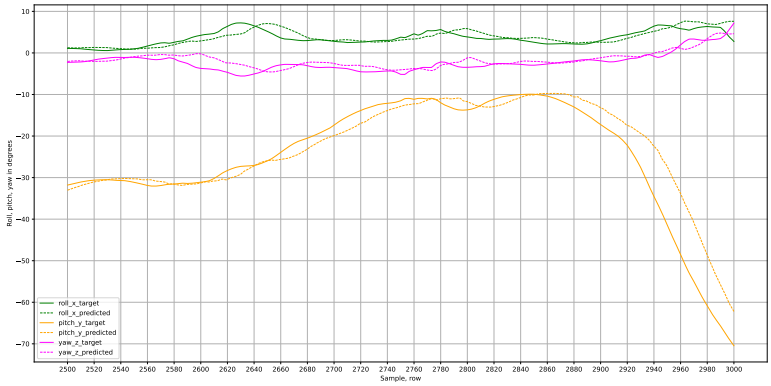


Figure 19: Outputs of LSTM1 model on interpolated dataset for qx, qy, qz and qw axes.

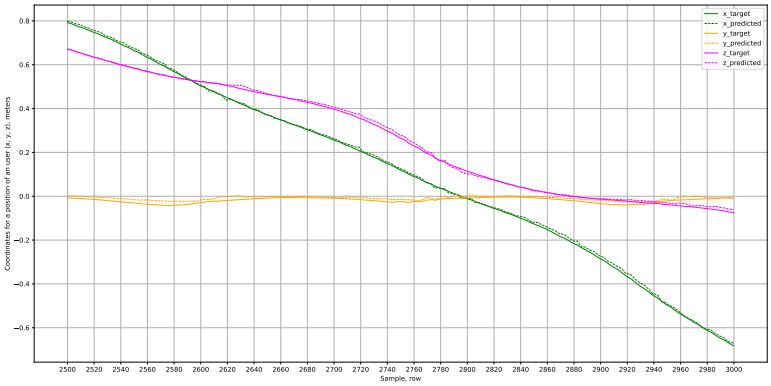


Figure 20: Outputs of LSTM1 model on dataset with flipped negative quaternions for x, y and z axes.

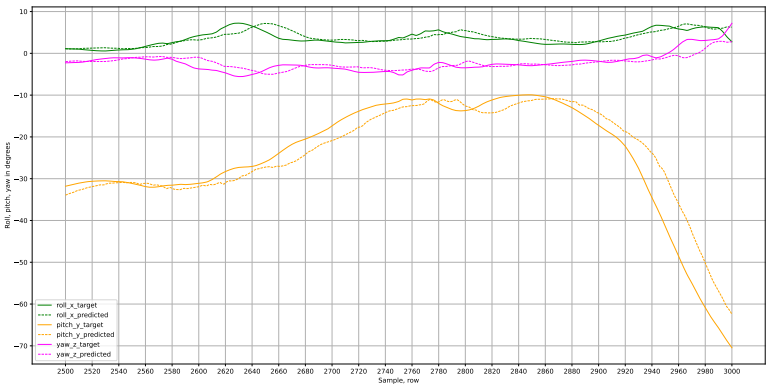


Figure 21: Outputs of LSTM1 model on dataset with flipped negative quaternions for qx, qy, qz and qw axes.

the next experiment is to evaluate whether the flipping of negative quaternions can improve rotational prediction as it was expected on the preprocessing step.

Figures 20 and 21 shows the predictions of LSTM1 model on dataset with flipped negative quaternions. The mean square error for the all three positional axes $MAE_{pos} = 0.011$m and root mean square error $RMSE_{pos} = 0.014$m meaning the average distance between predicted position and the real position reduced from 7 cm to 1.1 cm if LSTM1 on dataset with flipped negative quaternions is used instead of Baseline. For all four quaternion components calculated metrics are: $MAE_{rot} = 13.49°$ and $RMSE_{rot} = 18.47°$. Thus LSTM1 predicts the rotation on interpolated dataset slightly better than the Baseline model and there is also an improvement comparing to prediction with interpolated dataset. Next experiments aim to evaluate whether the normalization of positional data and dataset division can improve evaluation metrics.

For the sake of place saving the plots of the rest datasets will not be added in the thesis. The division of dataset on pure positional dataset and pure rotational does not improve the prediction error. For positional dataset MAE_{pos} increased to 0.078m and root mean square error jumped to to 0.091m what means that MAE_{pos} error increased by 116.41% and $RMSE_{pos}$ by $133,82\%$ compared to a Baseline. Surprisingly, opposed to the last experiment, in rotational dataset MAE_{rot} decreased to $11.81°$m and root mean square error reduced to $16.64°$. It seems that for a prediction of the rotation the positional data can be eliminated from a dataset. The most interesting experiment done with a normalized dataset, Only position (x, y, z) was scaled in range [0..1]. The MAE_{pos} is slightly lower than Baseline's and is equal to 0.056. This value must be considered as bad prediction because the better results are obtained with two previous datasets. Additionally, $RMSE_{pos}$ increased

significantly to 0.197 what is by 289% worse than a Baseline prediction. However, it is surprising that rotational MAE_{rot} decreased to $9.87°$m and $RMSE_{rot}$ reduced to $12.72°$.

For the LSTM1 model the best prediction is done for a future position using interpolated dataset with flipped negative quaternions and for rotation using a dataset with normalized position. The analysis of this phenomena is done in section 5.1.

Batch size

A significant impact on the performance e.g. the prediction accuracy has a batch size used in LSTM or GRU models. The batch-size helps to learn the common patterns as important features by providing a fixed number of samples at one time. So that the model thus can distinguish the common features by looking at all the introduced samples of the batch. In most cases, an optimal batch size is set to 64. When this batch size was initially used with LSTM model, it gave significant high MSE, RMSE, train and validation errors. Based on the performance observation during experiments with LSTM parameters, batch size fine-tuning was done. The experiments done by *Aykut et al* in their works [3] and [4] proved that appropriate batch size can be found in range 2^9 - 2^{11} (512 - 2048). Notice that a power of 2 is used as a batch size. The overall idea is to fit a batch of samples entirely in the the CPU/GPU. Since, all the CPU/GPU comes with a storage capacity in power of two, it is advised to keep a batch size a power of two. Using a number different from a power of 2 could lead to poor performance. Experimentally is proved in this thesis that similar to works [3, 4] batch size of 2^8 - 2^{10} (256, 512 and sometimes 1024) produces the best prediction result on 6-DoF dataset if the other hyperparameters are set correctly. The smaller batch sizes (32, 64 and 128) resulted in the high values of evaluation metrics and it was obvious to notice during experiments the improving the metrics with increasing the batch size.

Learning rate

Learning rate is a parameter of extended version of stochastic gradient Adam optimizer. The learning rate determines how much an updating step influences the current value of the weights.

If learning rate is large then a correspondingly large modification of the weights w_i happens on each epoch. In general too large learning rate overshoots the local minimum in a cost function.

The small learning rate does not allow model to neither successfully learn patterns in the data nor generalize them on the validation data. The prediction on the test data done by a model trained with a small value of learning rate results in significant high error. Fig. 22 shows the training and validation loss of the 150 epochs of training with learning rate of 1^{-6} that finished with $MAE = 3.70$. Model can neither learn on training data not predict the new data on validation dataset.

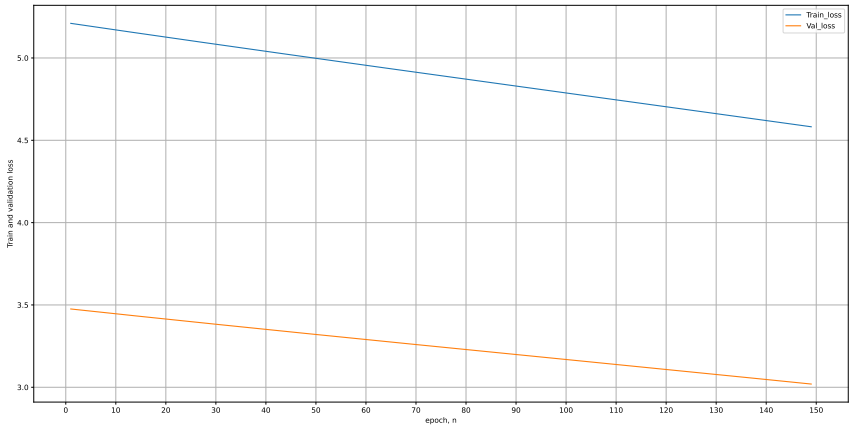


Figure 22: Plot of training and validation loss with small learning rate of Adam optimizer.

In works [3, 4] by *Aykut et al* the adaptively reducing learning rate is used. So that in master thesis a learning rate decay scheduler was used to decrease the initial learning rate of 0.001 every 50 epochs by 50%. The initial value for learning rate, the amount of epochs to keep the value the same and the multiplier were found out during experiments with grid parameters search.

Weight decay

Adam optimizer has an additional term in the weight update rule that causes the weights to exponentially decay to zero, if no other update is scheduled. With weight decay after each update, the weights are multiplied by a factor less than 1. This prevents the weights from growing too large, and can be seen as gradient descent on a quadratic regularization term. Thus weight decay ss a regularisation technique used to avoid over-fitting. Indeed experiments showed, that relatively large weight decay equal to $1^{-4}..1^{-8}$ make it possible for a model to overtrain so that training loss permanently decreases but validation loss fluctuates on the hight level. Model can not generalize the learned patter and predict successfully future values on never seen before data.

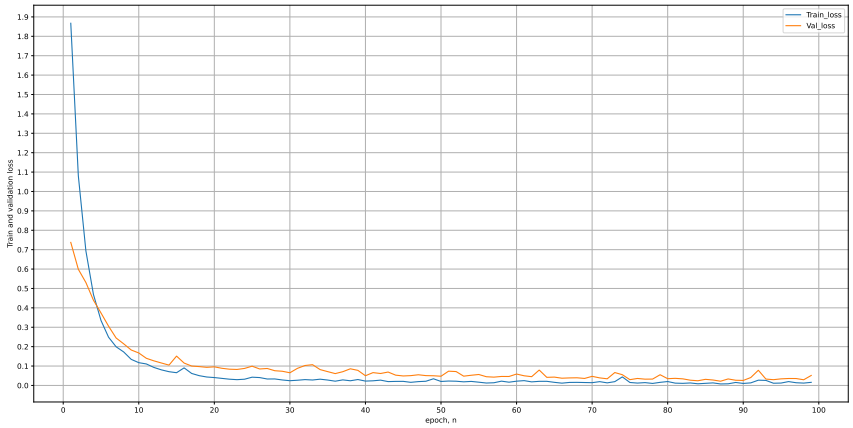


Figure 23: Plot of training and validation loss with large weigh decay of Adam optimizer.

Fig. 23 illustrates the both training and validation loss with large weigh decay set to 1^{-6} of Adam optimizer. For the illustrative goal to avoid an elimination the feeling of distance difference due to scale and to emphasize it, plot shows only 100 first epochs. It is to seen than validation loss remain on high lever and starts to increase.

Best values for weigh decay parameter of Adam optimizer set to 1^{-12} for LSTM and GRU model after parameter grid search on GPU cluster.

4.4.2 Prediction with LSTM

This section presents the results of best prediction using LSTM models and also shows the predictions with LSTM variants that considered to fail either to predict better than Baseline or to be used as improvement comparing to Baseline predictions. The role of different parameters such as batch size, learning rate and weight decay is described in section 4.4.1. The training and evaluation is this and next sections are done on interpolated dataset with flipped negative quaternions.

During preprocessing step Euler angles (yaw, pitch, roll) were calculated from quaternions and these parameters are used for visualization purposes. The quaternions of model's predictions are also converted to Euler angles so that MAE and $RMSE$ units in degrees are similar to plotted information.

The best prediction result with LSTM1 are already presented in figures 20 and 21 in section above. Thereby LSTM1 model has best performance on interpolated dataset with flipped negative quaternions and all evaluation metrics were improved.

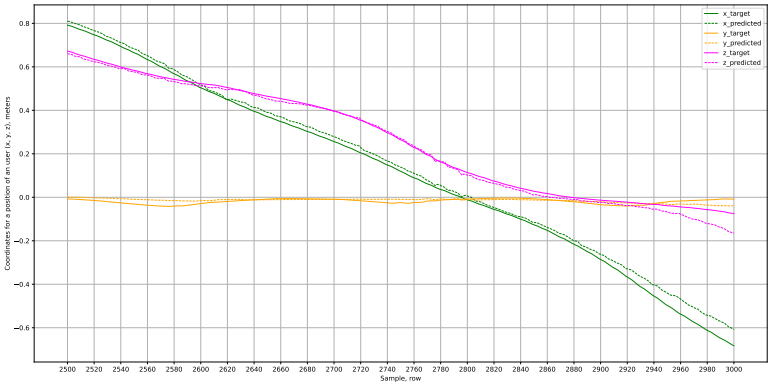


Figure 24: Outputs of LSTM2 model with ReLU activation function for x, y and z axes.

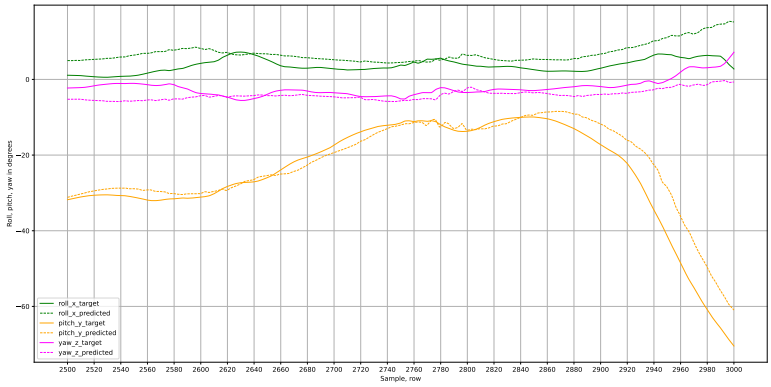


Figure 25: Outputs of LSTM2 model with ReLU activation function for qx, qy, qz and qw axes.

$MAE_{pos} = 0.011$m, $RMSE_{pos} = 0.014$m, $MAE_{rot} = 13.49°$ and $RMSE_{rot} = 18.47°$. Comparing to Baseline this means 80% improvement of prediction for position and almost 10% improvement for rotation. The average distance between predicted position and the real position reduced from 7 cm to 1.1 cm.

Fig. 24, 25 displays the same range of prediction outputs for LSTM2 model with $ReLU$ activation function. It was mentioned in section 3.2.2 that this architecture experimentally leads to higher evaluation errors comparing to LSTM1 model. Indeed, $MAE_{pos} = 0.055$m, $RMSE_{pos} = 0.185$m, $MAE_{rot} = 22.86°$ and $RMSE_{rot} = 30.88°$. In Fig. 24 and 25 it is clearly to see that distance between predicted values and real values is larger compared to LSTM1 and is similar and somewhere worse than with Baseline model. Thus graphs for $roll$ and $pitch$ have bigger gaps between real and predicted values. The predictions for x and z axes do not exactly follow

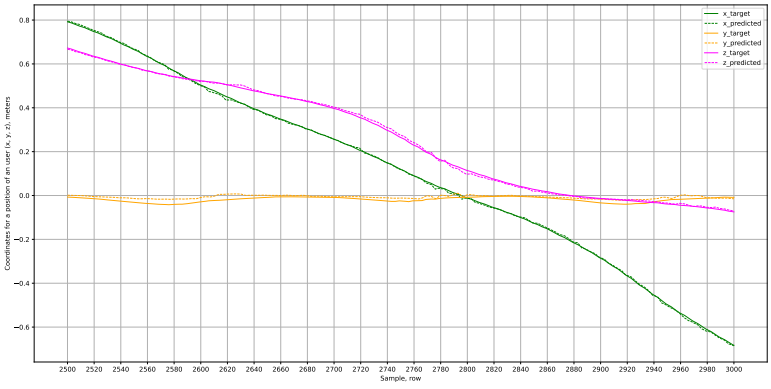


Figure 26: Outputs of LSTM3 model with Mish activation function for x, y and z axes.

the trend of the graph and real and predicted values have obvious distance between plotted graphs.

The same outputs for LSTM3 model with $ReLU$ activation function are illustrated in Fig. 26, 27. This architecture experimentally can improve the evaluation metrics comparing both to LSTM1 and Baseline models. The best metrics are as follows: $MAE_{pos} = 0.012$m, $RMSE_{pos} = 0.014$m, $MAE_{rot} = 13.18°$ and $RMSE_{rot} = 17.28°$. In Fig. 24 and 25 The predictions for x and z can almost exactly follow the trend of the graph (that was is smaller MAE means graphically) and real and predicted values have almost no obvious big distance between plotted graphs. There are also slight improvements of $roll$ and $pitch$ predictions but the evaluation metrics are still not low enough to allow predictions exactly repeat the graph of real values.

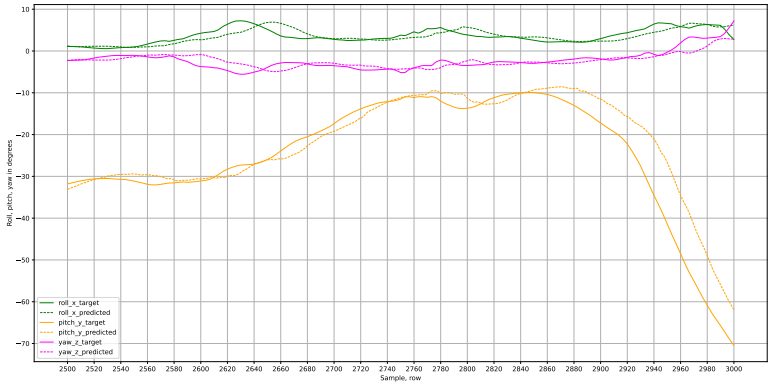


Figure 27: Outputs of LSTM3 model with Mish activation function for qx, qy, qz and qw axes.

4.4.3 Prediction with GRU

4.4.4 Prediction with Bidirectional GRU

5 Conclusion

The Python application $UserPrediction6DOF$ is a result of this work and can be used for future preprocessing of the new obtained datasets, training routine and prediction of user position and rotation in 6-DoF virtual environment.

5.1 Analysis

Same as in work of *Chang et al., 2020*, the basic GRU performs the best among all models. We surmised that it could be possibly due to the short-term correlation of human actions, so it isn't often required to consider the longterm complexity. In other words, Bi-LSTM network and TCN which architectures are more complex could be possibly overpredicted instead so that their prediction results aren't as good as LSTM network in this case.

The paper of *Chung et al., 2014* also provides an interesting comparison and evaluation of the performance of recurrent units LSTM and GRU on sequence modeling. Authors mentioned the ability of LSTM to keep the existing memory via the introduced gates and thus to detect an important feature from an input sequence at early stage, to easily carry this information (the existence of the feature) over a long distance, hence, capturing potential long-distance dependencies [8]. The GRU takes linear sum between the existing state and the newly computed state similar to the LSTM but does not have any mechanism to control the degree to which its state is exposed, but exposes the whole state each time [8]. *Chung et al., 2014* emphasize the fact that any important feature, decided by either the forget gate of the LSTM unit or the update gate of the GRU, will not be overwritten but be maintained as it is [8]. LSTM unit controls the amount of the new memory content and does not have any separate control of the amount of information flowing from the previous time step. The GRU differs and controls the information flow from the previous activation when computing the new and does not independently control the amount of the candidate activation being added via update gate [8].

It seems that for prediction of rotational data the positional data can be eliminated from a dataset. The key to this phenomena can lay in the user behavior during

5.2 Limitations

5.3 Suggestions for future work

Bibliography

[1] A. Deniz Aladagli, Erhan Ekmekcioglu, Dmitri Jarnikov, and Ahmet Kondoz. *Predicting head trajectories in 360° virtual reality videos*. https://ieeexplore.ieee.org/document/8251913. date access on 29.03.22. 2017. DOI: 10.1109/IC3D.2017.8251913.

[2] R.S. Allison, L.R. Harris, M. Jenkin, U. Jasiobedzka, and J.E. Zacher. *Tolerance of temporal delay in virtual environments*. https://www.researchgate.net/publication/2945506. date access on 17.03.22. 2001. DOI: 10.1109/VR.2001.913793.

[3] Tamay Aykut, Christoph Burgmair, Mojtaba Leox Karimi, and Eckehard Steinbach. *Delay Compensation for a Telepresence System With 3D 360 Degree Vision Based on Deep Head Motion Prediction and Dynamic FoV Adaptation*. https://arxiv.org/abs/2007.14084. date access on 23.02.22. 2018. DOI: 10.1109/WACV.2018.00222.

[4] Tamay Aykut, Eckehard Steinbach, and Jingyi Xu. *Realtime 3D 360-Degree Telepresence With Deep-Learning-Based Head-Motion Prediction*. https://www.researchgate.net/publication/330861228. date access on 25.03.22. 2019. DOI: 110.1109/JETCAS.2019.2897220.

[5] Y. Bengio, P. Simard, and P. Frasconi. *Learning long-term dependencies with gradient descent is difficult*. http://www.cs.unc.edu/techreports/93-010/93-010.pdf. date access on 31.03.22. 1994. DOI: 10.1109/72.279181.

[6] Devesh K Bhatnagar. *Position trackers for Head Mounted Display systems: A survey*. http://www.cs.unc.edu/techreports/93-010/93-010.pdf. date access on 31.03.22. 1993.

[7] Yun-Kai Chang, Mai-Keh Chen, Yun-Lun Li, Hao-Ting Li, and Chen-Kuo Chiang. *6DoF Tracking in Virtual Reality by Deep RNN Model*. https://ieeexplore.ieee.org/document/9394069. date access on 02.04.22. 2020. DOI: 10.1109/IS3C50286.2020.00057.

[8] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. https://arxiv.org/abs/1412.3555. date access on 30.03.22. 2014. DOI: 10.48550/arXiv.1412.3555.

[9] Xavier Corbillon, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. *Viewport-adaptive navigable 360-degree video delivery*. https://ieeexplore.ieee.org/document/7996611. date access on 05.04.22. 2017. DOI: 10.1109/ICC.2017.7996611.

[10] Alessandro Crivellari and Euro Beinat. *LSTM-Based Deep Learning Model for Predicting Individual Mobility Traces of Short-Term Foreign Tourists*. https://www.researchgate.net/publication/338377314. date access on 08.04.22. 2020. DOI: 10.3390/su12010349.

[11] Fanyi Duanmu, Eymen Kurdoğlu, S. Hosseini, Yong Liu, and Yao Wang. *Prioritized Buffer Control in Two-tier 360 Video Streaming*. https://www.researchgate.net/publication/319048432. date access on 05.04.22. Aug. 2017. DOI: 10.1145/3097895.3097898.

[12] Serhan Guel, Sebastian Bosse, Dimitri Podborski, Thomas Schierl, and Cornelius Hellge. *Kalman Filter-based Head Motion Prediction for Cloud-based Mixed Reality*. https://arxiv.org/abs/2007.14084. date access on 19.02.22. 2020. DOI: 0.1145/3394171.3413699.

[13] Serhan Gül, Dimitri Podborski, Thomas Buchholz, Thomas Schierl, and Cornelius Hellge. *Low-latency Cloud-based Volumetric Video Streaming Using Head Motion Prediction*. https://arxiv.org/abs/2001.06466. date access on 19.02.22. 2020. DOI: 0.1145/3394171.3413699.

[14] Sepp Hochreiter and Jürgen Schmidhuber. *Long Short-term Memory*. https://www.researchgate.net/publication/13853244. date access on 31.03.22. Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[15] Fazle Karim, Somshubra Majumdar, Houshang Darabi, and Shun Chen. *LSTM Fully Convolutional Networks for Time Series Classification*. https://arxiv.org/abs/1709.05206. date access on 14.04.22. 2017. DOI: 10.48550/arXiv.1709.05206.

[16] Hao-Ting Li, Yung-Pin Liu, Yun-Kai Chang, and Chen-Kuo Chiang. *Action recognition and tracking via deep representation extraction and motion bases learning*. https://www.researchgate.net/publication/358012181. date access on 01.04.22. 2022. DOI: 10.1007/s11042-021-11888-8.

[17] Diganta Misra. *Mish: A Self Regularized Non-Monotonic Activation Function*. https://doi.org/10.48550/arxiv.1908.08681. date access on 24.09.22. 2019. DOI: 10.48550/ARXIV.1908.08681. URL: https://arxiv.org/abs/1908.08681.

[18] Anh Nguyen, Zhisheng Yan, and Klara Nahrstedt. *Your Attention is Unique: Detecting 360-Degree Video Saliency in Head-Mounted Display for Head Movement Prediction*. https://www.researchgate.net/publication/328370817. date access on 15.03.22. 2018. DOI: 10.1145/3240508.3240669.

[19] Feng Qian, Lusheng Ji, Bo Han, and Vijay Gopalakrishnan. *Optimizing 360 video delivery over cellular networks*. https://dl.acm.org/doi/10.1145/2980055.2980056. date access on 12.03.22. 2016.

[20] Silvia Rossi, Irene Viola, Laura Toni, and Pablo Cesar. *A New Challenge: Behavioural Analysis Of 6-DOF User When Consuming Immersive Media*. https://ieeexplore.ieee.org/document/9506525. date access on 03.04.22. 2021. DOI: 10.1109/ICIP42928.2021.9506525.

[21] Silvia Rossi, Irene Viola, Laura Toni, and Pablo Cesar. *From 3-DoF to 6-DoF: New Metrics to Analyse Users Behaviour in Immersive Applications*. https://www.researchgate.net/publication/357172010. 1-7. 2021. date access on 13.04.22. 2021.

[22] Afshin Taghavi, Anahita Mahzari, Joseph Beshay, and Ravi Prakash. *Adaptive 360-Degree Video Streaming using Scalable Video Coding*. https://www.researchgate.net/publication/320542716. date access on 05.04.22. Oct. 2017. DOI: 10.1145/3123266.3123414.

[23] Howie Choset; Kevin M. Lynch; Seth Hutchinson; George A. Kantor; Wolfram Burgard; Lydia E. Kavraki; Sebastian Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. The MIT Press, 2005, p. 608. ISBN: 978-0262-03327-5.

[24] Zhiguang Wang, Weizhong Yan, and Tim Oates. *Time Series Classification from Scratch with DeepNeural Networks: A Strong Baseline*. https://www.researchgate.net/publication/318332658. date access on 25.03.22. 2017. DOI: 10.1109/IJCNN.2017.7966039.

[25] Lan Xie, Zhimin Xu, Yixuan Ban, Xinggong Zhang, and Zongming Guo. *360ProbDASH: Improving QoE of 360 Video Streaming Using Tile-based HTTP Adaptive Streaming*. https://www.researchgate.net/publication/320542716. date access on 05.04.22. Oct. 2017. DOI: 10.1145/3123266.3123291.

[26] Emin Zerman, Radhika Kulkarni, and Aljosa Smolic. *User Behaviour Analysis of Volumetric Video in Augmented Reality*. https://ieeexplore.ieee.org/document/9465456. date access on 13.04.22. 2021. DOI: 10.1109/QoMEX51781.2021.9465456.