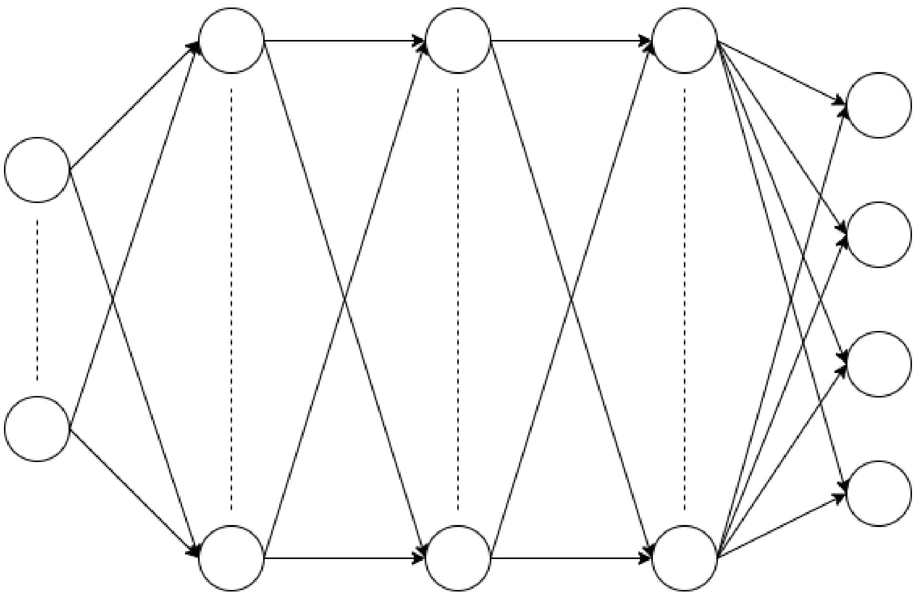
Field of View Prediction based on Neural Networks for 360° Video Streaming

A thesis presented for the degree of
B.Sc. of Computer Science

Christoph Nötzli



Supervisor
Professor Dr. Torsten Braun

Communication and Distributed Systems (CDS)
Institute of Computer Science (INF)
University of Bern
Switzerland
May 2020

Declaration of Authorship

I, Christoph Nötzli, declare that this thesis titled, "Field of View Prediction based on Neural Networks for 360° Video Streaming" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UNIVERSITY OF BERN

Abstract

Faculty of Science

Institute of Computer Science (INF)

B.Sc. of Computer Science

Bachelor thesis: FOV prediction based on neural networks for 360° video streaming

Christoph Nötzli

With 360° videos it is possible for the user to select the field of view (FOV) by changing the orientation of the smart phone or virtual reality glasses. Sending and receiving full 360° videos requires a large bandwidth. One approach to reducing the bandwidth is to only transmit the part of the video at full quality that is in the user's FOV. This, however, requires accurately predicting the FOV in a short amount of time. A new way to do so is the focus of this bachelor thesis. We implemented a Madgwick filter that estimates the orientation of a mobile phone and two neural networks with different approaches that predict the upcoming FOV with the help of the orientation and sensor data of the mobile device. Based on two videos we gathered data for training and evaluating the networks. The results show that the targeted values regarding calculation time were met and the error rate at 2s prediction time is under 10%. For short-term predictions (under 0.5s) the false negative error is less than 1%. With this approach it is possible to save more than 50% of the bandwidth with the respective error rate. We provide a new way of predicting the FOV that could be an alternative to state-of-the-art regression algorithms and may even allow live streaming.

Contents

List of Figures

List of Tables

1. Introduction

With 360° videos it is possible to view a video in all directions from the position of the camera. 360° videos offer the possibility to watch videos interactively. The videos are watched either with virtual reality (VR) devices or smartphones. With the orientation of the device the user determines which part of the video is shown on the screen. 360° videos are produced by special camera setups. Several cameras film in an arrangement that covers the full 360° range and then the images are stitched together. 360° videos are increasingly popular in art, documentaries and social media. In 2017, Facebook announced that more than 1 million 360° videos have been uploaded to the social media platform [11]. Vimeo already offers the possibility to upload and stream 360° videos in a quality of up to 8k resolution [12].

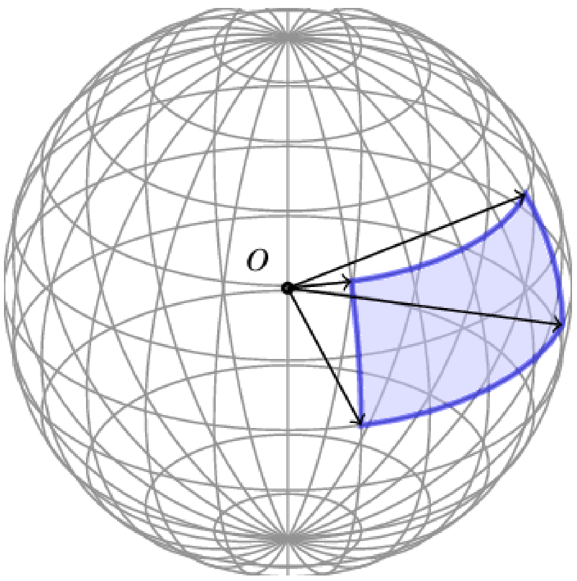


Figure 1.1.: Field of view [1]

1.1. Motivation

There are several challenges with 360° videos that need to be considered. One of them is the efficient delivery of the image material through a network. If the whole 360° image has to be delivered, it requires high bandwidth for the end user as well as the content provider. For example, for HD quality (quality displayed on the screen) 400Mbps and for 4k even a bandwidth of 1Gbps is necessary [13]. In addition, there are requirements for the latency. A movement of the smartphone must result in a movement of the image with the smallest delay possible. If the latency between movement and image shifting becomes too large, the delay will be noticeable for the user. For the latency an ideal value would be 7ms. Currently more realistic values for local systems are close to or slightly below 20ms [13].

Different approaches for a more efficient delivery of the videos exist. One approach is to send the parts in the user's field of view (FOV) with as little quality loss as possible. The parts outside of the field of view are not sent at all or are delivered in lower quality. The videos are divided into tiles (subsets of all the pixels) and either only individual tiles or tiles of different quality are delivered to the user.

To save network bandwidth with such an approach, it is necessary to send information about the orientation to the server that provides the image material. Time passes between sending the orientation and receiving the next image material. In 5G networks this latency is between 20 and 30ms [14]. When using edge servers these latencies go down to 10-20ms [15]. The latencies are expected to be even lower in the future.

In this work we assume latencies of 10ms or higher. To compensate for latencies, we have to predict the user's future FOV. For this reason we aim to compute the predictions in under 10ms. The predictions must be highly accurate, otherwise the quality of experience (QoE) for the user declines. Regression algorithms achieve at a latency of 0.5s an accuracy of up to 97% [16]. Longer-term predictions are rather difficult, because at 2s state-of-the-art approaches only achieve an accuracy of 72% (predictions in this case are accurate if they differ less than 10° from the actual angle) [16]. Previous work on such approaches measured 60-80% bandwidth savings with the respective error rates [16, 17].

The goal of this bachelor thesis is to investigate whether machine learning can be leveraged to improve predictions about the user's FOV. The used neural networks are kept small enough so that they meet the latency requirements.

1.2. Contributions

To fulfill the goals we implement a Madgwick filter that estimates the orientation of a smartphone [6]. The orientation is needed to display the correct part of the video and also serves as an input for the neural networks.

Further we implement and evaluate two different approaches of neural networks. One neural network approach estimates the orientation of the mobile phone. The second neural network approach directly predicts the tiles that will be used in the future.

To validate the neural networks we had to collect data. We collected data of five users in two small studies where the users watched a drone flight video and a short movie [18, 19]. In the first study, data was collected in average every 100 ms to test the validity of the approach. In the second study, data was then collected in an average interval of 10 ms to fulfill our latency requirements.

1.3. Structure

This thesis has the following structure. First, we introduce how 360° videos are displayed on phones, introduce quaternions and show how neural networks are constructed (Chapter 2). In Chapter 3, we introduce the system components of this work. These include the Madgwick filter, which estimates the orientation of a mobile phone, and the neural networks, which predict the FOV of a user. In a third step, the implementation of the Android application and the neural networks are described in more detail (Chapter 4). In Chapter 5, we introduce the evaluation methods and evaluate the two different approaches of neural networks. Furthermore, we discuss the results and give some advice for improvements. In Chapter 6 we conclude and present ideas for further research projects.

2. Theoretical Background

In this chapter, we explain the basics of the algorithms used in our work. We discuss how 360° videos are displayed, how the orientation of an object can be represented, and how neural networks work.

2.1. Displaying 360° video

An important task of computer graphics is to render three-dimensional scenes and to display them on screens. Objects can be placed in the three-dimensional scene and a virtual camera defines from which direction the scene is viewed (Figure 2.1) [2].

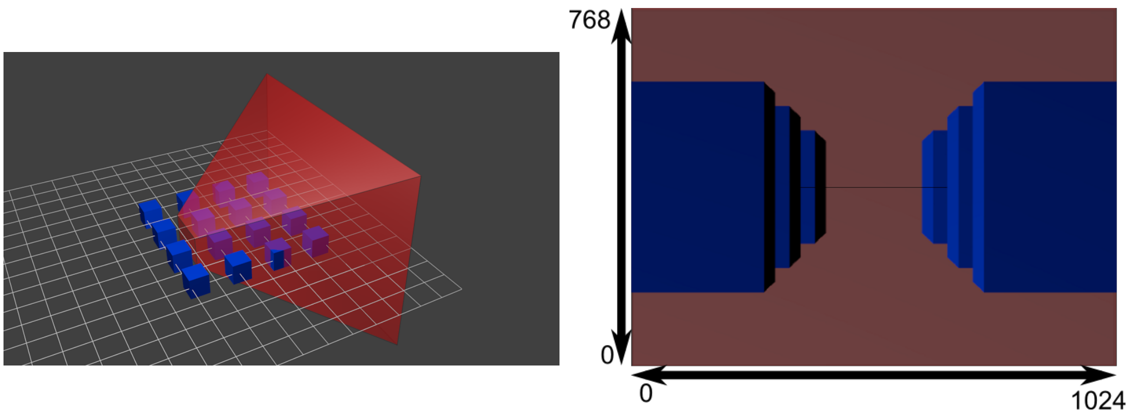


Figure 2.1.: Left: Three-dimensional scene with objects (blue) and the frustum of the camera (red) [2], right: Three-dimensional scene rendered on screen [2]

The objects are defined by vertices. These vertices are defined relative to the center of the object. A first matrix moves all vertices from the center of the objects to their respective places in the three-dimensional scene i.e. the model coordinates are converted to world coordinates (Figure 2.2) [2].

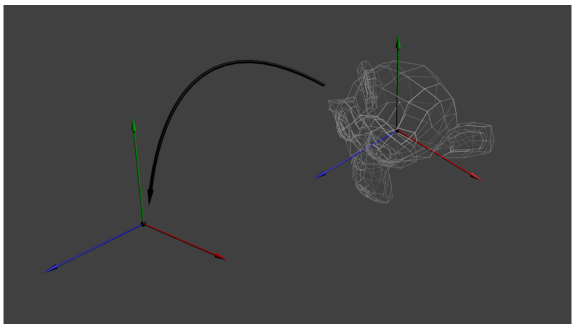


Figure 2.2.: Model to world transformation [2]

In a second step a virtual camera is placed in the three-dimensional scene. The camera determines from where the scene is viewed. A second matrix is used to display the objects from the camera's perspective (Figure 2.3) [2]. This matrix is called the view matrix.

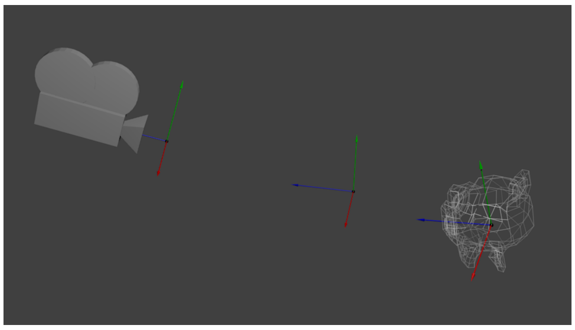


Figure 2.3.: World to camera transformation [2]

With the help of a third matrix, it is determined how the objects are displayed on the screen of the viewing device (Figure 2.4) [2]. This is done with the help of the aspect ratio of the screen and the distance of the objects to the camera. The so-called projection matrix projects the objects onto the screen of the mobile phone:

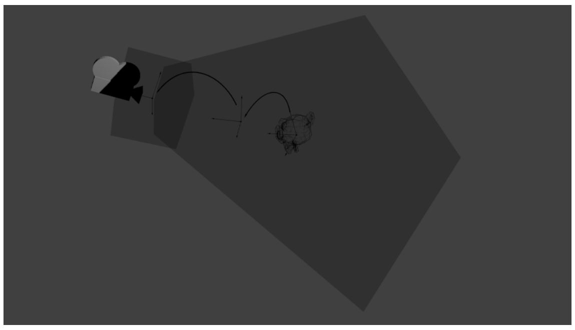


Figure 2.4.: Camera to homogeneous transformation [2]

When displaying 360° videos, the video is projected into a sphere. It is necessary to render the video, which is streamed in two dimensions, into the three-dimensional sphere. The image is not projected on the outside but on the inside of the sphere. The most commonly used projection is the equirectangular projection [20]. The two-dimensional image is distorted in a way that when it is projected into a sphere the image appears no longer distorted. The two-dimensional image is, therefore, more distorted at the top and bottom than in the middle of the frame. Figure 2.5 illustrates the projection.

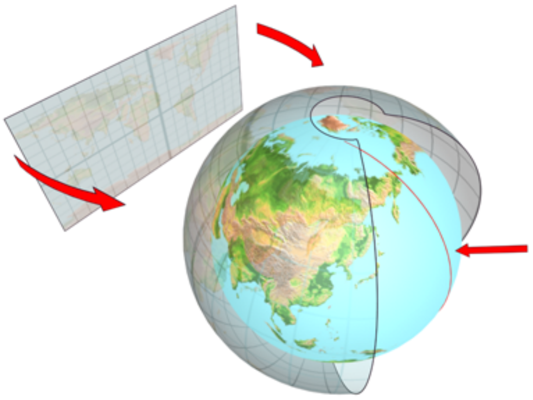


Figure 2.5.: Projection of a two dimensional image onto a sphere [3]

Since this sphere never moves and is always at the same place of the three-dimensional scene, the model matrix is constant. The center of the sphere is always at the point $[0, 0, 0]$ of the world. In a second step the camera is placed in the center of the sphere. By changing the orientation of the mobile phone this view matrix is adjusted. This changes the orientation of the virtual camera and thus the image section. The view matrix can be described by four vectors: The up-, forward-, right-, and position-vector. The up-, forward- and right-vector determine the orientation of the virtual camera and the position-vector the position in the world of the virtual camera in our three-dimensional scene:

$$V = \begin{bmatrix} right_x & up_x & forward_x & position_x \\ right_y & up_y & forward_y & position_y \\ right_z & up_z & forward_z & position_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.0.1}$$

In our case the camera is always placed in the middle of the sphere and does not change the position. The position-vector is therefore always $[0, 0, 0]$ and the view matrix consists only of the other three vectors:

$$V = \begin{bmatrix} right_x & up_x & forward_x & 0 \\ right_y & up_y & forward_y & 0 \\ right_z & up_z & forward_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1.0.2}$$

Section 3.1.4 describes how we determine the three remaining vectors with the help of the orientation of the mobile phone.

Since neither the aspect ratio of the screen of the mobile phone nor the distance to the sphere changes, the projection matrix remains constant as well.

2.2. Theory for Orientation Estimation

Orientations of objects can be represented differently. This chapter explains Euler angles, why they are not suited for the application, and what quaternions are.

2.2.1. Euler Angles

Euler described in the 18th century rotations in the following theorem [21]:

In whatever way a sphere is turned about its centre, it is always possible to assign a diameter, whose direction in the translated state agrees with that of the initial state.

In other words, if two coordinate systems are orthonormal there always exists a rotation that turns one coordinate system into the direction of the second one [4]. The angles between the two coordinate systems can be described with three parameters called Euler angles.

Sequences Depending on the sequence used to describe the rotation, Euler angles can have different values. This is why it is important to define the sequence used.

Sequences that can be used are the aerospace (z-y-x) and the orbit (z-x-z) sequences [4]. Some sequences use the same axis twice and other sequences use each axis once.

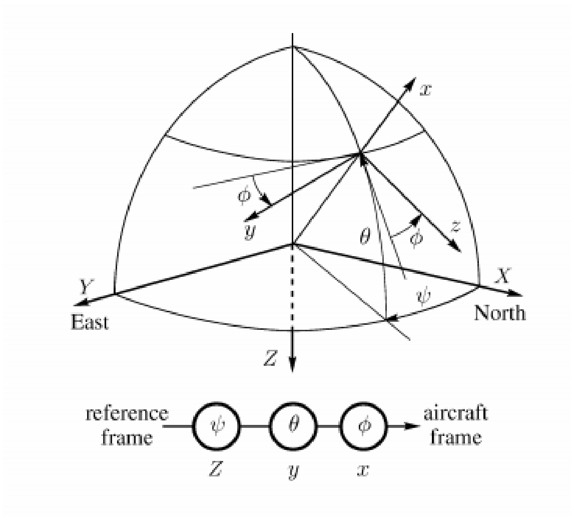


Figure 2.6.: Aerospace sequence [4]

For example for the aerospace sequence the axes would be defined: x-axis ϕ (roll), y-axis θ (pitch) and z-axis ψ (yaw), as illustrated in Figure 2.6.

Gimbal lock Euler angles have a drawback compared to the representation with quaternions (Section 2.2.2). Euler angles can suffer from gimbal lock. With the first rotation around the y-axis (aerospace sequence) the orientation of the x- and z-axis is changed [22]. If they end up parallel they rotate around the same axis and one degree of freedom is lost [22]. This could be prevented by changing the rotation sequence. But in every sequence we risk to end up in a gimbal lock.

2.2.2. Quaternions

Complex numbers usually only have two dimensions, where a is the real and b the imaginary part:

$$z = a + bi \tag{2.2.2.1}$$

A *Quaternion* extends complex numbers to four dimensions, one real part and imaginary parts $bi + cj + dk$ [4]:

$$q = a + bi + cj + dk \tag{2.2.2.2}$$

Quaternions can also have different sequences. This thesis uses the Hamilton representation: [4]:

$$i^2 = j^2 = k^2 = ijk = -1 \tag{2.2.2.3}$$

$$ij = -ji = k \tag{2.2.2.4}$$

$$jk = -kj = i \tag{2.2.2.5}$$

$$ki = -ik = j \tag{2.2.2.6}$$

If q Equation (2.2.2.2) is our quaternion, the complex conjugate of a quaternion is [4]:

$$q^* = a - bi - cj - dk \tag{2.2.2.7}$$

The product of two quaternions is defined in the following equation [4]:

$$\begin{aligned} q \times p = (q_1 + q_2 i + q_3 j + q_4 k)(p_1 + p_2 i + p_3 j + p_4 k) = \\ q_1p_1 - q_2p_2 - q_3p_3 - q_4p_4 \\ +(q_1p_2 + q_2p_1 + q_3p_4 - q_4p_3)i \\ +(q_1p_3 - q_2p_4 + q_3p_1 + q_4p_2)j \\ +(q_1p_4 + q_2p_3 - q_3p_2 + q_4p_1)k \end{aligned} \tag{2.2.2.8}$$

Quaternions can be used to represent the orientation in a three-dimensional space of an object relative to an other object [6]. Quaternions can be used to rotate a vector [4] around a certain axis. The vector that is rotated first needs to be extended with a 0 at the first position. The vector then can be rotated with the standard quaternion multiplication v [4]:

$$w = q \times v \times q^*, \qquad \begin{bmatrix} 0 \\ w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \times \begin{bmatrix} 0 \\ v_x \\ v_y \\ v_z \end{bmatrix} \times \begin{bmatrix} q_1 \\ -q_2 \\ -q_3 \\ -q_4 \end{bmatrix} \tag{2.2.2.9}$$

$$\begin{bmatrix} w_x \\ w_y \\ w_z \end{bmatrix} = \begin{bmatrix} 2v_x(\frac{1}{2} - q_3^2 - q_4^2) + 2v_y(q_1q_4 + q_2q_3) + 2v_z(q_2q_4 - q_1q_3) \\ 2v_x(q_2q_3 - q_1q_4) + 2v_y(\frac{1}{2} - q_2^2 + q_4^2) + 2v_z(q_1q_2 + q_3q_4) \\ 2v_x(q_1q_3 + q_2q_4) + 2v_y(q_3q_4 - q_1q_2) + 2v_z(\frac{1}{2} - q_2^2 - q_3^2) \end{bmatrix} \tag{2.2.2.10}$$

2.3. Neural Networks

In this section we show the components of neural networks and how neural networks are constructed.

2.3.1. Neurons

Neurons are the building blocks of a neural network. This section (2.3.1) describes the architecture of a neuron as shown in Figure 2.7.

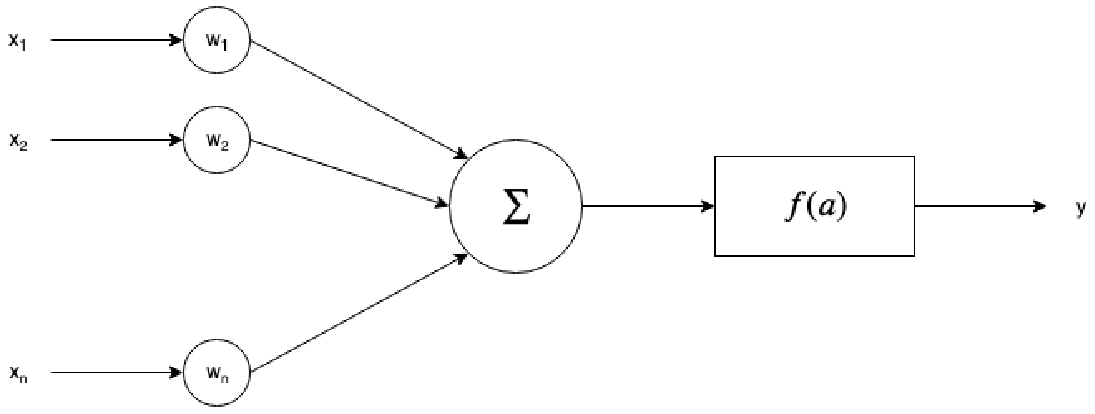


Figure 2.7.: Image of an artificial neuron

This is a simple version of a neuron and other variations exist. A neuron has one or many inputs $x_1, \dots, x_n$ [23]. The inputs are multiplied with the weights $w_1, \dots, w_n$ [23]. After multiplying the inputs with the weights the inputs are added together. This value is then manipulated with an activation function $f(a)$ [23]. Neurons can have different activation functions. Figure 2.8-2.10 show examples of common activation functions.

Linear [23]

$$f(a) = a \tag{2.3.1.1}$$

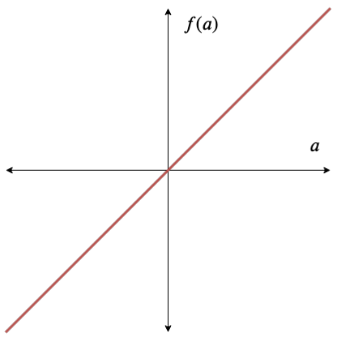


Figure 2.8.: Linear activation function

Sigmoid [23]

$$f(a) = tanh(a) \tag{2.3.1.2}$$

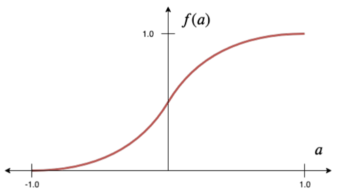


Figure 2.9.: Sigmoid activation function

ReLU (Rectified Linear Unit) [24]

$$f(a) = \begin{cases} a & a \geq 0 \\ 0 & a < 0 \end{cases} \tag{2.3.1.3}$$

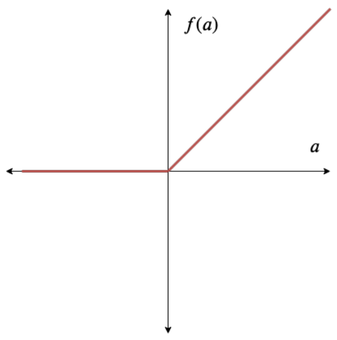


Figure 2.10.: Rectified linear unit activation function

Output After these steps we get a new value, the output y [23]. This value can either be used as an input for another neuron or as an output of the neural network.

2.3.2. Neural Networks

A neural network is a collection of connected neurons. The neurons are arranged in so-called layers. A distinction is made between three different layers as shown in Figure 2.11.

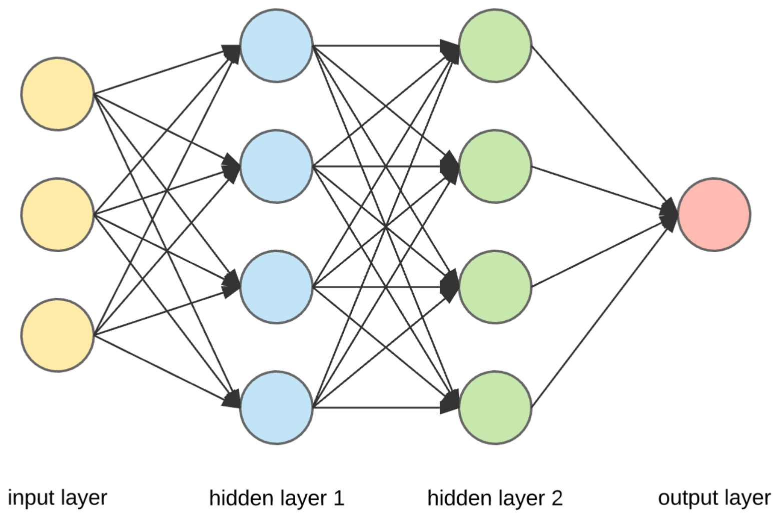


Figure 2.11.: Simple neural network with labeled layers [5]

Input layer The raw data (e.g. sensor data) is transferred to the input layer [5].

Hidden layer These are the layers where the calculations take place [5]. It is possible to have multiple hidden layers in a network.

Output layer The output layer gives a result for a given input [5].

Only fully-connected layers are used in this project. This means that each neuron is connected to each neuron in the next layer.

2.3.3. Training and Evaluation

Two independent data sets are used for training and evaluation of the neural networks. For the training of an artificial neural network a loss function is determined first. A neural network tries to minimize the loss during training by adjusting the weights between the individual neurons. In this bachelor thesis two different loss functions were used. The neural network approach that predicts the quaternion tries to solve a regression problem. In this case we use the mean squared error as a loss function. The second neural network approach that predicts the tiles solves a classification problem and uses the binary cross entropy loss function:

Mean squared error We use the average of the squared error to get the loss value [25].

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_{expected} - y_{predicted})^2 \tag{2.3.3.1}$$

Binary cross entropy The binary cross entropy loss function assumes that the individual outputs can only be 1 or 0 [26]. The binary cross entropy loss function computes the loss between the predicted and actual value [26].

With the second data set the neural networks can be evaluated. The data sets are used to check how well the neural networks perform on data that they have not been directly trained on.

3. System Components

This chapter explains the individual components of the work. In a first part it explains how an Android application estimates the orientation of a mobile device. Then it describes the two different variants of neural networks that we investigated.

3.1. Android Application

The Android application has the task to estimate the orientation of a smart phone. This task serves two purposes: On one hand the estimation is used to display the correct part of the video on the screen and on the other hand the orientation is used as input data to train the neural networks. The Figure 3.1 describes how the orientation estimation is structured.

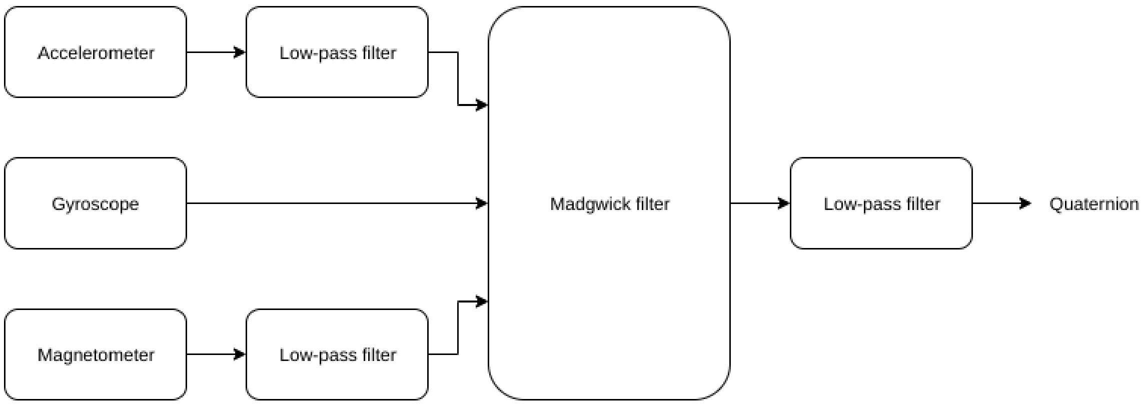


Figure 3.1.: Overview of the orientation estimation algorithm

First, the application applies a low-pass filter to the accelerometer and magnetometer data (Chapter 3.1.2). The reason for the filtering is that both sensors are susceptible to noise. Then the data from the accelerometer, gyroscope and magnetometer are passed to the Madgwick filter. This filter estimates the orientation and returns it in the form of a quaternion (Chapter 3.1.3) [6]. Since the output data is only an estimate, we apply another low-pass filter to the quaternions.

3.1.1. Sensors

For the two studies we use a mobile phone as a measurement unit. Modern mobile phones have several different sensors of which we use the accelerometer, magnetometer and gyroscope.

3.1.1.1. Accelerometer

The accelerometer is a sensor that measures the acceleration of the mobile phone in three axes. When measuring the acceleration it is possible to differentiate between two components, the linear acceleration and the gravitational acceleration. In this study only the gravitational part of the acceleration is needed. The gravitational part can be isolated by low-pass filtering (Chapter 3.1.2) the raw accelerometer-sensor data.

3.1.1.2. Magnetometer

The magnetometer is able to measure the magnetic field in three axes. The magnetometer is a sensor that is exposed to noise. Especially inside of buildings the magnetic field can be influenced by different materials and electric fields. To compensate for short term noise changes (Chapter 3.1.2) the data of this sensor is also low-pass filtered.

3.1.1.3. Gyroscope

This sensor measures the angular velocity on three axes. For the orientation estimation the gyroscope data is integrated to get the angular position.

3.1.2. Low-pass filters

The goal of low-pass-filters is to filter out the components with high frequencies from a signal. Signals with high frequencies can be noise or in the case of the accelerometer the linear part of the acceleration [6]. An often used low-pass-filter in digital signal processing is the recursive moving average filter [27]. In the following, we briefly describe common choices.

3.1.2.1. Simple moving average filter

If x_n is the current data point and $x_{n-1}...x_{n-(m-1)}$ are the last data points and m is the number of data points, y is the output [27]:

$$y_n = \frac{x_n + x_{n-1} + \ldots + x_{n-(m-1)}}{m} \tag{3.1.2.1}$$

$$y_n = \frac{1}{m}\sum_{i=0}^{m-1} x_{n-i} \tag{3.1.2.2}$$

3.1.2.2. Recursive moving average filter

The recursive moving average filter adds one or multiple outputs of the past $y_{n-1}, \ldots, y_{n-(p-1)}$ to the filter:

$$y_n = \frac{x_n + x_{n-1} + \ldots + x_{n-(m-1)}}{m} + \frac{y_{n-1} + \ldots + y_{n-p}}{p} \tag{3.1.2.3}$$

$$y_n = \frac{1}{m}\sum_{i=0}^{m-1} x_{n-i} + \frac{1}{p}\sum_{j=1}^{p} y_{n-j} \tag{3.1.2.4}$$

3.1.2.3. Weighted moving average filter

With weights $w_1, \ldots, w_m$ it is possible to give the values different importance.

$$y_n = \frac{w_0 x_n + w_1 x_{n-1} + \ldots + w_{m-1} x_{n-(m-1)}}{w_1 + \ldots + w_{m-1}} \tag{3.1.2.5}$$

$$y_n = \frac{1}{w_1 + \ldots + w_{m-1}}\sum_{i=0}^{m-1} w_i x_{n-i} \tag{3.1.2.6}$$

3.1.2.4. Weighted recursive moving average filter

The filter that is used in this Bachelor thesis is a combination of the weighted and the recursive filter. In this case α is our weight with a value between 0 and 1 [28].

$$y_0 = x_n \tag{3.1.2.7}$$

$$y_n = \alpha y_{n-1} + (1-\alpha) x_n \tag{3.1.2.8}$$

The low-pass filters of the accelerometer and magnetometer use an α of 0.95 and the filter after the Madgwick filter uses an α of 0.99.

3.1.3. Madgwick filter

Today, orientation estimation is usually done with Kalman filters [6]. However, Kalman filters that are used for orientation estimation need high sampling rates because of their linear regression iterations and have a high computational load because of their large state vectors that describe the rotational kinematics [6]. The Madgwick filter addresses these issues by using an analytically derived and optimised gradient descent algorithm to make the filter work at low sampling rates [6].

The Figure 3.2 shows an overview of the calculations of the Madgwick filter [6].

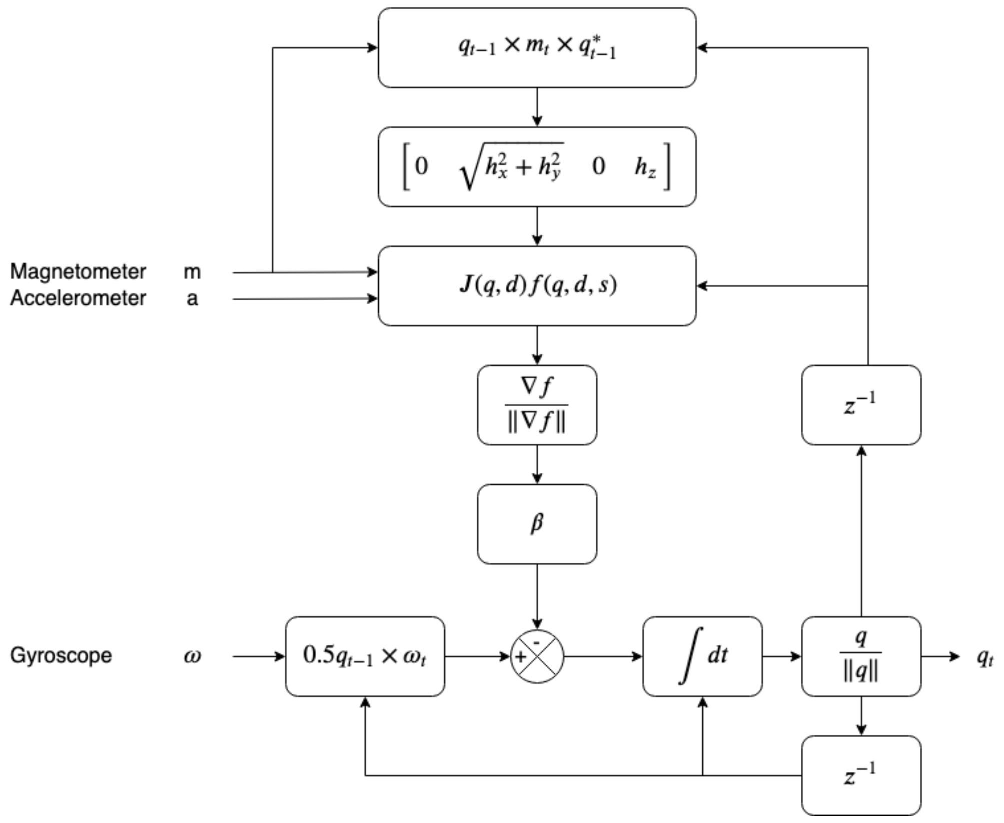

$q_{t-1} \times m_t \times q^*_{t-1}$
$\begin{bmatrix} 0 & \sqrt{h_x^2 + h_y^2} & 0 & h_z \end{bmatrix}$
Magnetometer m
Accelerometer a
$J(q,d)f(q,d,s)$
$\frac{\nabla f}{\|\nabla f\|}$
z^{-1}
β
Gyroscope ω
$0.5q_{t-1} \times \omega_t$
+
-
$\int dt$
$\frac{q}{\|q\|}$
q_t
z^{-1}

Figure 3.2.: Overview of the madgwick algorithm [6]

The Madgwick filter estimates the orientation twice, once with the gyroscope and a second time with the accelerometer and magnetometer. To get the advantages of all three sensors the estimations are fused into one quaternion.

3.1.3.1. Coordinate System

Orientations of objects are always relative to an other object. This is why it is necessary express the orientation of an object with two coordinate systems i.e. a reference system and a second coordinate system. The fixed coordinate system arises from the properties of the earth. The z-axis of the coordinate system points to the direction of gravity and the x-axis points to the magnetic north pole of the earth. The y-axis is orthogonal to these axes, it's direction is east. The Figure 3.3 shows the coordinate system of the earth.

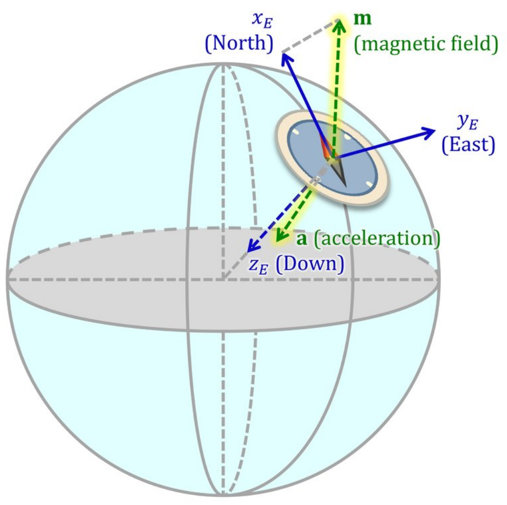


Figure 3.3.: The coordinate system of the earth [7]

The second coordinate system is of the mobile device and is explained in figure 3.5.

3.1.3.2. Orientation of the Sensors

The standard axes (phone in portrait mode) of the sensors are defined in Figure 3.4.

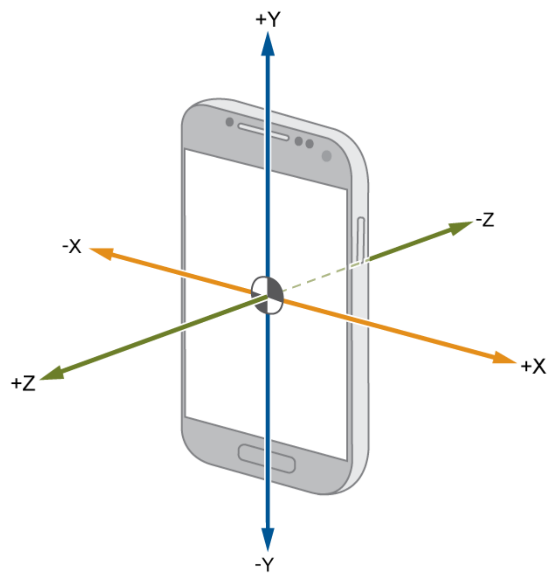


Figure 3.4.: The coordinate system in portrait mode [8]

In the application we change the definition of this coordinate system because we define the standard position in landscape mode instead of the portrait mode. To change the definition it is necessary to change the definition of the x- and y-axis. This is illustrated in the Section 3.1.3.2.

Mode	x, y and z-axis of exported data
Portrait (0°)	x, y, z
Landscape (90°)	y, -x, z
Inverse portrait (180°)	-x, -y, z
Inverse landscape (270°)	-y, x, z

Table 3.1.: We use the portrait mode in the android application

The coordinate system that is used in the application is defined in Figure 3.5.

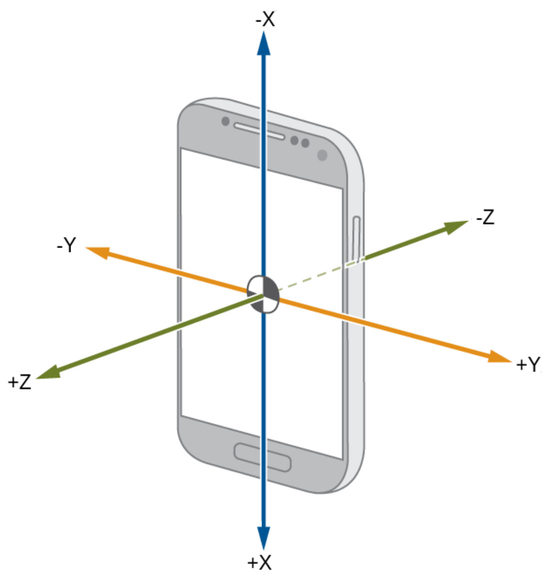


Figure 3.5.: The coordinate system in landscape mode [8]

3.1.3.3. Orientation Estimation with the Gyroscope

The gyroscope measures the angular velocity. To estimate the orientation with the gyroscope the values need to be integrated. The integration delivers the angular position of the smart phone. First the vector of the gyroscope needs to be extended with a fourth dimension in the first position to be able to use the quaternion multiplication (Equation (2.2.2.8)) [6]:

$$\omega_t = [0, \omega_x, \omega_y, \omega_z] \tag{3.1.3.1}$$

The integration is done with the help of the trapezoidal rule [29, 6]:

$$q_{\omega,t} = (0.5q_{t-1} \times \omega_t)\Delta t \tag{3.1.3.2}$$

The integrated value is then added to the quaternion of the last time step [6]:

$$q_t = q_{t-1} + q_{\omega,t} \tag{3.1.3.3}$$

The quaternion 3.1.3.3 is the estimated angular position / orientation estimation. The problem with only using the gyroscope for the orientation estimation is that the orientation has a drift. The drift arises through the measurement errors of the gyroscope [6]. The measurement error is integrated as well. The errors are added up in every time step of the integration and lead to the drift of the orientation estimation. The drift can be compensated with a second orientation estimation with different sensors (Section 3.1.3.4).

3.1.3.4. Orientation Estimation with Accelerometer and Magnetometer

For this orientation estimation first the coordinate systems need to be defined. The used systems are described in chapter 3.1.3.1. The three dimensional vectors of the sensors need to be extended with a zero at the first position. In the general case, q is the resulting quaternion, d the reference direction of the earth frame and s the measured values of the sensor frame [6]:

$$q = [q_1, q_2, q_3, q_4] \tag{3.1.3.4}$$

$$d = [0, d_x, d_y, d_z] \tag{3.1.3.5}$$

$$s = [0, s_x, s_y, s_z] \tag{3.1.3.6}$$

The Madgwick-filter uses the gradient descent method to calculate the orientation estimation with the accelerometer and the magnetometer. The gradient can be calculated by multiplying the Jacobian matrix J with the object function f in equation 3.1.3.7 [6]:

$$\begin{aligned} f(q,d,s) &= q \times d \times q^* - s \\ &= \begin{bmatrix} 0 \\ 2d_x(\frac{1}{2} - q_3^2 - q_4^2) + 2d_y(q_1q_4 + q_2q_3) + 2d_z(q_2q_4 - q_1q_3) - s_x \\ 2d_x(q_2q_3 - q_1q_4) + 2d_y(\frac{1}{2} - q_2^2 + q_4^2) + 2d_z(q_1q_2 + q_3q_4) - s_y \\ 2d_x(q_1q_3 + q_2q_4) + 2d_y(q_3q_4 - q_1q_2) + 2d_z(\frac{1}{2} - q_2^2 - q_3^2) - s_z \end{bmatrix} \end{aligned}$$

$$J(q,d) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 2d_yq_4 - 2d_zq_3 & 2d_yq_4 - 2d_zq_3 & -4d_xq_3 + 2d_yq_2 - 2d_zq_1 & -4d_xq_4 + 2d_yq_1 + 2d_zq_2 \\ -2d_xq_4 + 2d_zq_2 & 2d_xq_3 - 4d_yq_2 + 2d_zq_1 & 2d_xq_2 + 2d_zq_4 & -2d_xq_1 - 4d_yq_4 + 2d_zq_3 \\ 2d_xq_3 - 2d_yq_2 & 2d_xq_4 - 2d_yq_1 - 4d_zq_2 & 2d_xq_{+}2d_yq_4 - 4d_zq_3 & 2d_xq_2 + 2d_yq_3 \end{bmatrix}$$

The gradient is then defined in the following equation [6]:

$$\Delta f(q,d,s) = J(q,d)f(q,d,s) \tag{3.1.3.7}$$

3.1.3.5. Orientation Estimation with the Accelerometer

According to the description of the accelerometer in chapter 3.1.1 it is necessary to replace the earth and sensor frame of the general equations with the gravitational vectors. The gravity/earth frame has only a component on the z-axis g and the sensor frame is the low-pass-filtered acceleration measured with the accelerometer. With these definitions it is possible to define our objective function and Jacobian matrix [6]:

$$g = [0, 0, 0, 1] \tag{3.1.3.8}$$

$$a = [0, a_x, a_y, a_z] \tag{3.1.3.9}$$

$$f_g(q, a) = \begin{bmatrix} 2(q_2q_4 - q_1q_3) - a_x \\ 2(q_1q_2 + q_3q_4) - a_y \\ 2(\frac{1}{2} - q_2^2 - q_3^2) - a_z \end{bmatrix} \tag{3.1.3.10}$$

$$J_g(q) = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix} \tag{3.1.3.11}$$

3.1.3.6. Orientation Estimation with the Magnetometer

The same procedure as in chapter 3.1.3.5 is used with the magnetometer measurements. The magnetic field of the earth does not only have the whole resulting value on one axis. The inclination of the magnetic field of the earth is changing with the latitude. This leads to a horizontal component (z-axis) in the earth frame b [6].

$$b = [0, b_x, 0, b_z] \tag{3.1.3.12}$$

$$m = [0, m_x, m_y, m_z] \tag{3.1.3.13}$$

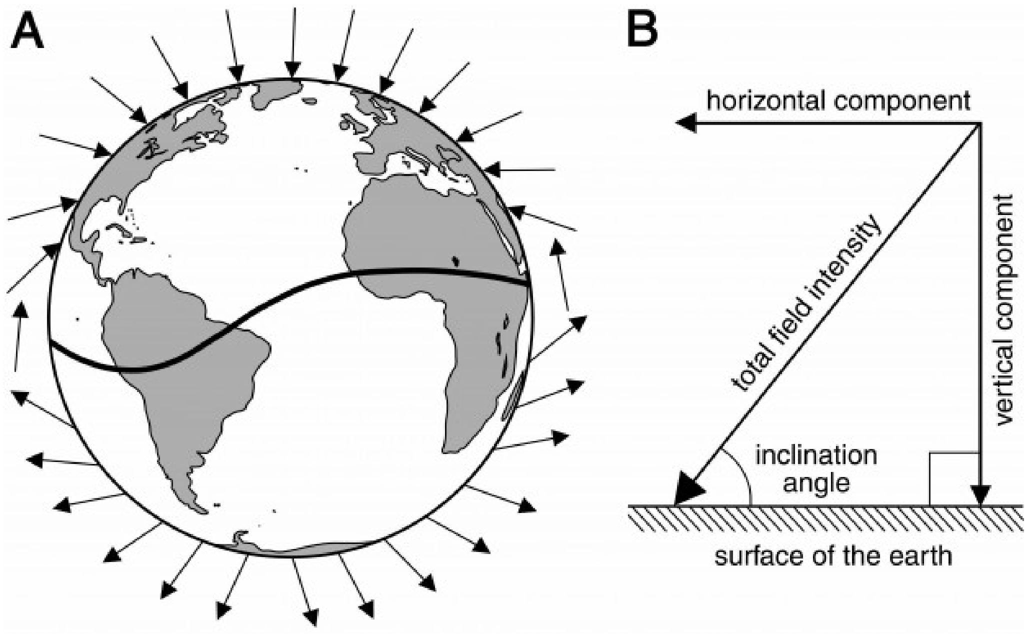


Figure 3.6.: Varying inclination with latitude [9]

To get the vector b the measured values of the low-pass-filtered magnetometer m is first rotated by the last estimated quaternion [6]:

$$h_t = [0 \quad h_x \quad h_y \quad h_z] = q_{t-1} \times m_t \times q^*_{t-1} \tag{3.1.3.14}$$

The horizontal component is then moved on to the x-axis [6]:

$$b_t = \begin{bmatrix} 0 & \sqrt{h_x^2 + h_y^2} & 0 & h_z \end{bmatrix} \tag{3.1.3.15}$$

With the calculated vector b it is again possible to replace the earth and sensor frame of the general equation with the magnetic vector values. The Jacobian matrix can be defined with the help of the objective function [6]:

$$f_b(q,b,m) = \begin{bmatrix} 2b_x(0.5 - q_3^2 - q_4^2) + 2b_z(q_2q_4 - q_1q_3) - m_x \\ 2b_x(q_2q_3 - q_1q_4) + 2b_z(q_1q_2 + q_3q_4) - m_y \\ 2b_x(q_1q_3 + q_2q_4) + 2b_z(0.5 - q_2^2 - q_3^2) - m_z \end{bmatrix} \tag{3.1.3.16}$$

$$J_b(q,b) = \begin{bmatrix} -2b_zq_3 & 2b_zq_4 & -4b_xq_3 - 2b_zq_1 & -4b_xq_4 + 2b_zq_2 \\ -2b_xq_4 + 2b_zq_2 & 2b_xq_3 + 2b_zq_1 & 2b_xq_2 + 2b_zq_4 & -2b_xq_1 + 2b_zq_3 \\ 2b_xq_3 & 2b_xq_4 - 4b_zq_2 & 2b_xq_1 - 4b_zq_3 & 2b_xq_2 \end{bmatrix} \tag{3.1.3.17}$$

3.1.3.7. Combining Accelerometer and Magnetometer

The orientation cannot be determined by measuring the gravity or the magnetic field alone, because only one axis of the coordinate system is fixed by each measurement (gravity shows the direction of the z-axis of the coordinate system). Therefore, the objective function and the Jacobian matrix of the accelerometer and the magnetometer need to be combined [6]:

$$f_{g,b}(q,a,b,m) = \begin{bmatrix} f_g(q,a) \\ f_b(q,b,m) \end{bmatrix} \tag{3.1.3.18}$$

$$J_{g,b}^T(q,b) = \begin{bmatrix} J_g^T(q) \\ J_b^T,(b,b) \end{bmatrix} \tag{3.1.3.19}$$

By combining the two functions/matrices we get a unique orientation of the sensor [6].

3.1.3.8. Sensor Fusion

In the last step the two orientation estimations are combined. The gyroscope measurements $q_{\omega,t}$ are corrected with the estimated direction of the error $\frac{\nabla f}{\|\nabla f\|}$ and the value of the error β [6]. The error ($\beta = 0.9$) was decided by trial and error because there was no information about the error of the gyroscope of the mobile device.

$$\dot{q}_t = q_{\omega,t} - \beta \frac{\nabla f}{\|\nabla f\|} \tag{3.1.3.20}$$

$$q_t = q_{t-1} + \dot{q}_t \Delta t \tag{3.1.3.21}$$

3.1.4. Changing the View Matrix with Quaternions

In order for a video to be displayed correctly in the Android application, the view matrix (Section 2.1) can be adjusted by the quaternions calculated by the Madgwick filter. The quaternion is converted into a rotation matrix. The columns of the rotation matrix equal the three vectors that have to be calculated for the view matrix [30]:

$$\begin{bmatrix} right_x & up_x & forward_x \\ right_y & up_y & forward_y \\ right_z & up_z & forward_z \end{bmatrix} = \begin{bmatrix} -(1-(2q_3^2+2q_4^2)) & 2(q_2q_3-q_1q_4) & 2(q_2q_4+q_1q_3) \\ -(2(q_2q_3+q_1q_4)) & 1-2(q_2^2+q_4^2) & 2(q_3q_4-q_1q_2) \\ -(2(q_2q_4-q_1q_3)) & 2(q_3q_4+q_1q_2) & 1-2(q_1^2+q_3^2) \end{bmatrix} \tag{3.1.4.1}$$

The negation in the right vector is caused by the orientation of the axes of the coordinate system. A negative sign leads to a right-handed and a positive sign to a left-handed coordinate system.

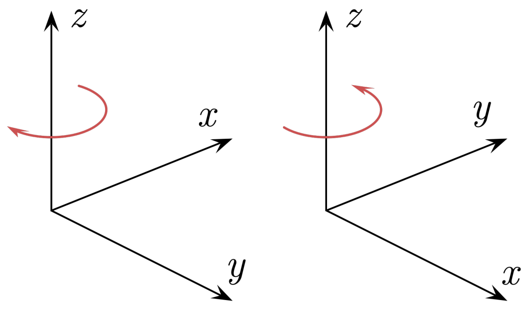


Figure 3.7.: Left-handed (left) and right-handed (right) coordinate system [10]

In this work a right-handed coordinate system is used.

3.1.5. Extracting Tiles from Quaternions

The vectors of the view matrix (Section 2.1) are used to find the FOV on the two-dimensional video frame. The center of the FOV is in the opposite direction of the forward vector. In other words, the forward vector points into the opposite direction of the virtual camera of the three-dimensional scene. With the following formulas and the knowledge that the we use an equirectangular projection (Section 2.1), it is possible to find the center of the FOV on the two-dimensional video frame [31]:

$$x = \frac{\pi - \arctan(forward_y / forward_x)}{2\pi} \tag{3.1.5.1}$$

$$y = \frac{\arccos(forward_z)}{\pi} \tag{3.1.5.2}$$

The FOV angle is the angle that decides how much of the sphere is displayed on the phone. This angle is hard coded in the application. The angle that is displayed in the Android application on the long side of the phone is 70° (The FOV angle was determined by trial and error). With the aspect ratio of the smart phone, it is possible to calculate the angle on the short side of the phone. With the right- and the up-vector the front-vector is step wise rotated by these angles. With the equations 3.1.5.1 and 3.1.5.2 the points on the two dimensional frame are calculated again. For the rotation of the three vectors again a quaternion is used. A rotation around the right-vector by the angle β:

$$q_{rotation} = [\cos(\beta), \sin(\beta) * right_x, \sin(\beta) * right_y, \sin(\beta) * right_z] \tag{3.1.5.3}$$

The new vectors are then rotated. The rotation is defined by the following equations:

$$forward_{new} = q_{rotation} \times forward \times q^*_{rotation} \tag{3.1.5.4}$$

$$right_{new} = q_{rotation} \times right \times q^*_{rotation} \tag{3.1.5.5}$$

$$up_{new} = q_{rotation} \times up \times q^*_{rotation} \tag{3.1.5.6}$$

The coordinates of the two dimensional FOV are then linked to a tile. First, the two-dimensional image is divided into tiles. Then the calculated x- and y-coordinate are assigned to the respective tiles. The tiles that are in the FOV are the tiles that need to be predicted.

3.2. Neural Networks

In this section, we present the neural networks that we used. The first neural network approach predicts the quaternions and the second approaches predicts the tiles that are used in the future directly. Further, we describe the hyperparameters of the networks (number of layers/nodes and loss-functions). The training and evaluations are described in Section 4.4 and Section 5.1.

3.2.1. Input Layer

For both networks, we used the data listed in Table 4.1 as input parameters. We used the quaternion (4 inputs), the accelerometer (3 inputs), the magnetometer (3 inputs) and the gyroscope (3 inputs) as input values (In total 13 values). But not only the data from the current point in time was used, but also sensor data and orientation estimation from the past. The data points 100ms, 200ms, 300ms and 400ms before the actual timestamp were also used as an input. Because each data point has 13 input values and 5 data points were used as input parameters, the input layer has 65 nodes.

3.2.2. Predicting the Quaternion

This prediction was inspired by neural networks in robotics [32]. The neural network is based on Frigeni's work [32]. We started out with the same hyperparameters and then adjusted them by trial and error by adding more or less nodes/layers. The loss function that is used for this neural network is the mean squared error. The resulting network has three hidden layers and each hidden layer has two hundred nodes. The output layer has four nodes because the network predicts a quaternion that consists of four values. The resulting layers are described in Section 3.2.2 and Figure 3.8.

Layer	**Nodes**	**Activation function**
Input layer	65	
Hidden layer 1	200	ReLU
Hidden layer 2	200	ReLU
Hidden layer 3	200	ReLU
Output layer	4	Sigmoid

Table 3.2.: Layers for predicting the orientation estimation

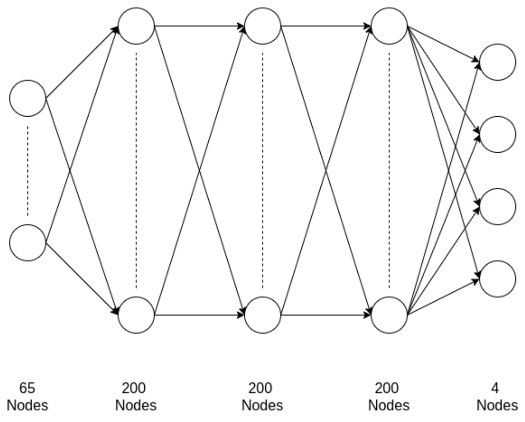


Figure 3.8.: Neural network for predicting the quaternion

3.2.3. Predicting the Image Tiles

The prediction of which tiles will be used in the future is a classic classification problem. In this case the outputs of the network are the individual tiles and can have a value of 0 and 1. If the value is 1, the tile is predicted to be in the FOV of the user. This is a good fit for the binary cross entropy loss function. This neural network is based on classical classification problems [33]. We first guessed the hyperparameters of the network and then improved it by trial and error. The resulting layers are described in Section 3.2.3 and Figure 3.9.

Layer	**Nodes**	**Activation function**
Input layer	65	
Hidden layer 1	1000	ReLU
Hidden layer 2	1000	ReLU
Hidden layer 3	500	ReLU
Hidden layer 4	250	ReLU
Output layer	Number of tiles	Sigmoid

Table 3.3.: Layers for predicting the used tiles

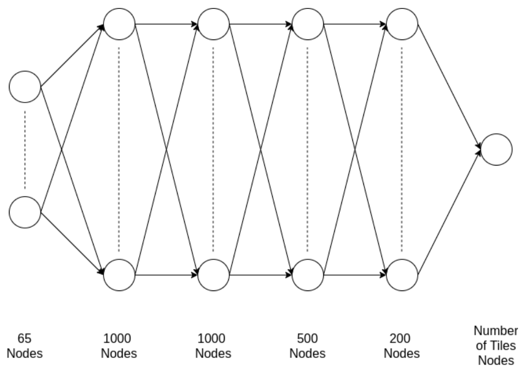


Figure 3.9.: Neural network for predicting the used tiles.

4. Implementation

The implementation of this thesis consists of three components. Figure 4.1 shows the parts of this thesis and how they are connected.

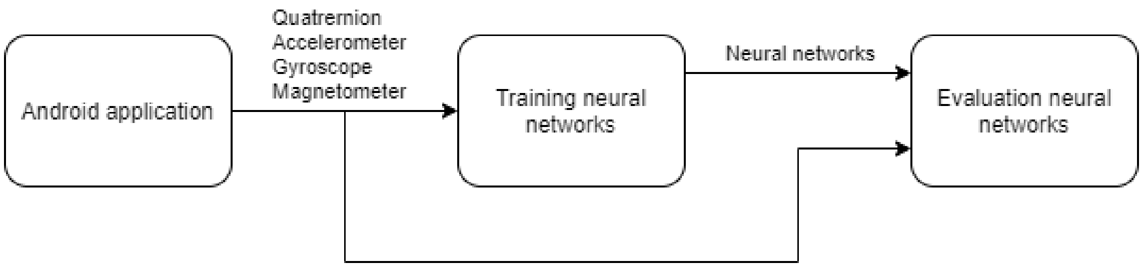


Figure 4.1.: Overview of the parts of the thesis

The first part consists of an Android application [34]. The application has two purposes. On one hand it is possible to watch a local 360° video with the Android application and on the other hand the application exports the data described in table 4.1, which is used for the training and evaluation of the neural networks. For both purposes, the application has to estimate the orientation of the smartphone.

In a second part we train the neural networks with the data stored that was exported in part the first part. For this purpose, we divide the data into a training and an evaluation data set. Then we save the trained networks.

In the last part we test the neural networks with the evaluation data sets. We evaluate the bandwidth savings and the compliance with the latency. We discuss the results in Chapter 5.

4.1. Devices and Software

The following devices were used to implement this work:

- For the Android application we used a Google Pixel 3a
  - OS: Android 10
  - CPU: Qualcomm SDM670 Snapdragon 670
  - RAM: 4GB
- A Dell XPS 13 9370 notebook was used for the training and evaluation of the neural networks
  - OS: Ubuntu 18.04 LTS
  - CPU: Intel Core i7-8550U CPU 1.80 GHz $\times$ 8
  - RAM: 16 GB

The software was developed with the following tools:

- PyCharm with Python version 3.6 was used for the development of neural networks [35, 36]
- Android Studio with the API level 27 (Android 8.1) for the development of the Android application [34]

4.2. Android Application

When opening the Android application a local 360° video is loaded and played. To render 360° videos, we extended a sample project provided by Facebook [37].

At a high level, the application is structured as follows. The accelerometer, gyroscope and magnetometer are initialized. The class *DataCollector* collects the data of the sensors when they update their values. Both the values of the accelerometer and the values of the magnetometer are low-pass filtered before they are passed to the *DataCollector*. If the gyroscope sends new values to the *DataCollector*, the Madgwick filter recalculates the orientation of the mobile device (Section 3.1.3). The quaternion calculated by the Madgwick filter is then low pass filtered as well.

This new calculated quaternion is stored in the *DataCollector*. The class that renders the 360° video onto the sphere gets the quaternion for every frame of the video. The view matrix is then adapted with the new value of the quaternion and the correct part of the video is displayed (Section 3.1.4).

The measured and calculated data is exported as well. Each **data point** contains the values in Table 4.1.

Measured / Calculated value	**Column in CSV**
Timestamp in ms	timestamp
Low-pass filtered Quaternion w	average q w
Low-pass filtered Quaternion x	average q x
Low-pass filtered Quaternion y	average q y
Low-pass filtered Quaternion z	average q z
Low-pass filtered Accelerometer x-axis in m/s^2	acc x
Low-pass filtered Accelerometer y-axis in m/s^2	acc y
Low-pass filtered Accelerometer z-axis in m/s^2	acc z
Gyroscope x-axis in rad/s	gyro x
Gyroscope y-axis in rad/s	gyro y
Gyroscope z-axis in rad/s	gyro z
Low-pass filtered Magnetometer x-axis in μT	mag x
Low-pass filtered Magnetometer y-axis in μT	mag y
Low-pass filtered Magnetometer z-axis in μT	mag z

Table 4.1.: Data recorded by the Android application. Only the values used as features for training the neural networks are listed.

The values are exported when a certain amount of time passed after the last data point was exported. In our case the time between the data points is in average 10 or 100ms.

4.3. Experiment

For this work, data was collected in two studies. The data was collected to train and evaluate the neural networks. In the first study one data point was exported from the Android application in average every 100ms and in the second study one data point was exported in average every 10ms. We recruited five subjects for each study. Different people were used for the two studies.

We handed the participants an Android mobile phone (Section 4.1). We instructed the users that by changing the position of the mobile phone they can see a different section of the 360° video.

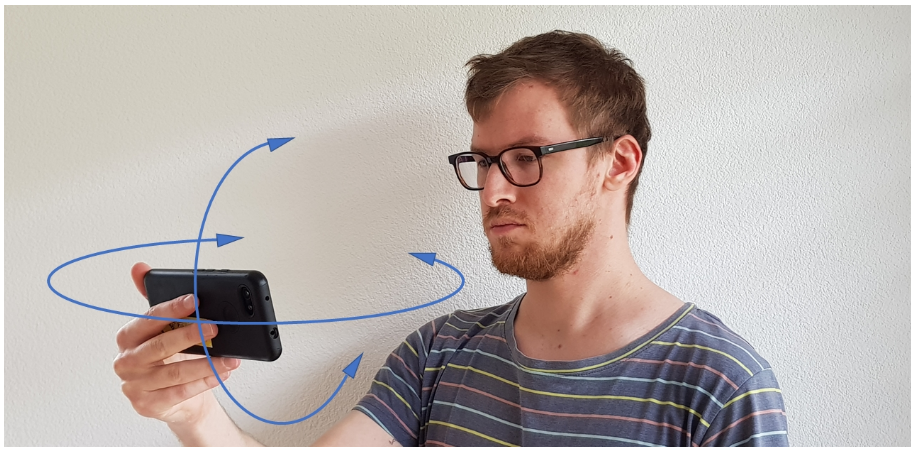


Figure 4.2.: Putting the phone in a different position brings a different section of the 360° video into view

In the first video, a drone flight in Thailand [19], the participants were instructed to look at the landscape by changing the orientation of the mobile phone. The video is 31 seconds long and was repeated after reaching the end. The participants were instructed to have a look at the landscape into all directions and to get used to the handling of the Android application. When they had seen enough of the landscape they stopped the application. The users usually watched 2-3 minutes.

In a second part of the study, the participants watched a different 360° video, the short film "The invisible man" [18]. We did not instruct the participants to look in a specific direction. After finishing the video they stopped the Android application.

The number of data points and the average and standard deviation of the time between the data points, that were gathered in the two studies for the training and the evaluation of the neural networks, is summarized in Table 4.2.

Data set	**Interval data points training**	**#data points for training**	**Interval data points evaluation**	**#data points for evaluation**
First data set	(100 ± 3.2)ms	19205	(100 ± 3.2)ms	12273
Second data set	(10.2 ± 1.6)ms	281291	(10 ± 1.5)ms	72140

Table 4.2.: Number of data points and time between the data points for training and evaluation

4.4. Training of Neural Networks

The neural networks are implemented in Python with the Tensorflow framework [38]. We implemented two different neural networks that predict the parts of the video that are used. The first approach tries to predict the orientation of the smart phone and the second approach predicts the tiles directly.

The training of the neural networks proceeds shown in Figure 4.3. In a first part the data is read in, then the data is processed and the inputs are generated. Then, first the outputs for the prediction of quaternions are generated, the neural networks are trained and stored. Then the outputs for the prediction of tiles are generated, the neural networks are trained and stored. The scripts nn.py and nn10milli.py (Appendix A) are used for the training of neural networks. In the following section the procedure is explained in more detail.

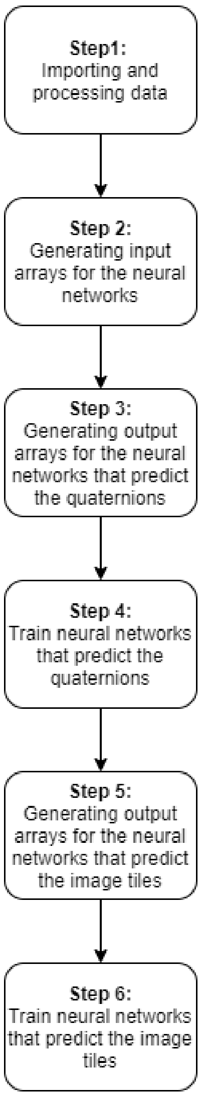

Step1: Importing and processing data
Step 2: Generating input arrays for the neural networks
Step 3: Generating output arrays for the neural networks that predict the quaternions
Step 4: Train neural networks that predict the quaternions
Step 5: Generating output arrays for the neural networks that predict the image tiles
Step 6: Train neural networks that predict the image tiles

Figure 4.3.: Overview of the parts for training the neural networks

Step 1: The training data is imported. Each CSV file stored in a specific folder is read in line by line. The FOV is determined directly for each data point (line in the CSV). This is done with the method described in section 3.1.5. The data is processed directly, so that the FOV for each data point has to be determined only once. After determining the FOV, the tiles that would be used for this data point are determined. We compute the results for 4, 16, 36, 64, and 100 tiles and store them in the data point.

Step 2: The input arrays for the neural networks are generated. Since for the input not only the data point with the current timestamp is used, an array with several data points is required. Consider a data point d_i. The first data set has in average one data point every 100ms in the CSV files. The last four data points are also used as an input, i.e. d_{i-1}, d_{i-2}, d_{i-3} and d_{i-4}. For the second data set with data points in average every 10ms, the same time interval was used for the additional input data. In this case d_{i-10}, d_{i-20}, d_{i-30} and d_{i-40} were also used for the input array.

Step 3: The output arrays for the case that the neural networks the quaternion predictions. For this purpose arrays with quaternions of data points of a point with a later timestamp are created. For each data point several output arrays are generated to evaluate different prediction times. For the first data set, the quaternions are predicted in 0.5s, 1s and 2s. This means that one array is created with d_{i+5}, one with d_{i+10} and one with d_{i+20}. For the second record, more prediction times are considered: 10ms, 20ms, 50ms, 0.1s, 0.2s, 0.5s, 1s, 2s. This means that one output array is created for each d_{i+1}, d_{i+2}, d_{i+5}, d_{i+10}, d_{i+20}, d_{i+50}, d_{i+100}, d_{i+200}.

Step 4: The neural networks with the approach to predict the quaternions are trained. For both data sets a neural network is created for each prediction time. For the training the input array from step 2 and the output arrays from step 3 are used. The training of the neural network has always 20 epochs. An epoch is when the neural network is trained with the whole training data set once. After the training the neural networks are saved.

Step 5: The output arrays for the neural networks that predict the tiles are generated. For both datasets arrays are generated for the same prediction times as in step 3. However, instead of arrays with quaternions, arrays with the results for 4, 16, 36, 64, 100 tiles are generated.

Step 6: The neural networks that predict the image tiles are trained. For both data sets a neural network is created for each prediction time. For the training the input array from step 2 and the output arrays from step 3 are used. In this case the training of the neural network has always 20 epoches as well. The neural networks are saved as well.

5. Evaluation

In this chapter the evaluation methods of the neural networks are shown. Furthermore the results are described and presented. Finally the results are reviewed in a discussion.

5.1. Evaluation of Neural Networks

By evaluating the neural networks it is possible to find out how accurately a neural network can predict the image tiles that will be used in the future. In this section it is explained in more detail how the two different approaches of neural networks were tested.

5.1.1. Evaluation of Prediction Quaternion

For the evaluation of neural networks that predict the quaternions the scripts evaluation.py and evaluation10milli.py are used (Appendix A: evaluation.py for the first dataset and evaluation10milli.py for the second data set). The following steps are performed for the evaluation of the neural networks that predict the quaternions:

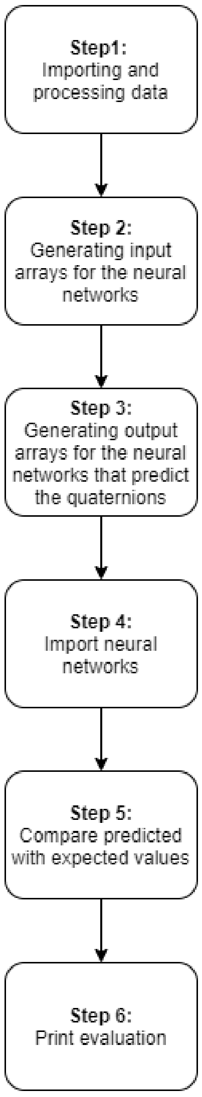

Step1:
Importing and processing data
Step 2:
Generating input arrays for the neural networks
Step 3:
Generating output arrays for the neural networks that predict the quaternions
Step 4:
Import neural networks
Step 5:
Compare predicted with expected values
Step 6:
Print evaluation

Figure 5.1.: Steps in the evaluation of predicting a quaternion

Step 1-3: These steps are performed like steps 1-3 in the training of neruonal networks (Section 4.4). Instead of the training data, the evaluation data is imported, processed and converted to input and output arrays.

Step 4: Import of the neural networks. The evaluation.py script imports the networks generated with the first data set and the evaluation100milli.py script imports the networks generated with the first data set (Appendix A).

Step 5: The imported networks from step 4 are then used to make a prediction with the given inputs from the evaluation data. First an input I_i is passed to the neural network and the network calculates a prediction p_i in form of a quaternion. This quaternion can then be converted to the predicted tiles t_{p_i} as described in section 3.1.5. The prediction was created by the data point d_i. Thus it is possible to compare the predicted tiles from the input I_i with the expected tiles of the data points $t_{expected,i+5}$ or $t_{expected,i+10}$ or $t_{expected,i+20}$ (first data set) imported in step 1.

Step 6: The last step is generating the evaluation parameters described in section 5.2

5.1.2. Evaluation of Prediction Image Tiles

The scripts for evaluating the neural networks that predict the image tiles are evaluationTiles.py and evaluationTiles10milli.py (Appendix A, evaluationTiles.py for the first dataset and evaluationTiles10milli.py for the second data set). The evaluation of neural networks that predict the image tiles differs only in step 3, 4 and 5 form the evaluation in section 5.1.1.

Step 1-2 and 6: Same steps as in section 5.1.1

Step 3: Step 3 equals Step 5 of the training of the neural networks (Section 4.4).

Step 4: Import of neural networks. This time the networks are imported that predict the image tiles instead of the networks that predict the quaternions.

Step 5: Again the networks are used to make a prediction with the help of the inputs from the evaluation data. In this case the tiles t_{p_i} are predicted directly by the neural networks. These values are then again compared to the expected values.

5.2. Evaluation Parameters

In this section, the individual tables columns of the results are explained in more detail.

5.2.1. Number of Tiles

The number of tiles value (# Tiles) tells into how many tiles the video is divided into. If the number of tiles is 4 the image is divided into 2x2 tiles. If the number of tiles equals 100 then the frame is divided into 10x10 tiles.

5.2.2. Prediction

The prediction time (Prediction) is the average and standard deviation of the time between the input value and the predicted value of the training data set (Prediction Training) as well as the evaluation data set (Prediction Evaluation).

5.2.3. FOV angles

The FOV angles tell us with which angle the networks were trained and evaluated. The first value is the training angle and the second value is the evaluation angle. To improve the results of the networks it is necessary to make a compromise between bandwidth and error. The network can be trained with a larger FOV angle than the one that is displayed on the screen of the user. More bandwidth is needed because more parts of the video are sent to the user, but this should also lead to less false negatives and therefore a better QoE for the user. We trained the neural networks with 90° or 70° FOV angle. The neural networks were always evaluated with an FOV angle of 70°. In the table, the networks that were trained with 90° are labeled in column ”FOV angle” with 90, 70. The networks that were trained and evaluated with the same angle are labeled with 70, 70.

5.2.4. Error determination

In the evaluation, we differentiate between the following labels:

True positives (TP) are tiles that are predicted to be inside the FOV and that are in fact inside the FOV.

True negatives (TN) are tiles that are predicted to be outside the FOV and that are in fact outside the FOV.

False positives (FP) are tiles that are predicted to be inside the FOV, but are in fact outside the FOV.

False negatives (FN) are tiles that are predicted to be outside the FOV, but are in fact inside the FOV.

Figure 5.2 visualizes the different labels. The blue rectangle is the FOV, the red squares are predicted to be outside of the FOV and the green squares are predicted to be inside the FOV.

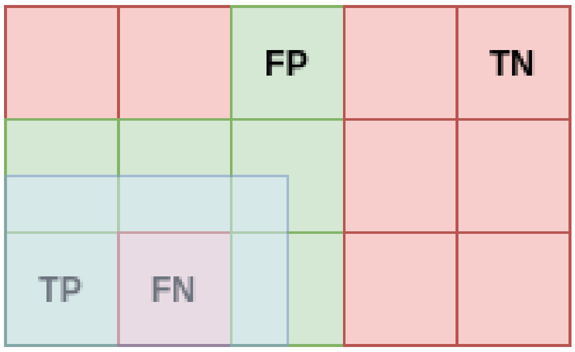


Figure 5.2.: FOV (blue), TP = True positives, TN = True negatives, FP = False positives, FN = False negatives

5.2.5. Errors

The number of errors (#Errors) includes both tiles that were incorrectly predicted and tiles that were incorrectly not predicted. The errors are the total of the false negatives and the false positives.

$$Errors = \frac{FP + FN}{TP + FP + FN + TN} \tag{5.2.5.1}$$

5.2.6. Accuracy

The accuracy describes the correct labeled tiles in the whole set of tiles [39]. But accuracy is only a good measure if the counts of false negatives and false positives are close [39].

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} \tag{5.2.6.1}$$

5.2.7. Recall

The recall measurements are used if the false negatives are unaccepted [39]. This measurement is good when there are more false positives then false negatives [39].

$$Recall = \frac{TP}{TP + FN} \tag{5.2.7.1}$$

5.2.8. Number of Predicted Tiles

The number of the predicted tiles (#Predicted) tells us how many tiles are predicted to be in the FOV. If a video would be streamed the used bandwidth compared to the whole frame would be the number of predicted tiles.

$$\#Predicted = \frac{TP + FP}{TP + FP + FN + TN} \tag{5.2.8.1}$$

With the number of predicted tiles we can then calculate the theoretical bandwidth savings. We assume that only the predicted tiles are transmitted to the user. This means that the saved bandwidth consists of the non-predicted tiles and is calculated in Equation (5.2.8.2).

$$BandwidthSavings = 1 - \#Predicted = \frac{TN + FN}{TP + FP + FN + TN} \tag{5.2.8.2}$$

5.2.9. Time of the Neural Network Predictions

The time of neural network predictions (Time) is the average time the neural networks need to predict the quaternions or image tiles.

5.3. Results Prediction Quaternion

5.3.1. Results of First Data Set (Data point in average every 100ms)

#Tiles	Prediction Train	Prediction Eval	FOV angle Train, Eval	#Errors	#FP	#FN	Accuracy	Recall	#Predicted	Time
4	0.5s±4ms	0.5s±5ms	70, 70	7.2%	4.1%	3.0%	92.8%	95.8%	73.4%	0.4ms
16	0.5s±4ms	0.5s±5ms	70, 70	5.2%	3.5%	1.7%	94.8%	95.1%	35.8%	0.4ms
36	0.5s±4ms	0.5s±5ms	70, 70	4.7%	2.1%	2.6%	95.3%	92.3%	32.4%	0.6ms
64	0.5s±4ms	0.5s±5ms	70, 70	3.7%	2.0%	1.7%	96.3%	93.8%	27.4%	0.4ms
100	0.5s±4ms	0.5s±5ms	70, 70	4.0%	2.2%	1.8%	96.0%	92.6%	24.8%	0.4ms
4	1s±5ms	1s±7ms	70, 70	14.0%	9.6%	4.4%	86.0%	94.0%	77.6%	0.4ms
16	1s±5ms	1s±7ms	70, 70	9.0%	5.8%	3.2%	91.0%	90.6%	36.5%	0.4ms
36	1s±5ms	1s±7ms	70, 70	8.2%	4.4%	3.8%	91.8%	88.4%	33.4%	0.4ms
64	1s±5ms	1s±7ms	70, 70	6.9%	3.6%	3.3%	93.1%	88.0%	27.4%	0.4ms
100	1s±5ms	1s±7ms	70, 70	7.1%	3.7%	3.4%	92.9%	85.8%	24.6%	0.4ms
4	2s±7ms	2s±10ms	70, 70	19.5%	12.0%	7.5%	80.5%	89.7%	79.4%	0.4ms
16	2s±7ms	2s±10ms	70, 70	15.5%	9.8%	5.7%	84.5%	83.1%	37.9%	0.4ms
36	2s±7ms	2s±10ms	70, 70	14.1%	7.2%	6.9%	85.9%	79.0%	33.1%	0.4ms
64	2s±7ms	2s±10ms	70, 70	12.1%	6.3%	5.8%	87.9%	78.7%	27.6%	0.4ms
100	2s±7ms	2s±10ms	70, 70	12.2%	6.4%	5.8%	87.8%	76.1%	24.9%	0.4ms
4	0.5s±4ms	0.5s±5ms	90, 70	27.5%	27.5%	0.0%	72.5%	100.0%	99.8%	0.4ms
16	0.5s±4ms	0.5s±5ms	90, 70	24.0%	23.9%	0.1%	75.9%	99.8%	57.8%	0.4ms
36	0.5s±4ms	0.5s±5ms	90, 70	13.6%	13.5%	0.1%	86.4%	99.8%	46.3%	0.4ms
64	0.5s±4ms	0.5s±5ms	90, 70	12.3%	12.2%	0.1%	87.7%	99.8%	39.2%	0.4ms
100	0.5s±4ms	0.5s±5ms	90, 70	12.7%	12.6%	0.1%	87.3%	99.7%	37.0%	0.4ms
4	1s±5ms	1s±7ms	90, 70	27.5%	27.4%	0.1%	72.5%	99.9%	99.7%	0.4ms
16	1s±5ms	1s±7ms	90, 70	28.1%	27.5%	0.6%	71.9%	98.3%	60.9%	0.4ms
36	1s±5ms	1s±7ms	90, 70	14.9%	14.2%	0.7%	85.1%	97.9%	46.4%	0.4ms
64	1s±5ms	1s±7ms	90, 70	14.9%	14.2%	0.7%	85.1%	97.4%	40.6%	0.4ms
100	1s±5ms	1s±7ms	90, 70	14.1%	13.4%	0.7%	85.9%	96.9%	37.0%	0.4ms
4	2s±7ms	2s±10ms	90, 70	27.6%	27.5%	0.1%	72.4%	99.9%	99.8%	0.4ms
16	2s±7ms	2s±10ms	90, 70	31.1%	28.7%	2.4%	68.9%	93.0%	60.3%	0.4ms
36	2s±7ms	2s±10ms	90, 70	19.3%	16.7%	2.6%	80.7%	91.3%	46.9%	0.4ms
64	2s±7ms	2s±10ms	90, 70	18.6%	16.0%	2.6%	81.4%	90.4%	40.4%	0.4ms
100	2s±7ms	2s±10ms	90, 70	18.1%	15.5%	2.6%	81.9%	89.4%	37.3%	0.4ms

Table 5.1.: Quaternion prediction with first data set (Data point in average every 100ms)

5.3.2. Results of Second Data Set (Data point in average every 10ms)

#Tiles	Prediction Train	Prediction Eval	FOV angle Train, Eval	#Errors	#FP	#FN	Accuracy	Recall	#Predicted	Time
4	(10 ± 2)ms	(10 ± 2)ms	70, 70	1.2%	0.7%	0.5%	98.8%	99.4%	86.5%	0.4ms
16	(10 ± 2)ms	(10 ± 2)ms	70, 70	1.7%	0.8%	0.9%	98.3%	98.2%	50.0%	0.4ms
36	(10 ± 2)ms	(10 ± 2)ms	70, 70	1.8%	0.9%	0.8%	98.2%	97.8%	40.9%	0.4ms
64	(10 ± 2)ms	(10 ± 2)ms	70, 70	1.7%	0.8%	0.9%	98.3%	97.5%	35.0%	0.4ms
100	(10 ± 2)ms	(10 ± 2)ms	70, 70	1.8%	0.9%	0.9%	98.2%	97.2%	32.0%	0.4ms
4	(20 ± 2)ms	(20 ± 2)ms	70, 70	1.4%	0.8%	0.6%	98.5%	99.3%	86.6%	0.6ms
16	(20 ± 2)ms	(20 ± 2)ms	70, 70	1.8%	0.9%	0.9%	98.2%	98.2%	50.1%	0.6ms
36	(20 ± 2)ms	(20 ± 2)ms	70, 70	1.8%	0.9%	0.9%	98.2%	97.8%	40.8%	0.6ms
64	(20 ± 2)ms	(20 ± 2)ms	70, 70	1.7%	0.8%	0.9%	98.3%	97.5%	35.0%	0.6ms
100	(20 ± 2)ms	(20 ± 2)ms	70, 70	1.7%	0.8%	0.9%	98.3%	97.2%	32.0%	0.6ms
4	(51 ± 4)ms	(51 ± 3)ms	70, 70	1.3%	0.6%	0.7%	98.7%	99.2%	86.4%	0.6ms
16	(51 ± 4)ms	(51 ± 3)ms	70, 70	1.8%	0.9%	0.9%	98.2%	98.1%	50.0%	0.6ms
36	(51 ± 4)ms	(51 ± 3)ms	70, 70	1.9%	0.9%	1.0%	98.1%	97.7%	40.8%	0.6ms
64	(51 ± 4)ms	(51 ± 3)ms	70, 70	1.8%	0.9%	0.9%	98.2%	97.4%	35.1%	0.6ms
100	(51 ± 4)ms	(51 ± 3)ms	70, 70	1.8%	0.9%	0.9%	98.2%	97.2%	32.1%	0.6ms
4	0.1s±5ms	0.1s±5ms	70, 70	1.5%	0.8%	0.7%	98.5%	99.2%	86.6%	0.6ms
16	0.1s±5ms	0.1s±5ms	70, 70	1.9%	0.9%	1.0%	98.1%	98.0%	50.1%	0.6ms
36	0.1s±5ms	0.1s±5ms	70, 70	2.1%	1.1%	1.0%	97.9%	97.5%	40.8%	0.6ms
64	0.1s±5ms	0.1s±5ms	70, 70	2.0%	1.0%	1.0%	98.0%	97.2%	35.1%	0.6ms
100	0.1s±5ms	0.1s±5ms	70, 70	2.0%	1.0%	1.0%	98.0%	96.9%	32.1%	0.6ms
4	0.2s±7ms	0.2s±7ms	70, 70	1.6%	0.9%	0.7%	98.4%	99.2%	86.7%	0.6ms
16	0.2s±7ms	0.2s±7ms	70, 70	2.2%	1.1%	1.1%	97.8%	97.8%	50.1%	0.6ms
36	0.2s±7ms	0.2s±7ms	70, 70	2.3%	1.1%	1.2%	97.6%	97.0%	40.7%	0.6ms
64	0.2s±7ms	0.2s±7ms	70, 70	2.2%	1.1%	1.1%	97.8%	96.8%	35.0%	0.6ms
100	0.2s±7ms	0.2s±7ms	70, 70	2.3%	1.1%	1.2%	97.7%	96.4%	32.0%	0.6ms
4	0.5s±12ms	0.5s±12ms	70, 70	4.3%	2.3%	2.0%	95.7%	97.7%	86.8%	0.6ms
16	0.5s±12ms	0.5s±12ms	70, 70	6.1%	3.0%	3.1%	93.9%	93.7%	50.0%	0.6ms
36	0.5s±12ms	0.5s±12ms	70, 70	6.2%	3.0%	3.2%	93.8%	92.3%	40.7%	0.6ms
64	0.5s±12ms	0.5s±12ms	70, 70	6.0%	3.0%	3.0%	94.0%	91.4%	35.0%	0.6ms
100	0.5s±12ms	0.5s±12ms	70, 70	6.0%	3.0%	3.0%	94.0%	90.5%	32.0%	0.6ms
4	1s±17ms	1s±16ms	70, 70	10.8%	6.2%	4.6%	89.2%	94.7%	88.0%	0.6ms
16	1s±17ms	1s±16ms	70, 70	15.7%	8.0%	7.7%	84.3%	84.6%	50.4%	0.6ms
36	1s±17ms	1s±16ms	70, 70	15.7%	7.8%	7.9%	84.3%	80.6%	40.6%	0.6ms
64	1s±17ms	1s±16ms	70, 70	15.0%	7.5%	7.5%	85.0%	78.6%	35.1%	0.6ms
100	1s±17ms	1s±16ms	70, 70	14.8%	7.3%	7.5%	85.2%	76.7%	32.0%	0.6ms
4	2s±25ms	2s±22ms	70, 70	18.7%	9.8%	8.9%	81.2%	89.7%	87.4%	0.6ms
16	2s±25ms	2s±22ms	70, 70	32.3%	16.7%	15.6%	67.7%	68.9%	51.2%	0.6ms
36	2s±25ms	2s±22ms	70, 70	31.9%	15.7%	16.2%	68.1%	60.3%	40.3%	0.6ms
64	2s±25ms	2s±22ms	70, 70	30.8%	15.2%	15.6%	69.2%	55.7%	34.8%	0.6ms
100	2s±25ms	2s±22ms	70, 70	29.8%	14.6%	15.2%	70.2%	52.7%	31.6%	0.6ms
4	(10 ± 2)ms	(10 ± 2)ms	90, 70	10.6%	10.6%	0.0%	89.4%	100.0%	97.1%	0.4ms
16	(10 ± 2)ms	(10 ± 2)ms	90, 70	17.5%	17.5%	0.002%	82.5%	100.0%	67.7%	0.4ms
36	(10 ± 2)ms	(10 ± 2)ms	90, 70	15.5%	15.5%	0.003%	84.5%	100.0%	56.3%	0.4ms
64	(10 ± 2)ms	(10 ± 2)ms	90, 70	15.6%	15.6%	0.003%	84.4%	100.0%	50.7%	0.4ms
100	(10 ± 2)ms	(10 ± 2)ms	90, 70	15.2%	15.2%	0.003%	84.8%	100.0%	47.3%	0.4ms
4	(20 ± 2)ms	(20 ± 2)ms	90, 70	10.7%	10.7%	0.008%	89.3%	100.0%	97.1%	0.7ms
16	(20 ± 2)ms	(20 ± 2)ms	90, 70	17.7%	17.7%	0.005%	82.3%	100.0%	67.8%	0.7ms
36	(20 ± 2)ms	(20 ± 2)ms	90, 70	15.5%	15.5%	0.009%	84.5%	100.0%	56.3%	0.7ms
64	(20 ± 2)ms	(20 ± 2)ms	90, 70	15.6%	15.6%	0.008%	84.4%	100.0%	50.7%	0.7ms
100	(20 ± 2)ms	(20 ± 2)ms	90, 70	15.2%	15.2%	0.008%	84.8%	100.0%	47.3%	0.7ms
4	(51 ± 4)ms	(51 ± 3)ms	90, 70	10.6%	10.6%	0.0%	89.4%	100.0%	97.0%	0.6ms
16	(51 ± 4)ms	(51 ± 3)ms	90, 70	17.7%	17.7%	0.004%	82.3%	100.0%	67.8%	0.6ms
36	(51 ± 4)ms	(51 ± 3)ms	90, 70	15.6%	15.6%	0.008%	84.4%	100.0%	56.3%	0.6ms
64	(51 ± 4)ms	(51 ± 3)ms	90, 70	15.6%	15.6%	0.008%	84.4%	100.0%	50.7%	0.6ms
100	(51 ± 4)ms	(51 ± 3)ms	90, 70	15.2%	15.2%	0.008%	84.8%	100.0%	47.3%	0.6ms
4	0.1s±5ms	0.1s±5ms	90, 70	10.7%	10.6%	0.009%	89.3%	100.0%	97.1%	0.6ms
16	0.1s±5ms	0.1s±5ms	90, 70	17.7%	17.7%	0.01%	82.3%	100.0%	67.8%	0.6ms
36	0.1s±5ms	0.1s±5ms	90, 70	15.5%	15.5%	0.01%	84.5%	100.0%	56.3%	0.6ms
64	0.1s±5ms	0.1s±5ms	90, 70	15.6%	15.6%	0.01%	84.4%	100.0%	50.7%	0.6ms
100	0.1s±5ms	0.1s±5ms	90, 70	15.2%	15.2%	0.01%	84.8%	100.0%	47.3%	0.6ms
4	0.2s±7ms	0.2s±7ms	90, 70	10.7%	10.6%	0.01%	89.3%	100.0%	97.0%	0.6ms
16	0.2s±7ms	0.2s±7ms	90, 70	17.7%	17.7%	0.03%	82.3%	99.9%	67.8%	0.6ms
36	0.2s±7ms	0.2s±7ms	90, 70	15.5%	15.4%	0.03%	84.5%	99.9%	56.2%	0.6ms
64	0.2s±7ms	0.2s±7ms	90, 70	15.6%	15.6%	0.03%	84.4%	99.9%	50.7%	0.6ms
100	0.2s±7ms	0.2s±7ms	90, 70	15.2%	15.2%	0.03%	84.8%	99.9%	47.2%	0.6ms
4	0.5s±12ms	0.5s±12ms	90, 70	10.8%	10.7%	0.1%	89.2%	99.9%	97.1%	0.7ms
16	0.5s±12ms	0.5s±12ms	90, 70	18.1%	17.9%	0.2%	81.9%	99.7%	67.8%	0.7ms
36	0.5s±12ms	0.5s±12ms	90, 70	16.0%	15.7%	0.3%	84.0%	99.3%	56.2%	0.7ms
64	0.5s±12ms	0.5s±12ms	90, 70	16.2%	15.9%	0.3%	83.8%	99.1%	50.6%	0.7ms
100	0.5s±12ms	0.5s±12ms	90, 70	15.8%	15.5%	0.3%	84.2%	99.0%	47.2%	0.7ms
4	1s±17ms	1s±16ms	90, 70	11.8%	11.1%	0.7%	88.2%	99.2%	96.9%	0.6ms
16	1s±17ms	1s±16ms	90, 70	22.9%	20.4%	2.5%	77.1%	95.0%	68.1%	0.6ms
36	1s±17ms	1s±16ms	90, 70	21.4%	18.5%	2.9%	78.6%	93.0%	56.5%	0.6ms
64	1s±17ms	1s±16ms	90, 70	21.4%	18.5%	2.9%	78.6%	91.7%	50.7%	0.6ms
100	1s±17ms	1s±16ms	90, 70	21.1%	18.1%	3.0%	78.9%	90.7%	47.2%	0.6ms
4	2s±25ms	2s±22ms	90, 70	14.7%	12.5%	2.2%	85.3%	97.5%	96.8%	0.6ms
16	2s±25ms	2s±22ms	90, 70	35.6%	27.0%	8.7%	64.4%	82.7%	68.4%	0.6ms
36	2s±25ms	2s±22ms	90, 70	35.3%	25.6%	9.7%	64.7%	76.1%	56.6%	0.6ms
64	2s±25ms	2s±22ms	90, 70	35.1%	25.3%	9.8%	64.9%	71.9%	50.5%	0.6ms
100	2s±25ms	2s±22ms	90, 70	34.4%	24.6%	9.8%	65.5%	69.4%	46.9%	0.6ms

Table 5.2.: Quaternion prediction with second data set (Data point in average every 10ms)

5.4. Results Prediction Image Tiles

5.4.1. Results of First Data Set (Data point in average every 100ms)

#Tiles	Prediction Train	Prediction Eval	FOV angle Train, Eval	#Errors	#FP	#FN	Accuracy	Recall	#Predicted	Time
4	0.5s±4ms	0.5s±5ms	70, 70	8.5%	6.5%	2.0%	91.5%	97.2%	76.7%	1.0ms
16	0.5s±4ms	0.5s±5ms	70, 70	4.6%	3.0%	1.6%	95.4%	95.2%	35.3%	1.0ms
36	0.5s±4ms	0.5s±5ms	70, 70	4.6%	2.4%	2.2%	95.4%	93.3%	33.0%	1.0ms
64	0.5s±4ms	0.5s±5ms	70, 70	3.5%	1.7%	1.8%	96.5%	93.3%	27.0%	1.0ms
100	0.5s±4ms	0.5s±5ms	70, 70	4.5%	2.1%	2.4%	95.5%	90.1%	24.0%	0.9ms
4	1s±5ms	1s±7ms	70, 70	15.1%	10.7%	4.4%	84.9%	94.0%	78.7%	1.1ms
16	1s±5ms	1s±7ms	70, 70	7.2%	3.6%	3.6%	92.8%	89.4%	34.0%	1.0ms
36	1s±5ms	1s±7ms	70, 70	8.1%	3.7%	4.4%	91.9%	86.6%	32.2%	1.0ms
64	1s±5ms	1s±7ms	70, 70	5.5%	2.5%	3.0%	94.5%	89.0%	26.6%	1.0ms
100	1s±5ms	1s±7ms	70, 70	6.6%	3.1%	3.5%	93.4%	85.7%	24.0%	0.9ms
4	2s±7ms	2s±10ms	70, 70	19.2%	11.4%	7.8%	80.8%	89.2%	75.9%	1.0ms
16	2s±7ms	2s±10ms	70, 70	13.0%	7.2%	5.8%	87.0%	82.9%	35.2%	1.0ms
36	2s±7ms	2s±10ms	70, 70	12.3%	5.1%	7.2%	87.7%	78.1%	30.8%	1.0ms
64	2s±7ms	2s±10ms	70, 70	9.2%	3.9%	5.3%	90.8%	80.4%	25.6%	0.9ms
100	2s±7ms	2s±10ms	70, 70	10.0%	4.3%	5.7%	90.0%	76.6%	23.0%	0.9ms
4	0.5s±4ms	0.5s±5ms	90, 70	27.7%	27.7%	0.0%	72.3%	100.0%	100.0%	1.1ms
16	0.5s±4ms	0.5s±5ms	90, 70	25.8%	25.8%	0.04%	74.2%	99.9%	59.6%	1.0ms
36	0.5s±4ms	0.5s±5ms	90, 70	13.9%	13.8%	0.1%	86.1%	99.7%	46.5%	1.0ms
64	0.5s±4ms	0.5s±5ms	90, 70	12.7%	12.6%	0.1%	87.3%	99.6%	39.6%	1.0ms
100	0.5s±4ms	0.5s±5ms	90, 70	13.1%	12.9%	0.2%	86.9%	99.3%	37.1%	0.9ms
4	1s±5ms	1s±7ms	90, 70	27.7%	27.7%	0.0%	72.3%	100.0%	100.0%	1.0ms
16	1s±5ms	1s±7ms	90, 70	26.6%	26.1%	0.5%	73.4%	98.6%	59.6%	1.0ms
36	1s±5ms	1s±7ms	90, 70	14.5%	13.9%	0.6%	85.5%	98.3%	46.2%	1.0ms
64	1s±5ms	1s±7ms	90, 70	13.5%	12.9%	0.6%	86.5%	97.7%	39.3%	1.0ms
100	1s±5ms	1s±7ms	90, 70	13.7%	13.0%	0.7%	86.3%	97.2%	36.7%	0.9ms
4	2s±7ms	2s±10ms	90, 70	27.7%	27.7%	0.0%	72.3%	100.0%	100.0%	1.1ms
16	2s±7ms	2s±10ms	90, 70	29.2%	27.6%	1.6%	70.7%	95.2%	59.9%	1.0ms
36	2s±7ms	2s±10ms	90, 70	17.5%	15.5%	2.0%	82.5%	93.9%	46.3%	1.0ms
64	2s±7ms	2s±10ms	90, 70	16.0%	13.8%	2.2%	84.0%	92.0%	38.7%	1.0ms
100	2s±7ms	2s±10ms	90, 70	15.8%	13.7%	2.1%	84.2%	91.4%	36.0%	0.9ms

Table 5.3.: Tile prediction with first data set (Data point in average every 100ms)

5.4.2. Results of Second Data Set (Data point in average every 10ms)

#Tiles	Prediction Train	Prediction Eval	FOV angle Train, Eval	#Errors	#FP	#FN	Accuracy	Recall	#Predicted	Time
4	(10 ± 2)ms	(10 ± 2)ms	70, 70	2.5%	1.2%	1.3%	97.5%	98.6%	86.4%	1.2ms
16	(10 ± 2)ms	(10 ± 2)ms	70, 70	2.4%	1.2%	1.2%	97.6%	97.6%	50.1%	1.2ms
36	(10 ± 2)ms	(10 ± 2)ms	70, 70	2.5%	1.2%	1.3%	97.5%	96.9%	40.7%	1.2ms
64	(10 ± 2)ms	(10 ± 2)ms	70, 70	2.6%	1.3%	1.3%	97.4%	96.2%	35.0%	1.2ms
100	(10 ± 2)ms	(10 ± 2)ms	70, 70	2.8%	1.4%	1.4%	97.2%	95.7%	32.1%	1.1ms
4	(20 ± 2)ms	(20 ± 2)ms	70, 70	2.3%	1.0%	1.3%	97.7%	98.5%	86.1%	1.2ms
16	(20 ± 2)ms	(20 ± 2)ms	70, 70	2.6%	1.3%	1.3%	97.4%	97.5%	50.2%	1.2ms
36	(20 ± 2)ms	(20 ± 2)ms	70, 70	2.5%	1.3%	1.2%	97.5%	97.0%	40.8%	1.1ms
64	(20 ± 2)ms	(20 ± 2)ms	70, 70	2.6%	1.4%	1.2%	97.4%	96.5%	35.2%	1.2ms
100	(20 ± 2)ms	(20 ± 2)ms	70, 70	2.9%	1.5%	1.4%	97.1%	95.5%	32.1%	1.1ms
4	(51 ± 4)ms	(51 ± 3)ms	70, 70	2.6%	1.4%	1.2%	97.4%	98.7%	86.7%	1.2ms
16	(51 ± 4)ms	(51 ± 3)ms	70, 70	2.5%	1.3%	1.2%	97.5%	97.5%	50.1%	1.3ms
36	(51 ± 4)ms	(51 ± 3)ms	70, 70	2.6%	1.3%	1.3%	97.4%	96.9%	40.8%	1.2ms
64	(51 ± 4)ms	(51 ± 3)ms	70, 70	2.7%	1.4%	1.3%	97.3%	96.3%	35.1%	1.2ms
100	(51 ± 4)ms	(51 ± 3)ms	70, 70	2.9%	1.5%	1.4%	97.1%	95.7%	32.2%	1.1ms
4	0.1s±5ms	0.1s±5ms	70, 70	2.5%	1.4%	1.1%	97.5%	98.7%	86.8%	1.2ms
16	0.1s±5ms	0.1s±5ms	70, 70	2.6%	1.3%	1.3%	97.4%	97.4%	50.2%	1.2ms
36	0.1s±5ms	0.1s±5ms	70, 70	2.7%	1.3%	1.4%	97.3%	96.6%	40.8%	1.2ms
64	0.1s±5ms	0.1s±5ms	70, 70	2.8%	1.5%	1.3%	97.2%	96.3%	35.3%	1.2ms
100	0.1s±5ms	0.1s±5ms	70, 70	3.0%	1.6%	1.4%	97.0%	95.5%	32.2%	1.1ms
4	0.2s±7ms	0.2s±7ms	70, 70	3.2%	1.3%	1.9%	96.8%	97.8%	85.4%	1.3ms
16	0.2s±7ms	0.2s±7ms	70, 70	2.8%	1.5%	1.3%	97.2%	97.3%	50.3%	1.2ms
36	0.2s±7ms	0.2s±7ms	70, 70	3.1%	1.6%	1.5%	96.9%	96.4%	40.9%	1.2ms
64	0.2s±7ms	0.2s±7ms	70, 70	3.1%	1.6%	1.5%	96.9%	95.6%	35.1%	1.2ms
100	0.2s±7ms	0.2s±7ms	70, 70	3.4%	1.7%	1.7%	96.6%	94.8%	32.1%	1.1ms
4	0.5s±12ms	0.5s±12ms	70, 70	5.4%	2.8%	2.6%	94.6%	94.6%	86.6%	1.2ms
16	0.5s±12ms	0.5s±12ms	70, 70	6.0%	2.9%	3.1%	94.0%	93.7%	49.9%	1.2ms
36	0.5s±12ms	0.5s±12ms	70, 70	6.0%	2.9%	3.1%	94.0%	92.4%	40.6%	1.2ms
64	0.5s±12ms	0.5s±12ms	70, 70	5.8%	2.9%	2.9%	94.2%	91.7%	35.1%	1.2ms
100	0.5s±12ms	0.5s±12ms	70, 70	5.9%	2.9%	3.0%	94.1%	90.5%	31.9%	1.1ms
4	1s±17ms	1s±16ms	70, 70	11.0%	6.3%	4.7%	89.0%	94.6%	88.1%	1.2ms
16	1s±17ms	1s±16ms	70, 70	15.3%	8.2%	7.1%	84.7%	85.8%	51.2%	1.2ms
36	1s±17ms	1s±16ms	70, 70	15.2%	7.6%	7.6%	84.8%	81.4%	40.8%	1.2ms
64	1s±17ms	1s±16ms	70, 70	14.0%	6.8%	7.2%	86.0%	79.4%	34.6%	1.2ms
100	1s±17ms	1s±16ms	70, 70	13.8%	6.8%	7.0%	86.2%	78.1%	31.8%	1.1ms
4	2s±25ms	2s±22ms	70, 70	18.2%	10.7%	7.5%	81.8%	91.3%	89.7%	1.2ms
16	2s±25ms	2s±22ms	70, 70	31.1%	15.1%	16.0%	68.9%	68.1%	49.3%	1.2ms
36	2s±25ms	2s±22ms	70, 70	31.6%	14.9%	16.7%	68.4%	59.1%	39.1%	1.2ms
64	2s±25ms	2s±22ms	70, 70	29.5%	13.9%	15.6%	70.5%	55.5%	33.4%	1.2ms
100	2s±25ms	2s±22ms	70, 70	28.5%	12.8%	15.7%	71.5%	51.2%	29.2%	1.1ms
4	(10 ± 2)ms	(10 ± 2)ms	90, 70	11.2%	11.2%	0.04%	88.8%	100.0%	97.5%	1.2ms
16	(10 ± 2)ms	(10 ± 2)ms	90, 70	17.8%	17.8%	0.05%	82.2%	99.9%	67.8%	1.2ms
36	(10 ± 2)ms	(10 ± 2)ms	90, 70	15.7%	15.7%	0.1%	84.3%	99.9%	56.4%	1.3ms
64	(10 ± 2)ms	(10 ± 2)ms	90, 70	15.8%	15.7%	0.1%	84.2%	99.9%	50.8%	1.2ms
100	(10 ± 2)ms	(10 ± 2)ms	90, 70	15.5%	15.4%	0.1%	84.5%	99.8%	47.4%	1.2ms
4	(20 ± 2)ms	(20 ± 2)ms	90, 70	10.9%	10.9%	0.04%	89.1%	100.0%	97.3%	1.2ms
16	(20 ± 2)ms	(20 ± 2)ms	90, 70	17.8%	17.8%	0.05%	82.2%	99.9%	67.9%	1.1ms
36	(20 ± 2)ms	(20 ± 2)ms	90, 70	15.7%	15.7%	0.04%	84.3%	99.9%	56.4%	1.2ms
64	(20 ± 2)ms	(20 ± 2)ms	90, 70	15.8%	15.8%	0.05%	84.2%	99.9%	50.8%	1.3ms
100	(20 ± 2)ms	(20 ± 2)ms	90, 70	15.4%	15.4%	0.05%	84.6%	99.8%	47.4%	1.1ms
4	(51 ± 4)ms	(51 ± 3)ms	90, 70	11.1%	11.1%	0.04%	88.9%	100.0%	97.4%	1.2ms
16	(51 ± 4)ms	(51 ± 3)ms	90, 70	17.8%	17.7%	0.1%	82.2%	99.9%	67.8%	1.2ms
36	(51 ± 4)ms	(51 ± 3)ms	90, 70	15.7%	15.7%	0.04%	84.3%	99.9%	56.4%	1.2ms
64	(51 ± 4)ms	(51 ± 3)ms	90, 70	15.8%	15.7%	0.1%	84.2%	99.8%	50.8%	1.3ms
100	(51 ± 4)ms	(51 ± 3)ms	90, 70	15.5%	15.4%	0.05%	84.5%	99.8%	47.5%	1.1ms
4	0.1s±5ms	0.1s±5ms	90, 70	11.1%	11.1%	0.04%	88.9%	100.0%	97.5%	1.2ms
16	0.1s±5ms	0.1s±5ms	90, 70	17.7%	17.6%	0.1%	82.3%	99.9%	67.7%	1.2ms
36	0.1s±5ms	0.1s±5ms	90, 70	15.9%	15.8%	0.1%	84.1%	99.9%	56.5%	1.2ms
64	0.1s±5ms	0.1s±5ms	90, 70	15.8%	15.7%	0.1%	84.2%	99.8%	50.8%	1.3ms
100	0.1s±5ms	0.1s±5ms	90, 70	15.5%	15.4%	0.1%	84.5%	99.8%	47.5%	1.1ms
4	0.2s±7ms	0.2s±7ms	90, 70	11.1%	11.1%	0.03%	88.9%	100.0%	97.5%	1.2ms
16	0.2s±7ms	0.2s±7ms	90, 70	17.7%	17.6%	0.1%	82.3%	99.8%	67.7%	1.2ms
36	0.2s±7ms	0.2s±7ms	90, 70	15.7%	15.6%	0.1%	84.3%	99.8%	56.3%	1.2ms
64	0.2s±7ms	0.2s±7ms	90, 70	16.0%	15.9%	0.1%	84.0%	99.8%	50.9%	1.3ms
100	0.2s±7ms	0.2s±7ms	90, 70	15.6%	15.5%	0.1%	84.4%	99.8%	47.5%	1.1ms
4	0.5s±12ms	0.5s±12ms	90, 70	10.7%	10.6%	0.1%	89.3%	99.9%	96.9%	1.2ms
16	0.5s±12ms	0.5s±12ms	90, 70	18.2%	18.0%	0.2%	81.8%	99.6%	67.9%	1.2ms
36	0.5s±12ms	0.5s±12ms	90, 70	16.2%	16.0%	0.2%	83.8%	99.5%	56.6%	1.2ms
64	0.5s±12ms	0.5s±12ms	90, 70	16.3%	16.1%	0.2%	83.7%	99.4%	51.0%	1.2ms
100	0.5s±12ms	0.5s±12ms	90, 70	15.9%	15.7%	0.2%	84.0%	99.3%	47.5%	1.1ms
4	1s±17ms	1s±16ms	90, 70	12.4%	11.9%	0.5%	87.6%	99.4%	97.8%	1.2ms
16	1s±17ms	1s±16ms	90, 70	22.6%	20.5%	2.1%	77.4%	95.7%	68.5%	1.2ms
36	1s±17ms	1s±16ms	90, 70	20.8%	18.4%	2.4%	79.2%	94.0%	56.7%	1.2ms
64	1s±17ms	1s±16ms	90, 70	20.8%	18.3%	2.5%	79.2%	92.8%	50.8%	1.2ms
100	1s±17ms	1s±16ms	90, 70	20.0%	17.6%	2.4%	80.0%	92.4%	47.2%	1.1ms
4	2s±25ms	2s±22ms	90, 70	14.2%	12.9%	1.3%	85.8%	98.5%	98.0%	1.3ms
16	2s±25ms	2s±22ms	90, 70	35.0%	26.6%	8.4%	65.0%	83.2%	68.3%	1.2ms
36	2s±25ms	2s±22ms	90, 70	34.4%	25.1%	9.3%	65.6%	77.2%	56.7%	1.2ms
64	2s±25ms	2s±22ms	90, 70	33.9%	24.4%	9.5%	66.1%	72.9%	50.0%	1.2ms
100	2s±25ms	2s±22ms	90, 70	33.7%	24.1%	9.5%	66.3%	70.3%	46.7%	1.1ms

Table 5.4.: Tile prediction with second data set (Data point in average every 10ms)

5.5. Discussion

Overall, the results are as expected. With increasing prediction time, the errors increase. If more tiles are used, the bandwidth savings increase, because less additional image material is transmitted. In this section we discuss the results.

5.5.1. Videos

The selection of videos is important for the two studies that we describe in section 4.3. If the videos have a focus in one direction, some tiles are used much more often than others. One of the test videos "The invisible man" [18], a short movie, has this problem. The actions in the video are limited mainly in one direction. In the second video, a drone flight [19], there were multiple interesting spots spread over the whole video. In a larger study one would have to pay attention to find various types of videos. Additionally, more videos would have to be used to prevent the neural networks from training the videos instead of a generic solution. In future work it could be interesting to take the data of the video itself into account.

5.5.2. Data Sets

Figure 5.3 shows that some tiles were used more often than others. If a tile is darker it was looked at more often in the studies then the lighter ones. This can have a impact on the training of the neural networks.

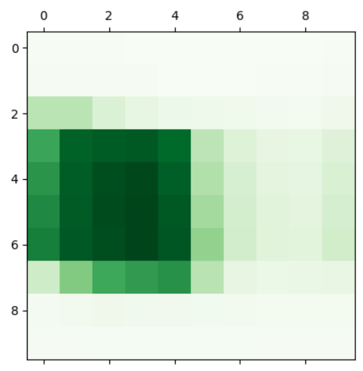


Figure 5.3.: Distribution of looked at tiles (100 tiles, first data set), darker tiles were viewed more often

The second data set with an average interval of 10ms has more data points. But when looking at the distribution of the expected tiles we see a bias again. Figure 5.4 shows that some tiles were used more often than others. Again the tiles that are darker were looked at more often in the studies then the lighter ones. This leads to a bias for single tiles when training the networks. For further studies it would be useful to use data with less bias. This could again bring significant changes into play.

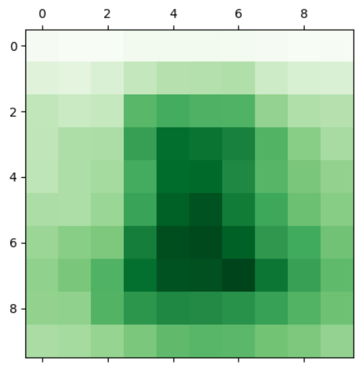


Figure 5.4.: Distribution of looked at tiles (100 tiles, second data set), darker tiles were viewed more often

5.5.3. Prediction

The prediction times are far below the requirements. Predicting the quaternions (section 3.2.2) takes less time than predicting the tiles (section 3.2.3). The calculations for the prediction on the laptop section 4.1 takes 0.4ms to 0.7ms. These numbers are expected to be lower when running the evaluation on a server, because servers usually have more resources. The numbers are far below the limit of the targeted value of 10ms. This gives the possibility to extend the network in future studies and thus to make better predictions.

Also the prediction of the tiles (section 3.2.2) is below the target value of 10ms [13]. The calculations take between 0.9ms and 1.3ms. Therefore, it is possible to extend the neural network in this variant as well. The measurements suggest that the latency is constant for the prediction of the tiles and quaternions. The real time calculations allow to use these two algorithms also for live streaming.

Due to false negative predictions it could happen, that tiles in the FOV are missing and therefore lower the QoE for the user. The countermeasure to lower the risk for false negatives is increasing the FOV angle for the training of the neural networks.

5.5.4. Bandwidth Savings (# Predicted)

The number of predicted (#Predicted) shows us how many tiles of the total number of tiles would be streamed in a streaming scenario. The calculations for the bandwidth savings are described in Section 5.2.8. The bandwidth savings for 100 tiles are between 53% and 77%. The two approaches of neural networks show similar results.

The bandwidth savings are higher for the first data set with an average 100ms interval. Some probable causes are discussed in section 5.5.2. The bandwidth savings for this approach are between 63% and 77%. For the second data set with more data points, the bandwidth savings for 100 tiles are between 53% and 68%. For both data sets the bandwidth savings decrease when predicting with a higher FOV angle. This is in line with expectations, because with a higher FOV angle more tiles are needed and therefore less bandwidth is saved. In future work it makes sense to find out whether the bandwidth savings will decrease further with larger and more diverse data sets.

5.5.5. Comparison

Both types of neural networks (predicting the quaternion and predicting the image tiles) show a total error of up to 35% at a prediction time of 2s, but it should be noted that in the FOV range there is only a maximum error of 16.7%. The total error is thus close to a solution with linear regression(accuracy of 72%) [16]. The error of interest to the user (false negatives 16.7%) is much lower. Both neural networks show similar results. The first data set with an average interval of 100ms shows slightly better results in terms of errors. When training with a higher FOV angle, the false negatives are 2.6% with a prediction time of 2s. In the second data set the false negatives are up to 10% with a prediction time of 2s. For short term predictions up to 0.5s the neural networks have less then 1% false negative errors. The recall value is used if there should be as few false negatives as possible. The networks that were trained with a 90° and evaluated with a 70° FOV anlge show that the recall value is higher than if you train and evaluate with the same FOV angle. When the training FOV angle is 20° higher the first data set shows recall values over 88% even with a prediction time of 2s. When the training FOV angle is 20° higher the second data set shows values over 70%. The accuracy stays for both data sets over 65%. With a larger data set it would be good to find out if the errors increase or stay constant on a certain level.

5.5.6. Madgwick Filter

The Madgwick filter is not accurate enough when using it for calculating the view matrix for displaying the video, because when holding the phone in a stable position the image shakes. The quaternions are unstable even with the low-pass filter. The image fluctuates when the phone is held still due to these instabilities. Additionally, a slight delay is noticeable when turning the smart phone. This is due to the low-pass filter. To make a better prediction about the FOV, it may be beneficial to use the orientation estimation implemented in Android which uses an extended Kalman filter [40]. The application is implemented in a way that allows to swap out the orientation estimation easily. In addition, two threads could be used to calculate the orientation estimation and the rendering of the image separately.

5.5.7. Neural Networks

Both neural networks can still be optimized. Since the computation times are far below the required latencies, the neural networks can be enlarged or other components can be added. The calculation time, QoE and bandwidth savings must be traded off. In a larger study with more diverse data, it would also be interesting to find out whether one network shows significantly better results than the other.

In this thesis the hyperparameters (number of hidden layers, number of nodes in the layers, ...) were chosen by trial and error. This means that there was no algorithm to improve the neural networks. This could be improved in further implementations with a grid or random search approach to tweak the hyperparameters.

6. Conclusion

6.1. Summary

Streaming 360° videos is a challenge as it requires a high bandwidth. In addition there are latency requirements. If the latency is too high, a delay between the movement of the mobile phone and the shifting of the field of view is noticeable and reduces the QoE.

In our work we follow the approach that viewers of a 360° video only need to receive the parts of the image that are within their FOV. To know in which direction they are looking we have implemented a Madgwick filter that calculates the orientation of a smart phone. Subsequently, we have collected, with the help of two videos, data for the training and evaluation of two neural networks in two small studies to predict the movements of the viewing device.

The results show that the selection of videos is important because they have an influence on the data sets. In both data sets we can see a bias of tiles, because one of the videos had the focus of action mainly in one direction. The experiments showed that the Madgwick filter is not suitable to use it for changing the view matrix. The filter is to instable and should be replaced with a different orientation estimation.

The results of the prediction time and the error rates seem promising. The prediction time is below the target value of 10ms. The quaternions are predicted in 0.4ms to 0.7ms and the tiles in 0.9ms and 1.3ms. The error rate at 2s prediction time is under 10%. For short-term predictions (under 0.5s) the false negative error is less than 1%. Our experiments suggest that our approach could save more than 50% of the bandwidth with the respective error while maintaining a good user experience.

These results need confirmation in a larger setting, as the current data set is small with limited diversity.

6.2. Future Work

The first future work suggestion is a prototype, that streams 360° videos from an edge server to an Android client. It would be interesting to tweak the hyperparameters of the neural networks with a grid or random search instead of a trial and error approach. This would either improve the performance or the accuracy of the neural networks. Further it is important to confirm the results in bigger settings with more participants and more various types of 360° videos. This would help to validate if a generic approach for streaming 360° videos is feasible. An additional possibility is to extend the input of the neural networks with the video material. This could improve the accuracy of the neural networks. The last suggestion is to exchange the Madgwick filter with another orientation estimation algorithm like a Kalman filter. Especially for a prototype it is important that the orientation estimation is as accurate as possible.

Bibliography

[1] L. Peñaranda, L. Velho, and L. Sacht, “Real-time correction of panoramic images using hyperbolic Möbius transformations,” *Journal of Real-Time Image Processing*, Dec. 2018. [Online]. Available: http://link.springer.com/10.1007/s11554-015-0502-x

[2] opengl tutorial, “Tutorial 3 : Matrices.” [Online]. Available: http://www.opengl-tutorial.org/beginners-tutorials/tutorial-3-matrices/

[3] Autodesk, “UVW Map Modifier,” Dec. 2014. [Online]. Available: https://knowledge.autodesk.com/support/3ds-max/learn-explore/caas/CloudHelp/cloudhelp/2015/ENU/3DSMax/files/GUID-78327298-4741-470C-848D-4C3618B18FCA-htm.html

[4] J. Kuipers, “Quaternions and Rotation Sequences,” Sep. 1999. [Online]. Available: https://www.emis.de/proceedings/Varna/vol1/GEOM09.pdf

[5] V. Zhou, “Machine Learning for Beginners: An Introduction to Neural Networks,” Mar. 2019. [Online]. Available: https://towardsdatascience.com/machine-learning-for-beginners-an-introduction-to-neural-networks-d49f22d238f9

[6] S. O. H. Madgwick, “An efficient orientation filter for inertial and inertial/magnetic sensor arrays.” [Online]. Available: https://www.x-io.co.uk/res/doc/madgwick_internal_report.pdf

[7] A. Yurtman, B. Barshan, and B. Fidan, “Activity Recognition Invariant to Wearable Sensor Unit Orientation Using Differential Rotational Transformations Represented by Quaternions,” *Sensors*, Aug. 2018. [Online]. Available: http://www.mdpi.com/1424-8220/18/8/2725

[8] “Accelerometer.” [Online]. Available: https://www.mathworks.com/help/supportpkg/android/ref/accelerometer.html

[9] K. Lohmann, P. Luschi, and G. Hays, “Goal navigation and island-finding in sea turtles,” *Journal of Experimental Marine Biology and Ecology*, Mar. 2008. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0022098107005783

[10] Wikipedia, “Right-hand rule.” [Online]. Available: https://en.wikipedia.org/wiki/Right-hand_rule

[11] B. Ayrey, “Introducing Facebook 360 For Gear VR.” [Online]. Available: https://about.fb.com/news/2017/03/introducing-facebook-360-for-gear-vr/

[12] V. Staff, “Vimeo 360: A home for immersive storytelling.” [Online]. Available: https://vimeo.com/blog/post/introducing-vimeo-360/

[13] S. Mangiante, G. Klas, A. Navon, Z. GuanHua, J. Ran, and M. D. Silva, “VR is on the Edge: How to Deliver 360^{circ} Videos in Mobile Networks,” in *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network - VR/AR Network ’17*. Los Angeles, CA, USA: ACM Press, 2017, pp. 30–35. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3097895.3097901

[14] Verizon, “Customers in Chicago and Minneapolis are first in the world to get 5G-enabled smartphones connected to a 5G network.” [Online]. Available: https://www.verizon.com/about/news/customers-chicago-and-minneapolis-are-first-world-get-5g-enabled-smartphones-connected-5g

[15] GSMA, “5G MEC – Based Cloud Game Innovation Practice.” [Online]. Available: https://www.gsma.com/futurenetworks/wiki/5g-mec-based-cloud-game-innovation-practice/

[16] F. Qian, L. Ji, B. Han, and V. Gopalakrishnan, “Optimizing 360 video delivery over cellular networks,” in *Proceedings of the 5th Workshop on All Things Cellular Operations, Applications and Challenges - ATC ’16*. New York City, New York: ACM Press, 2016, pp. 1–6. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2980055.2980056

[17] M. Hosseini and V. Swaminathan, “Adaptive 360 VR Video Streaming based on MPEG-DASH SRD,” *arXiv:1701.06509 [cs]*, Jan. 2017, arXiv: 1701.06509. [Online]. Available: http://arxiv.org/abs/1701.06509

[18] H. Keijzer, “The Invisible Man - 360^{circ} VR Short Film,” 2017. [Online]. Available: https://vimeo.com/207495856

[19] mettle, “Ayutthaya - needs stabilization and horizon correction | 360/VR Master Series | Free Download,” 2017. [Online]. Available: https://vimeo.com/214402865

[20] T. El-Ganainy and M. Hefeeda, “Streaming Virtual Reality Content,” Dec. 2016.

[21] L. Euler, “General formulas for any translation of rigid bodies.” [Online]. Available: http://www.17centurymaths.com/contents/euler/e478tr.pdf

[22] E. M. Jones and P. Fjeld, “Gimbal Angles, Gimbal Lock, and a Fourth Gimbal for Christmas,” 2000. [Online]. Available: https://www.hq.nasa.gov/alsj/gimbals.html

[23] D. J. C. MacKay, “Information Theory, Inference, and Learning Algorithms.”

[24] J. Brownlee, “A Gentle Introduction to the Rectified Linear Unit (ReLU),” Aug. 2019. [Online]. Available: https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/

[25] G. Ognjanovski, “Everything you need to know about Neural Networks and Backpropagation — Machine Learning Easy and Fun,” Jan. 2019. [Online]. Available: https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

[26] Tensorflow, “tf.keras.losses.BinaryCrossentropy,” Feb. 2020. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/losses/BinaryCrossentropy

[27] Tham, “Moving Average Filter.” [Online]. Available: https://web.archive.org/web/20100329134203/http://lorien.ncl.ac.uk/ming/filter/filmav.htm

[28] ——, “Exponentially Weighted Moving Average Filter.” [Online]. Available: https://web.archive.org/web/20100329135531/http://lorien.ncl.ac.uk/ming/filter/filewma.htm

[29] J. O. Smith, “Trapezoidal Rule,” Jan. 2020. [Online]. Available: https://ccrma.stanford.edu/~jos/pasp/Trapezoidal_Rule.html

[30] M. Ben-Ari, “A Tutorial on Euler Angles and Quaternions,” p. 22.

[31] E. W. Weisstein, “Spherical Coordinates.” [Online]. Available: http://mathworld.wolfram.com/SphericalCoordinates.html

[32] F. Frigeni, “Approximating dynamic models of industrial robots with neural networks.” [Online]. Available: https://towardsdatascience.com/approximating-dynamic-models-of-industrial-robots-with-neural-networks-2474d1a2eecd

[33] “Tensorflow tutorials,” Jan. 2020. [Online]. Available: https://www.tensorflow.org/tutorials

[34] Android, “Developer Guides,” Dec. 2019. [Online]. Available: https://developer.android.com/guide/index.html

[35] Python, “Python 3.6.” [Online]. Available: https://python.org

[36] JetBrains, “JetBrains.” [Online]. Available: jetbrains.com

[37] P. LaFayette, “360 Video Player for Android.” [Online]. Available: https://github.com/fbsamples/360-video-player-for-android

[38] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke,

Yuan Yu, and Xiaoqiang Zheng, *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*, 2015. [Online]. Available: https://www.tensorflow.org/

[39] S. Ghoneim, “Accuracy, Recall, Precision, F-Score & Specificity, which to optimize on?” Apr. 2019. [Online]. Available: https://towardsdatascience.com/accuracy-recall-precision-f-score-specificity-which-to-optimize-on-867d3f11124

[40] Android, “Position sensors,” Dec. 2019. [Online]. Available: https://developer.android.com/guide/topics/sensors/sensors_position#sensors-pos-orient

A. Git repositories

- https://gitlab.com/cnoetzli/sensorprediction
- https://gitlab.com/cnoetzli/neuralnetwork