Technische Universität Berlin

Audio Communication Group

Masterthesis

A Mobile 360° Audio and Video Rendering and Streaming Application

---

author	Marian Lesser
	[redacted]
supervisor	Prof. Dr. Stefan Weinzierl
	Markus Hädrich, M. A.
date	December 1, 2018

Contents

Declaration of Authorship . . . . . **III**

Abstract . . . . . **IV**

List of Figures . . . . . **VI**

List of Acronyms . . . . . **VII**

1 Introduction **1**

2 Motion Tracked Binaural system **4**

2.1 Spatial Hearing 5

2.2 Spatial Recording and Reproduction Systems 6

2.2.1 Binaural Technology 7

2.2.2 Artificial Head Technology 7

2.2.3 Binaural Synthesis 8

2.3 Discretisation and Interpolation 8

2.3.1 Sectoral Interpolation 9

2.3.2 Linear Interpolation 10

2.3.3 Separation in Frequency Range 11

2.3.4 Fixed Microphones 12

2.3.5 Linear Sectoral Interpolation 12

2.3.6 Interpolation in Frequency Range 12

2.4 Head Movement 13

3 Visual Computing **15**

3.1 Human Information Processing 16

3.2 Visual Cognition 17

3.2.1 Stereopsis 18

3.2.2 Motion Cognition 19

3.3 Displays 20

3.3.1 Head Mounted Displays 20

3.3.2 Retina Display 23

3.4 Latency 23

4 Development **25**
4.1 Streaming Service 27
4.1.1 Network Protocols 28
4.1.2 Web Server 31
4.2 iOS Application 33
4.2.1 Frameworks 37
4.2.2 File Download 41
4.2.3 Head-dependent Control Unit 43
4.2.4 Bluetooth Low Energy 48
4.2.5 Video Processing 51
4.2.6 Audio Processing 57
4.3 Discussion 62

5 Conclusion **66**

Appendix . . . . . **i**

References . . . . . **iv**

Declaration of Authorship

I hereby declare that the thesis submitted is my own unaided work. All direct or indirect sources used are acknowledged as references.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered into a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This paper was not previously presented to another examination board and has not been published.

Berlin. December 1, 2018

Abstract

The acoustic information in a room varies depending on the specific location. Virtual simulations of different conditions have been developed in recent decades. In 2004, Algazi et al. presented a Motion Tracked Binaural (MTB) System. It uses certain microphones to record different sources, e.g. an orchestra. The array of microphones is arranged in a circle and allows to combine different sources in a virtual way with the support of a head tracker. An MTB algorithm was developed by Sebastian Roos, at the Technical University of Berlin (TUB), for a PC based system (Roos, 2011). The task of the paper at hand is the conception and implementation of a binaural based 360° audio and video renderer, tracked by the motion of a mobile device.

An Application Programming Interface (API) is connected with an external head tracker or internal sensors. The mobile application is an interactive demonstration between a 360° audio and video signal. It could improve the immersion of the recipient through the combination of visual and binaural cues (Dörner et al., 2013, 13-15). The aim is the implementation of a mobile iOS application for a simultaneous presentation of 360° video streams and MTB audio recordings. It allows to convey a simulated spatial impression of a recorded event through the display of the mobile device to the media consumer.

List of Figures

1 Underlying principal of the motion-tracked binaural sound 4

2 Head related coordinate system 5

3 Directional and indirection components of HRTF 8

4 Sampling example for ear intermediate position 9

5 Linear interpolation for 8 microphones 11

6 Model of the Human Processor 17

7 Normal stereopsis, and stereo display manipulation 19

8 Field of view of monocular and binocular displays 21

9 Optical concept of direct view display 22

10 Scheme for total latency 24

11 ISO/OSI model linked to the AVR 26

12 Output Transcoding Process in FFMPEG 28

13 HLS basic configuration 30

14 Internetwork Operating System (iOS) Device Layer 34

15 AVR storyboard 36

16 Flowchart of the AVR 37

17 PickerView for downloading 42

18 Selection of remote control 44

19 iPhone device orientation 45

20 Flowchart of the sensor software driver 48

21 Bluetooth Low Energy protocol stack 49

22 Alert States for an External Motion Sensor 50

23 Flowchart of VideoProcessing 52

24 Frame processing in a SceneKit renderer 53

25 Landscape and Stereoscopic View 56

26 Amplifier configuration of electret capsules (compare Sennheiser (2018)) 58

27 MTB Array with its indices and azimuth 59

28 Flowchart of AudioProcessing 62

List of Acronyms

API Application Programming Interface

AR Augmented Reality

ATT Attribute Profile

AVR Audio and Video Rendering Application

BLE Bluetooth Low Energy

BRIR Binaural Room Impulse Response

CPU Central Processing Unit

DSP Digital Signal Processing

DFT Discrete Fourier Transformation

FFT Fast Fourier Transformation

FTP File Transfer Protocol

FOV Field of View

GATT Generic Attribute Profile

GAP Generic Access Profile

GFSK Gaussian Frequency Key Shiftings

GPU Graphics Processing Unit

GUI Graphical User Interface

HCI Host-Controller-Interface

HRTF Head-Related Transfer Function

HMD Head Mounted Display

HLS HTTP Live Streaming

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secure

I2C Inter Integrated Circuit

IDE Integrated Development Environment

ILD Interaural Level Difference

iOS Internetwork Operating System

IP Internet Protocol

ISM Industrial, Scientific and Medical Band

ISO International Standards Organisation

ITD Interaural Time Difference

L2CAP Logical Link Control And Application Protocol

LCD Liquid Crystal Display

LED Light Emitting Diode

MPEG Moving Picture Experts Group

MTB Motion Tracked Binaural

OLED Organic Light Emitting Diode

OS Operating System

OSI Open Systems Interconnect

PPI Pixel Per Inch

RGB Red Green Blue

RTP Real Time Transport Protocol

RTCP Real Time Control Protocol

RTMP Real Time Messaging Protocol

SDK Software Development Kit

STFT Short Time Fourier Transform

SMP Security Manager Protocol

SSH Secure Shell

SSL Secure Socket Layer

TLS Transport Layer Security

TUB Technical University of Berlin

UDP User Datagram Protocol

URL Uniform Resource Locator

USB Universal Serial Bus

UUID Universally Unique Identifier

VR Virtual Reality

VC Visual Computing

WLAN Wireless Local Area Network

1 Introduction

The prevalence of mobile devices and communication technologies has increased in the last decade. Societies are markedly influenced by those technologies. In case of Apple's iPhone, the marketplace increased from 1.39 mio. in 2007, to 211,88 mio. devices in 2016 (Nier, 2017). The iPhone has been Apple's most profitable device for the last 10 years, ahead of iTunes Software, Mac Computers and iPads (Loesche, 2017). It shows the marketplace is still growing for mobile devices. Technology used in the devices is in continual development. The Central Processing Unit (CPU), Graphics Processing Unit (GPU), power supply, displays, and software are more powerful than ever before. These evolutions suggest that mobility and the 'connected lifestyle' will continue to evolve too.

Augmented Reality (AR) and Virtual Reality (VR) are future innovation in research and industry as well. Complex consumer systems are able to realise, because of the fast development of hardware, like displays, interaction devices and tracking systems. Collateral to the engineering development, the costs decreased (Dörner et al., 2013, p. 8-12). VR is quite an old topic. The early simulation was founded by Edwin Link in the 1920's (Rosen, 2008).

Furthermore, the link between VR and MTB seems credible and worth an analysis. It could also be possible to connect a virtual sound.

Mobile applications and hardware development of smartphones have gained more importance recently. These facts make it possible to calculate with highly complex algorithms in real time on mobile devices. The good standard of streaming services (e.g. YouTube) and media technologies form a well productive basis in regards to the developments of audio content. An example of this new approach of media consumption is shown by the 360° camera. Even digital signal processing is a high quality standard for analysing audio waveforms. Sebastian Roos developed an multiple channel engine for rendering an microphone array in 2011 (Roos, 2011).

In relation to these developments the question arises, if a development of a mobile application for audio and video rendering with less immersion is possible. This could then allow the reproduction of a real environment by using VR glasses.

How is it possible to render streaming or local channels of an MTB microphone array to stereo signals synchronised to a 360° video? As well as through high latencies of bluetooth or Wireless Local Area Network (WLAN) transmission. For the application, it is necessary to implement a service that provides the MTB samples via a broadcast or streaming protocol. The Real Time Transport Protocol (RTP), a continuous transmission of audio, and video contents, was standardized in 2003 (Werner, 2005). The mobile application should have the ability to downstream multichannel formats and render the files depending on the direction of view by the usage of the MTB render algorithm (Algazi et al., 2004).

For simplification and best performances is it useful to work with the latest device in this paper at hand.

Various steps had to be followed during the development of this topic. The first approach is the development according to the International Standards Organisation (ISO) / Open Systems Interconnect (OSI) layer model. This is necessary because different ways of processing data are required and data must be provided across layers. With this model in mind, a backend server has to be implemented which offers encoded audio and video content based on a protocol for streaming or downloading.

In addition to the server solution, it is also necessary to integrate an external sensor to control the movement. This sensor should be connected via bluetooth, since the WLAN data rate can be relieved if two wireless technologies are used.

The external sensor is based on a gyroscope from Bosch, which is connected to a bluetooth module via a microcontroller. Therefore, it is used to transmit the data from the sensor to the transmitter.

The development of the application uses numerous frameworks, which are offered by Apple, therefore, the access to different hardware components can be guaranteed. For example, the `Audio and Video Rendering Application (AVR)` uses the `CoreBluetooth` framework to establish the connection to the external sensor.

The internal motion sensor, to ensure three dimensional motion oriented audio and video processing without a tracker. With the internal sensors the video processing has to be rendered, because the viewing angle can change in three planes. Motion control, on the other hand, only needs to be ensured on the horizontal plane.

The algorithm of Sebastian Roos should be integrated into the application as a rendering engine to ensure stable audio processing. Problems can occur when wrapping the C++ classes. Furthermore, the application must be extended by a rendering engine for video processing, in order to reproduce a copy of the real environment.

The data to be processed is offered by the server or locally as a demonstration and its output depends on the viewing direction. Therefore, the audio and video must be connected to the motion control. A double video output is necessary, so one can freely simulate the environment immersion with VR glasses.

2 Motion Tracked Binaural system

Algazi et al. presented the MTB system in 2004. MTB is an annular microphone array embedded in a sound reflecting spherical body, with an average head diameter. Those microphones are located at different angles and originate from the source. An MTB array could have 8, 16, 24, 36, or more channels. An MTB recording system simulates a partial human head. It contains the model of a sphere to simulate reflections and sound pressure levels. The pinna is not included. Microphones that are arranged in a circle and in the horizontal plane (see figure 1),are located at the widest point of the sphere. In contrast thereof, human ears are not exactly in the centre of a human head.

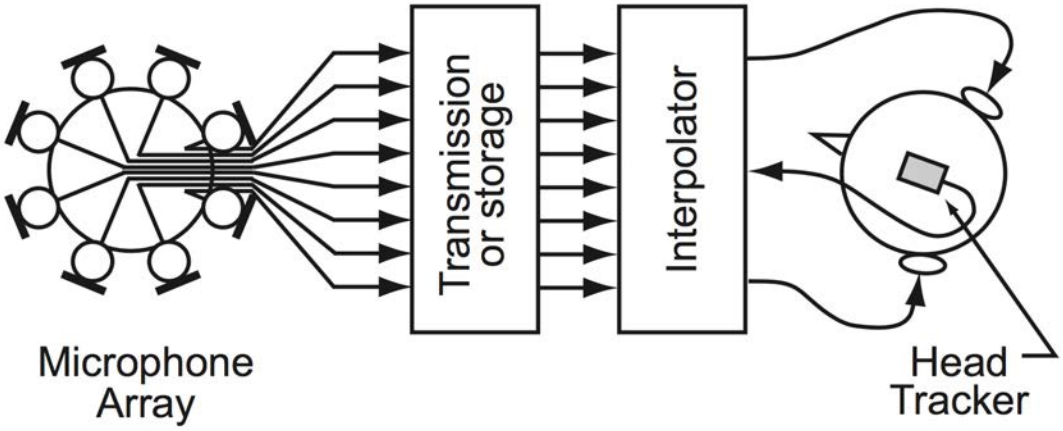

Figure 1: underlying principal of the motion-tracked binaural sound (Algazi et al., 2004)

The MTB system allows a multichannel audio recording with partial consideration of physical characteristics in binaural listening, e.g. directional flexibility, and sound pressure level. Pseudo-binaural audio signals can be generated under usage of a suitable rendering algorithm. The spherical head model creates Interaural Time Differences (ITDs), which help to calculate the source location. The main differences are the missing pinna and their shading, which prevent front and back localization (Roos, 2011, p. 18-29).

2.1 Spatial Hearing

The acoustic perception is within three orthogonal planes (see figure 2.1). Human ears measure the variations of physical quantities in the manner of a monaural or inter-aural method. These values are operated by the human brain. Monaural describes the changes in frequencies that are made by reflections on head, shoulder, body, and pinna. Their complex structure is combined with the characteristics of directional reflections, resonance, and shading. Interaural is the difference of level and run-time between the ear signals. It even provides important information regarding the direction in the horizontal plane (Weinzierl et al., 2008, p. 89-119).

A basic head (sphere) model of this kind is exampled in the Duplex Theory by Lord Rayleigh (Macpherson et al., 2002). It presents its usage of the ITD for low frequencies, until the critical frequency of approximately 1.5 kHz is reached. ITD is the phase difference between the left and right pinna (Strutt, 1904). The Interaural Level Difference (ILD) analyzes the range above 1.5 kHz. This is useful because the phase between the left and right ear canal is not precise (Macpherson et al., 2002).

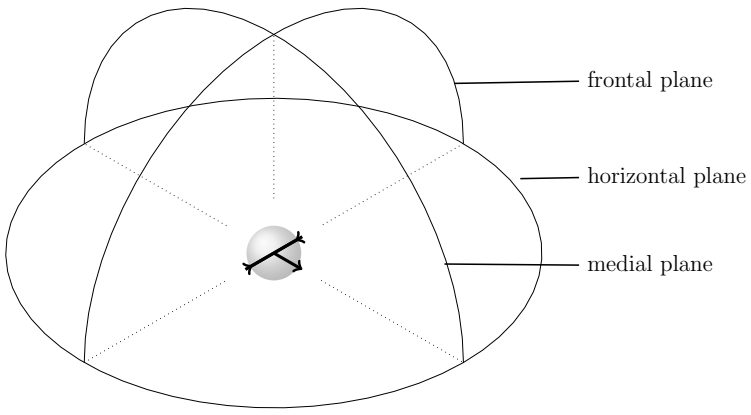

Figure 2: Head related coordinate system by Daniel et al. (2007)

Lord Rayleigh invented the mathematical calculation for the sphere based head simulations in 1904. The result of the sound pressure at the surface due to a plane wave is calculated in formula 1.

$$H(\mu,\theta) = \frac{1}{\mu^2}\sum_{m=0}^{\infty}\frac{(-i)^{m-1}(2m+1)P_m(\cos\theta)}{h'_m(\mu)} \tag{1}$$

$$\begin{aligned}
\mu &= kr = \tfrac{2\pi fr}{c} \\
r &= \text{sphere radius} \\
c &= \text{acoustic velocity} \\
\theta &= \text{angle of incidence} \\
P\ (m\text{-degree}) &= \text{Legendre polynomial} \\
h'\ (\text{order } m) &= \text{derivation of the} \\
&\quad \text{spheric Hankel function}
\end{aligned}$$

with $\mu = kr = \frac{2\pi fr}{c}$, sphere radius r, acoustic velocity c, angle of incidence θ, Legendre polynomial P (m-degree), and the derivation of the spheric Hankel function h' (order m) (Strutt, 1904).

2.2 Spatial Recording and Reproduction Systems

The human ear measures the variations of physical quantities in the characteristic of a monaural, or interaural manner.

These facts are the basics of directional, and acoustical simulations. Their manipulation are requirements for room acoustics, measurement systems, and even research. In addition to the MTB algorithm, further methods of headphone-bound spatial audio synthesis exist. Chapter 2.2.1, 2.2.2 and 2.2.3 will explain different spatial recording, and reproduction systems.

2.2.1 Binaural Technology

Binaural technology is an ear focused procedure which is based on synthetic ear signals. The signal is generated directly and separately between the two ear canals. It contains the influences of the transmitted sound field. A Binaural Room Impulse Response (BRIR) includes the Head-Related Transfer Function (HRTF) as well as room characteristics. BRIR enables the simulation of different sound sources, coupled with their characteristics. HRTF contains the interferences at body, head and pinna. It determines a signal of room impressions in the head phone (Dickreiter et al., 2014, p. 362). This enables reproduction with the pressure conditions at the respective auditory canal of the whole processing, from source to pinna (Møller, 1992, p. 171-218).

The synthesis is a broadly linear and time invariant system of the sound transmission between a random source and the human eardrum. Therefore, it is possible to characterize the time range by an impulse response and frequency range by a transfer function. This system is rather complicated to develop for a number of reasons. The individual HRTF of each person, the problematic interference from the microphone, the variations of head movements and its HRTF, the discrete convolution with low latencies as well as auditive cognition are all influenced by visual and non-auditive information (Weinzierl et al., 2008, p. 671-672).

2.2.2 Artificial Head Technology

A two channel recording system, based on binaural technology (Chapter 2.2.1), includes microphones at the beginning of each ear canal of a real, or model head. This allows a more accurate separation of different sound sources and their direction. The recording system shows a very good record simulation (Steickart, 1988, p. 314-316). It even filters the incoming acoustic signal by the HRTF. This transfer function contains the acoustical shading, deflection, delay, resonances, reflexions by torso, shoulder, head, pinna, and the ear channel (compare figure 2.2.2). HRTF is mostly influenced by the head and pinna. Furthermore, of major importance is the recording location within the ear channel for frequencies higher than 1 kHz. Microphones were located some millimetres within the ear channel,

because the individual differences of HRTFs are minimal (Weinzierl et al., 2008, p. 586).

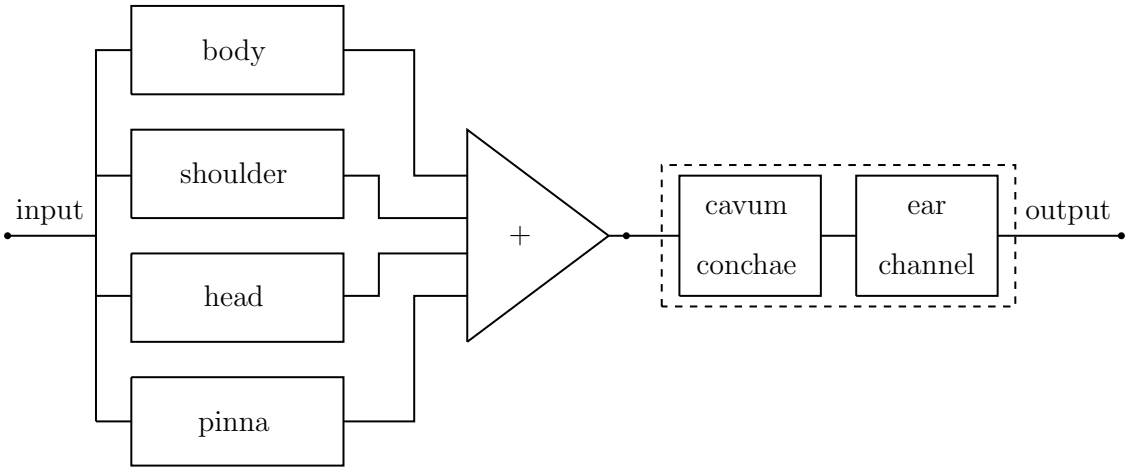

Figure 3: Directional and indirection components of HRTF (Weinzierl et al., 2008, p. 586)

2.2.3 Binaural Synthesis

An artificial head technology is located at the foundation of a computer based system of binaural technologies containing dynamic convolution. It provides numerous BRIR for potential sound sources. The convolution allows to shape a selected independent source with room and location information. Directions of sound sources can align by a connected head tracking system, which captures the head movement. It affords a spatial reproduction of sound sources with a huge plausibility rate (Lindau et al., 2007).

2.3 Discretisation and Interpolation

A discrete sampling is the result of limited microphones at the array. Signals for points at the array without microphones need to be approximated or interpolated

for a continuous process of ear signals depending on head movements (compare figure 2.3) (Roos, 2011, p.38).

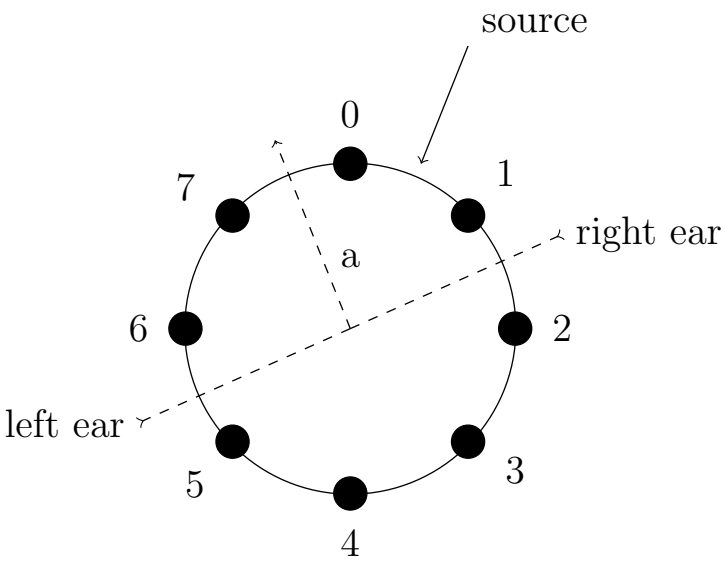

Figure 4: Sampling example for ear intermediate position by Algazi et al. (2004)

A similar problem is the auralization of sound sources by HRTF recordings, which are also based on a limited number of measuring points. HRTF recordings are based on echo free surroundings. MTB recordings are similar to HRTF datal, but horizontal microphone positions are simultaneously sampled, not successively. The differences are the microphone array, as it does not use impulse signals and spatial information is part of the signal as well. The MTB recordings are an HRTF convoluted with spatial and source information. Algazi et al. shows different interpolation processes which are suitable for MTB systems (chapter 2.3.1 - 2.3.6).

The reproduction will be the result of interpolation, continuity signals curved in each horizontal angle, and their characteristics (Algazi et al., 2004; Roos, 2011).

2.3.1 Sectoral Interpolation

A spheric surface will be divided in equal sectors, which keeps the angle of the generated system constant. The signal is switched to the next microphone output

at the sectoral border, and it then generates clicks, and timbre changes (Roos, 2011, p. 40-41). In 2009, Weinzierl et al. evaluated the minimal required solution of BRIR discretisation for different audio contents. They found out that 95% of experimental subjects could not analyse the most tender stimulus in the horizontal solution of 4°. A solution of 2° for critical audio contents was sufficient classified. The MTB system would need 180 microphones for this solution (Roos, 2011, p. 41).

2.3.2 Linear Interpolation

This procedure weighted adds the microphone signals with their inverse distance to the reference point of the ear. Signals with discontinuity could be prevented through an interpolation of the intermediate value of the two closest microphone signals (Roos, 2011, p. 41). According to Algazi et al. (2004), the interpolated signal is calculated in formula 2.

$$x(t) = \omega x_n + (1 - \omega) x_m \tag{2}$$

$$\begin{aligned} \omega &= \frac{\beta_n N}{2\pi} \\ x_n, x_m &= \text{nearby microphones} \end{aligned}$$

The cut-off range is calculated in formula 3. An essential difference of two adjacent microphones is the time delay from the sound path difference. The mixture of adjacent signals could be an effect of a comb filter by phase interference. A consequence will be a malfunction of localization characteristics, caused by the modulation effects of the head movement (Algazi et al., 2004, p 1148).

$$f_{max} = \frac{Nc}{8\pi r} \tag{3}$$

The number of microphones is proportional to the frequency of audible range. The frequency will be higher with the increase of microphones. A range up to

20 kHz needs 128 microphones. Figure 5 presents level spreading (left), and flaw effects (right) for linear interpolation for a spheric model with eight microphones calculated by Algazi et al. in 2004. Flaws are slim for a frequency below 1 kHz (spread line). Above the critical frequency flaws are shown at the points between nearby microphones and the frequency zeroing. Even the -3 dB flaw is shown through the solid line. This MTB system is bordered in the number of microphones (N). The result is a discrete sampling of the sound field.

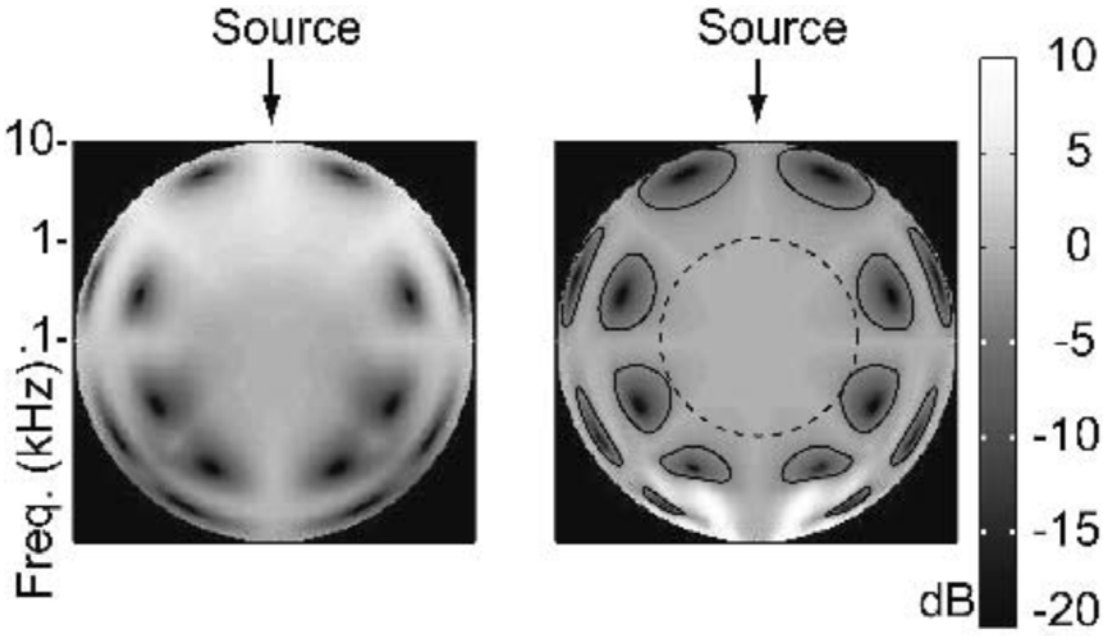

Figure 5: Linear interpolation for 8 microphones (Algazi et al., 2004)

2.3.3 Separation in Frequency Range

ITD represents the characteristic of localisation in the horizontal plane for low frequencies. Sensitivity decreases for the range above approximately 1.3 kHz. High frequently artefacts could be prevented through a low pass filtering (compare figure 5). The crossover frequency, depending on numbers of microphones, separates the low and high range (see formula 3) (Roos, 2011).

2.3.4 Fixed Microphones

A fixed omnidirectional microphone provides the high frequencies, with the signal reproducing the head movement independently. Spectral energy could be maintained in this frequency range. However, interaural and dynamic localisation characteristics will be lost. Also due to this effect, spatial impression could collapse and prevent direction determination. There is also a separated spatial source patterns, which is generated because of the differences of high and low frequencies of spatial information. Even the positioning of the microphone array is important for the timbre, through shading effects (Algazi et al., 2004).

2.3.5 Linear Sectoral Interpolation

A linear sectoral interpolation is a combination of low frequencies used by linear (see chapter 2.3.2), and high frequency sectoral interpolation (chapter 2.3.1). Based on the main energy in low frequencies of audio signals, and short energy in high frequency parts, the click will be less conspicuous. Linear interpolation will be used for an effective elimination of artefacts through a limited frequency. This combination generates the linear-sectoral interpolation with a better signal reconstruction than the previous methods. (Algazi et al., 2005)

2.3.6 Interpolation in Frequency Range

Low frequencies are interpolated in linear (chapter 2.3.2), and high frequencies in a spectral manner. Short Time Fourier Transform (STFT) is used by a transformation of the frequency range. STFT is calculated by Zhu et al. (2007) in formula 4.

$$X[m,\Omega] = \sum_{n=-\infty}^{\infty} x[n]W_a[n-mL]e^{-j\Omega n} \tag{4}$$

m	=	segment index	W_a	=	analyze tapering function
L	=	Hopsize	Ω	=	$\frac{2\pi k}{N}$
N	=	block size			

By contrast with the previous methods, the spectral is interpolation less error-prone. To the detriment of frequency range, interpolation processes signal more ambitiously, because of the complex Fourier transformation. Even the latency could be higher than in other methods (Roos, 2011; Algazi et al., 2004).

2.4 Head Movement

The head tracking is an essential part of an MTB system. For a head measurement system, it is useful to use a low latency and an efficient sensor. It is not necessary to analyze in three dimensions because the horizontal plane is sufficient. In 2002, Welch et al. described the different technologies of a head tracker.

- **Mechanical sensing** is the simplest option, with a typical direct connection between sensor and environment. An articulated series of fixed mechanic parts, which are connected with an electro mechanic converter (e.g. potentiometers, rotary encoder). If the target is moving, then the articulated series will shift.

- **Inertial sensing** was developed for airplanes, submarines, and ships in 1950s. It contains a highly precise gyroscope with a small packaging. The gyroscope measures three orthogonal angular-rates. A linear accelerometer on each axis generates the rotation matrix.

- **Acoustic sensing** uses a transmission, and cognition of sound waves. The duration of an ultrasonic impulse is measured, but signals could be destroyed by interferences of walls, and objects.

- **Magnetic sensing** is based on measurements of the local magnetic field with the use of magnetometers. Head movements can change the magnetic field, which is generated by an electromagnetic coil.

- **Optical sensing** is based on measurements of reflected, or emitted light. Two components (light source and optical sensors) are required. There are analogue, and digital targets with one or two dimension, and one dimension could be higher sampled than two dimensions.

It is useful to work with a magnetic sensor, as this indicates the detected rotational movements in a voltage change relative to the rotational speed.

3 Visual Computing

AR and Virtual Reality (VR) are not entirely contemporary concepts. They are based on a long history of inventions. The early developments in simulations, and VR started in the early 1920's of the 20th century, carried out by Edwin Link. He designed a flight training simulator for novice pilots (Rosen, 2008). Morton Heilig wrote an article about `'The Cinema of the Future'` in 1955. It describes a place that will appeal to several senses:

> „Thus, individually and collectively, by thoroughly applying the methodology of art, the cinema of the future will become the first art form to reveal the new scientific world to man in the full sensual vividness and dynamic vitality of his consciousness."

He even built the `Sensorama` in 1957. It is the first virtual simulation of a motorcycle ride through the streets of New York City. The recipient could smell the smog, feel the wind, and the vibration of the motorbike, and even had a 3D view shaded by stereo sound (Paech, 2018).

In 1965, Ivan Sutherland developed the first `'Head-Mounted Display'` for 3D simulation environments. These milestones, and further developments, built the fundamentals of contemporary AR/VR (Dörner et al., 2013, p. 19).

Visual Computing (VC) has become more popular in the last two years. It is necessary to distinguish between VR, AR, and 360° Videos. Generally, VR is the superordinate of AR, VC, 360° graphics, and visualization (Bebis et al., 2016).

The interaction of visual, and acoustical cognition could generate the sensation of being inside a virtual environment. A pseudo-real environment is generated when users awareness, and their illusory stimuli, are secondary; this effect is called immersion. If the degree of immersion is particularly high, it is also called `'presence'` (Brill, 2008, p. 6). While immersion concerns first and foremost, the degree of immersion and experience, the term presence aims at the state of being there (Brill, 2008, p. 6).

Augmented Reality

AR is the computer-based extension of the perception of reality. It describes the visual illustration of information. Virtual objects are added to videos and pictures via overlays. In contrast to VR, augmented reality focuses on presenting additional information. Good examples of AR can be found in the real time comments during a sport competition (Azuma et al., 1997, p. 2).

Virtual Reality

VR is a computer-based simulated reality, or artificial environment, which people allows an interactive experience with the help of technical devices and extensive software. It signifies a new interface for controlling, and interacting within a simulated reality. VR is used wherever users visualize, manipulate, and interact with complex data. It allows a 3D illustration of various existing, or non existing, objects. Even impressions could be mediated, which are not tactile in the reality, because they are to fast, or invisible (Brill, 2008, p. 6-12).

3.1 Human Information Processing

The way people perceive and process information, is essential for the design of virtual environments, as well as the interactions within them. Ultimately, every virtual world is consumed by humans. Figure 6 shows the human processor model of cognition, which works like a computer system with input, processing, and output.

This model starts with environment stimuli that is the input into the perceptual system. This setup is connected to memories (e.g auditory memory) and filter processors. The processing of those senses is managed, and saved in short and long-term memories by the cognitive system. It will prepare the interaction with the environment, which is conducted by the motoric system (Card et al., 1986).

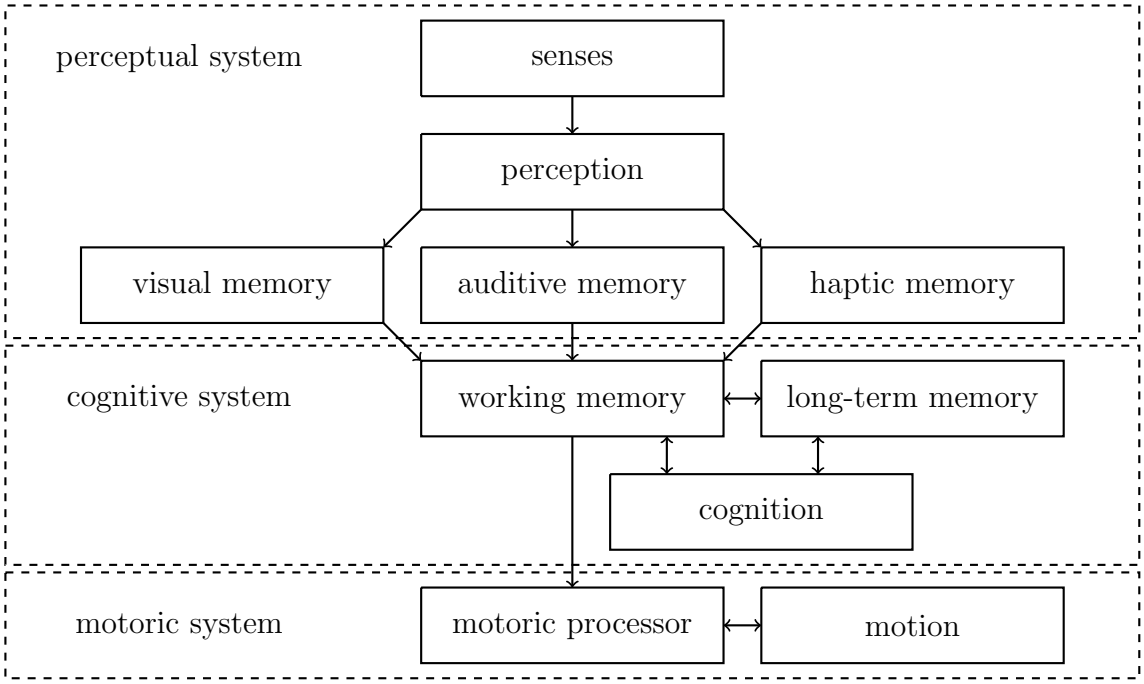

Figure 6: Model of the Human Processor (modified from Card et al. (1986)) and Dörner et al. (2013)

3.2 Visual Cognition

Human visual cognition is managed by the eyes and neural system. Light is projected to the retina through the lens. There are approximately 120 million visual cells, which are separated in light (rod cells) and colours (cone cells). Cone cells can differentiate between the Red Green Blue (RGB) spectrum. A sharp view at the retina is controlled by a muscle, which opens and closes the lens (accommodation). The highest concentration of visual cells and the best image sharpness is located at the fovea. The best resolution of image sharpness could be $0.5 - 1$ angular minute with the best conditions. Considering this resolution, it is possible to realize a 1 millimetre point, at a distance of 3 - 6 meters, from the human eye.

The visual sense enables the identification of objects. The image's brightness, contrast, colour, and motion are analysed by the retina, which will edit them if necessary. Detection of various elements and objects, and their relevance, works

probably with comparison of saved experiences in emotions, feelings, smell, and noises (Dörner et al., 2013, 35-36).

3.2.1 Stereopsis

Stereopsis describes the phenomena of manipulated human cognition by use of a VR system. Both eyes are not able to separate each individual image but the visual perception system can create a 3D impression from the 2D retina light stimulus. Figure 3.2.1 shows the difference between a normal stereopsis, and its manipulation by terminal environments, as well as a Head Mounted Display (HMD). While the eyes are focussed on the object, the projection is on both retinas. In the case where humans are wearing a HMD, the eye muscles need to be moved, depending on the distance. This motion is called convergence.

Each eye has got another convergence angle (θ) in relation to the considered object, because of the different eye location. It allows an absolute localization. Thereby the brain has two different images of the environment. The result of this image fusion is a three-dimensionality. 3D illustration within VR glasses are manipulations of the human cognition. This view is generated on a 2D display with objects in various depths in front of, or behind, the screen. This cue is called disparity. Depth cues are separated in pictorial, monocular, and binocular cues. (Dörner et al., 2013, p. 37-39) and (Brill, 2008).

- The monocular cue is a covering of a object by another. The brain can analyse the depth by experiences.
- The pictorial cue is based on monocular, cue but with a depth analysing in 2D images.
- The binocular cue is the property of disparity (Nagata et al., 2012).

According to Dörner et al., the validity of depth cues depend on the distance between observer and objects. Covering is solid in a normal visual field. Disparity is less solid in the instance of a long distance ($s_{max} = 10\text{m}$).

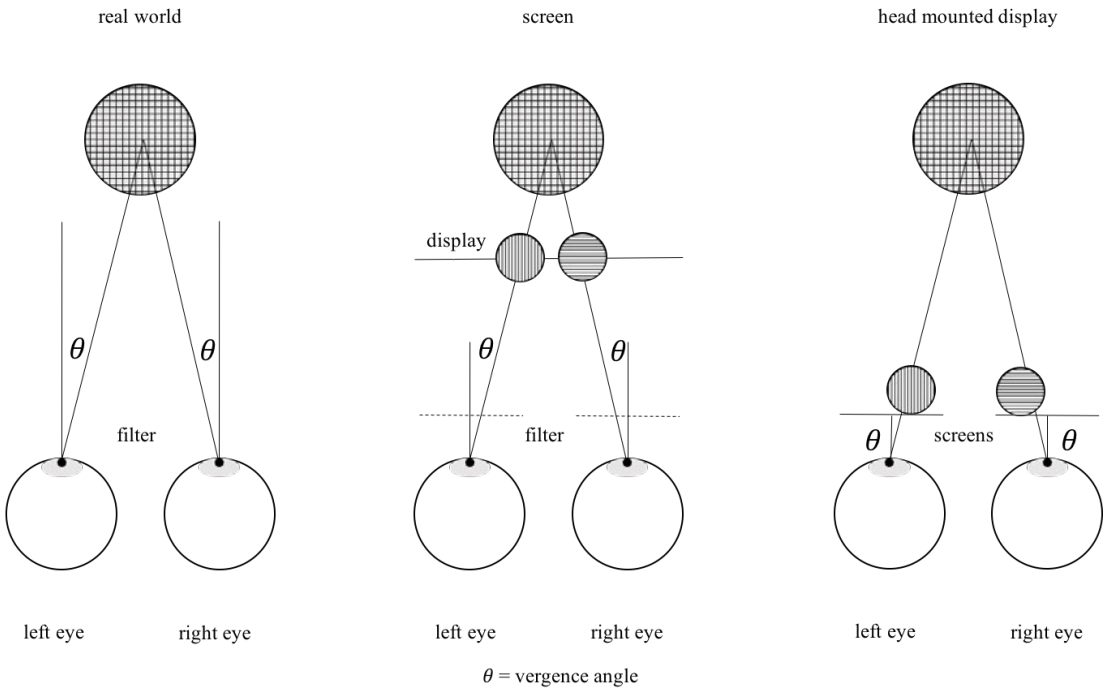

Figure 7: Normal stereopsis and stereodisplay manipulation adjusted according to Dörner et al., p. 37

3.2.2 Motion Cognition

Motions are fundamental processes in real, and VR environments. A stimulus will move whether or not a retina shifts and it also depends on the distance as well as depth cues. A stimulus that is further away generates a smaller retina and shifting, if the speed stays the same for both distances. Movements consist of local movement stimuli and humans are able to analyse complex movements with elementary detectors. Nevertheless, we mostly use the physical and not the retinal speed for motion cognition.

A second perception of movements is managed by the hair cells within the inner ear. They determine liquid changes inside of semicircular canals, which is called vestibular sense, and helps for linear, as well as rotary acceleration. It is possible to generate a pseudo-proper motion (vection). This illusion is mainly due to the perception of the optical flow. The optical flow can be modelled as

a vector field. It means that each point (P) on an image is assigned a vector. The image is not isolated, but is part of a sequence of images in which one can find corresponding pixels. The direction of this vector indicates the direction of movement of the pixel (P) in the image sequence. The speed of the movement can be determined by the length of the vector.

Therefore, the optical flow is a projection of the 3D velocity of vectors from visible objects onto the image plane. (Ernst, 2008; Dörner et al., 2013)

3.3 Displays

The basic equipment of a VR system consists of a computer and a graphic output device. There can also be audio and motion sensors, depending on the individual application. Early VR systems used HMDs. Those displays consists of two Liquid Crystal Display (LCD) for both eyes (Dörner et al., 2013, p.142 - 144)

3.3.1 Head Mounted Displays

A Head Mounted Display (HMD) is fixed on the user's head and is worn like an information helmet, or glasses. Central to its design is an image, generated by a miniaturized Light Emitting Diode (LED), or an Organic Light Emitting Diode (OLED) display, which are enhanced by an optical lens. Displays and sensors are linked to a controller and an interface will handle media files between computer and display (Warnecke and Bullinger, 1994, p. 22).

Field of View (FOV) has a wide influence on the presence (see chapter 3). For monocular displays, it describes a virtual horizontal and vertical angle of perceived virtual information by the user's eye (see figure 8). Resolutions of 442 x 238 pixel with a FOV of 108° and 76°, were the beginning of HMD. A favourable FOV angle supports a stronger virtual perception (Warnecke and Bullinger, 1994, p. 22) and (Dörner et al., 2013, p.143). Current HMDs working with a resolution of 2160x1200 Pixel. A high immersion would be generated by a resolution of 4k (3840x2160) (Thomeczek, 2016).

Angles of FOV in instances of binocular displays are shown in figure 8. Those displays contain two optical lenses and displays for each eye. Therefore, and for the users and display overlap, it is necessary to separate the angles in the right and left FOV. The brain will combine both impressions. Both eyes perceive the common middle of their individual image margin.

One option for a displays in an HMD could be a direct view display (see figure 9). In this case, it is impossible to reproduce the real environment. Only virtual information is supported. Those displays have to provide a slight front luminance (formula 5), because isolation is almost full.

Figure 9 shows a concept of direct view displays as well as figure 8. It works like a common magnifier which enlarges the presented image. The display on which the user looks through a lens, is positioned at a distance equal to the focal length of the lens (Melzer, 2001).

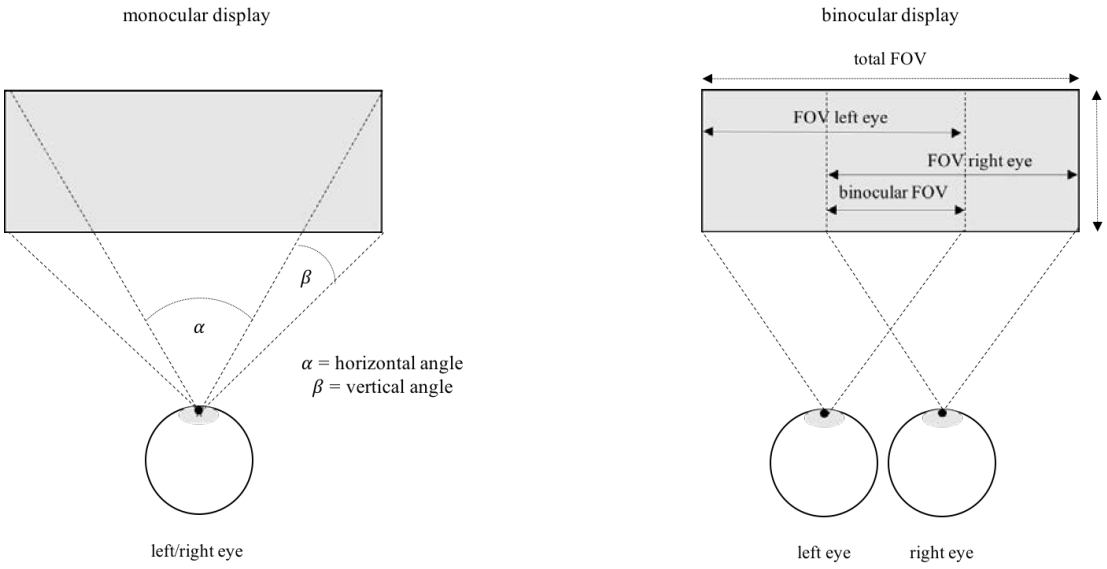

Figure 8: FOV of monocular and binocular displays (Dörner et al., pp. 143-144)

According to Dörner et al. (2013), further characteristics for a virtual environment cognition are:

- Front luminance states for the image brightness (see formula 5).

- The background contrast-ratio works closely with front luminance. It describes the relationship between bright and dark (see formula 6)

$$C_{front} = \frac{L_{pixel(max)}}{L_{pixel(min)}} \tag{5}$$

$$C_{back} = \frac{L_{front} - L_{back}}{L_{back}} \tag{6}$$

$$\text{FOV} = 2 \arctan\left(\frac{D_{lens}}{2L_e}\right) \tag{7}$$

FOV angles of figure 9 are calculated in formula 7, if ($D_{lens} < L_e \frac{S}{F}$). This angle depends on lens diameter and eye relief (L_e). Eye motion box (E, formula 8) conditional of L_e. If the eye is in such distance from the lens that the eye motion box is smaller than the viewing area of the eye, the virtual image appears partially cut off (Dörner et al., 2013, p. 149).

$$E = D_{lens} - \frac{L_e S}{F} \tag{8}$$

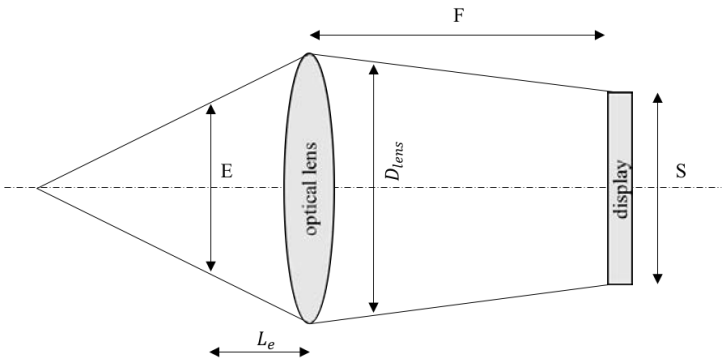

Figure 9: Optical concept of direct view displays (by Dörner et al., p 148 and Melzer)

3.3.2 Retina Display

Various application options of HMDs and constantly decreasing costs with coincident increase in components performance are very well qualified to use a HMD based on Apple iOS mobile devices with high resolution retinal displays. The high pixel density of an iPhone X (2436 x 1125 Pixel with 458 Pixel Per Inch (PPI)) (Apple, 2017a) integrated in a cardboard with optical lenses could convey a high immersion (compare chapter 3.3.1 and 3). The LCD of an iOperating System (OS) device needs a high response time to refresh the display. 60 Hz for an iPhone and maximum 120Hz for an iPad (Apple, 2017g). Even the variations of response time are important for balanced rendering. It is not profitable if you can see 99 Hz of 100 Hz in the first 5 ms. Finally, the video latency depends on the video quality (quantities of pixel).

3.4 Latency

VR systems have to work very well with low latencies. A high latency can causes motion sickness for many users. Cybersickness can occur, if the perception is not equal to the real environment. In the virtual world, it often happens that a movement is faked to the visual sense, which does not correspond exactly to the vestibular proprioceptive information. If a head tracker is used in combination with these displays, which is responsive for the displayed image to move in real time, symptoms may occur if the image display is adjusted too late or asynchronously (Dörner et al., 2013, p. 56).

It is useful to decrease system latencies to prevent this effect. Developments of current VR hardware has a minimum of requirements. A sufficient resolution per eye is necessary to achieve a certain level of immersion. A recommended value for visual rendering by physical movements of the headset is called `motion-to-photon time`. It is influenced by the head tracker, screen updates per second, and rendering engine. Current displays work with 90 Hz refresh rate. A jerk-free output of audio and video rendering needs a latency with less than 50 ms = 20 Hz (Lindau, 2009). In comparison, cinema projectors are working with a rate of 48 Hz. A

frame rate of 100 Hz is flicker-free for the human eye. However, there are different requirements of latency for video players (Thomeczek, 2016).

The latency of the head tracker includes the sampling rate and time of transmission of the device interface. Also, the MTB rendering and audio/video synchronisation need some processing time (function 9). Lindau (2009) evaluated the latency of 43 ms of the whole system. This is a good value for the audio and video synchronisation.

$$\Delta t_{system} = \sum_{k=1}^{3} \Delta t_{trans} + \Delta t_{sampling} + \Delta t_{rendering} \tag{9}$$

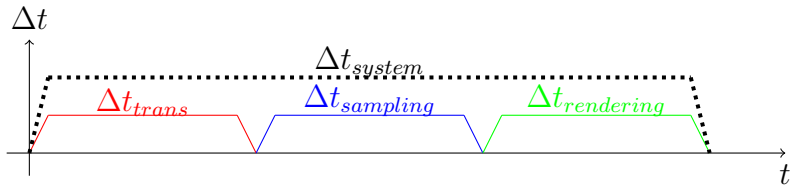

Figure 10: Scheme for total latency

4 Development

This chapter describes the development steps, including the streaming server, audio and video rendering, and how to control them.

A user-controlled computer system must consist of a frontend and a backend. The frontend serves as a user interface for data input, which can process the logic in the backend. As the name suggests, the frontend is located before the systems interface and the backend behind it. Usually these two terms occur in web development.

The user interface in the development of an application are the inputs and outputs via the display, speakers and sensors. The back-end is therefore everything that happens in the background depending on the users input.

Mobile phones support native and web-based applications. Native applications are developed based on the supported programming language, architecture and works without an Internet connection, because they are stored locally on the device. A web-based application is run by an Hypertext Markup Language (HTML) version 5 Internet browser that requires an internet connection, or network connection, because it needs to communicate with a server (Schumann et al., 2010, p. 3-9).

Based on the fact that the application will initially only work on iOS; AVR is based on a native application for iPhone and iPad.

The interaction of the AVR is based on the input configurations, which interacts with the streaming server (chapter 4.1), sensor data processing (see chapter 4.2.3), audio processing (chapter 4.2.6), and video processing (see chapter 4.2.5) with its outputs.

To put it all together, this chapter deals with the configuration of the media server, the coding and decoding options, and the associated internet protocols. Furthermore, the development of the application is covered with the tracker connection and audio, as well as video processing.

In the special case of application development, special features of the iOS have to be considered, as well as frameworks offered by Apple have to be used.

Based on this, the algorithm has to be developed and the connection to external peripherals has to be established.

ISO/OSI

Based on the ISO/OSI layer model (see figure 11), it is necessary to distinguish at which level the data processing is performed. The model is divided into the application layer and the transport layer. Figure 11 also shows in which layer the respective part of the AVR is to be assigned (Meinel and Sack, 2009, p. 142).

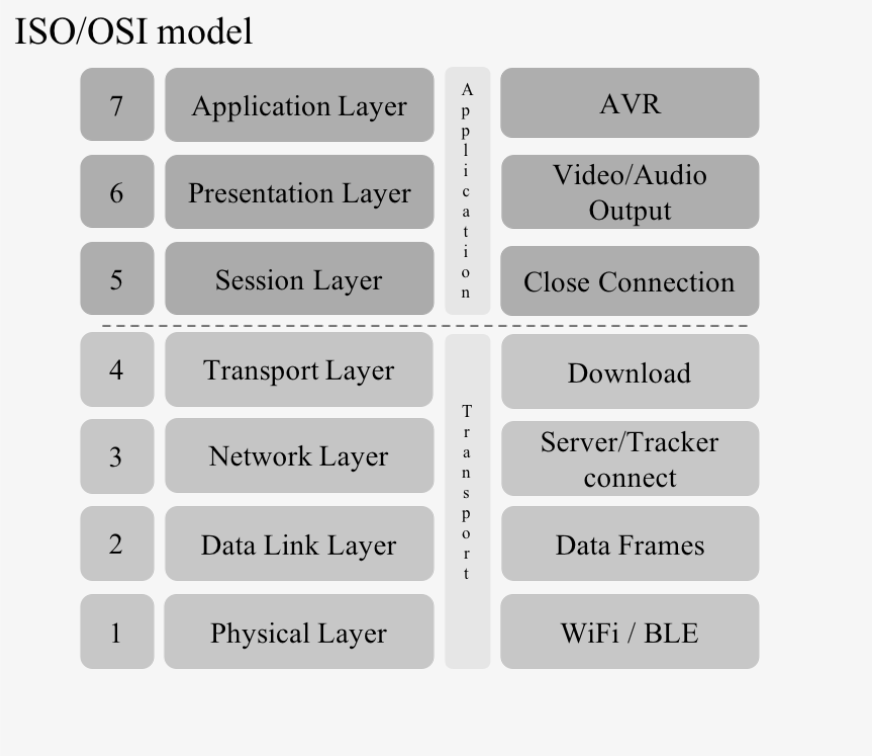

Figure 11: ISO/OSI model linked to the AVR (Meinel and Sack, 2009, p. 142)

The physical layer defines the transmission channel on the part of a network hardware, for example the connection to the external tracker and to the data server must be assigned to it. The Data Link Layer controls the communication between the individual network components. In the case of the AVR, it organizes

individual data frames between those components. The network layer provides the transmission of variable data packets. To complete the transport layer, it provides a reliable transport service for the application (Meinel and Sack, 2009, p. 143 - 145).

The application layers include three additional members. The session layer describes and controls the dialogue between two participants of one network. This means that server and iOS device, or iOS device and external head tracker are designated and controlled. The presentation layer ensures correct interpretation of the data from higher-level applications. Tracker data, or audio and video files can thus be displayed and played back. The application is the last layer and is an interface of the network participants using the underlying services (Meinel and Sack, 2009, p. 143 - 145).

4.1 Streaming Service

The principle of streaming is based on the fact that media data is sent in such a way that it can be received and simultaneously be played back by the user (client) in real time. This makes live streaming possible in addition to on-demand offers, which requires the real-time transmission of video and audio in a continuous data flow between server and client. There are various approaches to implement media content on the Internet or in a private network. The process of data transmission itself is called streaming and transmitted (`streamed`). While transmitting a broadcast service it is possible to call the stream by various clients (Hansch and Rentschler, 2012, p. 17).

This section deals in particular with the transmission protocols, decoding, and the associated web server. A Raspberry Pi version 3 may be a good low cost device for a prototyping web server. Server with more performance are welcomed. Furthermore is it useful to have a good internet or local network connection for the server.

File Encoding

In order to transmit media files via Internet communication units, they must be encoded. Moving Picture Experts Group (MPEG) is a procedure defined and approved by the International Organization for Standardization and International Electrotechnical Commission, which differs in different versions, for example layer 1-4. These differ in their complexity and audio or video content. For example, MPEG-layer 3 (MP3) is the most widely used method. Layer 2, the predecessor of layer 3, has established itself in some radio systems and Layer 1 is rarely used anymore. Layer 2 and higher can also be used to encode videos. Layer 4 enables full high definition processing (Weinzierl et al., 2008, p. 870- 872).

File Decoding

A decoder is required to play encoded audio and video content (compare figure 12). This encoder is known in the streaming context as `FFMPEG` and is merely a player that fetches and plays a stream via the command line. It even can convert between different sampling rates and change them during playback with a high-quality multi-phase filter. Audio and video data are often transmitted over the network by using the MPEG-TS (Transport Stream) format (FFmpeg, 2018).

Figure 12: Output Transcoding Process in FFMPEG (FFmpeg, 2018)

4.1.1 Network Protocols

Network communication can be separated into basic and further individual tasks. The basic task of all network communication is data transfer with individual advantages and disadvantages. The protocols send data separated into packets.

If a data packet is received correctly, a special acknowledgement procedure can be established to signal a successful data transfer. Acknowledgements can refer to individual data packets, or to various data packets called `Piggy Pack`.

Besides the transfer, another basic task of network communication is the establishment and termination of a data connection. The protocol mechanism used must be able to react according to successful or unsuccessful connection requests. Data transmitted via a switched connection must be delivered in the correct order. Also, fallback and recovery mechanisms must be provided to restore the communication to a consistent state after a disconnection. Further complex and important tasks are error handling, flow, rate and congestion control, and multiplexing (Meinel and Sack, 2009, p. 152-155).

File Transfer Protocol

The File Transfer Protocol (FTP) enables the download from a server by a client. Even file transfer between two hosts is possible. The advantage of this technology is the unrestricted format compatibility. The disadvantage is that the file can only be used after complete download. This can lead to long loading times for large files, depending on the Internet connection (Hansch and Rentschler, 2012, p. 17).

Hypertext Transfer Protocol

Hypertext Transfer Protocol (HTTP) was developed in 1999 for distributed collaborative, hypermedia information systems. It is even used for communication between user agents to other internet systems. HTTP allows basic hypermedia access. Therefore it is stateless protocol to load hypertext files (websites) into a web browser (Fieldling et al., 1999, p. 7). There are standards that supplement and build on HTTP, such as Hypertext Transfer Protocol Secure (HTTPS) for encrypting transmitted content.

HTTP Live Streaming

HTTP Live Streaming (HLS) streams audio and video content via a server, distributor to the client, by using HTTP. Input streams are coded and bundled in a suitable output format, by the server. It sends these bundles to the distributor, which is based on a basic web server. Incoming requests are handled and output files are provided. Finally the client downloads, bundle up, and shows the incoming data in a continuous stream (see figure 13).

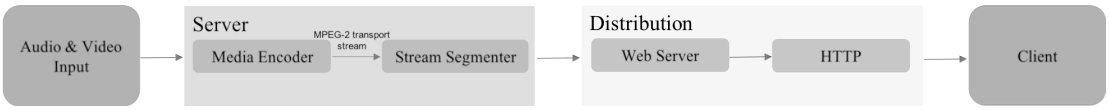

Figure 13: HLS basic configuration (Apple, 2018ah)

The server generates a video in `H.264` compression for MPEG-2 transport stream. This stream is separated into short media files, by a software segment, and stored to a web server. Even the Uniform Resource Locator (URL) is published at this server as well. The client software is showing the bundles encoded as media content without any pauses or gaps (Apple, 2018ah).

Real Time Transport Protocol

A standard for audio and video data transmission is the RTP. It is based on a protocol, lower level (e.g. User Datagram Protocol (UDP)), to transmit media data. However, it offers some important features for the communication of media data, such as content characterization, or data reception monitoring. Another important feature is that the protocol enables multicast communication. This has the advantage that a group of recipients can collect and process the data. RTP user data, for example, is transported via UDP (Schumann et al., 2010, p. 1).

UDP belongs to the transport layer of the Internet protocol family and enables applications to send datagrams in Internet Protocol (IP) based computer networks

without the guarantee that data will be received. Therefore, it is a minimal connectionless network protocol (Meinel and Sack, 2009, p. 150 - 151).

The data transport of the RTP is extended by a control protocol Real Time Control Protocol (RTCP), therefore the data transmission in multicast networks can be monitored, and minimal control and identification functions can be provided. Since RTCP and RTP only transmit data, but cannot check connection control, it is flexible in the choice of connection protocols and can transmit via other means besides the standard UDP (Schumann et al., 2010, p. 1).

Real Time Messaging Protocol

Real Time Messaging Protocol (RTMP) was developed by Adobe to enable live transmission. A broadcast software (e.g. Open Broadcast Software) can send a video directly to the web server. This protocol is based on RTP and therefore clients are able to call the live stream in real time. An internal software module which is called `FFMPEG` (see chapter 4.1) can encode the local video and broadcast it inside the server. For a good transmission it is necessary to have a fast upstream. The video quality is depending on the quality and type of origin materials, the compression method, and the data transfer rate (Hansch and Rentschler, 2012, p. 18).

4.1.2 Web Server

Based on a Raspberry Pi, the web server runs with a `nginx` module. This module offers modular techniques, such as name and IP-based virtual hosts, flash video streaming, websocket protocol, and much more (Nedelcu, 2010, p. 1-5).

The installation process for a Linux machine is managed by a Secure Shell (SSH) via the terminal window. A remote installation via SSH allows a network based configuration at the Raspberry Pi without a monitor or keyboard. A `NGINX` streaming server requires some software packages. The `NGINX-RTMP` module by Roman Arutyunyan contains the HLS protocol. It publishes a `MPEG-TS` format

over HTTP with the advantage of full performance, access control, logging, and so on (Arutyunyan, 2017).

Beside the RTMP module, some `Debian` libraries are needed. `Debian` is an open source OS based on Linux distributions. For secure Internet communication, the `libssl-dev` package must be installed. It is developed by OpenSSL. It contains the cryptography protocols Secure Socket Layer (SSL) and Transport Layer Security (TLS) (SPI, 1997 - 2018). The libraries needed during the process runtime are provided by the `libpcre3` package with its headers, static libraries and documentation (SPI (1997-2018a), SPI (1997-2018b)).

All necessary commands for installing the server on a Raspberry Pi are listed in appendix section `install NGINX`.

It is necessary to add configure lines into the `nginx.conf` file. In case of RTMP it includes the `standard port = 1935, chunk size = 4000`, and `application`. The maximum value of this chunk size for stream multiplexing is 4096. It allows a low Central Processing Unit (CPU) overhead with a big size. The name of the server streaming application is `live` or HLS. Its options are set with its path, fragments and playlist length (Arutyunyan, 2017).

After installing the engine with its streaming protocols, it is possible to stream via the HLS protocol without sending commands. The files are automatically linked to the right folder. It is possible to stream the files via the internet if there is a static IP address or a website. Port forwarding allows external access into the local network on the individual service port. In case of this web server, port `8080` is configured. The `VLC Media Player` offers a playback via the network (Arutyunyan, 2017). It is a solid option to test the server and its functionality with the following URL:

```
http://raspberrypi.local:8080/hls/360.mp4
```

It listens on port `8080` in the local network for the asked media file. The IP address will differ in other networks. Optionally use clear names for the Raspberry Pi.

4.2 iOS Application

This section includes the description of the iOS implementation. Apple's Integrated Development Environment (IDE) Xcode is required for an iOS application. Apple is very restrictive in different areas of using internal hardware. It should help to keep the surveillance for the development of applications.

IDEs has the facility to test the application, by using a simulator during the development process. Compared to a desktop computer, smart phones have some restrictions with regards to CPU, Graphics Processing Unit (GPU) and power supply, because of the different dimensions of each device.

The first version of Apple's new programming language „Swift"was released in 2014. It is a powerful development language which is used for iOS, Mac, Apple TV and iWatch applications. It is not a replacement for the older programming language Objective-C, but rather a supplement for the former program code. Very important features of Swift are the easy readability and the automatic managing of memory. According to Apple, it also has tupels and various returns, generic types, powerful error handling, and fast and precise iteration through a range or a collection. In September 2017 the latest version, Swift 4, has been released. It is used for programming of the AVR (Apple, 2017k, p. 58-59) (Kremenek, 2017).

Apple offers access to hardware components and system applications with dedicated APIs. These applications are not able to access the hardware directly. It is handled by the intermediate layer of the OS (see figure 14). Therefore, the OS and its abstraction of the hardware is the limit of development. It offers all essential interfaces with various levels of abstraction. Hardware Access has a lower abstraction than system applications.

The iOS is divided into private and public layers. Only the public one allows access by developers. It includes the Cocoa Touch, media, core services, and core OS protocols. The private layer contains the kernel and the hardware drivers (see figure 14).

Any code created in Swift within the frameworks, is only executable on the specific systems created for it. Independent libraries that can be used anywhere are supported by Swift.

Previous code is written in libraries, which are also used in different projects. Swift supports the integration of C, Objective-C, and C++. C is probably the simpler variant, since the functions can be called directly from Swift. Since the already developed audio processing was written in C++, some adjustments in the Swift code are necessary. This also necessitates the fact that there are open source libraries available, which are only available for C++, such as Digital Signal Processing (DSP). Because Swift can communicate directly with C, but not with C++, it is necessary to use a program wrapper. This allows C++ code to be read and executed by C (Singh, 2016).

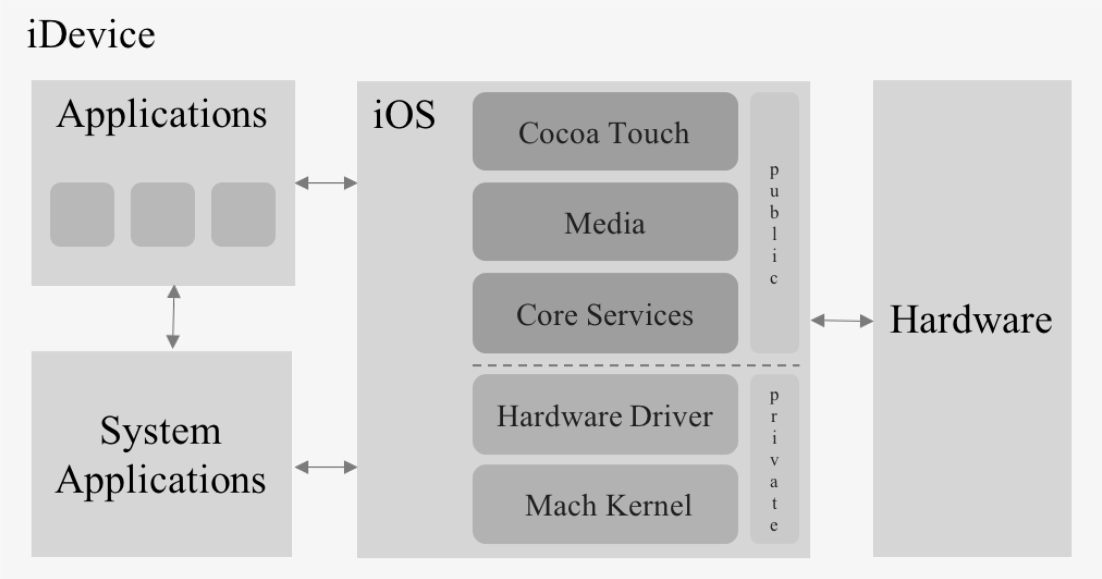

Figure 14: iOS Device Layer (Stäuble, 2009, p. 26-27)

View Controller

The interaction with an application is controlled by the Model-View-Controller. The controller updates the model and receives notifications. The other uses are that the controller updates the view and the controller receives user actions as well (Apple, 1997), (Reenskaug, 1979).

The design of the human interface needs to have clarity deference and depth. It means that text and icons are in every size precise and lucid. Also, the motions

need to be fluid and clear. A clean division is the basic for a self-explaining interaction with the content. The Graphical User Interface (GUI) controls the Model-View-Controller Apple (1997), it was founded by Reenskaug in 1979.

Every ViewController starts with its own `UIViewController` class, which provides the infrastructure for managing `UIKit` views (Apple, 2017l). There are two types of view controllers, according to Apple:

- Content view controllers are the main type of controller used and manage the application content.
- Container view controllers collect and present information from other controllers differently (Apple, 2017j)

These variations of `ViewController` are developed in the Xcode's internal main file of the storyboard designer. They are connected to their own Swift class, which is defined in a separate file. In the class, there is an override function, `viewDidLoad`. It is the entry point for the algorithm in the respective `View Controller` and will perform additional initialization on views. Of course, the classes can communicate with each other and change variables and outputs (Apple, 2018ai), (Apple, 2017k, 376-396).

The interface designer allows buttons to be added, as well as, sliders, labels and other in and output options. These interactions must be linked to the source code in order to enable individual functions, statements, inputs and outputs to be executed. Of course, it is also possible to develop a storyboard by using program code (Apple, 2018ai).

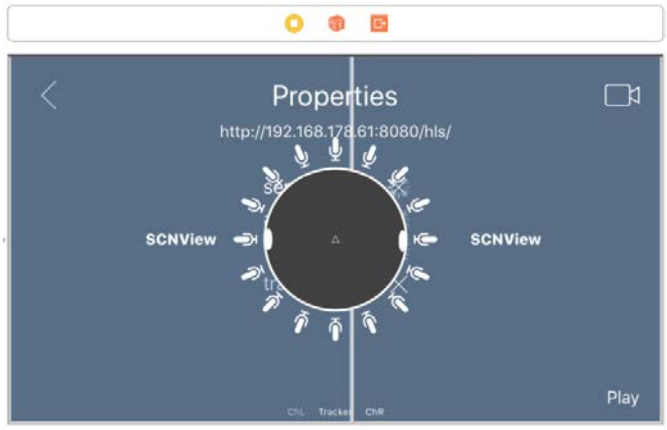

Figure 15: AVR storyboard

According to Apple, it is possible to use various `ViewController` types. The most important ones are: table-, collection-, navigation-, tab bar-, page-, and GLKit views controller. In the case of AVR, no predefined view controllers are used, because they are individually designed (see figure 15), furthermore, no tables, pages, or other features are displayed (Apple, 2018ai).

The application is therefore based on different views. They will be changed step by step, depending on the progress. After the start, the setting view is loaded, in which the user can influence the server address and select the type of media source. If the settings are successfully loaded, the next view is audio processing. To optionally add the video afterwards, the internal sensors will automatically control the audio processing, as well. The views are always stacked on top of each other or removed.

During playback, the operating objects can be shown or hidden by using the instance property that uses the boolean value `isHidden` (Apple, 2017i). This effect is primarily used in the video reproduction. The menu is hidden while the video is playing, however, it can be activated by a tapping the left side of the screen. In the case of the audio playback, non hidden objects (e.g. microphone circle) is an indicator of successful connection to Bluetooth Low Energy (BLE) peripherals or the other remote control opportunities.

In general, the application is based on different processes (see figure 16), partially parallel. After the user has started the application, selected the data source

and the tracker option, audio processing and video processing (chapter 4.2.5 and 4.2.6) are loaded with their respective rendering engines.

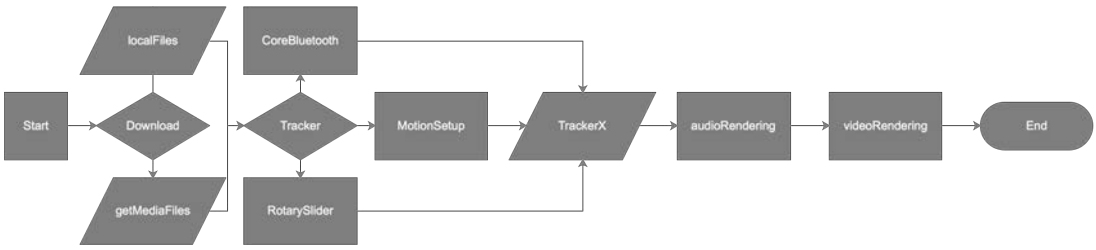

Figure 16: Flowchart of the AVR

4.2.1 Frameworks

The iOS Software Development Kit (SDK) contains numerous frameworks that provide the developer with interfaces for developing applications on different abstraction layers. The lowest two layers, Core OS and Core Services, provide fundamental iOS interfaces and are largely written in the C programming language. Among other things, low-level interfaces for the file and network input and output are made available. The frameworks contained in the upper-Layer, Media and Cocoa Touch offer object-oriented abstractions for low-level constructions and technologies of the lower two layers and are largely written in the Objective-C programming language. Encapsulation of complex features, such as sockets and threads, reduce the volume of individual program code needed the likelihood of errors. For this reason, high-level frameworks are preferred (Stäuble, 2009, p. 35-38).

Foundation

The `Foundation framework` is the base layer for all Objective-C and Swift classes from the iOS SDK because it creates a root object class `NSObject`. It also provides a fixed set of utility classes and introduces consistent conventions within

a class hierarchy. For example, it establishes a policy for memory management by bindingly defining what is responsible for dealing with an object. Its portable code also supports the platform independence of all frameworks and its code is based on it (Apple, 2018ag).

UI Kit

This framework contains all necessary classes for the development of the Graphical User Interface and is especially optimized for touch screen interfaces. The implementation of the framework ranges from the construction of the main loop in the application object, through the drawing model that belongs to windows and views, to event handling. Apple recommends using `UIResponder` or classes with a manipulated user interface only from the main thread or main dispatch queue (Apple, 2018av).

AV Foundation

This framework combines the four possibilities of audiovisual recording: processing, synthesis, control, and import or export. It also enables access to the built-in-camera. An Objective-C interface offers the action, creation, edition or re-encode of audio and video content. For these opportunities, the AV Foundation stack contains the frameworks of the core audio, video, media animation, and media and video toolboxes.

The main class of the framework is `AVAsset`, which represents a collection of several audio and video data, even their information (title and duration). No specific data format is required. It enables parallel various play-back of several different files with different resolutions. A player item object manages the presentation status of an asset. The track elements allow editing during playback of audio mix parameters and video composition settings. Queues of elements are possible as well. As already mentioned, the analysis of the files is also possible. A new asset for the output can be created by re-encoding an old one or saving it as a new asset.

The combination of reader and writer assets further options for transformation. This allows conversion from one asset to another, for example to define the output format or to display the audio waveform by using the reader. Apple recommends selecting the frameworks according to their individual task. For video playback the AVKit framework is required (Apple, 2018ab).

Core Bluetooth

The `CoreBluetooth framework` provides internal access to the BLE classes. They offer the scanning, discovering, connecting, and communication between BLE devices in the nearby environment. Since the iOS 6, iPhones and iPads can also work as a peripheral client.

BLE connections consist of two different members, both with individual tasks. The peripheral offers its presence and data over the air. As a central, the iPhone uses the classes of the `CoreBluetooth framework` to ensure communication. If necessary, the connection is closed by the receiving device as well (Apple, 2017b).

Core Audio

Even `Core Audio` is a framework for audio processing as an interface. It offers specialised data types for interacting with audio streams, complex buffers, and time stamps. Even data types are declared, which are used by other core audio interfaces (Apple, 2018bg)

Accelerate

Large mathematical computations and image calculations are managed by the `accelerate framework.` Vector processing offers high performance and concurrent energy efficiency on the CPU. This framework offers the `vDSP` library for digital signal processing in iOS development. It includes Fast Fourier Transformation (FFT), convolution, correlation, windows generation, and biquadratic filtering (Apple, 2018aa).

Graphics

Working with graphics is provided by the appropriate framework. The classes of the `CoreGraphic framework` allows interactions with 2D vector graphics. It is included if the standard `UIKit framework` does not offer the necessary functions, especially animations. It is based in the programming language C but contains an interface for the `Quartz 2D` API, for object oriented abstractions. The `OpenGL framework` provides an interface between 2D and 3D graphics. It became well known as an open source library for graphic processing (Stäuble, 2009, p. 33).

Besides these frameworks, Apple offers useful views and classes for video reproduction. This section will explain the advantages and disadvantages of four different ways to add a video player into the application.

AV Player View Controller

The `AVPlayerViewController.class` displays video content in a specific ViewConroller, managed by the player. This native system player includes remote buttons and sliders and even supports AirPlay in sharing the video content with an Apple TV. It is an easy method to transfer video content to an application, as it is not necessary to use an URL stream via the internet or a local video, which is linked inside the application.

This opportunity allows a quick implementation. It does not support a rendering of 360° videos by the `CoreMotion.framework` (Apple, 2018ad).

Web Kit

`WebKit Views` objects show integrated interactive web content. Embedded videos, e.g. YouTube videos, can be presented in a specific web browser. Therefore, it is only possible to stream the video via the internet. However, even this possibility does not support a 360° video rendering by device orientation (Apple, 2018aw).

SpriteKit

The `SpriteKit` framework offers an implementation of 2D animations and graphic rendering. The engine works with a loop. It determines the displayed content and the changing of the frames with the help of efficient graphic hardware usage. Because it is optimized for the arbitrary use of animations and changes, it is very well suited for games and flexible animation applications (Apple, 2018as).

SceneKit

These views offers an opportunity to combine 3D content with a high-performance engine and is part of the `ARKit`. It creates a link with the `CoreMotion framework` (see section 4.2.3) for device orientation that controls the video rendering (Apple, 2018al).

4.2.2 File Download

The MTB algorithm by Roos gets the audio files via the Jack Audio Connection Kit. It is an audio server, which provides a low latency processing for various OS. Several clients can be connected to share audio information between audio devices (JACK, 2014). It is a popular and professional audio connection software, but it is no longer available for iOS devices since 2013. Because of that fact, the AVR needs an own streaming service (compare section 4.1)

To keep latency as low as possible, all files should be download at the start of the application (see figure 17). This is necessary because, despite the HLS format, data volumes of 700 mega byte has to be streamed simultaneously. If the play is used for other files, the data stream can become very large, because the test files are only about three minutes long.

Optionally, there is also a demonstration version stored in the AVR. Apple offers the possibility to save files to a local file path on the iOS device via the internet or network (Apple, 2018). The download enables the possibility to play

back files from various recordings, which were previously copied to the Raspberry Pi in the */mnt/hls/files/* folder.

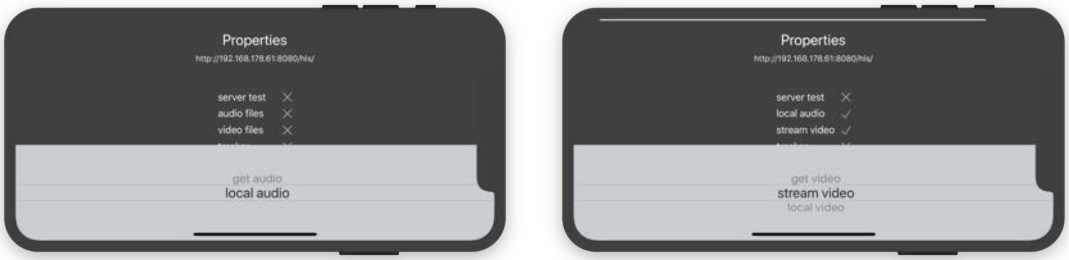

Figure 17: PickerView for downloading

The user interface offers the possibility of entering an IP address. If this is confirmed, a connection test is performed by downloading an empty text file and then deleting it. If this test is successful, the user is informed via the user interaction, and the audio download can be selected. This method is called by a PickerView. In case of reproducing the local demo files, the server test is not needed.

To download audio and video, some functions are required. There is a function that calls an empty array containing the server address, file name, and file extension. This array must be converted to an URL. The function linkstoURL(). It equates the array with an URL and converts it into a string. A filter checks that the array is not equal to NIL. The function getAudioDataFromURL() with the parameters audioURL, data expect a return value, because it is declared as void. This function starts a shared URL session that receives a data task. It calls the audioURL parameter from the function as an URL. The function ensures the complete download of the data and checks for any errors.

A progress bar is displayed, which continues once the various steps have been completed. There is also an activity indicator for the audio and video download, as the download of Raspberry Pi will take longer in this case, depending on the existing network speed. If the download was successful, a check mark will be displayed as an indicator. The server test can be skipped if only the demonstration version is to be played.

Depending on the selection of the data source, the respective files are played in the further course of the audio and video processing. The internal memory is accessed via the `file manager` for the loaded files. Local audio files are stored in the `mainBundle` based on the `NSBundle.h` library.

The advantage of a full download is that the files do not have to be streamed again. A new stream means that the requested files always generate new traffic. Downloaded files can be saved locally and used after closing the application. In addition, just a data server or cloud service is needed, and no streaming server for this type of file provision.

4.2.3 Head-dependent Control Unit

The mobile audio and video rendering application can be controlled by various opportunities in the function of motion values. It supports a hand-controlled simulation (chapter 4.2.3), internal motion processing (section 4.2.3) and an external motion sensor (chapter 4.2.3). Internal sensors are used to control the video rendering (compare section 4.2.5), because of the three-dimensional nature. Audio processing supports each case of sensor control. The user can choose the option via a picker view (see figure 18). BLE (compare chapter 4.2.4) is used for the external motion sensor. Values of these opportunities are saved in the `TrackerX` variable.

For audio processing, it is not necessary to develop a three dimensional control unit, because the microphones are just located in the horizontal plane. For the external sensor this means less data rate and also probably less latencies. For simplicity, the `circle slider` can also be used in the horizontal plane only, otherwise the application will need three different sliders, one for each level.

When comparing these three possibilities, with reference to latency, it is useful to choose the internal sensor. A connected external measurement generates a sharply higher latency than the internal which you can see in the animation by a fast movement. However, each option has advantages and disadvantages in their own operation.

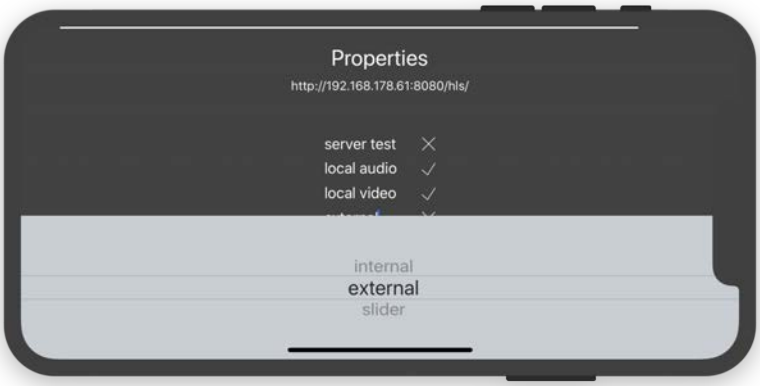

Figure 18: Selection of remote control

Manual Control System

A simulator is an option, beside the target device, to test the audio processing while the developmental stage, because the Xcode internal device simulator does not support a BLE connection and motion sensors as well.

This manual system is a simple way to change the rotation angle. In case of this property, the user can rotate the represented head manually via a slider. Ragonese developed a circle shape slider in 2016. Adapted to the AVR requirements is it a solid base to build on.

The developer offers a class that compares start and end coordinates to the circle center. Radian values are generated by the finger movement as a touch gesture. A function call of `rotarySlider()` will print a transparent circle with its touch diameter, touch point, touch tolerance, and trail. Generated values are returned as a `string` in Euler angles. Even the pre-developed class analyses when the circle is completed, or the user leaves the tolerance. These exceptions are printed in the Xcode internal terminal.

When the audio and video are playing concurrently, it is not possible to change the values of this simulator because it is not shown in the video reproduction. The only option to offer an autonomous reproduction of MTB. The following two sections are options to use an automated movement-dependent remote.

Internal Motion Sensors

Apple's iOS devices have an orientation sensor. This sensor contains an accelerometer, gyroscope, pedometer, and magnetometer. The Core Motion for device orientation is the internal opportunity to analyze movements of the device. This framework supports the access to the raw and processed accelerometer and the gyroscope data (Apple, 2017h).

Accelerometer and gyroscope data are presented in three axes through the device. All axes start in the root. X-directions are in order to use portrait mode from the root to the left ($-x$) and to the right ($+x$). The Y-axis goes to the bottom ($-y$) and top ($+y$). Last but not least, the x-axis is the perpendicular from the x and y axis. Positive direction ($+z$) runs through the top of the display and negative direction ($-z$) goes through the back of the device (see figure 19) (Cook, 2014).

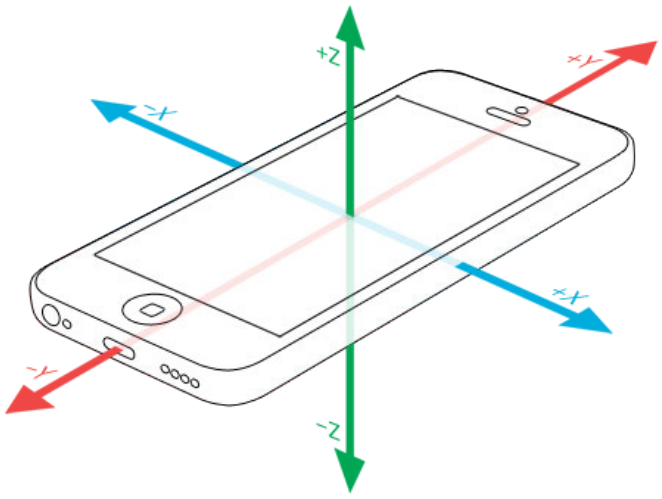

Figure 19: iPhone device orientation (Cook, 2014)

The initialisation of its `CMMotionManager` object is the base to report the onboard motion service. It enables access to the already listed sensors. Linked to this class, device orientation is managed by function `MotionSetup` for the initialisation of `deviceMotionInterval` and `startDeviceMotionUpdates`, which

both start the incoming data process (Apple, 2018ae,a). Motion interval is the time in seconds, which handles refreshment of incoming values (Apple, 2018af).

Incoming values of `CoreMotion` have to be stopped, if the user is selecting another control option for rotation (Apple, 2018at). Furthermore, Apple recommends to use only one motion manager in an mobile application, because one could conflict with another otherwise.

Access to the function `AccelerometerUpdate` is handled by `MotionData.gravity x-z`. Return variables contain a three digit number and one decimal format, for each direction.

Gyroscope updates (`MotionData.rotationRate.x, MotionData.rotationRate.y, MotionData.rotationRate.z`) returns a four digit format and one decimal in `func GyroscopeUpdate`. The fourth one is a place holder for the mathematical sign. Motion values of iOS devices are generated as a radian. For MTB it is useful to work with Euler values. The terms of the calculation and relations of the radian and Euler degrees are shown in equation 10 and 11 (Papula, 2009, p. 244-245).

$$1 \text{ rad} = \frac{360^\circ}{2\pi} = \frac{180^\circ}{\pi} \approx 57,296^\circ \tag{10}$$

$$1^\circ = \frac{2\pi}{360}\text{rad} = \frac{\pi}{180}\text{rad} \approx 0,017 \text{ rad} \tag{11}$$

External Motion Sensor

The sensor used within the audio communication group is the `BNO055`, by Bosch. It has a 9-axis with an absolute orientation, which includes a fusion sensor package of a triaxial 14-bit accelerometer, a triaxial 16-bit gyroscope with a range of $\pm$2000 degrees per second, a triaxial geomagnetic sensor, and a 32-bit microcontroller.

The accelerometer could work in the range of $\pm 2g/\pm 4g/\pm 8g/\pm 16g$. It also has a motion triggered interruption signal generation for any motion (slope) detection, slow or no motion recognition, and high-g detection. Features of the gyroscope range from $\pm 125°/s$ - $\pm 2000°/s$ with the low pass filter bandwidth range 523Hz - 12Hz. The On-chip interruption controller handles motion triggered interrupt-signal generation for any motion (slope) detection, and the high rate as well. Magnetic fields are ranged from $\pm 1300\mu$ for the x and y axis and, $\pm 2500\mu$T for z-axis. The sensor has a magnetic field resolution of $\sim 0.3\mu$T.

Software Driver

As already mentioned, the microcontroller is based on Arduino. Therefore, the driver (see figure 20) development can be made easier by using the corresponding IDE. Adafruit offers headers to initialize the sensor correctly and to read the data correctly. The driver is developed in C. An array with 18 digits must be defined, in which later the three axes can be loaded with its sign. A sub-function `AT-Read()` checks whether a BLE module is connected and sends the data from the `Serial.Monitor` to the chip. The `setup function` connects at 9600 Baud (number of transmitted symbols) to the sensor via the serial interface. A second connection is established with the BLE module.

Once both communication units are connected, BLE specific commands are sent to its module, for defining the name, and to turn off the notifications as iOS also turns off the notifications. An `if-condition` checks that the sensor data has been received.The `loop function` sets a new sensor event and receives the data at regular intervals. These are converted into Euler values and written to the array that it can be sent via BLE.

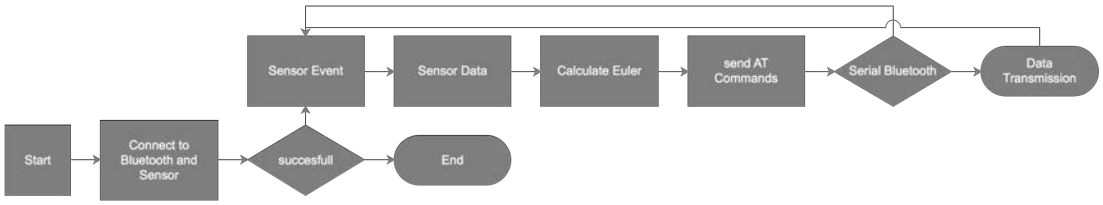

Figure 20: Flowchart of the sensor software driver

Connected with the Arduino based development board `Teensy 3.2`, it is possible to send the sensor data via a `Serial.Monitor` and the wired Universal Serial Bus (USB) connection to a terminal. It simulates a serial connection. This microchip has a 32 bit ARM processor with an Inter Integrated Circuit (I2C) and serial port (Teensy, 2017). The sensor is connected via the I2C bus. The source code for receiving sensor data also contains the option to send the data via BLE to a target device, beside the `Serial.Monitor`.

4.2.4 Bluetooth Low Energy

Bluetooth Low Energy (BLE) was launched in 2010 and is based on the Bluetooth 4.0 specification, which defines a set of communication protocols for low-energy devices. The `CoreBluetooh framework` offered by Apple defines the BLE protocol stack (see figure 21). There are two player in the communication, the central and peripherals with respective tasks. Peripherals make its presence known by offering its data via radio. The central search for offered periphery data in the local environment will request a connection to it (Apple, 2017b).

It uses the 2.4 GHz Industrial, Scientific and Medical Band (ISM) which provides 40 frequencies and two channels. The bandwidth is separated in three advertising channels for peripheral scanning and 37 data channels for connection and data exchange. Gaussian Frequency Key Shiftings (GFSK) is used to protect against interference through shifting between the 37 frequencies.

The BLE microcontroller is separated into two parts (figure 21). It contains the controller and the host, the former of which provides the technical part. It implements various protocols for BLE (e.g. Security Manager Protocol, or Logical

Link Control and Application Protocol). Both are linked in a Host-Controller-Interface (HCI). The physical layer and the link layer are implemented in a System-On-A-Chip. Pairing security is saved through the Security Manager Protocol (SMP) by the host. (Maier, 2010, p 2-3)

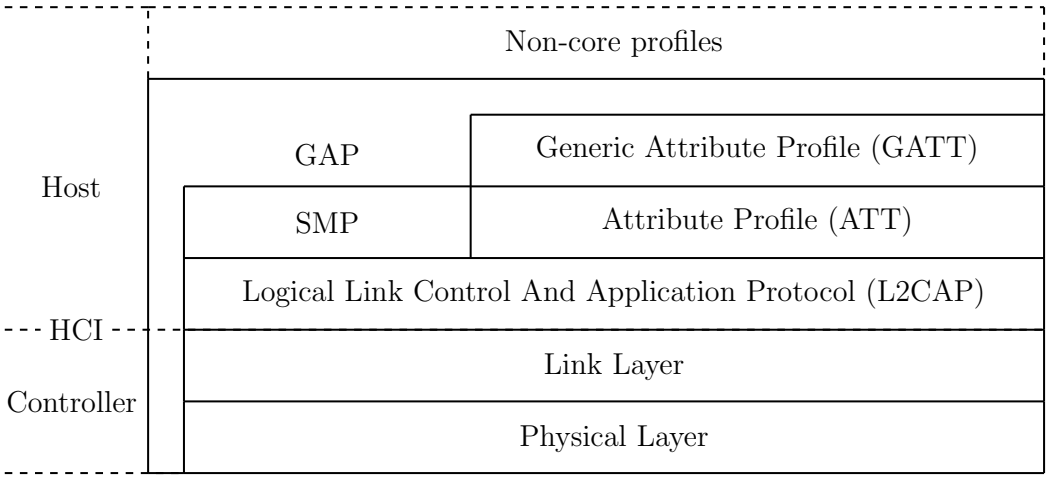

Figure 21: BLE protocol stack (Gomez et al., 2012)

For using the BLE connections in an iOS device, it is necessary to add `Core Blutetooth framework`, as it is the basis of getting access to the BLE hardware. It provides the basic profiles in the stack: Generic Access Profile (GAP), GATT and ATT (Apple, 2017b). The framework supports the roles of using:

- Performing Common Peripheral Role Task

  Peripherals will publish and advertise services and even responding of read and write.

- Performing Central Role Task

  The device will handle, scan and connect to available peripherals, while also exploring and dealing with data offered by the peripheral. This case is used in the AVR (Apple, 2017b).

Two more BLE specific protocols need to added into `ViewController.class`. These protocols are called `CBCentralManagerDelegate` and `CBPeripheral`

`Delegate`. The `CBCentralManagerDelegate` protocol delegates the monitoring of discovery, connectivity and the retrieval of peripherals. It also indicates the availability of the `CBCentralManager`. The protocol is calling states of updated CBCentralManager (Apple, 2017c). Also the `CBPeripheralDelegate` protocol uses the monitoring of discovering, exploration and the interaction of peripherals services and properties. It is even adopted by the `CBPeripheral` object (Apple, 2017d).

Furthermore some functions are essential:

- `func centralManager` discovers the peripherals. It will connect, if the potential head tracker is listed (Apple, 2017c).
- `func peripheral` uses the service of the peripherals as well as receiving and printing the data by the `characterArray` (Apple, 2017d).
- `func centralManagerDidUpdateState` checks the state updates by the `CB CentralManager`. BLE states are printed into an alert (see figure 22) (Apple, 2017c).

If the user selected the external tracker, and the tracker is not available, it is possible to select a new option after confirming the alert. All these input options are developed in an `if - else if condition`. It means that an entry must be made and depending on which option is chosen, any bluetooth or motion connections will be closed, or the manual circular slider will be removed from the ViewController.

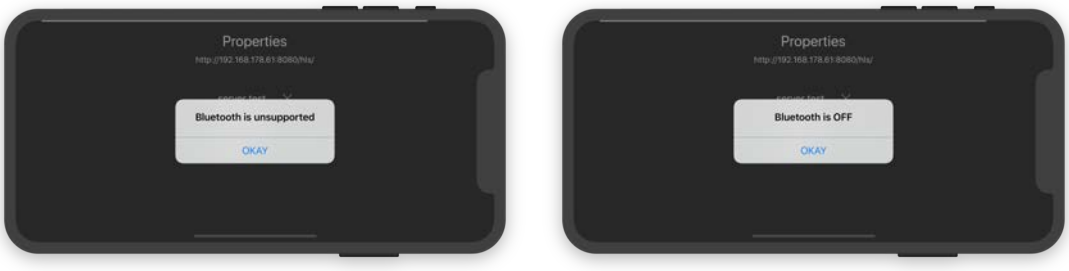

Figure 22: Alert States for an External Motion Sensor

There are two more instant methods for a link to the head tracker. The function `peripheral(didDiscoverServices)` will invoke when the application calls available services and function `peripheral(didDiscoverCharacteristicsFor)` the characteristics of a specified service, according to Apple (2017f,e). The connections to any service is managed by the Universally Unique Identifier (UUID), as a hexadecimal number, of the peripheral.

4.2.5 Video Processing

Video processing is an essential part of the present paper.

A video player has to render an already pre-recorded 360° video. It was recorded under usage of 12 GoPro version 4, action cameras attached on a sphere based rig, and was located next to the microphone array. All videos were generated in an one sphere video. There are different software tools for this process, for example Unity. In case of the AVR video player the video was already merged into the spheric video with its meta data and angle depending viewpoints.

In order to play the video, it must be linked to motion sensors so that the direction can be changed. Based on an iOS device, all required sensors are integrated. The source code has to connect these measurements to a video rendering engine.

An open source player was developed by Arthur Swiniarski in 2016. Adapted from this already written player, the AVR uses the `SceneKit` as well. It supports the possibility to play the video locally or stream it from the network or internet.

The associated algorithm is described with its class, protocols, variables, functions and instances. Even the methods to define stereoscopic opportunities will be explained.

Rendering Engine

The player is added to the `UIViewController protocol` in `AVR class`, to manage view hierarchies, update contents, and respond to user interactions (see Figure 23) (Apple, 2017l).

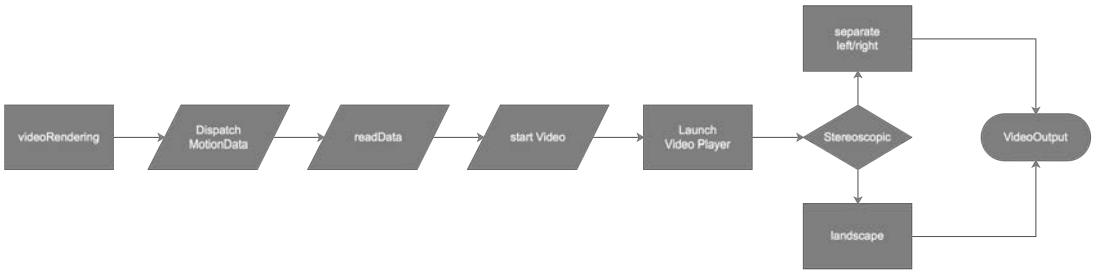

Figure 23: Flowchart of VideoProcessing

Even `SceneKit protocols`, like `SCNSceneRendererDelegate` are needed to display the video scene with its rendering. It is displayed in a SCNView, which is a subclass of UIView. The scene is managed by a loop, which handles the rendering in a succession of steps (see figure 24). A call of the render instance (render:updateAtTime:), which is called by the view, is the entry point into the loop. It is called once per frame until the video is paused (Apple, 2018ak), then the scene performs attached actions. Those actions are managed by the `SCNAction class` and `SCNAnimatable.protocol`. It offers the possibility of changing the scene frequently through the user's input (Apple, 2018am).

The protocol performs those objects on nodes for animations and movements. It checks any current running actions and can stop them, as well (Apple, 2018an). In step three, the view will call its delegates' renderer method for animation. It updates the scene view immediately through changes in the user's interaction (Apple, 2018aj).

Step four and five are not essential for the 360° processing, because the latter applies physics simulations to its various bodies and its delegate with a time renderer method. The present application does not have these objects.

At the end of the loop the view will call the `render:Scene:atTime` twice, which updates the scene view. The view will be updated by the scene, before call number two (Apple, 2018aq).

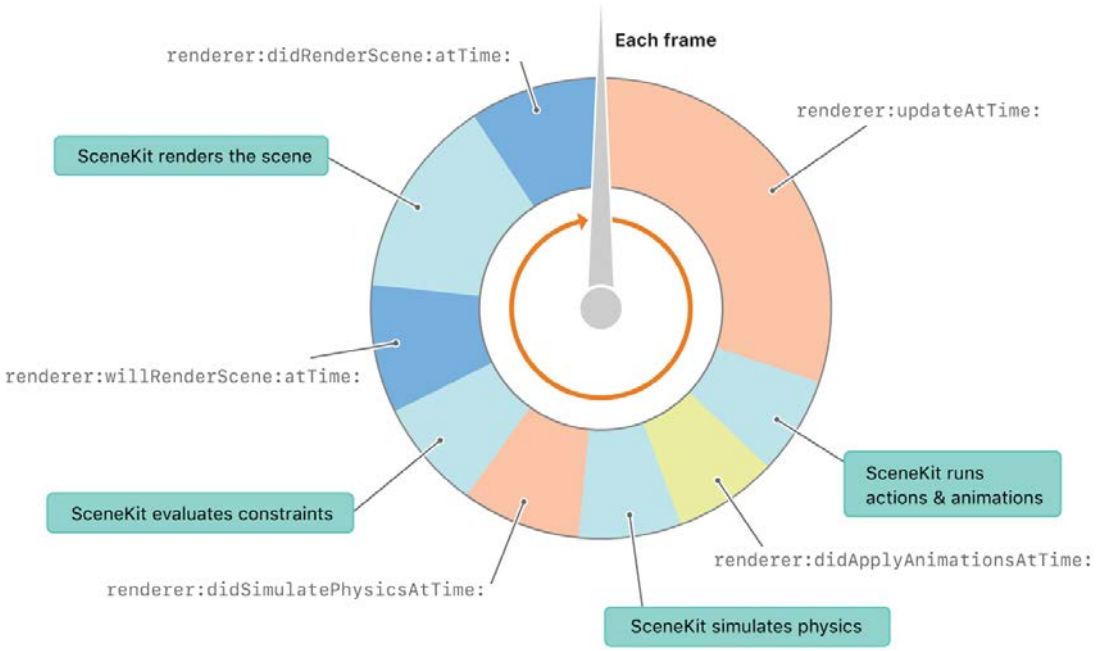

Figure 24: Frame processing in a SceneKit renderer (Apple, 2018aq)

`UIGestureRecognizerDelegate protocol` is necessary as well. The gesture recognizer will communicate with its delegates. It will manipulate gestures and their relationship to another gesture recognizer, such as allowing simultaneous recognition (Apple, 2018au).

Scenes and nodes need to be instantiated as a variable in the case of a `SceneKit View` with its spheric video rendering, to display the content. `SCNNode` classes are representing positions and 3D transformations of structural elements. It only represents positions, orientations, and scales (Apple, 2018ao). Three-dimensional nature and their angles (roll, yaw, pitch) need to be defined in various nodes for a 360° Player. These nodes are linked to `SCNScene`. It contains the represented 3D content. They are managed by its render `SCNSceneRendererDelegate` protocol (Apple, 2018ap).

The rendering function is based on a dispatcher queue, which processes the main threads asymmetrically. The asynchronous processing of the grinding allows a direct change of possible viewing angle changes. Thus, the outputs of the three planes are independent of each other. Within a plane there is dependency on the previous value. The queue organizes the processing of video data depending on

the process and transaction data. A `for loop` guarantees the permanent query of the transaction data during this processing, and transfers it to the output scenes.

The device motion and its properties are defined as well. It can be described as internal sensors working as a remote for video rendering with an update interval.

Reproduction

As already mentioned, the video player can play a local video file or even a stream. The options are offered in a `pickerView` as well. To start a player, it is necessary to connect the developed player to the `AV Player` instance, with the latter also receiving the file source value (the location and name) of the video it is linked to. In addition, the local display size and associated nodes must also be defined.

Furthermore, the possibility of stereoscopic display in the player must be linked. The outcome of being observation depends on the device size, with relation to the width of the individual display and how it is defined. It is kind of a constraint. Constraints are restrictions in the output of each view. Elements and objects on the output surface can be defined in a strict way.

The player options of wide screen or separated views have individual nodes for rendering. In case of device rotation, an observer detects the alignment and adjusts the video view scenes and their nodes.

The function `getNodeCamAngle` observes the landscape mode in right and left direction depending on the status bar. An `else-if` loop will compare the orientation of the device and set a new camera node angle. It returns the new calculated value to the calling function `startVideo()`.

Even `panGesture()` is used for calculating the shown content. Depending on the devices orientation, it turns the video by tilting the target. In order for this method to work, the angles will be saved in a new variable, using both, landscape and portrait mode. However, as it does not make sense to watch a video, with or without VR glasses, in portrait mode, this mode is switched off.

The video player itself is developed in function `LaunchVideoPlayer()`. The `AVPlayer` gets the file `URL` as a string for streaming options, or from the `Bundle.main.path` as the local option. This player is linked to the left video node depending on the full device screen scale. It will also automatically be divided into

two scenes in case of stereoscopic view, depending on its screen scale. By tabbing this option of views, the right scene will be shown. The left scene will be layered over the right one in case of a full screen.

Both scenes are defined as a `SKCropNode()` to allow a replay of the individual left and right scene. In general, a `SKCropNode()` is offered by Apple, as any effect with a set of nodes, should be drawn only inside a specific part of a scene (Apple, 2018ar). A loop will calculate the individual node position without increasing the sprite kit value. It is defined as `weak var` to deallocate to decay it to `nil`.

The actual rendering happens in the function `render` and defines the engine which is described in section 4.2.5 depending on its time interval. The engine uses a asynchronous queue, depending on its device orientation and renders the axis in its own loop. It sets the motion attitude equally as roll, pitch, and yaw. An observer checks the incoming motion values. It will remove the nodes and stop the video, if there is an exception.

For connection of all these functions and processes, it is necessary to use a separate function `startVideo()`. At the beginning the scenes, nodes, angles, rotational position, and `far value` (distance between the camera and the surface) are defined. Even added `ChildNodes` are linked to the left camera node, depending on the camera nodes and later, on the right one as well. These definitions are separated because the right one is just needed in a stereoscopic view.This option is checked by an `if-else loop`.

Further the camera node in general are set to the positions and angles in a `for loop`. It defines the three dimensions as a vector and generates the needed camera angles for both views as `float values`. The view point itself is to `NodeCamL` and `NodeCamR`, after both scenes get the `boolean value: true`. Even the motion manager is called in the start function as well. Just as outlined in section 4.2.3. It gets a motion refresh rate of $\frac{1}{60}$s. That means the values are updated 60 times in a second. At the end, the video player is launched and motion controlled video rendering starts.

Stereoscopic View

Just as outlined in section 3.2, it is useful to separate the presented video content into two individual displays when using VR glasses. Therefore, there are two different scenes in the storyboard of the application. Tabbing the right display part, activates the stereoscopic view. When turned off, only the left scene is shown on the complete display (see figure 25).

During the simulation in the Xcode internal simulator the image is rotated 90° to the right, probably because the simulator has no internal sensors to align the image correctly. On the physical target device, the image is automatically rotated in the viewing direction by the panorama recognizer and the motion manager as well.

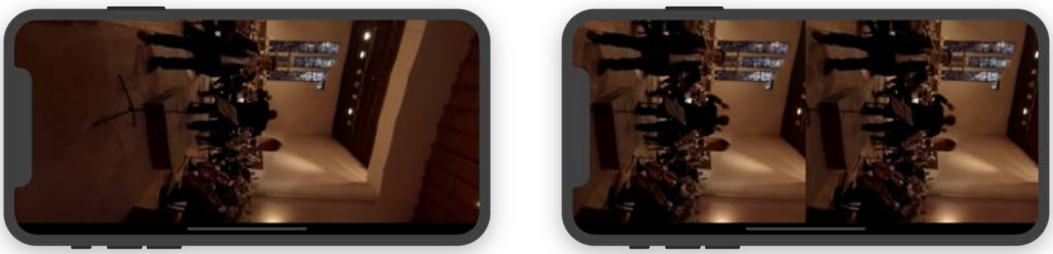

Figure 25: Landscape and Stereoscopic View

This option is defined in an if-else condition in the `startVideo()` function and will be called by the button action `@IBAction func activateStereo(_ sender: AnyObject)` to ensure a change between these views during playback. It compares the current view with its expectation. All local variables and instances are set in the `startVideo()` function. The camera angles are assigned with a starting position (Float = 0). And the nodes for rendering are added to their respective scenes.

The if-else condition distinguishes between the activation of the stereoscopic view and the landscape mode. If the program option is equal to `true`, left and right scenes are assigned to the camera properties and the `CoreMotion` sensor data. So the right scene gets its own display options and is not just a copy of the

left one. This is important for an independent video presentation. If stereoscopic view is equal to `false`, all associated properties are assigned to the left scene only.

A `fileprivate function` will manage the display scale of the individual device size. It even set constraints to the bounds of a target device. Depending on the stereoscopic view, the output scenes are projected as `SuperView` onto the existing elements. Functions declared as `fileprivate` are accessible from the same source file. It can only be accessed within their lexical range (Apple, 2017k, p. 792-821).

4.2.6 Audio Processing

The AVR has to render a synchronized MTB microphone array besides the video rendering. The array contains 16 microphones arranged by a circle, which simulates the human head (compare chapter 2). It uses omnidirectional electret condenser capsules (compare section 4.2.6) from Sennheiser (KE14) with a diameter of 14 mm (Fiedler, 2018, p. 7-8). Even the audio rendering is controlled by motion sensors for changing the microphones of the MTB array.

Various rendering algorithms of different qualities have been developed in a thesis by Roos, at TUBs. The control centre of the advocated one is the MTB Engine, it handles file receiving, signal processing, and file sending. These files are transmitted to a buffer after crossing a FFT and filter (Roos, 2011, pp 62-66).

Microphone Array

A microphone containing out of 16 pieces array was built at TUB, which allows a 360° recording. It uses permanently polarized electret microphones and is a subgroup of condenser acoustic transducer. The capsule contains a foil to simplify the schematics, because no external polarisation voltage has to be generated. The electrical charge is applied to heat the foil, which consists of Teflon, by electron shelling and remains frozen there after cooling down. These microphones especially capsules are a low cost variations of condenser acoustic transducer, mostly

used for telecommunication applications and target devices (Weinzierl et al., 2008, p. 326-327).

Microphones with the use of a permanently polarized electret foil have to be converted, because the phasing is either positive or negative. In case of the TUB internal microphone array, the phase of the capsule has to be turned because a negative output signal is generated by a positive pressure impulse. Figure 26 shows the configuration of this phasing. The sensitivity increases by 10 to 14 dB (Sennheiser, 2018).

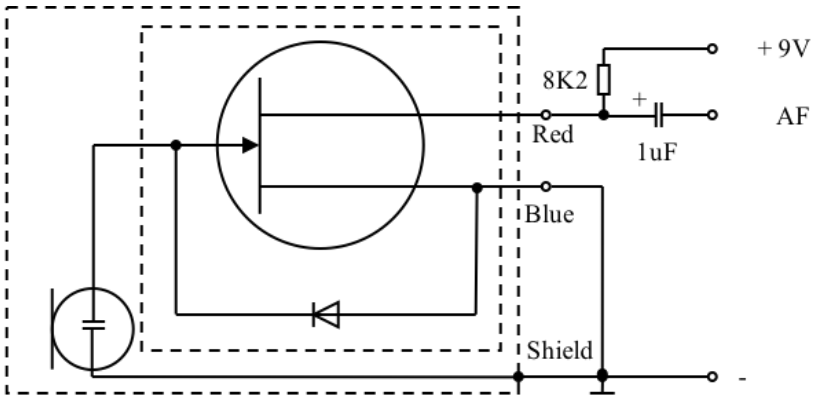

Figure 26: Amplifier configuration of electret capsules (compare Sennheiser (2018))

The individual angle of a microphone is calculated in formula 12. The angle calculates the product of a single angle with the number of total microphones and subtracts the product of one microphone. Otherwise the angle of individual microphone + 1 is calculated.

$$\alpha = \left(\frac{360°}{N_{Mics}} * Mic \right) - \frac{360°}{N_{Mics}} \tag{12}$$

Figure 27 shows the azimuth angle of the individual numbered microphones located in the array. It defines the horizontal angle based on 0° (north).

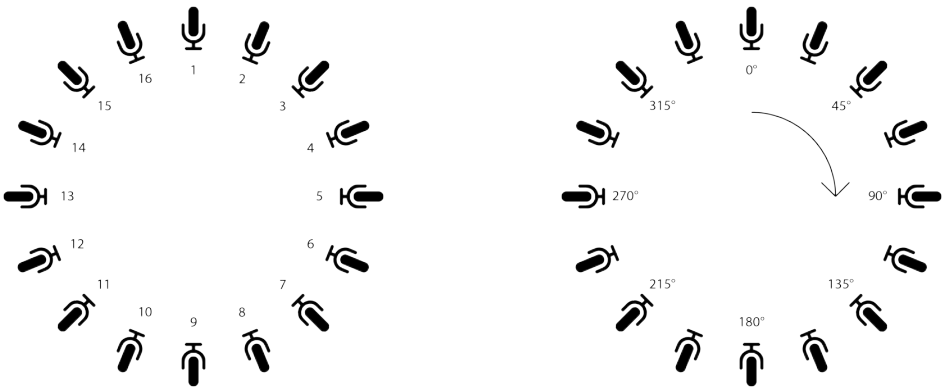

Figure 27: MTB Array with its indices and azimuth

Function `MicsAndAngles()` calculates based on formula 12 and scheme 27 the individual angle for various microphones in a `for-loop`. The loop defines `float` variables of the angle, individual ratio of left and right output, and the output itself. An `if-else` statement computes the values of left and right output depending on `TrackerX` value. The variables `ChR, ChL` gets the value of the individual channel depending on an observer to swap left and right output. The percentage is calculated, in case of a position between two microphones on each side. The individual microphone has an output impact depending on the percentage. Influences are higher, when the ears are inside of the individual microphone angle range.

Rendering Engine

The original MTB program code uses different libraries that are offered for various computer platforms. Most of these libraries are offered by Apple explicitly for iOS and Mac iOS development, such as vDSP. These allows efficient audio processing on the iOS layer (figure 14).

The audio playback in the AVR is based on `AV Foundation` framework and different Objective C classes, which are based on an already developed program code by Welbes in 2017. Figure 28 shows the principal algorithm of audio processing. The session is globally instantiated by the `AVRSession class`. A `static dispatcher` is defined, which returns a global concurrent queue by the system

with a specified quality of service class (Apple, 2018bh). The session is set to 0 for a defined state. The instance will return its `Global Instance` to the calling function.

Furthermore, the `AVRSession class` has three additional function which are offered for the audio processing. The `AVRSetup` sets the `AVRnstance` to a shared `AVAudioSession`, which acts as an intermediary between the AVR and the iOS (Apple, 2018ac). The `setup function` also defines the `instance category` to `AVSessionCategoryPlayback` for music content, `SampleRate = 48000 kHz` and `BufferTime = 0.005 ms` (Apple, 2018bb). Before the session can be started successfully, the interrupt and routing handler must be called. The interrupt handler consists of the interruption type, which reports a notification. If the interrupt can be terminated, the session will be set active again, if not, the session will be terminated. The routing handler also reports a notification and prints any changes.

The `AVREngine class` in `AVREngine.m`, defines an `AVAudioEngine` and connects the 16 channel nodes to `AVAudioPlayerNode`. In addition, two `BOOL` variables (Playing and audiolocal) are defined. The playing variable later checks whether the player is active or inactive. The `audiolocal` variable is set if the local files and not the downloaded files are to be played. The Playing variable is returned by the `playerRuns` function. To initialize the `AVREngine class` the `Init` is set as `id`. The value `self` is returned to the calling function. If `self != NIL`, the playing variable is set to false. This is basically important for interactions between audio engine and player.

The engine starts with the `callEngine` function. In this function, the `AVAudio Engine` is assigned to `Init`. For the different file paths there is a variable for each channel (`FileUrl01 - FileUrl16`). Depending on the user's selection, local or downloaded files are assigned to `FileUrl`, so that the files can be read into a buffer. Each audio node (`AVAudioPlayerNode`) is also assigned to the `init` and then added to the `audioEngine`.

Initializing the `AVAudioPlayerNode` enables the access to samples in the `AV AudioBuffer` at any time. This scheduling of `AVAudioPlayerNode` supports processes at a specific point by the `AVAudioFile`. The output format depends on the number of channels and buffers. If the output is not compliant, channels are deleted or added as required. A sample rate conversion can also be performed.

Therefore, the output rate should be selected file conform. Buffer playback is based on the implicit assumption that it operates at the same sampling rate as the node output. It is subject to the possibility that all previously scheduled buffers are disabled and sets the players timeline to 0. A second timeline is overlaid. This specifies the start time stamp of the player and stops intervals. When the buffer is emptied, the `completionHandler` is called to create a defined states. The result may also be 0. If required, the `completionHandler` can also be called before the start of rendering or playback (Apple, 2018bf).

The objects of an `AVAudio Engine` consist of various subclasses. The nodes always consist of input and output buses, which can be regarded as access points or connection points. For example, a mixer has several inputs, but only one output. The output formats consist of sampling rates and channel numbers. A connection between nodes requires the same formats. There are exceptions for `AVAudioMixerNode` and `AVAudioOutputNode` (Apple, 2018be).

`AVAudioFIle` opens a file for reading and writing, which processes the input format of `AVAudioPCMBuffer` as `AVAudioCommonFormat`, contained in the sample. It is the processing format, independent of the data input format. Read and write operations are always performed sequentially (Apple, 2018bc).

An `if condition` checks whether an error occurred while reading the file. If this happens, the error description is printed in the console. The audio file is assigned to an `AVAudioPCM` buffer. Depending on its properties, it is loaded into this buffer. That means its format and frame size depend on the file length. If this is known, the buffer is filled. An `if condition` also checks the buffering process and outputs of an error description. This reading and buffering is performed individually for each of the 16 channels.

When the files are read and buffered, each buffer is connected to a `AVAudio MixerNode`. This mixer can have various numbers of input. It accepts any sample rates and efficiently combines sample rate conversions (Apple, 2018bd). The channel nodes are connected to the individual buffer and a loop option, that the `startAVRPlayer` function can play all nodes and the playing variable becomes `YES`. The player is stopped by the engine and the variable is set to `NO`.

Parameter which are set by the users input are connected through the `AVR -Bridging-Header.h`. It contains the Objective-C header files of the `Audio Session`. For this interaction each channel has its own `pan and volume` value.

This link to the `AVR.swift` file is defined as the `avrManager:AVRManager!`. It is set as a global instance and becomes the engine in function `startAudio()`. Even the engine is loaded in this function.

The access of those functions are managed in the `MicsAndAngles()` function as well. An If condition checks, if the right ear is between microphone 1 and 9, so that `avrManager.panCh01 - avrManager.panCh08` be set to 1 (right) and -1 (left) for `avrManager.panCh09 - avrManager.panCh16`. If the left ear is between microphone 1 and 9, the left and right channels are switched.

Even the volume is managed by the `avrManager`. Depending on the location of the right and left position, the individual channels will be fadeIn, fadeOut, set to 1 or 0. For a defined start of the volume all values are set to 0 at the beginning and are only faded in and out after the first movement of the rotary slider, internal or external tracker.

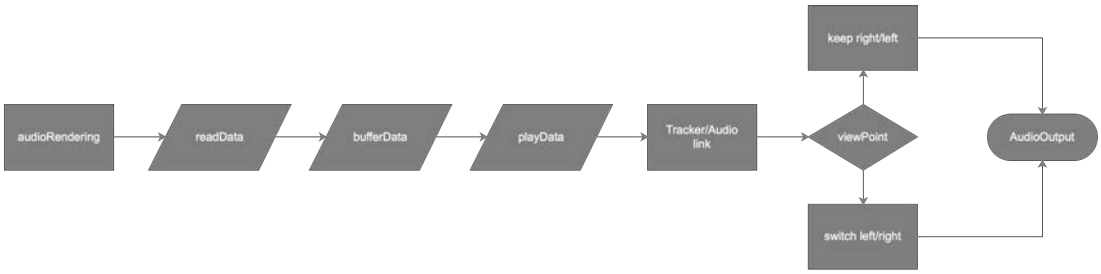

Figure 28: Flowchart of AudioProcessing

4.3 Discussion

As described in chapter 4.1, the streaming server offers the audio and video content to the application, and a BLE connection offers an outstanding remote control for head movements. Even 360° videos are processed by a rendering loop (see chapter 4.2.5).

The necessity to develop a streaming server is because the Audio Connection Jack is no longer available for iOS. It also means that the Audio Connection

Jack buffers used, by the algorithm. could not be applied for this application. Even a higher latency is expected, because the target device has to play all audio channels independently to each other. In the case of communication with an Audio Connection Jack, it plays the channels and streams them over to the network or locally on the computer. Once the data has been downloaded and rendered, buffers can be filled.

Continuing programming is necessary to figure out the wrapping of the MTB algorithm. It requires a communication between the Swift audio players and the original C++ code. That means a buffer must be filled for each player for reading it by the MTB algorithm.

That means the programming languages C++, Objective-C, C, and Swift, are compatible with each other. To allow functions to communicate between it and thus to merge them. Swift cannot communicate directly with C++. An Objective-C class must be developed that performs the exchange. In case of the MTB algorithm, the application must use the internal headers or buffers and make them available to the C++ MTB algorithm.

The migration of the code revealed that the Xcode Linker could not translate the code and thus make it readable. As a result of the development, the error output that symbols of the architecture x86_64 could not be found:

```
clang: linker command failed with exit code 1 (use -v to see
                        invocation)
```

It is recommended to merge both software projects in addition of a bridging header, which could be created automatically, as soon as C++ or Objective-C code is added. In this header the C++ libraries are made known to the Swift project. In order for the C++ program code to communicate with the Swift classes, they must be added with "AVR-SWIFT/AVR-SWIFT.h".

Different solutions, like the `Other Linker Flags` release values change to `BUILT_PRODUCTS_DIR/libCordova.a`, or the `Build Active Architecture Only` are set to `NO`. The solutions did not make the code translatable. Therefore, the assumption is that missing libraries causing the error what has to be considered in further work.

The `libc++.dylib` must be added to read C++ code in a Swift environment. It is a standard C++ library to translate such program code. Adding this library didn't solve the problem either.

A class to pass, simple integer values was created temporarily to check that the wrapping principle works. After successful testing, an attempt was made to access the C++ project classes based on this description. In almost all cases the `clang error` (see above) was produced again.

In order for a better implementation of the rendering of the MTB algorithm, it is necessary to figure out how the Apple headers can replaced the third party libraries for a bug free calculation of the audio output.

During a non-scientific test listening it was noticed that the audio processing has a higher runtime with increasing the used channels. This led to the effect that the right output channel was heard earlier than the left. As a result, no spatial perception can take place.

Furthermore, the effect was generated by acoustic artefacts that are audible when the microphones change angle-dependently. To eliminate this effect, the audio processing was switched to the basic algorithm, so that every microphone outside its effective range (22.5 °) is switched off hard.

This means that they can be heard in their respective angle ranges. At a viewing angle of 0-180°, microphones number 1-8 are switched to the left, and 9-16 to the right channel. If the viewing direction changes higher than 180°, the channels are switched between left and right. The newly developed 16 audio channel player is based on the `AVAudioPlayer class` and is declared class wide. The players are called in the local function for reproducing the stores files or network function in order to play the downloaded files. With this kind of development the artefacts, and runtime differences between right and left channel are eliminated. A spatial hearing can thus be experienced.

In order to be able to integrate the extended algorithms correctly, it is recommended to develop an individual audio unit for each algorithm.

Audio Units enables complex and sophisticated audio processing functions that are processed in a sub-application. It serves as an application extension that sends an internal stream to the processing application and return. These extension

points provide APIs enables mutual communication. In summary, AudioUnits are stand-alone programs that supplement to applications (Apple, 2018ba).

5 Conclusion

In conclusion, the MTB system is a reproduction of a room with acoustical information of direction, sound pressure, and room response. It provides the possibility to simulate the acoustical situation, depending on the head movement. The mixture of sounds depends on the angle of the listener towards the source (Algazi et al., 2004).

A possible virtual reconstruction of the real environment with the connection of spatial audio and video processing is demonstrated with work. Modern smartphones are able to perform complex processings with low latency and high immersion.

From a visual point of view, perception is the simulation of a virtual environment that allows a realistic reproduction of a pre-recorded situation, with a certain perceptibility of the technical devices, such as pix-elated displays, reproduction, or motion latencies. The display quality is particularly noticeable when using VR glasses, as the built-in lenses represent a zoom into the display and thus a change in focus is generated. This happens despite the possibility of high definition playback. If one uses the application without such glasses, thus in landscape mode, the video quality is indeed the same, but there is no optical zoom into the display. However, the lenses in VR glasses are necessary in order to be able to adjust individual sharpness. This also changes the angle to the image. Thus a lower immersion is created when wearing VR glasses. For the use of a cardboard the internal sensor evaluation is unavoidable, because all three axes must be evaluated.

The frameworks offered by Apple, to access the devices hardware, such as `CoreBluetooth`, considering the BLE protocol stack, or `CoreMotion`, provides a stable and high-performance connection to read or transmit motion data.

Audio control via one external and two internal connections is possible, however, with the external tracker it is important to note that a more modern BLE chip and a high-performance microcontroller is used, such as the `Adafruit HUZZAH32 - ESP32 Feather Board` with built-in BLE module, so that latencies can be kept low. The used tracker has relatively high latencies, because a fast angle change shows up in the application with a delayed movement of the vir-

tual head. It is also recommended that the internal sensors are evaluated, as of the latencies occurrence, which are not to be taken into account. Fast and slow movements do not lead to a delayed movement.

The server-based data provision via streaming protocols was also successfully demonstrated. At the end, it is possible to stream several audio and video files, but it makes sense to download the files completely when starting the application, as a data of about 700 megabyte could generate big traffic. Therefore, the amount of data happens, as well as because of UDP, due to no reception guarantee.

The necessity to develop a new modified rendering is that the source code of Sebastian Roos is based on the Audio Connection Jack, which is no longer supported for iOS. Therefore, the data provision is download based or local. This has the disadvantage that the data has to be played back on the device, and not as in the original algorithm, in the Audio Connection Jack, therefore, the algorithm only has to communicate with the player directly.

In conclusion, it is advisable to use Apple's vDSP libraries as they are optimized for iOS. There are conflicts for the buffering of all 16 channels during runtime, because the `AudioBufferList` did not play files synchronously. This is because the array of the `AudioBufferList` has one position and the samples of the files are loaded in multiple numbers into one memory of the array.

The merging of AVR application and MTB algorithm did not work as expected because the communication via bridging header did not work. Thus, several errors were generated, which could not be solved in different approaches.

One possibility for a MTB migration can be that the algorithm is completely newly developed and implemented in Swift. This requires the use of various frameworks to calculate interpolations, discretions, cut-off frequencies, or FFT. For this signal processing, Apple offers the vDSP and Accelerate Framework. The migration of my implementation of the MTB algorithm was based on the already used libraries. However, the wrapping went wrong and an alternative had to be developed.

In order to maintain clarity in the Xcode, all non-functional software parts have been removed and the rendering engine is using `FadeIn` and `FadeOut`. This leads to the fact that a certain latency or overlay can be heard during the transition. This may be due to the fact that the audio files are not read synchronously from the PCM buffer depending on the process.

In summary, this project shows that the virtual environment can be simulated on an iPhone or iPad, with the connection of external motion sensors and a streaming service or download.

Appendix

Configure the Raspberry Pi

1. Download the latest OS.
   `https://www.raspberrypi.org/downloads/raspbian/`
2. Take a Micro SD Card (at least 8GB) for the Raspberry Pi version 3.
3. Unzip the downloaded Image.
4. Open the Terminal.
5. List all Memories.
   `$ diskutil list`
6. Select the right disk (depending on its memory capacity)
7. Unmount the selected disk
   `$ sudo diskutil unmountDisk /dev/diskX` (X depends on the disk)
8. Create the boot disk.
   NOTE: Replace FOLDER through the folder where Raspbian OS is stored.
   NOTE: Replace RASPIANNME.img into the actual Filename.
   NOTE: Replace the X through your memory number.
   NOTE: It needs around 10 minutes without any feedback.
   `$ sudo dd bs=1m if=FOLDER/RASPBIANNAME.img of=/dev/rdiskX`
9. Eject the disk (X depends on the disk).
   `$ sudo diskutil eject /dev/diskX`
10. Remove the disk on your Macintosh.
11. Create an empty file `ssh.txt` in `boot`.
12. Create a `wpa_supplicant.conf` file with the following content in `boot`.

```
network={
ssid="Your SSID-Name"
psk="Your WIFi Password"
key_mgmt=WPA-PSK}
```

13. Or use a LAN cable connected to your Router.

14. Insert the SD Card into the Raspberry Pi

15. Start the Raspberry Pi

16. Log in (Passwort: raspberry)
    `ssh pi@raspberrypi.local`

17. Update the system.
    `sudo apt-get update`

18. Upgrade the system.
    `sudo apt-get upgrade`

Installing NGINX Streaming Server

NOTE: Please use the root user and type `sudo su` after log in.

19. Clone RTMP for NGINX.
    `$ git clone https://github.com/sergey-dryabzhinsky/nginx-rtmp-module.git`

20. Install NGINX dependencies.
    `$ apt-get install build-essential libpcre3 libpcre3-dev libssl-dev`

21. Download NGINX source code.
    `$ wget http://nginx.org/download/nginx-1.14.0.tar.gz`

22. Unzip the source code and enter the NGINX directory.
    `$ tar -xf nginx-1.14.0.tar.gz`

23. Move to the downloaded folder.
    `$ cd nginx-1.14.0`

24. Build NGINX (Please don't copy and paste. Type the first line!).
    ```
    $ ./configure --with-http_ssl_module --add-module=../nginx-rtmp-module
    $ make -j 1
    $ make install
    ```

25. Start the server.
    `$ /usr/local/nginx/sbin/nginx`

26. Make sure NGINX is running in your browser.
    `http://Server IP address`

27. Copy the nginx.conf file from the data stick to the default path of the web server.
    `$ cp -a /media/pi/USB/files /mnt/hls/`

28. Restart NGINX.
    `$ cp /media/pi/USB/nginx.conf /usr/local/nginx/conf/`

29. Copy media files to `/mnt/hls/` folder.

30. Restart NGINX. `$ /usr/local/nginx/sbin/nginx -s stop`
    `$ /usr/local/nginx/sbin/nginx`

31. Update the OS.
    `$ apt-get update`

32. Install the multimedia keyring.
    `$ apt-get install deb-multimedia-keyring`

33. Update again.
    `$ apt-get update`

34. Install FFMPEG.
    `$ apt-get install ffmpeg`

35. Update again.

36. Restart NGINX. `$ /usr/local/nginx/sbin/nginx -s stop`
    `$ /usr/local/nginx/sbin/nginx`

References

Algazi, V. Ralph; V. Ralph Algazi; and V. Ralph Algazi (2004): "Motion Tracked Binaural Sound." In: *J. Audio Eng. Soc*, **52**(11), pp. 1142–1156.

Algazi, V. Ralph; et al. (2005): "Immersive spatial sound for mobile multimedia." In: *Journal of the Audio Engineering Society*, **52**(11), pp. 1142–1156.

Apple (1997): *Model-View-Controller*. URL `https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html`. Access May 2nd 2017.

Apple (2017a): URL `https://www.apple.com/chde/iphone-x/specs/`.

Apple (2017b): *About Core Bluetooth*. URL `https://developer.apple.com/library/content/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html`. Access Nov. 13, 2017.

Apple (2017c): *CBCentralManagerDelegate*. URL `https://developer.apple.com/documentation/corebluetooth/cbcentralmanagerdelegate`. Access November 13th 2017.

Apple (2017d): *CBPeripheralDelegate*. URL `https://developer.apple.com/documentation/corebluetooth/cbperipheraldelegate`. Access November 13th 2017.

Apple (2017e): *didDiscoverCharacteristicsFor*. URL `https://developer.apple.com/documentation/corebluetooth/cbperipheraldelegate/1518821-peripheral`. Access November 21st 2017.

Apple (2017f): *didDiscoverServices*. URL `https://developer.apple.com/documentation/corebluetooth/cbperipheraldelegate/1518744-peripheral`. Access November 21st 2017.

Apple (2017g): *Displays*. URL `https://developer.apple.com/library/archive/documentation/DeviceInformation/Reference/iOSDeviceCompatibility/Displays/Displays.html`. Access July 2nd 2018.

Apple (2017h): *Framework Accelerate*. URL https://developer.apple.com/reference/coremotion. Access March 23th 2017.

Apple (2017i): *isHidden*. URL https://developer.apple.com/documentation/uikit/uiview/1622585-ishidden. Access November 14th 2017.

Apple (2017j): *The Role of View Controllers*. URL https://developer.apple.com/library/content/featuredarticles/ViewControllerPGforiPhoneOS/index.html#//apple_ref/doc/uid/TP40007457-CH2-SW1. Access 2017.

Apple (2017k): *The Swift Programming Language*. 4.1. Apple.

Apple (2017l): *UIViewController*. URL https://developer.apple.com/documentation/uikit/uiviewcontroller. Access 2017.

Apple (2018): *NSFileManager*. URL https://developer.apple.com/documentation/foundation/nsfilemanager?language=occ. Access September 24.

Apple (2018aa): *Accelerate Framework*. URL https://developer.apple.com/documentation/accelerate. Access July 17th 2018.

Apple (2018ab): *AV Foundation*. URL https://developer.apple.com/library/content/documentation/AudioVideo/Conceptual/AVFoundationPG/Articles/00_Introduction.html#//apple_ref/doc/uid/TP40010188. Access May 8th 2018.

Apple (2018ac): *AVAudioSession*. URL https://developer.apple.com/documentation/avfoundation/avaudiosession?changes=_3. Access July 3rd 2018.

Apple (2018ad): *AVPlayerViewController*. URL https://developer.apple.com/documentation/avkit/avplayerviewcontroller. Access May 3rd 2018.

Apple (2018ae): *CMMotionManager*. URL https://developer.apple.com/documentation/coremotion/cmmotionmanager. Access May 4th 2018.

Apple (2018af): *deviceMotionUpdateIntervall*. URL https://developer.apple.com/documentation/coremotion/cmmotionmanager/1616065-devicemotionupdateinterval. Access May 4th 2018.

Apple (2018ag): *Foundation*. URL https://developer.apple.com/documentation/foundation. Access May 8th 2018.

Apple (2018ah): *HTTP Streaming*. URL https://developer.apple.com/library/content/documentation/NetworkingInternet/Conceptual/StreamingMediaGuide/HTTPStreamingArchitecture/HTTPStreamingArchitecture.html#//apple_ref/doc/uid/TP40008332-CH101-SW2. Access May 13th 2018.

Apple (2018ai): *Interface Builder Built-In*. URL https://developer.apple.com/xcode/interface-builder/. Access May 7th 2018.

Apple (2018aj): *renderer(didApplyAnimationsAtTime)*. URL https://developer.apple.com/documentation/scenekit/scnscenerendererdelegate/1523038-renderer. Access May 3rd 2018.

Apple (2018ak): *renderer(updateAtTime)*. URL https://developer.apple.com/documentation/scenekit/scnscenerendererdelegate/1522937-renderer. Access May 3rd 2018.

Apple (2018al): *SceneKit*. URL https://developer.apple.com/documentation/scenekit. Access May 3rd 2018.

Apple (2018am): *SCNAction*. URL https://developer.apple.com/documentation/scenekit/scnaction. Access May 3rd 2018.

Apple (2018an): *SCNActionable*. URL https://developer.apple.com/documentation/scenekit/scnactionable. Access May 3rd 2018.

Apple (2018ao): *SCNNode*. URL https://developer.apple.com/documentation/scenekit/scnnode. Access May 4th 2018.

Apple (2018ap): *SCNScene*. URL https://developer.apple.com/documentation/scenekit/scnscene. Access May 4th 2018.

Apple (2018aq): *SCNSceneRendererDelegate*. URL https://developer.apple.com/documentation/scenekit/scnscenerendererdelegate. Access May 3rd 2018.

Apple (2018ar): *SKCropNode*. URL https://developer.apple.com/documentation/spritekit/skcropnode?changes=_5. Access June 20th 2018.

Apple (2018as): *SpriteKit*. URL https://developer.apple.com/documentation/spritekit. Access May 8th 2018.

Apple (2018at): *startDeviceMotionUpdates*. URL https://developer.apple.com/documentation/coremotion/cmmotionmanager/1616110-startdevicemotionupdates?language=occ. Access May 4th 2018.

Apple (2018au): *UIGestureRecognizerDelegate*. URL https://developer.apple.com/documentation/uikit/uigesturerecognizerdelegate. Access May 3rd 2018.

Apple (2018av): *UIKit*. URL https://developer.apple.com/documentation/uikit. Access May 8th 2018.

Apple (2018aw): *WKWebView*. URL https://developer.apple.com/documentation/webkit/wkwebview. Access May 3rd 2018.

Apple (2018ba): *App Extensions Increase Your Impact*. URL https://developer.apple.com/library/archive/documentation/General/Conceptual/ExtensibilityPG/index.html#//apple_ref/doc/uid/TP40014214. Access November 30th 2018.

Apple (2018bb): *Audio Session Categories and Modes*. URL https://developer.apple.com/library/archive/documentation/Audio/Conceptual/AudioSessionProgrammingGuide/AudioSessionCategoriesandModes/AudioSessionCategoriesandModes.html. Access October 5th.

Apple (2018bc): *AVAudioFile*. URL https://developer.apple.com/documentation/avfoundation/avaudiofile. Access October 8th.

Apple (2018bd): *AVAudioMixerNode*. URL https://developer.apple.com/documentation/avfoundation/avaudiomixernode. Access October 5th.

Apple (2018be): *AVAudioNode*. URL https://developer.apple.com/documentation/avfoundation/avaudionode. Access October 9th.

Apple (2018bf): *AVAudioPlayerNode*. URL `https://developer.apple.com/documentation/avfoundation/avaudioplayernode`. Access October 8th.

Apple (2018bg): *Core Audio*. URL `https://developer.apple.com/documentation/coreaudio?changes=_8`. Access July 25th 2018.

Apple (2018bh): *dispatch get global queue*. URL `https://developer.apple.com/documentation/dispatch/1452927-dispatch_get_global_queue?language=objc`. Access October 5th.

Arutyunyan, Roman (2017): *NGINX-based Media Streaming Server*. URL `https://github.com/arut/nginx-rtmp-module`. Access June 20th 2018.

Azuma, Ronald T.; et al. (1997): "A Survey of Augmented Reality." In: *Teleoperators and Virtual Environments*, **6**(4), pp. 2–34.

Bebis, George et al. (2016): "Advanced in Visual Computing." In: Tobias Isenberg (Ed.) *Proceedings Part I*, vol. 1. International Symposium on Visual Computing, Las Vegas, Nevada, USA: Springer Verlag.

Bosch (2016): *BNO055 Intelligent 9-axis absolute orientation sensor*. Bosch Sensortec GmbH, Gerhard-Kindler-Strasse 9, 72770 Reutlingen, 1.4.

Brill, Manfred (2008): *Virtuelle Realität*. Fachhochschule Kaiserslautern, Fachbereich Informatik und Mikrosystemtechnik, Amerikastraße 1, 66482 Zweibrücken: Springer Verlag.

Card, S.K.; T. Moran; and A. Newell (1986): "The Model Human Processor: An Engineering Model of Human Performance. Handbook of Perception and Human Performance." In: *Cognitive Processes and Performance*, **2**.

Cook, Nate (2014): *CMDeviceMotion*. URL `http://nshipster.com/cmdevicemotion/`. Access October 28th, 2014.

Daniel, Peter; et al. (2007): "Kunstkopftechnik - Eine Bestandsaufnahme." In: *ACUSTICA/acta acustica/Nuntius Acustucus*, **93**(1), p. 1.58.

Dickreiter, Michael; et al. (2014): In: Michael Dickreiter (Ed.) *Handbuch der Tonstudiotechnik*, vol. 2. De Gruyter Saur, p. 362.

Dörner, Ralf; Wolfang Broll; Paul Grimm; and Bernhard Jung (2013): *Virtual und Augmented Reality (VR/AR) - Grundlagen und Methoden der Virtuellen und Augmentierten Realität*, vol. 1. Berlin, Heidelberg, New York: Springer Verlag.

Ernst, Marc. O (2008): "Multisensory Integration: A Late Bloomer." In: *Current Biology*, **18**(12), pp. R519–R521.

FFmpeg (2018): *FFmpeg*. URL `https://www.ffmpeg.org/ffmpeg.html`. Access October 13th.

Fiedler, Felicitas (2018): *A motion-tracked binaural microphone array for recording spatial sound*. Masterthesis, Technische Universität Berlin, Straße des 17. Juni 135, D-10623 Berlin.

Fieldling, R.; UC Irivine; et al. (1999): *Hypertext Transfer Protocol - HTTP/1.1*. The Internet Society.

Gomez, Carles; et al. (2012): "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-Power Wireless Technology." In: *Sensors*, **12**(9), pp. 11734–11753. URL `http://www.mdpi.com/1424-8220/12/9/11734/htm`.

Hansch, Pierre and Christian Rentschler (2012): *Emotion@Web - Emotionale Websites durch Bewegtbild und Sound-Design*, vol. 1. Auflage. Springer Vieweg.

Heilig, Morton (1955): "The Cinema of the Future." In: .

JACK, Team (2013): "JACK on iOS 7." URL `http://www.crudebyte.com/jack-ios/ios7/`.

JACK, Team (2014): *JACK-AUDIO-CONNECTION-KIT*. Jack Audio, 0.124.0.

Kremenek, Ted (2017): *Swift 4.0 Released!* URL `https://swift.org/blog/swift-4-0-released/`. Access September 19, 2017.

Lindau, Alexander (2009): "The Perception of System Latency in Dynamic Binaural Synthesis." In: *Fortschritte der Akustik: Tagungsband der 35. DAGA*.

Lindau, Alexander; et al. (2007): "Binaural resynthesis for comparative studies of acoustical environments." In: *Proc. of the 122nd AES Convention*.

Loesche, Dyfed (2017): *The Smart Device That Made Apple Great*. URL `https://www.statista.com/chart/7477/apple-revenue-by-product-group/`. Access January 09, 2017.

Macpherson, E.; et al. (2002): "Listener weighting of cues for lateral angle: The duplex theory of sound localization revisited." In: *The Journal of the Acoustical Society of America*, **111**(5), pp. 2219–2236.

Maier, Dennis (2010): *Bluetooth Low Energy - Introducing iBeacon*. Technische Universität München, Arcisstraße 21, 80333 München.

Meinel, Christoph and Harald Sack (2009): *Digitale Kommunikation - Vernetzen, Multimedia, Sicherheit*. Springer Verlag.

Melzer, James E. (2001): "Head-Mounted Displays." In: *CRC Press LLC*.

Møller, Henrik (1992): "Fundamentals of Binaural Technology." In: *Applied Acoustics*, **36**(3-4), pp. 171–218.

Nagata, Takashi; Mitsumasa; Koyanagi; Hisao Tsukamoto; et al. (2012): "Depth Perception from Image Defocus in a Jumpin Spider." In: *Science*, **335**, pp. 469–471.

Nedelcu, Clément (2010): *Nginx HTTP Server*, vol. 1. Olton, birmingham, b27 6pa. 32 Lincoln Road: PACKT Publishing.

Nier, Hedda (2017): *Die Erfolgsgeschichte des iPhones*. URL `https://de.statista.com/infografik/7468/absatz-von-apple-iphones/`. Access January 09, 2017.

Paech, Anne (2018): *Das Aroma des Kinos Filme mit der Nase gesehen: Vom Geruchsfilm und Düften und Lüften im Kino*. URL `http://www.filmportal.de/sites/default/files/EBD16B727A094212B735E2E26A42331A_mat_paech_aroma.pdf`.

Papula, Lothar (2009): *Mathematik für Ingenieure und Naturwissenschaftler*, vol. Band 1. 12. GWV Fachverlage GmbH, Wiesbanden: Vieweg + Teubner.

Ragonese, Nicholas (2016): *CircleSlider*. URL `https://github.com/nichrago/CircleSlider`. Access May 4th 2018.

Reenskaug, Trygve (1979): “Models - View - Controllers.”

Roos, Sebastian (2011): *Implementierung und Evaluation eines Aufnahme- und Wiedergabesystems nach dem Motion-Tracked Binaural Verfahren.* Master’s thesis, Technische Universität Berlin, Straße des 17. Juni 135 in Berlin.

Rosen, Kathleen (2008): “The history of medical simulation.” In: *Journal of Critical Care*, **23**, pp. 157–166.

Schumann, Matthias; et al. (2010): *Webbasierte Anwendungen als Lösungsansatz für die Heterogenität im mobilen Internet.* Tech. Rep. 3, Georg-August-Universität Göttingen, Wilhelmsplatz 1, 37073 Göttingen.

Sennheiser (2018): *Facts About Electret Wiring.* Sennheiser, One Enterprise Drive - PO Box 987 - Old Lyme, CT 06371.

Singh, Harwinder (2016): “Speed Performance Between Swift and Objective-C.” In: *Internation Journal of Engineering Applied Science and Technology*, **1**(10), pp. 185–189.

SPI (1997 - 2018): “Paket: libssl-dev.” URL `https://packages.debian.org/jessie/libssl-dev`.

SPI (1997-2018a): *Paket: libpcre3.* URL `https://packages.debian.org/de/sid/libpcre3`. Access October 13th.

SPI (1997-2018b): *Paket: libpcre3-dev.* URL `https://packages.debian.org/de/sid/libpcre3-dev`. Access October 13th.

Stäuble, Markus (2009): *Programmieren fürs iPhone*, vol. 1. Ringstraße 19B, 96115 Heidelberg: dpunkt verlag.

Steickart, H. (1988): “Kopfbezogene Stereophonie - Neuere Erfahrungen bei Produktionen und Rezeption.” In: *Transkript zum Vortrag anläßlich der 15. Tonmeistertagung Mainz*, pp. 314–316.

Strutt, J.W. (1904): “On the acoustic shadow of a sphere.” In: *Hilosophical Transactions of the Royal Society of London*, **203**(A), pp. 87–110.

Swiniarski, Arthur (2016): “Simple 360 Video player for iOS.” URL `https://github.com/Aralekk/simple360player_iOS`.

Teensy (2017): *Teensy USB Development Board*. URL `https://www.pjrc.com/teensy/index.html`.

Thomeczek, Sebastian (2016): "Echtzeitanforderungen an Virtual Reality Systeme – Interaktive Anwendungen mit sechs Freiheitsgraden." In: *Informatik aktuell - Internet der Dinge - Echtzeit 2016*, **1**, pp. 101 – 106.

Warnecke, H.-J. and H.-J. Bullinger (1994): *Virtual Reality '94 Anwendungen und Trends*, vol. T42. Institutszentrum der Fraunhofer-Gesellschaft Stuttgart-Vaihingen: Springer Verlag.

Weinzierl, Stefan; et al. (2008): *Handbuch der Audiotechnik*. Berlin, Heidelberg, New York: Springer Verlag.

Weinzierl, Stefan; et al. (2009): "On the Spatial resolution of virtual acoustic environments for head movements in horizontal vertical and lateral." In: *Proc. of the EAA Symposium on Auralization*.

Welbes, William (2017): "CoreAudioMixer." URL `https://github.com/welbesw/CoreAudioMixer`.

Welch, G.; et al. (2002): "Motion tracking: No silver bullet, but a respectable arsenal." In: *IEEE Computer Graphics and Applications*, **22**(6), pp. 24–38.

Werner, Martin (2005): "Internet." In: Martin Werner (Ed.) *Netze, Protokolle, Schnittstellen und Nachrichtenverkehr*, 1, chap. 6. Vieweg, pp. 112–134.

Zhu, Xinglei; et al. (2007): "Real-Time Signal Estimation Form Modified Short-Time Fourer Transform Magnitude Spectra." In: *IEEE Transactions On Audio, Speech And Language Processing*, **15**(5), pp. 1645–1653.