

## **Bachelor Thesis**

# **„swarm off sound“**

Conception, planning and realization of an interactive sound installation

Submitted by:

Sebastian Obel (912707)

In Fulfillment for Requirements

For Bachelor of Engineering (B. Eng.)

Course of Study:

Theater- und Veranstaltungstechnik und -Management

Supervision: Prof. Dr. Alexander Lindau

Diplom Designer Carsten Stabenow

Review: Magister Artium Alois Späth

Submitted: 29.08. 2024

## **Abstract**

This thesis focuses on the artistic conception and development of the interactive sound installation *swarm of sound* which allows participants to shape soundscapes through their movements. Aim of this work is to show the connections between the artistic concept and the technical solutions, utilizing mainly open-source resources. A secondary goal is to explore the limitations and boundaries given by the artistic concept as intentional tools in a creative process.

The projects development included the analysis and realization of the artistic concept, the identification of problems and solutions for technical requirements through open-source resources. Test setups were used to validate the concept functionality and interactivity of the sound installation.

It was demonstrated that an immersive experience could be achieved without relying on expensive professional equipment and that those constraints could create unintended ways of interactions between participants, technology, and the installation.

## **Kurzfassung**

Diese Arbeit beschäftigt sich mit der künstlerischen Konzeption und Entwicklung der interaktiven Klanginstallation *swarm of sound*, die es den Teilnehmer:innen ermöglicht, durch ihre Bewegungen Klanglandschaften zu gestalten. Ziel dieser Arbeit ist es, die Verbindungen zwischen dem künstlerischen Konzept und den technischen Lösungen aufzuzeigen, wobei hauptsächlich Open-Source-Ressourcen verwendet werden. Ein sekundäres Ziel ist es, die Beschränkungen und Grenzen zu erforschen, die durch das künstlerische Konzept gegeben sind und diese als bewusste Werkzeuge im kreativen Prozess einzusetzen.

Die Projektentwicklung umfasste die Analyse und Umsetzung des künstlerischen Konzepts, die Identifizierung von Problemen und Lösungen für technische Anforderungen durch Open-Source-Ressourcen. Die Funktionalität des Konzepts und die Interaktivität der Klanginstallation wurden anhand von Testaufbauten überprüft.

Es wurde gezeigt, dass immersive Erfahrungen auch ohne teure professionelle Technik möglich sind und dass diese Einschränkungen unbeabsichtigte Interaktionen zwischen den Teilnehmern, der Technologie und der Installation schaffen können.

## Table of Contents

<b>I.</b>	<b>List of Figures .....</b>	<b>IV</b>
<b>II.</b>	<b>List of abbreviations .....</b>	<b>VI</b>
<b>III.</b>	<b>List of Symbols .....</b>	<b>VIII</b>
<b>IV.</b>	<b>List of formulas.....</b>	<b>VIII</b>
<b>1</b>	<b>Introduction.....</b>	<b>1</b>
<b>1.1</b>	<b>Overview.....</b>	<b>1</b>
<b>1.2</b>	<b>Motivation and Inspiration .....</b>	<b>1</b>
<b>2</b>	<b>Theoretical Framework .....</b>	<b>2</b>
<b>2.1</b>	<b>Sound Perception and Basic Panning Fundamentals .....</b>	<b>2</b>
2.1.1	Human Perception of Sound .....	2
2.1.2	Stereophonic Panning laws.....	3
<b>2.2</b>	<b>Digital Image Processing and Object Detection/Tracking Methods ....</b>	<b>5</b>
2.2.1	Digital Image Processing Fundamentals.....	5
2.2.2	Traditional Object Detection using Image Processing .....	8
2.2.3	Traditional Object Tracking using Image Processing.....	9
2.2.4	Object Detection and Tracking Methods Based on Deep Learning .....	9
<b>2.3</b>	<b>Network Communication and Protocols .....</b>	<b>10</b>
2.3.1	The TCP Protocol and its Application .....	11
2.3.2	The UDP-Protocol and its Application.....	11
2.3.3	The OSC content format and its Message Structure .....	13
<b>2.4</b>	<b>An Overview on Single-Board Computers .....</b>	<b>14</b>
<b>2.5</b>	<b>Software solutions for Audio-Visual Applications .....</b>	<b>15</b>
2.5.1	TouchDesigner.....	15
2.5.2	Max / RNBO .....	15
2.5.3	Pure Data .....	16
<b>2.6</b>	<b>Audio-Visual Art, Sound Art and the Connection to Interactivity .....</b>	<b>17</b>
2.6.1	Audio-Visual Art .....	17
2.6.2	Sound Art .....	17
2.6.3	Interactivity between Audience and Sound Art .....	17
<b>3</b>	<b>Identifying the Technical Approach for the Sound Installation .....</b>	<b>19</b>
<b>3.1</b>	<b>Requirements Analysis Based on the Artistic Concept.....</b>	<b>19</b>
<b>3.2</b>	<b>Ideation for Solutions Based on the Requirements .....</b>	<b>20</b>
<b>3.3</b>	<b>Finalizing the Technical Approach .....</b>	<b>21</b>
3.3.1	Detection and Tracking Approach.....	21
3.3.2	Solution for Sound Generation and D/A conversion .....	22
3.3.3	Amplification and Sound Reproduction Solution .....	22
3.3.4	Enabling Audience Engagement and Interaction .....	23
3.3.5	Signal Flow Diagram .....	24

<b>4</b>	<b>Development, Evaluation and Implementing Improvements of the Sound Installation .....</b>	<b>24</b>
<b>4.1</b>	<b>Approach for the Object Tracking Algorithm.....</b>	<b>24</b>
4.1.1	Choice of Camera Integration .....	25
4.1.2	Performing Image Processing to Enhance Tracking Capability .....	26
4.1.3	Implementing Traditional Object Detection .....	28
4.1.4	Implementing Traditional Object Tracking.....	29
4.1.5	Implementing a Calibration Method .....	29
<b>4.2</b>	<b>Control Data Processing for Human Computer Interaction .....</b>	<b>31</b>
4.2.1	Clear Assignment of Tracked Participants to the Sound Elements.....	32
4.2.2	Projecting the Sound Element to the Participants Location .....	32
4.2.3	Control Data for Interaction with Audio Effects .....	33
4.2.4	Transferring the Control Data from the RPI to the Bela .....	33
<b>4.3</b>	<b>Performing Audio Processing and Sound Design .....</b>	<b>35</b>
4.3.1	Receiving and Processing of the Control Data .....	35
4.3.2	Implementing Surround Panning for the Sound Element.....	36
4.3.3	Creating a Sound Design Framework .....	39
<b>4.4</b>	<b>D/A conversion, Speaker Array, and Amplification.....</b>	<b>39</b>
4.4.1	Implementing the Digital to Analog Conversion .....	39
4.4.2	Developing a Custom Speaker Design for the 8-Channel Array.....	40
<b>4.5</b>	<b>Evaluating and Improving the Technical Realization.....</b>	<b>41</b>
4.5.1	Assessing Key Functions Through a First Test Setup .....	41
4.5.2	Improvement Implementation based on the First Test Setup .....	43
4.5.3	Assessing the Improvements Through a Second Test Setup.....	45
<b>5</b>	<b>Discussion .....</b>	<b>46</b>
<b>6</b>	<b>Conclusion .....</b>	<b>47</b>
<b>7</b>	<b>Outlook .....</b>	<b>48</b>
<b>V.</b>	<b>Bibliography.....</b>	<b>VIII</b>
<b>VI.</b>	<b>Appendix .....</b>	<b>XII</b>
<b>VII.</b>	<b>Source code .....</b>	<b>XXIII</b>



## I. List of Figures

Figure 1: Stereo speaker setup with highlighted sweet spot (Weinzierl 2008, p. 611)	4
Figure 2: Standard stereophonic listening position (Pulkki and Karjalainen 2001, p. 740)	5
Figure 3: Example for a binary pixel matrix with 7 x 8 dimensions (Werner 2021, p. 10)	6
Figure 4: Applying the 3 x 3 median filter on to the picture element $(i, j)$ (Werner 2021, p. 64)	7
Figure 5: The TCP/IP and OSI models (Wendzel 2018, p. 13)	10
Figure 6: Structure of the OSC content format (Schmeder, Freed, and Wessel 2010)	13
Figure 7: Flow chart showing the technical concept, components, and functions....	24
Figure 8: Terminal output object_tracking.py FPS test between YUV420 and RGB26	
Figure 9: Creating a mask as an overlay in order to define the tracking area (Source code 2)	27
Figure 10: Image processing of target frame: inversion, filtering and thresholding (Source code 2)	28
Figure 11 Finding contours and computing moments for PXC extraction (Source code 2)	29
Figure 12: Graphic over few on the relations between the Python scripts executed on the RPI4	31
Figure 13: Speaker placement with corresponding azimuth angle and octants I - VIII	32
Figure 14: Calculating new xy coordinates and distance from PCX to origin (Source code 2)	33
Figure 15: Calculating the azimuth angle for the first quadrant (Source code 2)	33
Figure 16: Calculating normalized x, y coordinate and radius (Source code 2)	33
Figure 17: Setting the IP address and port for UDP socket (Source code 2)	34
Figure 18: Creating a message from PD_list and sending over UDP socket to the Bela (Source code 2)	35
Figure 19: Subpatch [upd_data], receiving UDP datagram and data handling	36
Figure 20: Subpatch [pd_avr], running average over a four-value window	36
Figure 21: Abstraction [panner-abs], panning operation for one sound element to 8 outlets	38
Figure 22: PD sound design framework showing the basic structure of the patch	39

Figure 23: Custom speaker design from Styrodur 3035 CS panels and the Dayton Audio DAEX32EP-4 exciter .....	41
Figure 24: Camera unit with HQ v1 Camera, RPI4, Bela + CTAG module .....	42
Figure 25: Web interface for gain adjustments.....	44
Figure 26: PD Subpatch [pd bella_gui_recive], receiving, handling and saving gain slider values from web interface .....	44
Figure 27: PD abstraction [id_active], handling fade-in/out of a sound element .....	45

## II. List of Abbreviations

ARM	Advanced RISC Machines
CNN	Convolutional Neural Networks
CPU	Central Processing Unit
D/A	Digital to Analog
DAC	Digital to Analogue Converter
DHCP	Dynamic Host Configuration Protocol
DIN	German Institute for Standardization
DNS	Domain-Name-System
FPS	Frame per Second
GID	Global Identification Number
GPIO	General Purpose Input/Output
FUDI	Fast Universal Digital Interface
HCI	Human-Computer Interaction
HDMI	High-Definition Multimedia Interface
HHI	Human-Human Interaction
HRTF	The Head-Related Transfer Function
HTTP	Hypertext Transfer Protocol
ICRAM	Institute for Research and Coordination in Acoustics/Music
ILD	Interaural Level Difference
ITD	Interaural Time Difference
LED	Light Emitting Diode
MiDaS	Monocular Depth Sensing
MIPI CSI-2	Mobile Industry Processor Interface
MTU	Maximum Transmission Unit
OpenCV	Open-Source Computer Vision library
OS	Operating Systems
OSC	Open Sound Control
OSI	Open Systems Interconnection
PD	Pure Data
PD	Pure Data
PXC	Pixel Coordinate
R-CNN	Region-based Convolutional Neural Network
RGB	Red-Green-Blue
RMS	Root Mean Square
RPI	Raspberry Pi
SBC	Single-Board Computer
SMTP	Simple Mail Transfer Proto
SSD	Single-Shot Multibox Detector
SSH	Secure Shell
TBD	Tracking by detection
TCP/IP	Transmission Control Protocol / Internet Protocol

UDP	User Datagram Protocol
URL	Uniform Resource Locators
VoIP	Voice over IP
XLR	eXternal Line Return
YOLO	You Only Look Once

### III. List of Symbols

#### Symbols Used in Formulas

$\theta$	Perceived Position to the Base Angle
$\theta_0$	Base Angle Stereo Pair
$g_1$	Gain Factor Speaker Left
$g_2$	Gain Factor Speaker Right
$g_L$	Gain Factor Speaker Left
$g_R$	Gain Factor Speaker Right
$\theta_x$	Perceived Position Between 0° and 90°
$N_{\theta_x}$	Perceived Position Normalized Between 0 and 1
$A$	Pixel Matrix
$M$	Number of Rows in the Matrix
$N$	Number of Columns in the Matrix
$a$	Picture Element
$m$	Row Index
$n$	Column Index
$\sin$	Sine
$\tan$	Tangent
$\cos$	Cosine

#### Units Used

Hz	Hertz
°	Degrees
$\pi$	Pi
Ghz	Gigahertz
Gbit/s	Gigabit per Second
kHz	Kilohertz
cm	Centimeter
m	Meter
dB	Decibel

### IV. List of formulas

Equation 1: Sine Law .....	5
Equation 2: Tangent Law .....	5
Equation 3: Sine-Cosine Law (degrees).....	5
Equation 4: Sine-Cosine Law (normalized) .....	5

# 1 Introduction

The following work deals with the conception, development and realization of an interactive sound installation. The aim was to develop a functional and low-cost prototype using open-source software and minimizing the use of professional audio and video equipment. Often an artist's vision is at odds with a realistic technical implementation due to limiting factors such as physical space, safety concerns, budget restrictions, etc. In this work, those limitations were used as an inspiration rather than a restriction.

## 1.1 Overview

Chapter 1.2 covers the motivation for this project and gives insights into the inspiration for the artistic concept. Chapter 2 establishes the fundamental concepts involved in this project, such as audio, video, network technology, object detection, and the basic concept of audience engagement in sound art. Chapter 3 describes the design phase, specifically with the purpose of identifying the technical requirements for the project in order to find suitable solutions and combine them into the final approach. This part of the process where the technical solutions are aligned with the artistic concept before development was crucial in order to achieve the best possible installation. Chapter 4 deals with the technical development of the installation, highlighting the technical approach for each solution in relation to the artistic concept. Finally, the installation was evaluated through real scenario test setups. These focused on the technical setup, calibration, identifying problems, and gaining insight into how the installation behaved and which improvements could be implemented. The discussion, conclusion and outlook analyze the findings and highlight future improvements.

## 1.2 Motivation and Inspiration

The main inspiration for creating this sound installation is rooted in the author's own neurodiversity. Loud and crowded environments can become quickly overstimulating. In contrast, quiet and calm spaces can be under-stimulating. Listening to music and body movement has always been a tool to self-regulate stimulation levels in order to stay functional in changing environments. Without the aid of music, thoughts can run amuck and focusing is challenging.

Due to these challenges, the author developed a personal interest in exploring the world of sounds. It even led him to pursue his first career as a hearing acoustic technician.

Despite always having an interest in art, traditional spaces like galleries, exhibitions, and museums did not feel like welcoming environments due to a perceived lack of stimulation. The author became fascinated by the concept of integrating an individual into the overall context of an artwork and was inspired to create an installation where movement and interaction with others is rewarded. This interaction should be playful and encourage a curiosity for discovery. The installation would create an intimate, interactive, and atmospheric soundscape that surrounds participants. The sound and its origin in the space would be influenced by the movement and collective interactions of the audience. It would encourage individuals to step out of just being passive observers and play with their movement individually and with others within a space to awaken the installation and create their own unique auditory experience. A first sketch and a mood board are provided in Appendix 1.

The technical concept is limitation as a tool was also used as an inspiration. Nowadays the seemingly infinite number of technical solutions, tools, and possibilities can be overwhelming. The author has largely dispensed with the use of expensive ready-made professional audio and video technology and opted to solve technical issues creatively with more affordable and accessible options. Not only was how the author's own creative process could be promoted through these limitations, but also how these limitations could open up the installation on a broader spectrum, both in terms of the spatial conditions and accessibility to a diverse range of communities were explored. Professional audio and video equipment is expensive and might not be easily available in many regions. A key motivation was building the installation in a manner to minimize geographic or economic constraints in order to make it accessible for anyone no matter who they are or where they are.

## 2 Theoretical Framework

### 2.1 Sound Perception and Basic Panning Fundamentals

#### 2.1.1 Human Perception of Sound

Sound is a mechanical wave that results from the back-and-forth vibration of the particles of the medium through which the sound wave is moving. Sound waves are longitudinal waves, where the particle movement is parallel to the direction of the wave propagation<sup>1</sup>.

---

<sup>1</sup> cf.: Weinzierl, 2008, Handbuch der Audiotechnik, p. 18-19,

Binaural localization refers to the ability of the auditory system to determine the location of a sound source using both ears.<sup>2</sup> This subject has been a topic of extensive study, leading to the discovery of several key concepts and cues underlying this skill.

Interaural Time Difference (ITD) describes the difference in arrival time of a sound between the two ears. This parameter is crucial for locating the direction of frequencies below 1.5 kHz.<sup>3</sup> The difference in sound pressure level reaching each ear is known as Interaural Level Difference (ILD). This is important for locating frequencies above 1.5 kHz.<sup>4</sup> The Head-Related Transfer Function (HRTF) explains how an ear receives a sound from a point in space, incorporating the effects of the, body, head and outer-ear. HRTFs are unique to each individual and help in spatial localization<sup>5</sup>. Variations in the sound spectrum caused by the shape of the ear and the head, which provide information about the vertical location of the sound source are generally referred to as spectral cues. Reflections of sound from surfaces can provide additional spatial cues to help localize sounds in an environment with reverberation and echoes.<sup>6</sup>

### 2.1.2 Stereophonic Panning Laws

To replicate the localization of single sound sources through loudspeaker systems, binaural localization principles have been used to develop various panning methods that accommodate different loudspeaker arrangements. In this context, the sound source is commonly referred to as a virtual source or phantom source, as the sound does not originate from a natural source.<sup>7</sup>

This thesis restricts itself on focusing on stereophony and a standard panning concept which was adapted to a 360° speaker setup is covered in chapter 4.3.2. This concept applies level differences to the audio signal.<sup>8</sup> A pair of speakers create an auditory soundscape on the horizontal plane, replicating the localization of the phantom sources in front of the listening position from left to right. The usual speaker arrangement involves an equilateral triangle between the loudspeakers and the listening position, resulting in an opening angle of 60 degrees. The DIN 15996 standard allows for an opening angle tolerance of plus or minus 15 degrees.<sup>9</sup> Small movements from left to right of the listening position can lead to misinterpretations of

---

<sup>2</sup> cf.: Möser, 2018, Psychoakustische Messtechnik, p. 23,

<sup>3</sup> cf.: Möser, 2018, Psychoakustische Messtechnik, p. 23-24,

<sup>4</sup> cf.: Möser, 2018, Psychoakustische Messtechnik, p. 23-24,

<sup>5</sup> cf.: Möser, 2018, Psychoakustische Messtechnik, p. 24-25,

<sup>6</sup> cf.: Möser, 2018, Psychoakustische Messtechnik, p. 25,

<sup>7</sup> cf.: Pulkki and Karjalainen, 2001, Localization of amplitude-panned virtual sources I: stereophonic panning, p. 739,

<sup>8</sup> cf.: Weinzierl, 2008, Handbuch der Audiotechnik, p. 727,

<sup>9</sup> cf.: Weinzierl, 2008, Handbuch der Audiotechnik, p. 611,



the localization of the phantom source, resulting in a narrow corridor of optimal listening positions, commonly known as the sweet spot, as illustrated in Figure 1.<sup>10</sup>

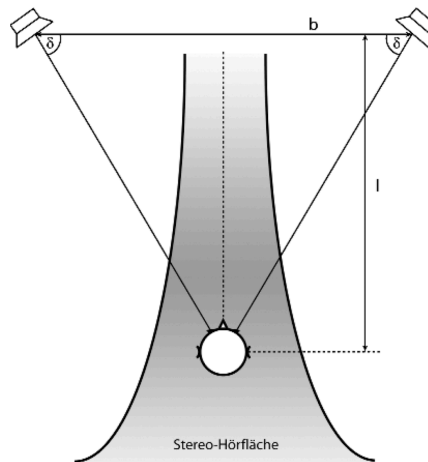


Figure 1: Stereo speaker setup with highlighted sweet spot (Weinzierl 2008, p. 611)

In this setup, two principles, ITD and ILD, can be used to reproduce a binaural listening experience. For ITD, the audio signal for the phantom source can be delayed, while for ILD, the audio signal can be adjusted to give more level to one speaker. In most applications, only ILD is used to ensure mono compatibility for the audio signal, as short time delays can cause comb filtering effects.<sup>11</sup> Each loudspeaker receives the same audio signal with different levels in amplitude. This concept is referred to as amplitude or intensity panning.<sup>12</sup>

To determine the appropriate level for driving each loudspeaker to reproduce the location of a phantom source, several panning laws have been developed. The goal is to maintain a constant perceived loudness, regardless of the panning position of the phantom source. Figure 2 describes the standard stereophonic listening position in relation of the azimuth angle  $\theta$  representing the perceived position of the phantom source to the base angle  $\theta_0$ .<sup>13</sup>

<sup>10</sup> cf.: Weinzierl, 2008, Handbuch der Audiotechnik, p. 612,

<sup>11</sup> cf.: Weinzierl, 2008, Handbuch der Audiotechnik, p. 727,

<sup>12</sup> cf.: Pulkki and Karjalainen, 2001, Localization of amplitude-panned virtual sources I: stereophonic panning, p. 739-740,

<sup>13</sup> cf.: Pulkki, 2001, Spatial sound generation and perception by amplitude panning techniques, p. 11-12,

B. B. Bauer formulated Equation 1: Sine Law which comes with the limitation that it is only valid with a forward head position and for frequencies below 600 Hz, due to the frequency-dependent properties of ILD:<sup>14</sup>

$$\frac{\sin \theta}{\sin \theta_0} = \frac{g_1 - g_2}{g_1 + g_2} \quad \text{Equation 1: Sine Law}$$

To account for deviations in head positions, J.C. Bennet reformulated the sine law into the Equation 2: Tangent Law:<sup>15</sup>

$$\frac{\tan \theta}{\tan \theta_0} = \frac{g_1 - g_2}{g_1 + g_2} \quad \text{Equation 2: Tangent Law}$$

The tangent law can be reformulated as the sine-cosine law with  $\theta_x = 0^\circ$  all left and  $\theta_x = 90^\circ$  all right applies to Equation 3 normalized to  $N_{\theta_x} = 0$  all left and  $N_{\theta_x} = 1$  all right applies to Equation 4:<sup>16</sup>

$$\begin{aligned} g_L &= \cos(\theta_x) \\ g_R &= \sin(\theta_x) \end{aligned} \quad \text{Equation 3: Sine-Cosine Law (degrees)}$$

$$\begin{aligned} g_L &= \cos\left(N_{\theta_x} \times \frac{\pi}{2}\right) \\ g_R &= \sin\left(N_{\theta_x} \times \frac{\pi}{2}\right) \end{aligned} \quad \text{Equation 4: Sine-Cosine Law (normalized)}$$

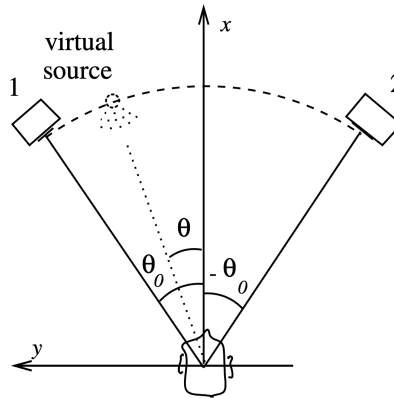


Figure 2: Standard stereophonic listening position (Pulkki and Karjalainen 2001, p. 740)

## 2.2 Digital Image Processing and Object Detection/Tracking Methods

### 2.2.1 Digital Image Processing Fundamentals

Digital image processing is essential for performing and applying computer vision tasks, such as object detection. To execute computing tasks, digital images are usually interpreted as a rectangular pixel matrix  $A_{M \times N} = (a_{m,n})$ , where M represents

<sup>14</sup> cf.: Pulkki, 2001, Spatial sound generation and perception by amplitude panning techniques, p. 12,

<sup>15</sup> cf.: Pulkki and Karjalainen, 2001, Localization of amplitude-panned virtual sources I: stereophonic panning, p. 740-741,

<sup>16</sup> cf.: Pulkki, 2001, Spatial sound generation and perception by amplitude panning techniques, p. 12,

the rows and N the columns. As shown in Figure 3, each index pair corresponds to a pixel value that defines the state of each pixel or picture element.<sup>17</sup>

		$n \rightarrow$							
		1	2	3	.....	8			
$m \downarrow$	1	1	1	1	1	1	1	1	1
	2	1	0	0	1	0	0	0	1
	3	1	0	1	1	1	0	1	1
	.....								
	4	1	0	0	1	1	0	1	1
	5	1	0	1	1	1	0	1	1
	6	1	0	0	1	1	0	1	1
	7	1	1	1	1	1	1	1	1

Figure 3: Example for a binary pixel matrix with 7 x 8 dimensions (Werner 2021, p. 10)

These picture element values depend on the color model used, such as Red-Green-Blue (RGB). In this thesis, binary images, which consist of only two values, 0 and 1, are also considered. In addition to more commonly known image manipulations, like contrast adjustment and color correction, a few other techniques are highlighted below.

**Thresholding:** Thresholding, or binarization, is a technique used to transform a grayscale image, also known as an intensity image, into a binary image. Every pixel value below a defined threshold is set to 0, while values above the threshold are set to 1. This can be used to detect objects in front of a uniform background, as long as the pixel values of the background and the object differ in intensity.<sup>18</sup>

**Binning:** To highlight objects with similar pixel values, a technique known as binning can be used. This technique consolidates neighboring pixel values of the grayscale image, resulting in a coarser image with areas of uniform pixel value.<sup>19</sup>

**Inversion:** One simple but useful operation is grayscale inversion, which subtracts the value of each pixel from the maximum value, resulting in an inverted image where darker areas are displayed as bright areas and vice versa. The same technique can be applied to RGB images.<sup>20</sup>

**Linear Filtering:** As these image processing methods are only concerned with each pixel's value individually, other methods, categorized as neighboring processing, take into account the state of neighboring pixel values. These types of operations are usually referred to as rank-order filtering, a type of nonlinear filters.<sup>21</sup> This technique

<sup>17</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 11,

<sup>18</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 55,

<sup>19</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 60,

<sup>20</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 62,

<sup>21</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 63,

operates by collecting the grayscale values of pixels within a defined mask matrix around a target pixel, sorting these values, and then selecting one value from the sorted list to replace the grayscale value of the target pixel. The specific value chosen from the sorted list determines the type of filter. Figure 4 illustrates this concept on the example of a median filter.

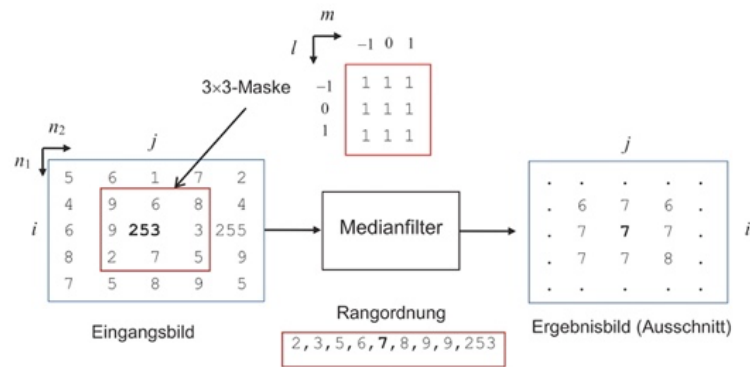


Figure 4: Applying the 3 x 3 median filter on to the picture element (i, j) (Werner 2021, p. 64)

The median filter elects the median value from the sorted list, replacing the target pixel with this median value.<sup>22</sup> The minimum and maximum filters select and replace accordingly the minimum or maximum value. Those type of filters are typically used to reduce noise or apply a blurring effect to the image.<sup>23</sup>

**Contour and edge filtering:** Another important aspect of image processing for pattern recognition and object detection algorithms is edge and contour detection. Edges can be defined as rapid changes in the intensity of pixel values within small areas, following a directional propagation. Contours are formed by continuous edges that define the boundaries of objects.<sup>24</sup> Common methods for edge and contour detection include the Sobel method, Laplacian-of-Gaussian method, and Canny method. These techniques are based on vector analysis principles to evaluate intensity changes concerning direction and propagation.<sup>25</sup>

Another essential tool for further image processing, especially for object detection, is the use of moments as standard characteristics. In the context of image processing, moments can generally be categorized as moments of points, lines, and areas. They are used to describe the position and orientation of objects, assist in classifying objects, and provide information about geometric standard forms like circles, rectangles, etc., which can be helpful for detecting and matching objects.<sup>26</sup>

<sup>22</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 64,

<sup>23</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 65,

<sup>24</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 138,

<sup>25</sup> cf.: Werner, 2021, Digitale Bildverarbeitung, p. 138-139,

<sup>26</sup> cf.: Süße and Rodner, 2014, Bildverarbeitung und Objekterkennung, p. 515,

These are just a few tools used in image processing to help understand the basics of how images can be evaluated to recognize patterns and associate or match them with objects or predefined classification criteria.

### 2.2.2 Traditional Object Detection using Image Processing

To effectively utilize the information about an image provided by the methods mentioned above, traditional object algorithms are typically divided into several key steps: preprocessing, segmentation, classification, and the output of analytical data.

Preprocessing is performed to enhance the image, making it more suitable for further analysis. This step often includes filtering to remove noise, adjusting contrast to improve visibility, and converting the image into different formats such as greyscale or binary, depending on the requirements of the specific algorithm.<sup>27</sup> The choice of preprocessing techniques depends on the specific use case and the characteristics of the data being analyzed.<sup>28</sup> For example, converting an image to greyscale might be necessary to simplify the analysis, while binarization could be useful for tasks involving edge detection or object recognition.

Segmentation involves dividing the image into meaningful segments or regions that represent specific features, such as objects, contours, or lines. This step is crucial because it isolates the areas of interest within the image, which are then analyzed in subsequent steps.<sup>27</sup>

Classification is the process of analyzing each segment to determine which predefined class it belongs to. This involves extracting features from the segmented regions and comparing these features against a set of predefined classes. The accuracy of the classification depends on both the quality of the segmentation and the effectiveness of the feature extraction process. Typical features might include texture, shape, color, or intensity patterns within the segments.<sup>29</sup>

Finally, the output of analytical data involves presenting the results of the classification in a meaningful way. This could be in the form of labeled images, statistical summaries, or other data representations that provide insight into the characteristics of the image.<sup>30</sup> The nature of the output will depend on the goals of the image processing task, whether it is for further analysis, decision-making, or visualization purposes. Typically, data like distance between objects or set coordinates, geometrical properties or localization of the object inside the image are of interest. In

---

<sup>27</sup> cf.: Süße and Rodner, 2014, Bildverarbeitung und Objekterkennung, p. 211,

<sup>28</sup> cf.: Kadam, Fang, and Zou, 2024, Object Tracking Using Computer Vision: A Review, p. 18,

<sup>29</sup> cf.: Süße and Rodner, 2014, Bildverarbeitung und Objekterkennung, p. 212,

<sup>30</sup> cf.: Süße and Rodner, 2014, Bildverarbeitung und Objekterkennung, p. 211,

traditional object detection, each of these stages are manually tweaked to fit the needed use case or application which narrows down the area in which those detecting algorithms lead to accurate results.<sup>31</sup>

### 2.2.3 Traditional Object Tracking Using Image Processing

Tracking methods take place after object detection. The goal is to reidentify and track the location of a previously detected object over a sequence of video frames.<sup>32</sup> One method is known as tracking by detection (TBD), where each frame is analyzed and tracked objects are associated with the objects detected in the next frame using methods like data association, template matching optical flow, descriptor-based tracking, or the Kalman filter.<sup>33</sup>

Data association uses information accrued in the previous detection phase about the objects, movement pose, or change in appearance to reidentify the object. Template matching is an image processing technique used to find and locate a specific pattern or template within a larger image by comparing portions of the image against the template. It works by sliding the template over the larger image and calculating a similarity measure at each position to identify the best match.<sup>34</sup> Optical flow estimates the motion of objects or features between consecutive frames in. It provides information on the direction and speed of motion, helping to track objects or understand scene dynamics over time.<sup>35</sup> The Kalman Filter is an algorithm which predicting the future state by combining previous estimates with new measurements tracking moving objects by predicting their future positions and adapting these predictions in real-time.<sup>36</sup>

### 2.2.4 Object Detection and Tracking Methods Based on Deep Learning

Deep learning is a subset of machine learning that uses artificial neural networks with multiple layers to model and understand complex patterns in data. For object detection, typically convolutional neural networks (CNN) are used which have been trained on feature sets to classify and locate objects in a given image.<sup>37</sup> Commonly used CNNs with pretrained datasets that are available are Single-Shot Multibox Detector (SSD), You Only Look Once (YOLO), Region-Based Convolutional Neural Network (R-CNN) and Monocular Depth Sensing (MiDaS).<sup>38</sup>

---

<sup>31</sup> cf.: Süße and Rodner, 2014, Bildverarbeitung und Objekterkennung, p. 18,

<sup>32</sup> cf.: Kadam, Fang, and Zou, 2024, Object Tracking Using Computer Vision: A Review, p. 16-17,

<sup>33</sup> cf.: Kadam, Fang, and Zou, 2024, Object Tracking Using Computer Vision: A Review, p. 22,

<sup>34</sup> cf.: Kadam, Fang, and Zou, 2024, Object Tracking Using Computer Vision: A Review, p. 22-23,

<sup>35</sup> cf.: Kadam, Fang, and Zou, 2024, Object Tracking Using Computer Vision: A Review, p. 23,

<sup>36</sup> cf.: Kadam, Fang, and Zou, 2024, Object Tracking Using Computer Vision: A Review, p. 24-25,

<sup>37</sup> cf.: Sarkar, Bali, and Sharma, 2018, Deep Learning for Computer Vision, p. 499,

<sup>38</sup> cf.: Kadam, Fang, and Zou, 2024, Object Tracking Using Computer Vision: A Review, p. 20-21,

TBD methods can also be combined with deep learning models in order to train tracking algorithms following those concepts. In addition to TBD, another approach is joint detection and tracking, which incorporates the detection and tracking processes into a single stage which makes it possible to introduce end-to-end trainable models. This means that a single model could be trained to perform both the detection and tracking rather training models individually for each stage.<sup>39</sup>

## 2.3 Network Communication and Protocols

For modern network communication, the one relevant architecture framework is the Transmission Control Protocol/Internet Protocol (TCP/IP) suite, which is a collection of compatible communication protocols. These protocols follow the TCP/IP model, which is constructed of logical layers representing the communication architecture, similar to the Open Systems Interconnection (OSI) model.<sup>40</sup> The main difference between the two is that the TCP/IP model consists of four layers instead of the seven OSI layers, as shown in Figure 5: The TCP/IP and OSI models (Wendzel 2018, p. 13). The application layer combines the functions of the OSI's application, presentation, and session layers and handles high-level protocols like Hypertext Transfer Protocol (HTTP), Simple Mail Transfer Protocol SMTP, etc. The Transport Layer corresponds to the OSI's Transport layer, providing reliable data transfer via protocols like TCP and User Datagram Protocol (UDP). The internet layer is similar to the OSI's network layer, managing IP addressing and routing of data packets. The link layer corresponds to the OSI's data link and physical layers, dealing with the physical transmission of data in the form of bytes over network interfaces.<sup>41</sup>

TCP/IP-Modell:	OSI-Modell:
Application Layer	Application Layer Presentation Layer Session Layer
Transport Layer	Transport Layer
Internet Layer	Network Layer
Link Layer (auch: Network Access Layer)	Data Link Layer Physical Layer

Figure 5: The TCP/IP and OSI models (Wendzel 2018, p. 13)

<sup>39</sup> cf.: Kadam, Fang, and Zou, 2024, Object Tracking Using Computer Vision: A Review, p. 25,

<sup>40</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 12,

<sup>41</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 12-13,

### 2.3.1 The TCP Protocol and its Application

The TCP protocol is a Transport Layer protocol, and in this context, data packets are referred to as segments. These segments are not sent as separate messages but as a data stream.<sup>42</sup> It is important to keep track of the order and any potential loss of the transferred data. For this purpose, TCP uses sequence numbers to label each segment sent by the host. If a segment is successfully delivered to the client, the client sends an acknowledgment number to the host. If no acknowledgment is received by the host, the missing segment is resent to the client.<sup>43</sup>

To perform successful communication between the host and client, the client initiates a process called passive opening, which involves dedicating a port for TCP communication. The host performs an active opening by establishing a connection between the host and client through a process known as the Three-Way Handshake.<sup>44</sup> This involves three TCP segments being exchanged between the host and client. First, the host sends a synchronization flag to the client, indicating a connection attempt and including a synchronization sequence number. The client accepts by sending an acknowledgment number and its own synchronization sequence number. The third segment is an acknowledgment by the host, and the connection is established.<sup>45</sup> To properly end the communication, the connection must be closed using another Three-Way Handshake, with an active close performed by the host and a passive close by the client.<sup>46</sup>

### 2.3.2 The UDP-Protocol and its Application

The UDP is also a Transport Layer protocol, and in this context, data packets are referred to as datagrams. UDP falls under the category of connectionless protocols, which means a connection between the host and client does not need to be established.<sup>47</sup> Datagrams are sent regardless of the connection status. Datagrams are sent as sequential messages, but unlike TCP, UDP does not have a process to ensure the correct order or to resend missing packets, meaning lost datagrams cannot be recovered. Handling this would only be possible at the application layer, which requires metadata about packet control to be included in each datagram message

---

<sup>42</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 50,

<sup>43</sup> cf.: Plenk, 2024, Angewandte Netzwerktechnik kompakt: Dateiformate, Übertragungsprotokolle und ihre Nutzung in Java-Applikationen, p. 159-160,

<sup>44</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 54-55,

<sup>45</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 56,

<sup>46</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 57,

<sup>47</sup> cf.: Plenk, 2024, Angewandte Netzwerktechnik kompakt: Dateiformate, Übertragungsprotokolle und ihre Nutzung in Java-Applikationen, p. 185,



and specified by the application used.<sup>48</sup> This means that, in general, applications using UDP should be insensitive to data loss and the mixed arrival of data packets.

Despite its limitations, UDP offers several advantages that make it a valuable communication protocol. UDP does not require a connection to be established before data is sent, eliminating the need for the three-way handshake process used by TCP. Additionally, UDP does not perform error checking, acknowledgment, or retransmission, which allows data to be transmitted more quickly, resulting in lower latency and faster transmission. This makes UDP suitable for real-time applications such as live video streaming, online gaming, and Voice over IP (VoIP), where slight delays or loss of data packets are preferable to the delays caused by retransmission in TCP.<sup>49</sup>

In addition, UDP is easier to implement in both hardware and software, making it a preferred choice for simple, resource-constrained systems where minimizing complexity is important. UDP is typically used in scenarios where datagrams need to be transmitted at regular intervals or where the loss of a message is not problematic because requests can be easily repeated, as in Domain-Name-System (DNS) or Dynamic Host Configuration Protocol (DHCP) requests.<sup>50</sup> When implementing the UDP protocol, an important factor to consider is IP fragmentation. Every network has limitations on the amount of data that can be transferred over a given time, depending on the protocols used and, more importantly, the physical network interfaces implemented. Since UDP does not offer the ability to reorder datagrams, fragmentation and the mixed arrival of those fragmented frames could corrupt the data.<sup>51</sup>

Fragmentation occurs at the link layer within the TCP/IP model. At the link layer, packets are generally referred to as frames, and the maximum size of each frame is defined by the Maximum Transmission Unit (MTU). A typical MTU for Ethernet and WLAN connections is 1,500 Bytes. If a data packet exceeds the MTU, it is fragmented into two or more fragments and sent as individual frames.<sup>52</sup> To prevent fragmentation, V. Plenk advises to keep the maximum data payload for each datagram under 508 Bytes, accounting for network MTU's and metadata like IP and UDP headers.<sup>53</sup>

---

<sup>48</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 49,

<sup>49</sup> cf.: Plenk, 2024, Angewandte Netzwerktechnik kompakt: Dateiformate, Übertragungsprotokolle und ihre Nutzung in Java-Applikationen, p. 186,

<sup>50</sup>cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 49,

<sup>51</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, 38,

<sup>52</sup> cf.: Wendzel, 2018, IT-Sicherheit Für TCP/IP-und IoT-Netzwerke, p. 37,

<sup>53</sup> cf.: Plenk, 2024, Angewandte Netzwerktechnik kompakt: Dateiformate, Übertragungsprotokolle und ihre Nutzung in Java-Applikationen, p. 188,

### 2.3.3 The OSC Content Format and its Message Structure

In the context of multimedia applications Open Sound Control (OSC) is an established communication protocol that was developed in 1997 by Adrian Freed and Matt Wright.<sup>54</sup> It is designed to control real-time audio applications and operates at the Application Layer in the TCP/IP model or the Presentation Layer in the OSI model. For network communication, OSC relies on underlying transport protocols like UDP or TCP.<sup>55</sup>

OSC streams consist of sequences of frames, each associated with a specific moment in time known as a time tag. These frames are referred to as bundles. Within each bundle, there are multiple messages, each representing the state of a sub-stream at the referenced time tag. These sub-streams are identified by a human-readable string called an address, which is constructed as a hierarchical namespace similar to Uniform Resource Locators (URL). A visualization of the message hierarchy is provided in Figure 6.<sup>56</sup> Messages within the same bundle are treated as atomic, meaning their effects should be executed simultaneously by the receiver. This is crucial in multimedia applications where certain values need to be set at the exact same time.<sup>57</sup>

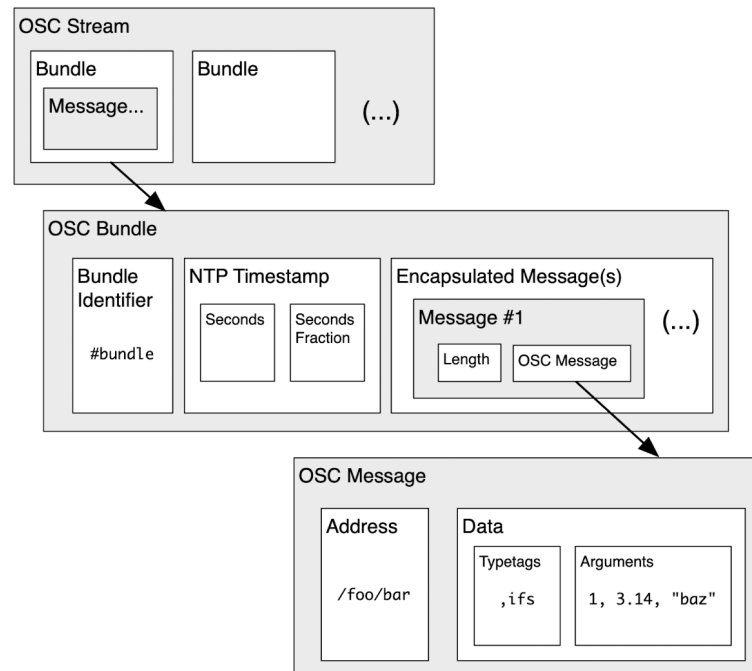


Figure 6: Structure of the OSC content format (Schmeder, Freed, and Wessel 2010)

<sup>54</sup> cf.: Wright and Freed, 1997, Open SoundControl: A new protocol for communicating with sound synthesizers,

<sup>55</sup> cf.: Freed and Schmeder, 2009, Features and Future of Open Sound Control version 1.1 for NIME,

<sup>56</sup> cf.: Schmeder, Freed, and Wessel, 2010, Best practices for open sound control,

<sup>57</sup> cf.: Wright and Freed, 1997, Open SoundControl: A new protocol for communicating with sound synthesizers,

## 2.4 An Overview on Single-Board Computers

A Single-Board Computer (SBC) is a compact, self-contained computing device that integrates all the essential components of a computer, such as the processor, memory, input/output interfaces, and storage, onto a single circuit board. Unlike traditional desktop computers, which typically consist of multiple separate components connected via a motherboard, an SBC is designed to be a low-cost, energy-efficient solution for a wide range of applications, including embedded systems, educational tools, hobbyist projects, and industrial automation. SBCs like the Raspberry Pi (RPI), BeagleBone, and Arduino are particularly popular due to their affordability, small form factor, and versatility.<sup>58</sup>

**Raspberry Pi:** The RPI is an SBC developed by the Raspberry Pi Foundation designed to promote computer science in education. Known for its affordability, small size, and versatility, the Raspberry Pi features a powerful Advanced RISC Machines (ARM) based processor, multiple Universal Serial Bus (USB) ports, High-Definition Multimedia Interface (HDMI) output, and General-Purpose Input/Output (GPIO) pins for connecting hardware components. It runs a variety of operating systems (OS), most notably Raspberry Pi OS.<sup>59</sup>

**BeagleBone Black:** An open-source SBC platform developed by the nonprofit organization BeagleBoard.org Foundation and designed for developers and engineers requiring more processing power and input/output capabilities than typical SBCs like the RPi. The BeagleBone boards, such as the BeagleBone Black, feature an ARM 1Ghz Cortex-A8 processor, various GPIO options, and support for real-time processing tasks, making them ideal for industrial automation, robotics, and embedded systems.<sup>60</sup>

**Bela:** The Bela is a licensed version of the BeagleBone Black with light modifications. It is designed for real-time audio applications integrating audio programming languages like Pure Data (PD), Supercollider, and Csound.<sup>61</sup>

**Arduino:** This SBC is a popular open-source electronics platform that combines easy-to-use hardware and software to create interactive projects. Unlike typical single-board computers, Arduino boards are microcontroller-based rather than microprocessor-based, focusing on control and interfacing tasks rather than running full operating systems. Arduino is widely used in education, prototyping, and hobbyist

---

<sup>58</sup> cf.: Ariza and Baez, 2022, Understanding the role of single-board computers in engineering and computer science education: A systematic literature review,

<sup>59</sup> cf.: Raspberry-Pi, Raspberry Pi hardware, <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>,

<sup>60</sup> cf.: BeagleBoard.org-Foundation, Our Mission, <https://www.beagleboard.org/about>,

<sup>61</sup> cf.: Bela.io, Bela & Bela Mini, <https://bela.io/products/bela-and-bela-mini/>,

projects due to its simplicity, affordability, and extensive community support. It is used in tasks such as controlling sensors, motors, and light emitting diodes (LED), making it a suitable choice to build custom hardware solutions.<sup>62</sup>

**Sensors/Modules:** Most of those SBC platforms support a wide range of sensors or modules, enabling them to interact with the environment by monitoring various physical properties such as temperature and humidity, motion and proximity, light, or gas as well as camera modules, accelerometers and gyroscopes, and other environmental sensors.<sup>63</sup>

## 2.5 Software Solutions for Audio-Visual Applications

### 2.5.1 TouchDesigner

TouchDesigner is a visual development platform that enables users to create interactive multimedia applications and real-time digital art. Developed by Derivative, it provides a node-based interface for procedural design, making it highly versatile for tasks from interactive installations, generative design, and real-time 3D graphics, to audio-reactive visuals. Its strength lies in its ability to handle complex multimedia systems and real-time data inputs like audio, video, image processing data, allowing artists, designers, and developers to create dynamic, immersive experiences. TouchDesigner integrates seamlessly with various hardware devices, making it a popular choice for creating interactive installations, live performances, and projection mapping projects.<sup>64 65</sup>

### 2.5.2 Max / RNBO

Max is a visual programming language originally developed by Miller Puckette at the Institute for Research and Coordination in Acoustics/Music (ICRAM) and later the rights were acquired by Cycling '74. Max is designed to create complex audio-visual applications. It operates through a graphical interface where users connect objects representing different functions, such as sound synthesis, visual rendering, and data processing which are called patches. Max is particularly known for its flexibility in audio-visual programming, making it a popular tool for creating custom software instruments and audio effects, interactive installations, and live performance systems. Its extensive library of objects and compatibility with various external hardware

---

<sup>62</sup> cf.: Arduino-S.r.l., Hardware, <https://www.arduino.cc/en/hardware>,

<sup>63</sup> cf.: Sertronics-GmbH, sensoren-module, <https://www.berrybase.de/sensoren-module/>,

<sup>64</sup> cf.: Kraus-[GbR], About TouchDesigner, <https://thenodeinstitute.org/about-touchdesigner/>,

<sup>65</sup> cf.: derivative, Features, <https://derivative.ca/feature>,

devices make Max a powerful environment for creative coding and multimedia experimentation.<sup>66 67</sup>

RNBO is an extension of the Max environment designed for creating audio applications that can be easily exported to multiple platforms, including web, mobile, and embedded systems. RNBO enables users to design audio signal processing algorithms visually within Max and then compile them into portable code that can run independently outside of the Max environment like JavaScript, C++, or directly compiling onto an RPI. This makes RNBO a valuable tool for developers looking to deploy their custom audio tools and instruments across different platforms without rewriting code.<sup>68</sup>

### 2.5.3 Pure Data

Pure Data (PD) is a visual programming language developed by Miller Puckette as an open-source alternative to Max for creating interactive multimedia applications, with a particular emphasis on real-time audio processing. Like its commercial counterpart Max, PD allows users to build complex systems by visually connecting objects that perform various functions, such as sound synthesis, signal processing, and data manipulation which are also referred to as patches. PD is widely used in experimental music, sound art, and interactive installations due to its flexibility, extensibility, and free accessibility. It also supports the creation of custom audio effects, virtual instruments, and interactive interfaces. The modular architecture and cross-platform compatibility of PD make it a powerful tool for artists, musicians, and developers seeking to explore creative coding and digital media.

The PD community has developed numerous additional functions, known as externals or external libraries, which extend its capabilities to include video processing, 3D object manipulation, and more.<sup>69</sup> The integration of PD on the Bela comes with certain limitations and challenges. It is important to note that this is not the full Pd Vanilla version but rather libpd a lightweight, headless version that allows patches to be loaded and run but not edited directly on the platform. This can be limiting for users who need to tweak patches on the fly, as the standard PD interface is not available.

---

<sup>66</sup> cf.: Cycling'74, What is Max, <https://cycling74.com/products/max>,

<sup>67</sup> cf.: Institute-of-Electronic-Music-and-Acoustics, Home, <https://puredata.info/>,

<sup>68</sup> cf.: Cycling'74, Introducing RNBO, <https://cycling74.com/products/rnbo>,

<sup>69</sup> cf.: Institute-of-Electronic-Music-and-Acoustics, Home, <https://puredata.info/>,

## 2.6 Audio-Visual Art, Sound Art and the Connection to Interactivity

### 2.6.1 Audio-Visual Art

Audio-visual art is an artistic form that combines auditory and visual elements to create immersive experiences. Within this context, sound art plays a crucial role, contributing to the overall sensory impact of the work.<sup>70</sup> Sound art in audio-visual contexts encompasses elements of the tonal, spatial, and visual.<sup>71</sup> This integration challenges traditional boundaries between disciplines, drawing from fine art, music, and technology to create a complex and rich artistic medium.

### 2.6.2 Sound Art

The emergence of sound art as a distinct form within audio-visual art can be traced back to the early 20th century. Sound artist and composer Alan Licht points to the development of technologies like the telephone and sound recording equipment as key moments when sound could be detached from its physical sources and cultural contexts.<sup>72</sup> The incorporation of sound in non-musical artistic contexts represented a profound challenge to established aesthetic norms, offering artists new ways to explore space, time, and sensory perception.

### 2.6.3 Interactivity between Audience and Sound Art

Interactivity in the context of art refers to the dynamic relationship between the artwork and its audience, where the viewer or participant plays an active role in influencing or shaping the experience of the piece. Unlike traditional art forms, where the audience passively observes, interactive art invites engagement, allowing participants to alter or affect the outcome, appearance, or behavior of the artwork through their actions, inputs, or decisions.<sup>73</sup>

In this paper, the definition of Interaction in sound art mainly followed the framework from Visda Goudarzi and Artemi-Maria Gioti which they discussed engagement and interaction in participatory sound art.<sup>74</sup> It explored the role of different actors in interactive sound systems, focusing on how technology and creative processes intersect. They discussed the blurring of roles between technology creators, composers, performers, and spectators in participatory sound art and the challenges associated with ownership of technical and aesthetic components.<sup>75</sup>

---

<sup>70</sup> cf.: Shoer, Kopru, and Erzin, 2022, Role of Audio in Audio-Visual Video Summarization,

<sup>71</sup> cf.: Holmes, 2022, Sound Art: Concepts and Practices,

<sup>72</sup> cf. Garrelfs, 2015, From inputs to outputs: an investigation of process in sound art practice, p. 21,

<sup>73</sup> cf.: Candy, Interaction in Art and Technology, <https://crossings.tcd.ie/issues/2.1/Candy/>,

<sup>74</sup> cf.: Goudarzi and Gioti, 2016, Engagement and interaction in participatory sound art,

<sup>75</sup> cf.: Goudarzi and Gioti, 2016, Engagement and interaction in participatory sound art,

They laid down different aspects of audience participation within participatory sound systems, highlighting several key parameters that influenced how audiences engage with such systems into three main categories: Audience Engagement, Human-Computer Interaction, and Human-Human Interaction.<sup>76</sup>

**Audience Engagement** can range from *crowdsourcing*, where the audience passively provides data for the sound creation process. *Performance agency*, where they take on an active role, controlling sound parameters in real-time within a framework set by the composer or designer. The highest level of engagement is *co-authorship*, where the audience not only performs but also participates in the creative decision-making process, shaping the artwork itself and also democratizes the creative process.<sup>77</sup>

**Human-Computer Interaction (HCI)** focuses on strategies for audience-system interaction, focusing on the following aspects. *Multiplicity of control* separates between single-user and multi-user systems, with the latter allowing simultaneous interaction by multiple users. The *type of control* the various interfaces used for interaction, which can range from haptic to non-tactile inputs, influencing the nature of the interaction. The *mapping of control action* is the process of mapping user actions to sound parameters, which can be linear or dynamic, affecting how transparent and intuitive the control feels to the user. *Control parameters* refer to specific sound elements that users can manipulate. And *control modality* is whether the control is discrete or continuous influencing the precision and style of interaction.<sup>78</sup>

**Human-Human Interaction (HCI)** emphasizes the concept of collaborating between participants and highlights factors such as location, remote or co-located interactions, levels of communication and collaboration between experts and not experts.<sup>79</sup>

The core idea is that the artwork is not complete without the viewer's participation, making the experience of the art unique and personal for each individual. Interactivity transforms the role of the audience from passive observer to active participant, softens the boundaries between artist and viewer, and often leading to unpredictable and evolving outcomes.

---

<sup>76</sup> cf.: Goudarzi and Gioti, 2016, Engagement and interaction in participatory sound art,

<sup>77</sup> cf.: Goudarzi and Gioti, 2016, Engagement and interaction in participatory sound art,

<sup>78</sup> cf.: Goudarzi and Gioti, 2016, Engagement and interaction in participatory sound art,

<sup>79</sup> cf.: Goudarzi and Gioti, 2016, Engagement and interaction in participatory sound art,

### 3 Identifying the Technical Approach for the Sound Installation

In this project, the artistic concept and the technical realization through engineering were closely linked together. In order to find suitable solution for this installation, it was crucial to understand the artistic motivation and the reason why a specific requirement was important to the final result.

Technical solutions come with limitations, which could contradict the artistic concept. External limitations, such as budgeting, schedules, personal capacities, supply chains, etc. could also stand in the way of the technical implementation process. This project was an exercise in acknowledging limitations and transforming them into artistic features. This chapter analyzes the artistic concept described in chapter 1.2 to identify the requirements necessary to create a successful installation.

#### 3.1 Requirements Analysis Based on the Artistic Concept

Before it was possible to create an adequate list of the most important requirements, it was important to understand the purpose of the installation and what it should do. The most important part was the interaction between participants, and the installation itself. The idea was to assign one sound element to each participant when they entered the installation space. The participants' movement in the horizontal plane should influence the sonic characteristics of the sound. At the same time, the location of the origin of the sound should correspond to the location of the participant. This should enable nonverbal communication between the participants through observation and listening.

At the same time, it was important to keep the use of professional audio-visual equipment to a minimum in order to keep the overall cost of the installation as low as possible. A second cost-minimizing approach was to use as many open-source software solutions as possible to enable communities to adapt the installation to their needs or interests as needed. Keeping the size and weight at a minimum would ensure shipment of the installation was easier and cheaper as well maintaining a small footprint for storage. It was also important to ensure that the setup process could be performed by a non-professional individual.

A special requirement for a minimum footprint for the installation was set for a minimum ceiling height of 3.5 meters and a diameter of 5 meters.



In summary, the key technical requirements were:

- Detection and tracking
- Sound generation
- Digital to Analog (D/A) conversion
- Amplification
- Sound reproduction
- Communication protocols
- Cost, size and weight efficiency
- Use of open-source software
- Easy set up process
- No professional audio/video equipment

### 3.2 Ideation for Solutions Based on the Requirements

The next step involved identifying solutions that best met the majority of the technical requirements. In order to demonstrate the variety of possible solutions, the requirement to not to use professional audio and video technology was ignored. The chosen solutions are highlighted in bold.

#### Detection and tracking

- Ultrasonic sensors
- Pressure sensors in the floor
- **Camera object tracking**
- Manual tracking with operators
- Tracking transmitter
- Production-ready software (TouchDesigner, Max)

#### D/A conversion

- Traditional audio interface
- Stage box
- Network audio converter
- **SBC + D/A converter module**

#### Amplification and sound reproduction

- Active loudspeaker system
- Passive loudspeaker system
- **Custom speaker design**
- 8 x 1-channel amplifier
- **8-channel amplifier**
- Network amplifier

#### Sound generation

- Modular synthesizer
- Digital audio workstation
- Live performance
- **Visual programming language (Max or PD)**
- Audio programming language (Faust, Csound, Supercollider)

#### Communication protocols

- Network protocol (OSC, **TCP, UDP**)
- Serial
- GPIO connection

### 3.3 Finalizing the Technical Approach

Chapter 3.2 gives a rough insight into the available solutions, most of which could be interchangeably combined to create a functional overall solution in order to realize the sound installation from a technical perspective. The use of a professional small form factor Network Device Interface (NDI) camera with a connection to a computer running TouchDesigner to control an Ableton session or Max patch feeding an active array of 8 to 12 speakers over a digital to analogue converter (DAC) would be a suitable choice.

However due to the requirements outlined in chapter 3.1, a more out-of-the-box thinking approach had to be explored, which is detailed below.

#### 3.3.1 Detection and Tracking Approach

##### **Camera-Based Object Tracking**

Object tracking was accomplished through a camera-based tracking solution that followed a traditional TBD approach based on the principles discussed in chapters 2.2.2 and 2.2.3. This solution had some technical and artistic benefits. From a technical perspective, one advantage was that prebuilt hardware and software solutions were widely available, and resources and knowledge bases were accessible. This enabled a wide range of combinations of hardware and software to choose from.

Since the goal was to create a small, light, and cost-effective solution, the use of a small but powerful SBC was a fitting choice. The Raspberry Pi 4 was a suitable choice not only since it is widely available for an affordable price but one was already available on hand. There is an active community around the RPI which enabled easy access to resources, code examples, and tutorials online. This platform also offered a huge variety of aftermarket hardware extensions like battery packs, DACs, sensors, and cameras.<sup>80</sup> Due to the small form factor cameras which are available, it would be possible to create a compact computing and camera unit.

The above-mentioned arguments supported the camera-based tracking approach from an artistic perspective as it was crucial for the participants to enter the installation space seamlessly to avoid spoiling what might happen after entering. This would lead to a more connected experience and encourage participants to explore and experiment.

---

<sup>80</sup> cf.: Sertronics-GmbH, senasoren-module, <https://www.berrybase.de/sensoren-module/>,

### 3.3.2 Solution for Sound Generation and D/A conversion

#### **SBC + D/A Converter Module**

Similar to the approach taken for object tracking, the preferred choice was also for the SBC to handle the audio processing. Max, RNBO or PD would be fitting software solutions to perform the audio processing on an SBC. While Max has the capability to export patches using the RNBO extension directly onto an RPI, it required a license to operate and therefore conflicts with the artistic premise to keep the project open source as much as possible.<sup>81</sup> PD was the open-source alternative and therefore preferred for this application.

Once the sound engine was decided, it was necessary to find a suitable SBC to run PD. At first, an RPI seemed to be a fitting solution to handle the audio processing and was actively considered. However, during the first prototyping phase, it was discovered that outputting an 8-channel audio signal on a RPI would be more challenging than expected. It was possible to connect the RPI to USB audio interfaces that support multichannel D/A conversion, but this was considered to be too big of a compromise to the artistic concept and was not pursued. A more suitable SBC was the Bela. It offers native PD support and, in combination with the CTAG module, D/A conversion for up to 16 channels was possible. The optimized architecture for real-time applications was an additional benefit for this project<sup>82</sup>.

### 3.3.3 Amplification and Sound Reproduction Solution

#### **Custom Speaker Design + 8-channel Amplifier**

Since a key part of the artistic concept was the localization of the origin of each sound element corresponding to the participants' positions inside the installation, a multi-channel surround speaker array was desired. Early in the process, it became apparent that a speaker array with 16 or 32 individual speakers would not be possible due to access to resources. Therefore, the decision was made to reduce the number of speakers to 8 to form a 360-degree speaker array in the horizontal plane.

The speakers used in this project were self-built, using mostly materials available in hardware stores. Even though active studio monitors would have saved valuable development time, this extra effort was important for the artist. Aesthetically, a speaker could immediately provide some insight about the installation and hint towards sound reproduction. A 360-degree speaker array could also give a lot of insight into the installation before a participant even has entered the installation space.

---

<sup>81</sup>cf.: Cycling'74, Max, <https://cycling74.com/shop/max>,

<sup>82</sup> cf.: Bela.io, CTAG multichannel audio board, <https://learn.bela.io/products/multichannel/ctag-multichannel-board/#ctag-multichannel-audio-board>,

The goal was to integrate them in such a way that they did not immediately reveal themselves as speakers this could influence the participants' experience.

Due to time and capacity constraints, it was decided to use a professional-grade 8-channel Class D amplifier, leaving this as the last area in the entire signal chain not conforming to the set requirements.

#### 3.3.4 Enabling Audience Engagement and Interaction

To enable participants to engage and interact with the installation, several concepts presented in Chapter 2.6 were incorporated. The installation remains silent when unoccupied to capture the participants attention, and when they interact with it, their direct influence on the sound should be immediately apparent. The primary goal was to design the installation according to the concept of performance agency, allowing the audience to control key aspects like pitch and effect parameters in real-time. HCI should support simultaneous multi-user interaction, with participants' movements and collaboration within the installation serving as the main form of control. Chapter 4.2 delves deeper into the process of mapping control actions and explains how control parameters and values are generated.

### 3.3.5 Signal Flow Diagram

A flow diagram visualizing the intended signal chain is shown in Figure 7.

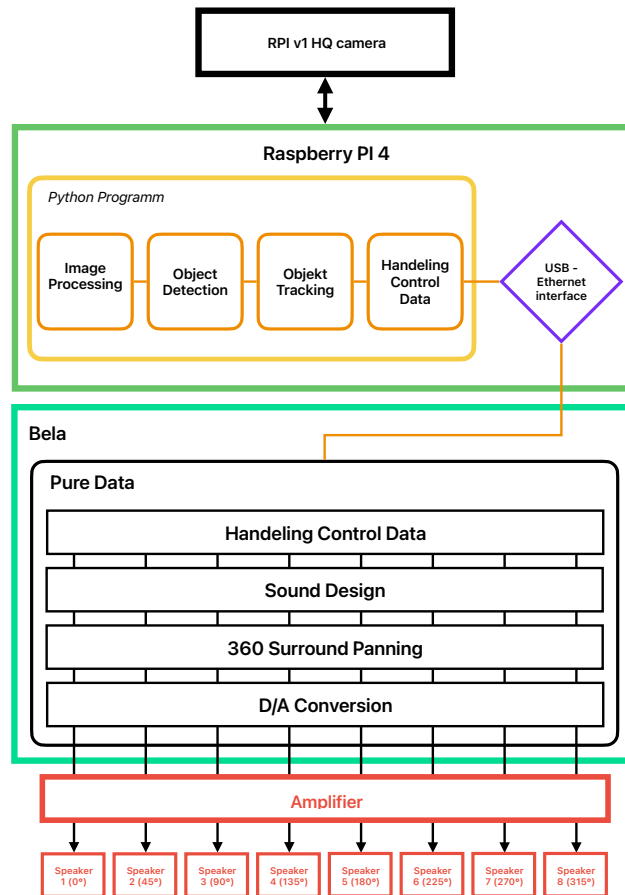


Figure 7: Flow chart showing the technical concept, components, and functions

## 4 Development, Evaluation and Implementing Improvements of the Sound Installation

This chapter highlights the development, evaluation and implementing improvement in relation to the most important components and solutions: the object tracking algorithm, control data processing, audio processing, D/A conversion and speaker array. The aim is to give insight into the development process and the ideas behind it. Furthermore, an attempt is made to show the decision-making process for each technical solutions and to clarify their technical and artistic motivations.

### 4.1 Approach for the Object Tracking Algorithm

As already discussed in chapter 2.2, many approaches to realize camera-based object tracking were available. At first, it was considered to use deep learning-based methods. Tests in the early stages showed that pretrained models like YOLO caused

high Central Processing Unit (CPU) loads on the RPI4 resulting in frame rates of one frame per second (FPS), which was not responsive enough. It was decided to follow a different strategy instead of developing a self-trained model and optimizing the approach.

Inside of Max, it is possible to create a simple tracking technique. David Tinapple highlights his approach, which he refers to as blob-tracking, in his online course "Motion Responsive Soundscapes".<sup>83,84</sup> His technique served as the main influence to develop a similar approach in Python to perform the image processing and computing operations. The concept is to detect differences in pixel intensity by assuming moving participants entering the scene appeared less bright in front of a white background. Because the installation takes place in a controlled environment, the floor color and lighting could be predetermined.

The object tracking algorithm was programmed in Python using the Open-Source Computer Vision library (OpenCV), which is designed for image processing and computer vision operations.<sup>85</sup> The algorithm is handled in the script `object_tracking.py`<sup>86</sup> and followed a traditional approach using standard image processing and the concept of TBD. The foundation of this tracking algorithm is based on the code example by Sergio Canu, which used the YOLO4 algorithms for detection and classification.<sup>87</sup> This foundation has been modified in a variety of ways to accommodate the technical approach and given requirements. The detection and classification have been replaced; however, the re-identification aspect of the algorithm is still present.

#### 4.1.1 Choice of Camera Integration

The camera chosen was the RPI v1 HQ camera, as this camera was already in inventory. To maximize the field of view for spaces with limited ceiling heights, an aftermarket fisheye lens was chosen. The camera uses the Sony IMX477 sensor with an optical size of 1/2.3", 12.3 megapixels, and a sensor resolution of 4056 x 3040 pixels. In video mode, the maximum resolution is 2028 × 1520 for 40 FPS.<sup>88</sup>

The physical connection between the RPI and the camera was a second-generation Mobile Industry Processor Interface (MIPI CSI-2). It is a high-performance, multi-layer

---

<sup>83</sup>cf.: Tinapple, Max/MSP - Blob-tracking, <https://www.youtube.com/watch?v=cytx9NqSQNA>,

<sup>84</sup> cf.: Tinapple, Tracking - Blob Tracking, <https://tinapple.notion.site/Tracking-Blob-Tracking-9b5b314087074429808b53bf4598e4de>,

<sup>85</sup> cf.: OpenCV, Introduction, <https://docs.opencv.org/4.x/d1/dfb/intro.html>,

<sup>86</sup> Source code 2: Object tracking `object_tracking.py`

<sup>87</sup> cf.: Sergio-Canu, Object tracking from scratch – OpenCV and python, <https://pysource.com/2021/10/05/object-tracking-from-scratch-opencv-and-python/>,

<sup>88</sup> cf.: Raspberry-Pi, About the Camera Modules, <https://www.raspberrypi.com/documentation/accessories/camera.html#hq-camera>,

protocol designed for low-power video applications and supports a maximum bandwidth of 10 Gbit/s. The support for the MIPI D-PHY layer enables embedded communication with the application processor.<sup>89</sup> The MIPI CSI-2 interface is more efficient for this application. Even though the SDI standard SMPTE ST-2082 supports a bandwidth of 12 Gbit/s, the need for format conversions and interfacing over the USB standard would make this a less efficient solution.

#### 4.1.2 Performing Image Processing to Enhance Tracking Capability

Before the object detection algorithms can operate, a few image processing steps were necessary after establishing a connection to the camera module.

The first step was setting the basic camera configurations. The color format was set to YUV420, which has the benefit of accessing a grayscale image directly through its Y channel, saving the step of converting an RGB signal to grayscale.<sup>90</sup> The resolution was set to 1520 x 1520 to maximize vertical information and minimize data size, as the extra horizontal information was not necessary since the region of interest was a circle, corresponding to a 1:1 aspect ratio. The frame rate was set to 30 FPS.

To quantify processing efficiency and the benefit of using the YUV420 color format, the time needed to compute a single frame by the tracking algorithm was averaged over 30 seconds. Using the total number of computed frames during that timeframe, the average frame rate could be calculated.

<code>final test yuv420</code>	<code>final test with rgb</code>
-----	-----
Total frames: 859	Total frames: 593
Total time: 29.18 seconds	Total time: 29.46 seconds
Average FPS: 29.44	Average FPS: 20.13

Figure 8: Terminal output `object_tracking.py` FPS test between YUV420 and RGB

The results showed that using the YUV420 format, it was almost possible to operate within the predefined frame rate range. In comparison, using the RGB format caused the average frame rate to drop by about 32%.

Because the tracking was based on detecting brightness levels, a grayscale image contained all the information necessary. The image processing and tracking algorithm was performed for each frame individually. Therefore, the entire processing was wrapped inside a `for` loop, where, at the beginning of each iteration, a new frame was captured from the camera as a bit array and stored as `frame`.

---

<sup>89</sup> cf.: MIPI-Alliance, DRAFT MIPI Alliance Specification for Camera Serial Interface 2 (CSI-2), [https://caxapa.ru/thumbs/799244/MIPI\\_Alliance\\_Specification\\_for\\_Camera\\_S.pdf](https://caxapa.ru/thumbs/799244/MIPI_Alliance_Specification_for_Camera_S.pdf),  
<sup>90</sup> cf.: Schmidt, 2013, Professionelle Videotechnik: Grundlagen, Filmtechnik, Fernsehtechnik, Geräte-und Studiotechnik in SD, HD, DI, 3D,

To prevent unnecessary processing the tracking algorithm should only be applied to the area inside the speaker array. For this purpose, a mask called `mask_read` is created to overlay onto the `frame`. The `mask_read` is filled as a pixel array with the same dimensions as the `frame` with white pixel values. A black circle is then drawn with its center point in the center of the image and the radius defined by the variable `radius_circle`. This variable can be set by the user to adjust the size of the tracking area. The `mask_read` is applied to the `frame`, creating `frame_mask`, using a bitwise OR operation, which keeps the region of the original image inside this circle intact while turning the area outside the circle white see Figure 9.

```

300 # create a mask to define tracking area and overlay
    onto frame
301 mask_read = np.zeros_like(frame)
302 mask_read = 255 - mask_read
303 mask_read = cv2.circle(mask_read, center_xy,
    radius_circle, (0,0,0), -1)
304 frame_mask = cv2.bitwise_or(frame, mask_read)

```

Figure 9: Creating a mask as an overlay in order to define the tracking area (Source code 2)

The next steps highlighted in Figure 10 involved the inversion of the image into `img_inv` and applying a linear filter function `cv2.blur`, similar to the ones presented in chapter 2.2.1, However, this filter is called a box filter, which performs arithmetic averaging to create a blurring effect.<sup>91,92</sup> The goal is to merge local intensity differences, adjusting for variations in brightness due clothing, backpacks, and other factors to create a more uniform and trackable object. The purpose of the if statement was for error handling, due to user input for system calibration.

The last step involved thresholding to separate the objects of interest from the background using the `cv2.threshold` function.<sup>93</sup> The threshold can also be adjusted by user input for calibration purposes. To further enhance the thresholding, the adaptive OTSU algorithm is used to account for local intensity differences.<sup>94</sup>

<sup>91</sup> cf.: McDonnell, 1981, Box-filtering techniques,

<sup>92</sup> cf.: OpenCV, Smoothing Images, [https://docs.opencv.org/4.x/d4/d13/tutorial\\_py\\_filtering.html](https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html),

<sup>93</sup> cf.: OpenCV, Image Thresholding, [https://docs.opencv.org/4.x/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html),

<sup>94</sup> cf.: Xu et al., 2011, Characteristic analysis of Otsu threshold and its applications,



A visualization of all image processing steps is provided in Appendix 2 using a custom test image showing different shapes in different colors.

```
309 # processes frame invert blurr and convert to binary
310 img_inv = cv2.bitwise_not(frame_mask)
311 # apply blur (liniar filter - type box)
312 if kernel_size > 1:
313     img_blurred = cv2.blur(img_inv, (kernel_size,
314                                   kernel_size))
315 else:
316     img_blurred = img_inv.copy()
317 # perform thresholding
ret, img_binary = cv2.threshold(img_blurred,
                                binary_thresh1, binary_thresh2, cv2.THRESH_BINARY +
                                cv2.THRESH_OTSU)
```

Figure 10: Image processing of target frame: inversion, filtering and thresholding (Source code 2)

#### 4.1.3 Implementing Traditional Object Detection

The object detection method primarily relied on the OpenCV function `cv2.findContours` to identify all contours of nonzero regions.<sup>95</sup> The contours are stored as pixel arrays referred to as `cnts`. A for loop is used to analyze all contours found in the current frame. To filter for nonzero regions too small to represent a person, a minimum threshold for the number of pixels forming a contour is defined through user input as `contour_size`. Only contours above this threshold are processed further.

The next step shown in Figure 11 identified the center point of each contour using the `cv2.moments` function. This function calculates image moments for all detected contours, such as `m00` the number of pixels forming the contour, `m10` the sum of all x-axis pixel coordinates, and `m01` the sum of all y-axis pixel coordinates<sup>96</sup>. Dividing `m10` by `m00` and `m01` by `m00` results in the average x and y pixel positions. Each set of pixel coordinates (PXC) is converted to an integer, as PXC values are discrete, and saved to the list `center_points_cur_frame`.

<sup>95</sup> cf.: OpenCV, Contours : Getting Started, [https://docs.opencv.org/4.x/d4/d73/tutorial\\_py\\_contours\\_begin.html](https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html),

<sup>96</sup> cf.: OpenCV, Image Moments, [https://docs.opencv.org/3.4/d0/d49/tutorial\\_moments.html](https://docs.opencv.org/3.4/d0/d49/tutorial_moments.html),

```

329 # find contours and find total area
330 cnts = cv2.findContours(img_binary, cv2.RETR_EXTERNAL,
331                          cv2.CHAIN_APPROX_SIMPLE)
332 cnts = cnts[0] if len(cnts) == 2 else cnts[1]
333 for c in cnts: # find center of contour
334     pxc = np.count_nonzero(c)
335     if pxc > contour_size:
336         # compute the center of the contour
337         M = cv2.moments(c)
338         if (M["m00"] != 0):
339             cX = int(M["m10"] / M["m00"])
340             cY = int(M["m01"] / M["m00"])
341             center_points_cur_frame.append((cX, cY))
342         else:
343             cX, cY = 0, 0

```

Figure 11 Finding contours and computing moments for PXC extraction (Source code 2)

#### 4.1.4 Implementing Traditional Object Tracking

All relevant objects were detected in the current frame and their positions were saved as a PXC. As TBD was used, it was necessary to compare and identify objects detected in the previous frame with those in the current frame. Each PXC saved in `center_points_cur_frame` was assigned a Global Identification number (GID), which was an integer that incremented by one for each new object detected. Both the PXCs and the GID were saved in `tracking_objects`, and the contents of `center_points_cur_frame` were copied to `center_points_prev_frame`. After the first frame had been processed, the distance between all PXCs from the current and previous frames was calculated. If the distance was below a user defined threshold called `tracking_distance`, it could be assumed that both PXCs belonged to the same object, and the updated PXC could be assigned to the object's GID. All detected objects in `center_points_cur_frame` that did not meet these criteria were added as new objects and given a new GID. The final step involved checking `center_points_prev_frame` for old objects that also did not meet these criteria and could now be deleted. This was handled in lines 346-379 in Source code 2.

#### 4.1.5 Implementing a Calibration Method

To minimize tracking errors, system calibration was essential. Since the tracking process relied heavily on detecting variations in brightness, its reliability depended on several key factors, including ambient lighting, background color, camera exposure, aperture, positioning, and the color contrast between the visitor and the background. Given that the installation is situated in a controlled environment, ambient brightness and background color could be predetermined. Ideally, a bright, evenly lit environment with a light-colored floor was preferred.

While the goal was to create a setup that eliminated the need for manual calibration, an automated and adaptive calibration process would be ideal. However, given that this was a prototype, the focus was placed on more critical aspects of development, with the possibility of integrating automated calibration in the future.

Despite this, the ability to calibrate the system during the prototype stage remained crucial. Visual feedback at various stages of image processing and tracking, real-time parameter adjustments, and a wireless connection were necessary to streamline the setup process and maintain flexibility in connecting to the unit without relying on long HDMI cables.

To meet these requirements, a web interface was developed to receive a video stream from the RPI4 and provide sliders for controlling image processing and tracking parameters. This web interface is hosted on the RPI4.

During the prototype phase, running the main installation program required connecting to the Wireless Local Area Network (WLAN) hotspot that was automatically initialized on the RPI at startup. A Secure Shell (SSH) connection could be established to launch the program via terminal commands. Upon executing the main Python program called `main.py`<sup>97</sup>, user input is required to either initialize a web server through executing the `http_video.py`<sup>98</sup> script or running the tracking algorithm script `object_tracking.py`<sup>99</sup> independently for enhanced performance.

If the user opted to initialize the web server, the HTML `login.html`<sup>100</sup> and `index.html`<sup>101</sup> are sequentially executed. The user could then access the calibration web interface through a standard browser using the IP address printed in the terminal. Parameter changes are handled with the `config.py`<sup>102</sup> script, saving changes to the `config.json`<sup>103</sup> file, ensuring that settings could be reloaded after a system reboot. Figure 12 shows a graphic overview of the Python scripts executed on the RPI4. Appendix 3 shows the calibration web interface and demonstrates a basic calibration process: adjustment of the tracking area, toggling of the binary image, adjustment of the threshold, and specifying the amount of average filtering. The footage was taken during the second test setup which will be covered in chapter 4.5.3.

---

<sup>97</sup> Source code 1: Object tracking

<sup>98</sup> Source code 3: Object tracking `http_video.py`

<sup>99</sup> Source code 2: Object tracking `object_tracking.py`

<sup>100</sup> Source code 7: Object tracking `login.html`

<sup>101</sup> Source code 6: Object tracking `index.html` calibration web interface and receiving video stream

<sup>102</sup> Source code 4: Object tracking

<sup>103</sup> Source code 5: Object tracking `config.json` saved

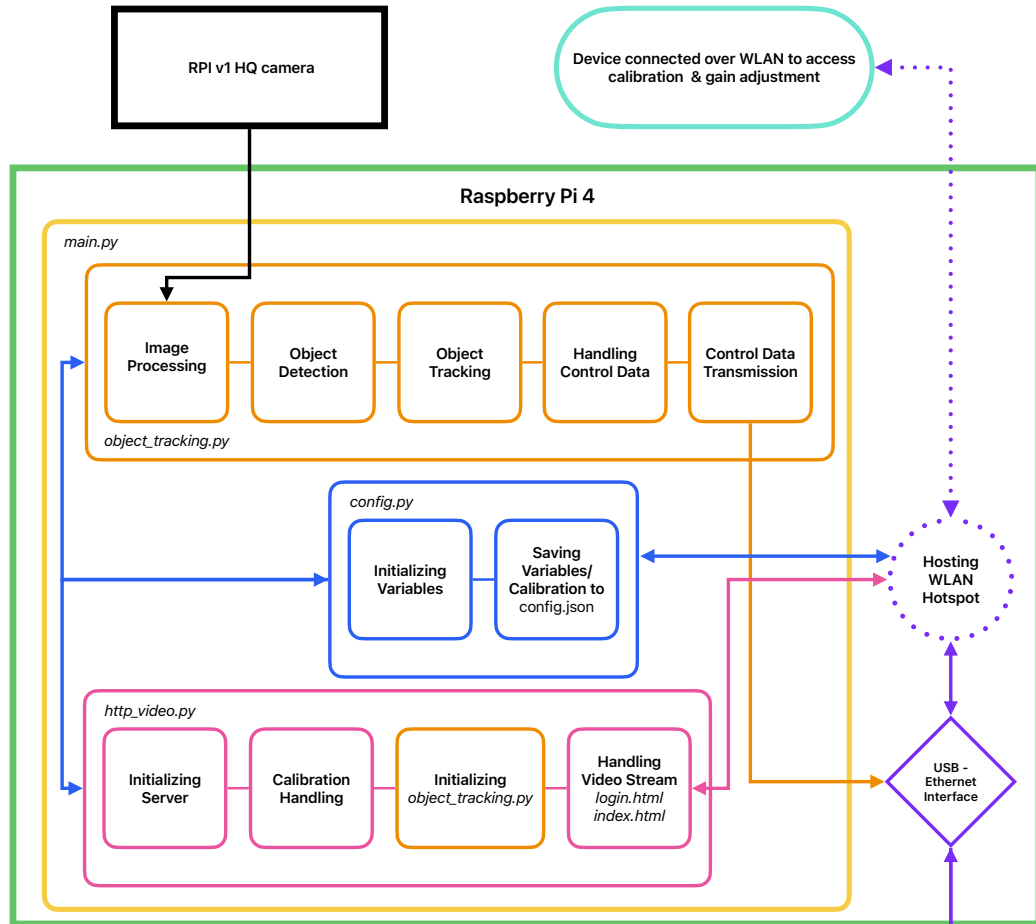


Figure 12: Graphic over few on the relations between the Python scripts executed on the RPI4

## 4.2 Control Data Processing for Human Computer Interaction

The results of the tracking algorithm must be converted into control data optimized for PD to control various aspects of the sound design to enable HCI. Therefore, it was important to understand the key requirements of the artistic concept, even though the final sound design was not yet determined at this stage.

The key requirements were a clear assignment of each tracked participant to a specific sound element, spatial-acoustic projection of the assigned sound element to the spatial position of the tracked object, and control of effect parameters of the sound engine, such as reverberation times and delay times through participants' movement.

From this, it followed that the control data should be as neutral as possible to allow flexibility in applying it to later parameters, without knowing the exact threshold values, quantity, or scope at this time. Since the audio processing took place on the Bela a communication between the RPI4 and Bela needed to be established.

#### 4.2.1 Clear Assignment of Tracked Participants to the Sound Elements

Regardless of the type and format of the control data, it had to be ensured that each object was uniquely assigned to a sound element. As there was a finite number of 8 sound elements, but the GID is assigned in an infinite sequence. Another requirement by the artistic concept was to ensure that the 8 available slots are continuously refilled starting from the first slot. To prevent confusion within the tracking algorithm, an additional data structure was created which consists of a limited number of possible entries all indexed from 1 to 8. called `rnbo`. Each indexed entry was filled sequentially with the current detected GID's and PXC's. This was handled in lines 380-577 in Source code 2.

#### 4.2.2 Projecting the Sound Element to the Participants Location

It was important to convert the PXC's into a more suitable format for later use in PD. Since the speaker array and tracking area were circular, a polar coordinate system with its origin located at the center of the image was used. The azimuth angles and octants were defined as shown in Figure 13. A coordinate transformation was performed, with the center PXC of the image as the new origin. The distance from the center point to the new coordinate was calculated using trigonometric operations as shown in Figure 14.

To calculate the azimuth angle, it was necessary to determine in which quadrant the object is positioned. This step was handled by two nested if-else statements that checked the signs of the coordinates. For example, if the x-coordinate was less than zero, the object is either in quadrant 2 or quadrant 3. If the x-coordinate was greater than zero, it was either in quadrant 1 or quadrant 4. Once the quadrant was identified, the angle relative to the quadrant could be calculated using the arctangent operation. Adding the appropriate angle components  $90^\circ$ ,  $180^\circ$ , or  $270^\circ$  resulted in the azimuth angle relative to the entire circle. The operation for quadrant 1 was shown in Figure 15.

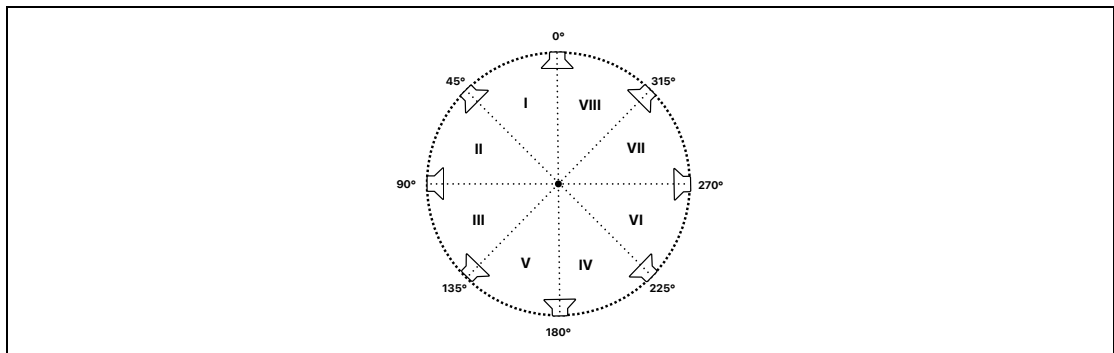


Figure 13: Speaker placement with corresponding azimuth angle and octants I - VIII

```

444 #calculate relative xy koordinats to the center of
    Frame
445 angel_xy = tuple(map(operator.sub, pt, center_xy))
446 #calculate distance angle_xy to center_xy
447 angel_radius =
    np.sqrt((angel_xy[0]**2)+(angel_xy[1]**2))

```

Figure 14: Calculating new xy coordinates and distance from PCX to origin (Source code 2)

```

448 # calculate the azimuth angel of each position
449 if angel_xy[0] < 0:
450     if angel_xy[1] < 0:
451
452         angel_devision= (angel_xy[1]/ angel_xy[0])
453         #print(angel_devision)
454         azimuth = (math.degrees(math.atan(angel_devision)))
455         azimuth_1 = (90-azimuth)

```

Figure 15: Calculating the azimuth angle for the first quadrant (Source code 2)

### 4.2.3 Control Data for Interaction with Audio Effects

As a method to find control data that could be used to control effects in PD, there was the artistic directive to transfer the movement of the objects as an XY-matrix. Two types were already established, the pixel and polar coordinate systems. Using one of those two might result in similar control patterns and could appear repetitive, so it was artistically sensible to use a new coordinate origin to increase variation. The choice falls on the classic Cartesian coordinate origin, only using positive axes. The origin was defined by the intersection of the 90° and 180° tangent and the axes maximum limits are defined by the diameter of the tracking circle. The coordinate system scales with the adjustable circle radius of the tracking mask. A coordinate transformation was performed and, to ensure the usability for PD, the x and y coordinates were normalized. The last data type was the distance from the PXC to the center point of the image which was also normalized. Both steps were handled as shown in Figure 16.

```

480 x_dict[k] = "%.3f" % ((pt[0] - x1) / (2 *
    radius_circle))
481 y_dict[k] = "%.3f" % ((y1 - pt[1]) / (2 *
    radius_circle))
482 radius_dict[k] = "%.3f" % (angel_radius /
    radius_circle)

```

Figure 16: Calculating normalized x, y coordinate and radius (Source code 2)

### 4.2.4 Transferring the Control Data from the RPI to the Bela

After all of the control data as processed and assigned to the corresponding objects, the next step involved transferring the data from the Python script running on the RPI4 to PD running on the Bela. Several communication methods could be established between the RPI4 and the Bela: serial connection through USB or breakout pins, a

network connection through USB, or an ethernet connection. Since the Bela opened a network connection with a static IP address by default over the mini-USB port, this proved to be the quickest implementation. PD Vanilla has pre-built objects to handle network communication using UDP, TCP, or OSC protocols.<sup>104</sup> While OSC would be a suitable choice for music and audio applications as highlighted in chapter 2.3.3, it was not necessary for this application since the message structure was expected to be simple. Therefore, the choice was between UDP and TCP.

As discussed in chapter 2.3.2, UDP had the downside of potential packet loss and no guarantee of packet ordering during communication. However, it was still chosen for this application. The main reason was the prioritization of a continuous running sound installation over precise network communication. An unexpected loss of connection leading to a program crash if the TCP server and client were not written with precise error handling was anticipated and would extend development time and require more debugging processes. As this was an interactive real-time application, the transportation speed was preferred over eventual data loss. The maximum size for a floating-point number is 8 bytes defined by the Institute of Electrical and Electronics Engineers.<sup>105</sup> The total size of each message consists of a set of 4 x 8 floating-point values, resulting in a total message size of 256 bytes. Since these messages are sent sequentially and each message does not exceed the MTU of 507 bytes, UDP was deemed sufficient for transmission and leaves room for expansion of the data set.

The first step involved the preparation of the data sets: azimuth angle, x-coordinate, y-coordinate, and radius. Each of these consisted of eight float or integer values, all in the order of the corresponding object id. These four data sets were consolidated into the list `PD_list`, which was then converted into a single string and saved as the variable `MESSAGE`. Next, an internet and UDP socket was created, and for every processed frame, one message was sent using the `socket.sendto` function. This was done by encoding the `MESSAGE` into a binary format and specifying the client's IP address and port number, as shown in Figure 17 and Figure 18.

This was the last essential step performed on the RPI4, all remaining processes were handled directly on the Bela.

```
207 | #set variables for UDP-Socket for Controldata to Bella
208 | UDP_IP1 = ip_address #Bella IP
209 | UDP_PORT1 = 3001
```

Figure 17: Setting the IP address and port for UDP socket (Source code 2)

---

<sup>104</sup> cf.: IEM, [netreceive], <https://pd.iem.sh/objects/netreceive/>,

<sup>105</sup> The Institute of Electrical and Electronics Engineers, 2008, IEEE Standard for Floating-Point Arithmetic,

```

550 | # Sending Message to Bella/Pure Datat over UDP Socket
551 | MESSAGE = " ".join(str(x) for x in PD_list)
552 |
553 | sock = socket.socket(socket.AF_INET, # Internet
554 | socket.SOCK_DGRAM) # UDP
555 | sock.sendto(MESSAGE.encode(), (UDP_IP, UDP_PORT))

```

Figure 18: Creating a message from PD\_list and sending over UDP socket to the Bela (Source code 2)

### 4.3 Performing Audio Processing and Sound Design

#### 4.3.1 Receiving and Processing of the Control Data

In order to receive the messages sent from the RPI4 over UDP to PD the `[netreceive -u -b 3000]` object acted as the UDP client inside the subpatch `[upd_data]` as illustrated in Figure 19. The parameter `-u` specified the use of the UDP protocol, the parameter `-b` set the message type to binary and the `3000` set the port number.<sup>106</sup> The following object `[fudiphrase]` converted the received message to the PD message protocol Fast Universal Digital Interface (FUDI) which resulted in the initially sent list `PD_list`.<sup>107,108</sup> To separate the list and isolate each value, the object `[list split]` and `[unpack]` was used in sequence. Following the `[unpack]` object, a simplified data smoothing function was performed. The object `[pd avr]` was a custom subpatch in which a running average over a four-value window is calculated as illustrated in Figure 20. All incoming numeric values were added to the first position of a list limited to a total of four elements in the `[pack f f f f]` object. This list was unpacked, and element 1 is added to the second position, element 2 to the third position, and element 3 to the fourth position of the original list, and element 4 was discarded. At the same time, the `[unpack]` object forwarded the numeric values into a summing operation. The multiplication by the factor 0.25 resulted in the running average.

After the receiving and smoothing process is handled inside the PD subpatch, every value is sent into a `[send 1x]` object. The first number referred to the assigned tracking id and the letter specified the data type: `x` = x-coordinate, `y` = y-coordinate, `a` = azimuth angel, and `r` = radius. This assignment using the send object ensured the access to each value inside the entire PD patch. For easy access, all corresponding returns were listed at the beginning of the main patch.

<sup>106</sup> cf.: IEM, `[netreceive]`, <https://pd.iem.sh/objects/netreceive/>,

<sup>107</sup> cf.: IEM, `[fudiparse]`, <https://pd.iem.sh/objects/fudiparse/>,

<sup>108</sup> cf.: IEM, `[fudiformat]`, <https://pd.iem.sh/objects/fudiformat/>,



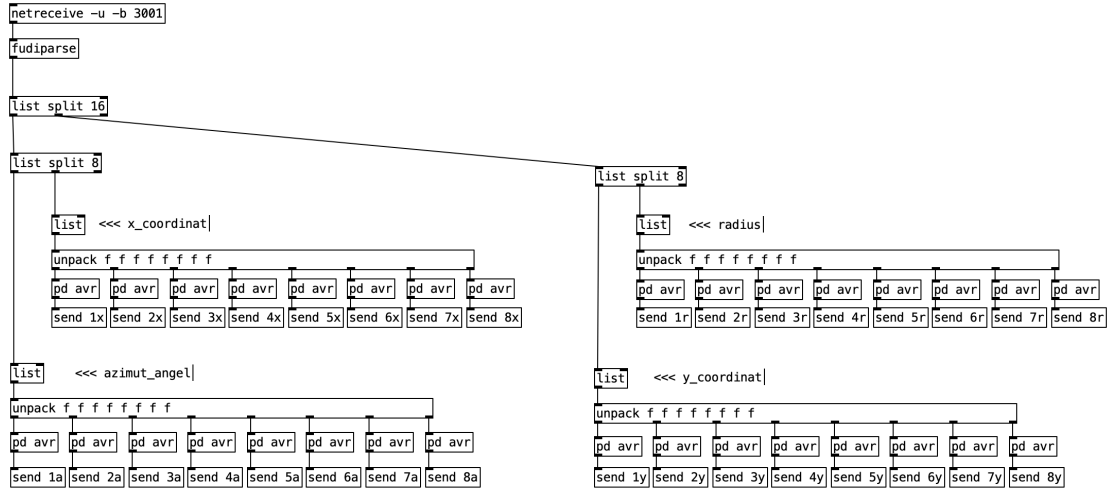


Figure 19: Subpatch [upd\_data], receiving UDP datagram and data handling

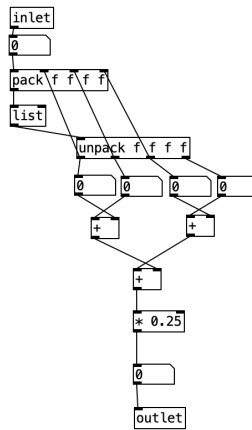


Figure 20: Subpatch [pd avr], running average over a four-value window

### 4.3.2 Implementing Surround Panning for the Sound Element

A key feature of the installation was positioning each sound element within the installation space at the same location as the corresponding person controlling the sound element. This allowed for observing others and recognizing which individual sound element was assigned to them based on their location.

The fastest solution to fit the scope of the project considered was to be a direct implementation within the PD patch. Several open-source surround panning externals were available for PD, such as Ambisonics Toolbox, abclib, and spat8~.<sup>109,110,111</sup> Compatibility issues arose with the libpd versions.<sup>112</sup> Bela supported version 5.2 of PD, which was not fully compatible with the current PD Vanilla version 5.4. This discrepancy affected the ability to easily use externals on the Bela. Externals relying

<sup>109</sup> cf.: Pisano and Lindborg, 2023, Introducing the Open Ambisonics Toolkit,

<sup>110</sup> cf.: Bonardi, abclib, <https://github.com/alainbonardi/abclib>,

<sup>111</sup> cf.: solipd, AudioLab, <https://github.com/solipd/AudioLab>,

<sup>112</sup> cf.: Bela.io, Libpd, <https://learn.bela.io/using-bela/languages/pure-data/#libpd>,

on other audio programming languages like Csound, SuperCollider, or Faust could be especially challenging to implement on Bela, as they often require complex compilation directly on the board using CMake.<sup>113</sup>

During testing, it was revealed that these externals could not be successfully compiled on the Bela or exceeded the processing capabilities of the Bela for this application. As the installation aimed to provide spatial orientation rather than precise spatial representation, a custom, low-performance, stereo-based surround panner had to be developed. To keep the panner simple and quick to program, the reformulated tangent law expressed as Equation 4 covered in chapter 2.1.2 was selected.

Figure 3 shows the created PD abstraction `[panner-abs]` to pan the incoming audio signal from `[inlet~ $1]` to eight `[outlets~]` corresponding to the speaker outputs. Four feed values are calculated: `[feed1]` is the azimuth angle divided by  $45^\circ$  resulting in a floating-point number from 0 to 8 and determines which two speakers receive the audio signal. `[feed2]` is the decimal part of `[feed1]` representing the normalized ratio of the angular position between a stereo pair, and `[feed3]` being the inversion of `[feed2]`. `[feed4]` is the integer part of `[feed1]` and determines if `[feed2]` or `[feed3]` is used to calculate the gain factor.

The stereo panning was only performed once and to calculate the correct gain factor using `[feed2]` or `[feed3]` an if else statement was used to check if `[feed4]` was even or odd. If `[feed4]` was even, the phantom source was located in octant one, three, five or seven. Speaker one, three, five or seven would act as the right channel, receiving `[feed3]` and speaker two, four, six or eight would act as the left channel receiving `[feed2]`. For odd numbers, this relation was conversely.

After the panning process, another series of if statements determined which of the eight speakers received the panned audio signals. To ensure a smooth transition and minimize artifacts, a ramp was used to fade the signal from 1 to 0 in 0.01 seconds.

---

<sup>113</sup> cf.: Bela.io, Abstractions, <https://learn.bela.io/using-bela/languages/pure-data/#abstractions>,

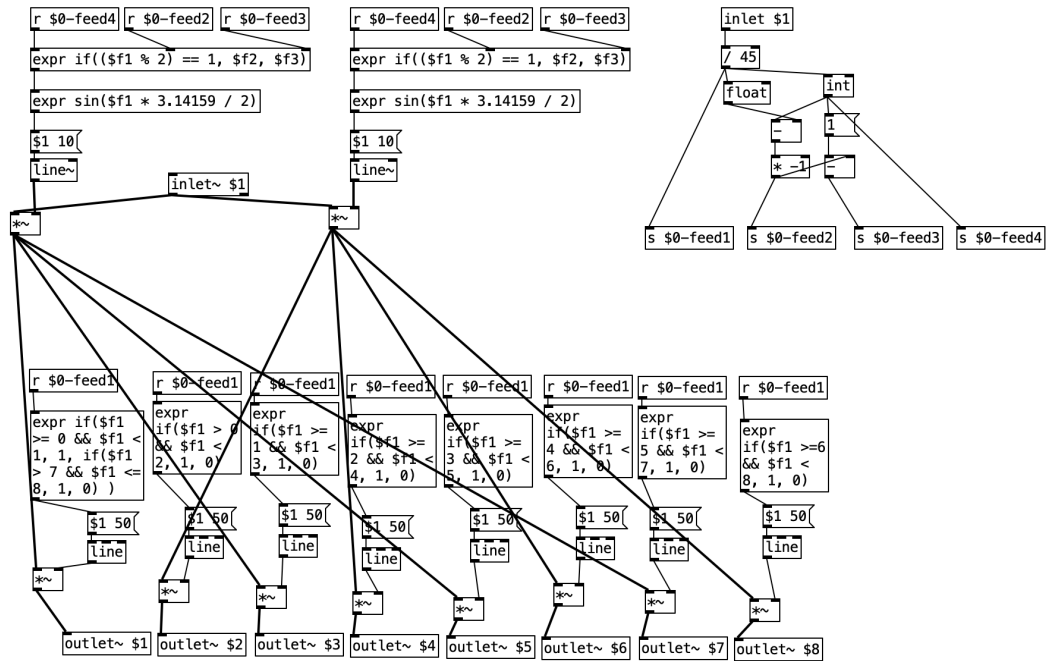


Figure 21: Abstraction [panner-abs], panning operation for one sound element to 8 outlets

Eight [panner-abs] abstractions were combined to [panner-360] to pan the eight sound elements individually, then all outlets are summed and sent to DAC.

This approach was chosen to ensure efficient performance, considering the limited CPU capacity of the Bela. An initial version consisted of eight individual panning functions for each outlet, performing the panning function. This resulted in a 60% CPU load running the [panner-360] in isolation without running audio signals through. Primarily this was caused due to the [line~] object, which acted as an audio ramp generator running at sample rate, enabling sample-accurate value changes of an audio signal without creating audible artifacts.<sup>114</sup> It took two float numbers as its argument: the first was the value change, and the second was the ramp speed in milliseconds. Even if no new value changes were sent, the [line~] object remained active, generating an audio signal.<sup>115</sup> As eight panners for eight audio signals were needed, this resulted in 64 [line~] objects constantly running at sample rate.

The final approach, using two panning functions for each audio signal, reduced the number of [line~] objects to 16. Both version of the [panner-360] were loaded onto the Bela in isolation for CPU load comparison, showing a reduction from 60% to 34%.

<sup>114</sup> cf.: IEM, [vline~], <https://pd.iem.sh/objects/vline~/>,

<sup>115</sup> cf.: IEM, [line~], <https://pd.iem.sh/objects/line~/>,

### 4.3.3 Creating a Sound Design Framework

Due to time constraints, the final sound design could not be fully developed. However, it was possible to provide a suitable framework with room for adaptability during future development. To later evaluate the functionality of the installation, experimental sound designs were tested during the test setups covered in chapter 4.5.

The basic framework shown in Figure 22 consisted of eight individual sound elements. For the first sound design experiments, a few different patches offering various approaches were prepared. These patches used noise generators and oscillators to provide simple placeholder sources to represent future sound elements. These elements could be panned using the [panner-360] and included some pitch and filter modulation. Another approach utilized pre-produced audio samples. Three audio samples represented one sound element: one being the dry signal, one a 100% wet signal of a delay, and one a 100% wet signal of a reverb. The x and y coordinates of the control data were used to fade in the two effect samples according to the x and y position of a participant. Another experiment involved a simple synthesizer sound with its pitch modulated by the distance from the participant to the center point.

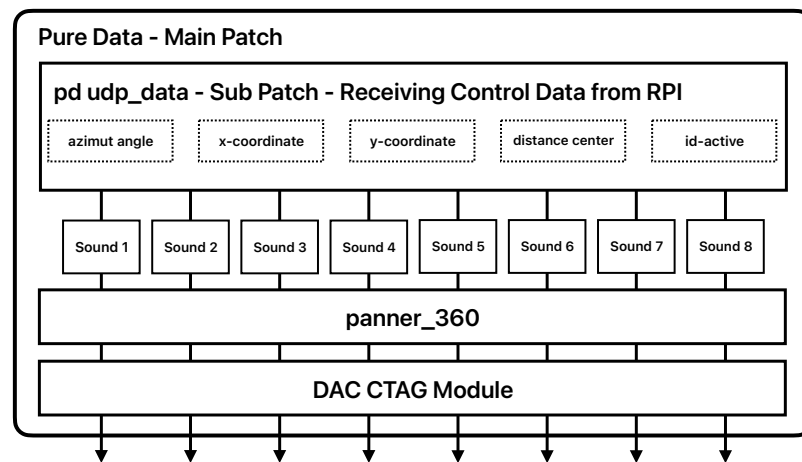


Figure 22: PD sound design framework showing the basic structure of the patch

## 4.4 D/A conversion, Speaker Array, and Amplification

### 4.4.1 Implementing the Digital to Analog Conversion

The digital-to-analog conversion of the eight audio signals was managed by the Bela in combination with the FACE Cape by CTAG. The FACE Cape is a sample-accurate ADAC that supports four analog input and eight analog output channels, as well as real-time environment compatibility using Pure Data, resulting in overall latency times

as low as 1 millisecond.<sup>116</sup> For this specific application, the FACE Cape is configured to run with a 22 kHz sample rate, 16-bit depth, and a buffer size of 256 samples. Analog signals are outputted over four unbalanced stereo channels.

#### 4.4.2 Developing a Custom Speaker Design for the 8-Channel Array

The choice of speaker was primarily dictated by the artistic concept of the installation. The goal was to find a custom speaker design that met the following specific requirements: low cost, no professional equipment, and not instantly discernible as a traditional speaker. The speakers were to be an integral part of the installation, essential for defining a space for participants to enter. The speakers were intended not just as sound reproduction tools but also as aesthetic components of the installation.

The fabrication and assembly of the speakers would need to be possible with minimal tools and materials, allowing for future design adaptations. It was decided to create a self-standing speaker design using exciters as the drivers. The driver assembly consists of a magnet, voice coil, suspension system, and electrical connection terminals. This type of driver was designed to be attached to a rigid surface to stimulate vibrations and create sound. The chosen exciter was the Dayton Audio DAEX32EP-4, a 4-ohm driver capable of handling 40 watts RMS and 80 watts peak power<sup>117,118</sup>.

To accommodate the fabrication and assembly requirements, materials and tools commonly available in construction stores were selected. The main construction material chosen was Styrodur 3035 CS panels. They are lightweight, rigid, inexpensive, and have a B1 fire class rating, making them suitable for the desired application<sup>119</sup>. Connection elements such as gaffer tape, double-sided tape, and nails were used. To dampen vibrations between panel connection points, self-adhesive felt gliders were employed.

All design choices aimed to minimize material waste and maintain a straightforward fabrication approach, with a cutter knife as the only required tool. The main panel, acting as the loudspeaker's membrane, can be used in its original shape. Additionally, two panels were cut into eight equal strips from the shorter side to serve as the base of the loudspeaker. The main panel and the base were connected with two nails at equal distances apart. For additional support, a 15 x 15 cm panel was used to provide

---

<sup>116</sup> cf.: Bela.io, CTAG multichannel audio board, <https://learn.bela.io/products/multichannel/ctag-multichannel-board/#ctag-multichannel-audio-board>,

<sup>117</sup> cf.: Dayton-Audio, EXCITERS & TACTILE TRANSDUCERS 101, <https://www.daytonaudio.com/topic/excitersbuyerguide>,

<sup>118</sup> Appendix 4: Data sheet Dayton Audio exciter

<sup>119</sup> Appendix 5: Safety certificate Styrodur 3035 **Error! Reference source not found.**

more stability, connected with two nails through the base and one nail through the main panel at the front. At each contact point between the panels, 2-4 felt gliders (2 cm x 2 cm) were positioned. To improve the adhesive connection between the felt gliders and the panels, longer pieces of gaffer tape were used to maximize the adhesive area to the panel. The exciter placement followed the manufacturer's recommendations.

Since the speaker design was not intended for source-accurate playback and acceptable results for this application were achieved, analyzing the frequency response of the speaker was not considered to be necessary due to time constraints. Likewise, the selection of a suitable amplifier was influenced by time and resource constraints which led to the Sirius I-Amp 8.150 to be chosen. It is an 8-channel Class-D amplifier with 8x 130 Watt @ 4 Ohm / 8x 70 Watt @ 8 Ohm RMS power. A visualization of the entire processing chain is shown in Appendix 6.



*Figure 23: Custom speaker design from Styrodur 3035 CS panels and the Dayton Audio DAEX32EP-4 exciter*

## 4.5 Evaluating and Improving the Technical Realization

### 4.5.1 Assessing Key Functions Through a First Test Setup

After the initial development phase, a test setup was conducted to assess the key functions of the installation and gain insights for improvements. The first test setup was scheduled for one day at the light studio at BHT-Berlin.

For this setup, the minimum spatial requirement of a 5m diameter and the minimum camera height of 3.4m was replicated. The entire camera unit consisted of the camera, the RPI4, the Bela, and four External Line Return (XLR) sockets wired to

output eight unbalanced audio channels, which were all housed between two aluminum plates shown in Figure 24. To accommodate different rigging applications, three M6 eye nuts were added to the back plate in a triangular formation, and one M6 eye nut was added in the center. A custom-made 20m multicore cable was used to connect the four XLR outputs to the amplifier.

The camera unit was rigged 3.4m above the center point of the tracking area using an 8mm steel wire cable with an additional safety line. The video stream from the RPI4 to was used to adjust the tracking area to the required 5m diameter. All eight speakers were placed at their predefined azimuth angle positions using basic measurements from the center point of the camera position. It was discovered that the visual feedback from the video stream provided a beneficial reference to verify the speaker positions.

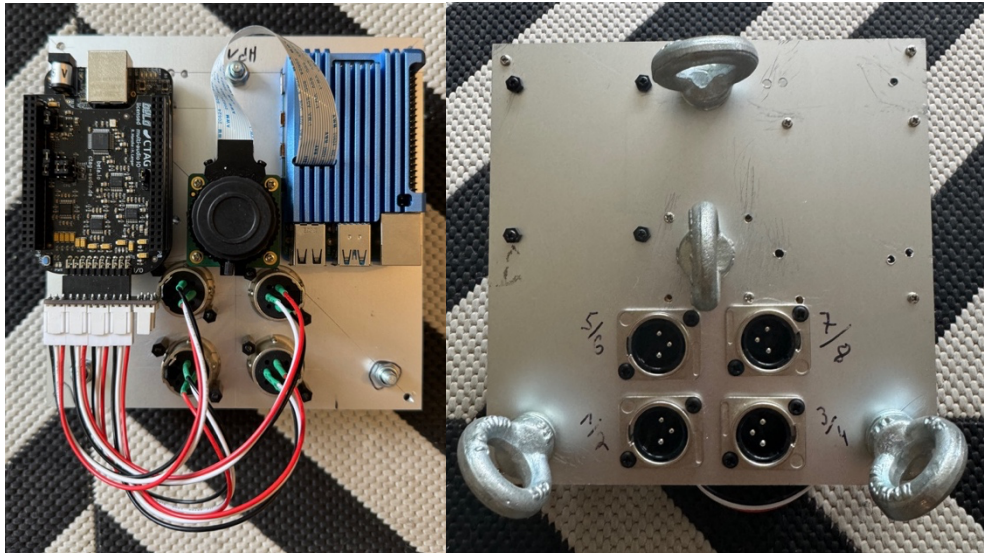


Figure 24: Camera unit with HQ v1 Camera, RPI4, Bela + CTAG module

After all speakers were connected to the amplifier, the output gain for each speaker had to be set. To keep this process approachable for untrained personnel, the plan was to position a chair as a guide for equal positioning and use a smartphone with a decibel meter app to set equal gains for all speakers by playing white noise sequentially from each speaker. However, it was discovered that the gain adjustment on the amplifier was not sufficient for precise gain adjustment. Due to time constraints during setup, the gain was roughly adjusted by ear.

The calibration process as covered in 4.1.5 and Appendix 3 was straightforward and worked as planned. The thresholding and blur factor could be set until useful tracking results were achieved. The adjustability of the tracking circle also worked as expected and even provided unexpected assistance in the speaker placement. At this stage, a final sound design had not yet been developed. The tracking ability and spatial

reproduction of sound elements in relation to the position of each person within the installation were tested using simple noise and sine wave signals. The surround panning feature functioned within the project's scope. It was clearly audible that sounds followed a person's movements, which was sufficient for this installation and aligned with the intent of the installation.

To identify the systems limitations, a new patch was loaded in which a composition draft was played as eight individual samples. It became evident that with more complex audio signals, the surround panning feature lost some of its impact as the higher sonic complexity could overwhelm the listener. Also, due to tracking errors, sound elements would cut in and out quickly if a person was misdetected, lost or erroneously detected more than once. This provided critical insights into the possibilities and requirements for the sound design.

#### 4.5.2 Improvement Implementation Based on the First Test Setup

##### **Creating Custom Solution for Output Gain Adjustment**

The first test setup provided useful ideas for improving the installation's features. To further enhance the setup process, markings were added to the displayed tracking circle for each angular speaker position, speeding up speaker placement and reducing reliance on on-site measurements.

To perform better output gain adjustments, a gain adjustment web interface was created that is hosted on the Bela, written in P5.js, and called `sketch.js`<sup>120</sup>. The web interface displayed eight gain sliders `vol1-8` for setting the output gain and eight sliders for adjusting the internal gains `snd1-8` for the sound elements, along with a save button to store all gain values as shown in Figure 25. The `snd1-8` sliders were only relevant during development and would not be visible to the end user. All gain values were combined into an array to be received within the PD patch. The PD version preinstalled on the Bela included pre-built objects for sending and receiving data to and from the web interface. Inside the main PD patch, a subpatch `[pd bellapdbella_gui_receive]`, is created to perform the data handling.<sup>121</sup> To ensure the gain levels are saved and reloaded after a system reboot, the subpatch `[pd bella_gui_receive]`, writes all values to a text file, located inside the main patch directory, when the save button on the web interface is pressed. This patch is shown in Figure 26.

---

<sup>120</sup> Source code 8: `sketch.js` web interface script for output gain adjustment

<sup>121</sup> cf.: Bela.io, Controlling Bela from a GUI, <https://learn.bela.io/tutorials/pure-data/communication/controlling-bela-from-a-gui/>,



Accessing the web interface works similar as accessing the calibration web interface by connecting a Wi-Fi-capable device to the RPI4 WLAN hotspot and entering the URL into a browser. This made the process of setting the output gains straightforward by using one smartphone for the dB measurement and another smartphone or tablet to adjust the output gains via the web interface.

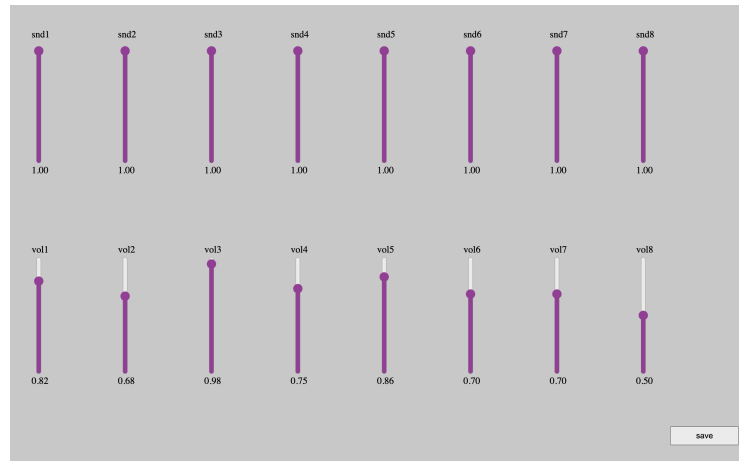


Figure 25: Web interface for gain adjustments

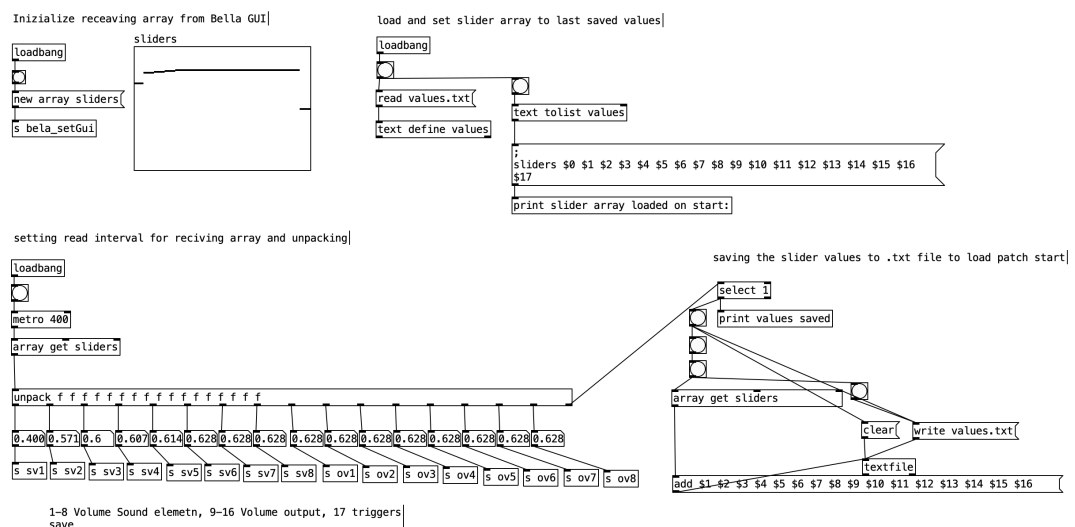


Figure 26: PD Subpatch [pd bella\_gui\_recive], receiving, handling and saving gain slider values from web interface

## Improving the Error Handling of the Tracking Algorithm

The tracking algorithm produced some unintended behaviors. These included the misdetection of a single person due to drastic intensity changes introduced by clothing colors that are too close to the set threshold value and by detecting more than one person who were close to each other as one individual. As discussed in chapter 4.1.5, these errors could be mitigated by adjusting the threshold, blur factor, and contour size. However, a complete elimination of these errors could not be achieved at this time.

These errors resulted in audible cutting in and out of the audio signals. Additionally, if one than one person were too close to each other, they were recognized as one object, leading to the disappearance of multiple sound elements, which was not desired. To counteract this behavior, a fifth control value called `id_active` was created and sent from the RPI4 to the PD patch. If a previously detected object was lost, the GID is updated to -1 within the `object_tracking.py`<sup>122</sup> script, and all GIDs are sent to the PD patch.

Since the tracking area was a closed environment, it could be assumed that a lost object was valid only if the participant has left the tracking circle. By using the radius as an additional argument, the status of the object could be verified. To handle this scenario, an abstraction called `[id_active]` was created to verify the state by checking if the GID is -1 and the last tracked radius value is greater than 0.9. If this condition returns true, it could be assumed the person had left the circle, and an audio ramp faded out the signal from 1 to 0 over 2 seconds. The audio ramp was also applied when reassigning a person to a sound element to gradual fade in the audio signal. This approach was also useful in the event of rapid loss and reassignment, making the entire system less reactive to abrupt identifications. The `[id_active]` object is shown in Figure 27.

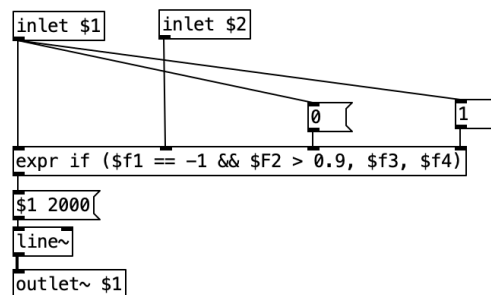


Figure 27: PD abstraction `[id_active]`, handling fade-in/out of a sound element

#### 4.5.3 Assessing the Improvements Through a Second Test Setup

In order to validate the effect of these improvements, a second test setup was conducted over 4 days at the Laboratory for Theater and Event Technology at BHT-Berlin.

At the second test setup, these improvements performed sufficiently. Setting up the speakers was improved by using a smartphone to view the real-time camera footage and using the speaker position marks as a reference. Setting the output gains worked

<sup>122</sup> Source code 2: Object tracking `object_tracking.py`

as intended. And the improvements made to the identification assignment and reassignment improved the behavior of the system. A pleasant discovery during the second test setup was that the tracking algorithm performed even less misdetections than anticipated despite nonideal floor color conditions.

It was revealed as well that that more subtle adjustments to the generated sounds such as volume, low pass filter, delay and reverb times in response to participants' movements did not lead to the hoped connection between participant and the sound element. Those adjustments had to be greatly exaggerated to have the intended effect. It stood out that changes in pitch and tonality could form a quicker connection between the participant and sound element. This formulated the idea that the final sound design could incorporate a drone-like synthesizer where each participant controls the pitch of a voice, tuning and modulating the sound in collaboration with other participants. Photos of the rigged camera unit and the speaker array are shown in Appendix 7.

## 5 Discussion

The goal of this work was to develop an interactive sound installation that is functional, cost-effective, utilizes open-source software and has minimal reliance on professional-grade audio and video equipment. The use of open-source resources to develop the installation was proven. Open-source software such as Python, OpenCV and PD were responsible for object detection, tracking, data transfer, sound creation and panning. Utilization of the open-source SBCs, RPI and Bela, also proved to perform reliably. Additionally, the self-built speakers proved to be a suitable choice for the intended application.

The evaluations in chapter 4.5 demonstrates that the installation's key functions met the initial artistic intent. The test setups validated that all key functions of the installation operated according to the artist's requirements. Specifically, the tracking algorithm implemented on the RPI4 performed reliably, with tracking errors remaining within an acceptable range, despite the suboptimal floor color. This suggests that the system could function effectively under less-than-ideal conditions. Participants' movements had a direct impact on the perceived location of sound elements within the installation and on the sonic characteristics of each sound. This demonstrates the capability for audience engagement through the concept of performance agency of the installation as outlined in chapter 2.6.3. The chosen panning approach highlighted in chapter 4.3.2 produced satisfactory results. This allowed for the clear localization

of different sound elements and enabled collaboration between participants through observation, thus demonstrating that the installation was capable of not only HCI but also HHI.

The secondary goal of this project to utilize limitations as a tool and source of inspiration was fulfilled. The limitations imposed by the processing power of the RPI and Bela required a simplified approach to both the tracking and sound design. The tracking algorithm, while functional, demonstrated anticipated sensitivity to environmental factors such as lighting, floor color, and participants' clothing, leading to the occasional misdetections of participants. These imperfections became integral features of the installation, introducing an element of unpredictability that enhanced the artistic experience. For example, the multi-assignment of one participant or the single assignment for multiple people introduced an organic and unpredictable element to the auditory experience by bringing more sound elements into the space than there were detected participants.

The CPU load constraints of the Bela forced an "out-of-the-box" panning approach, a reduction in sound design elements, and a strict focus on only essential components. These constraints ensured that every element of the PD patch contributed crucially to the installation, keeping the CPU load within an acceptable range. Additionally, a calibration solution was developed to optimize the tracking behavior and adjust the output gain. By using the RPI4 as a Wi-Fi hotspot, the user could connect to it and access a URL via a browser to achieve a suitable calibration.

## 6 Conclusion

This work has successfully demonstrated that it is possible to create a compelling artistic experience with limited resources by embracing the constraints of open-source technology. In addition, it was shown how technical constraints can be harnessed creatively to enhance artistic expression in an interactive sound installation. The project not only met its technical objectives but also opened up new possibilities for HCI that could inspire future developments. Importantly, the unintended HCI facilitated by the technical limitations represent a significant outcome of this study. The need for technical simplification created a unique connection between the participant, the sound, and the technology, highlighting an interesting aspect of unintended HCI as well as HHI.

## 7 Outlook

Moving forward, several enhancements could further improve the system's performance and artistic potential. One of the primary areas for future development is the finalization of the sound design to explore the full potential of the current state of the sound installation. Furthermore, refinement of the tracking algorithm, implementation of an automated calibration feature to adjust the tracking algorithm for varying lighting conditions, and an automated output gain adjustment would enhance usability.

Another promising direction is exploring the concept of unintended HCI more deeply. This could involve studying how participants perceive and react to system behaviors that deviate from their expectations, potentially leading to new insights in both HCI and interactive art.

Additionally, expanding the installation to include networked or distributed environments, where multiple installations communicate and interact with each other, could open up new possibilities for collective and shared experiences, further pushing the boundaries of interactive sound art.

While this project has reached its initial goals, the possibilities for expansion and refinement are present. The insights gained lay a solid foundation for future improvements that could lead to exciting developments at the intersection of art, technology, and human interaction.

## V. Bibliography

- Arduino-S.r.l. "Hardware." Accessed 14.08.2024.  
<https://www.arduino.cc/en/hardware>.
- Ariza, Jonathan Á., and Heyson Baez. 2022. "Understanding the role of single-board computers in engineering and computer science education: A systematic literature review." *Computer Applications in Engineering Education* 30 (1): 304-329. <https://doi.org/https://doi.org/10.1002/cae.22439>.
- BeagleBoard.org-Foundation. "Our Mission." Accessed 16.08.2024.  
<https://www.beagleboard.org/about>.
- Bela.io. "Abstractions." Accessed 16.08.2024. <https://learn.bela.io/using-bela/languages/pure-data/#abstractions>.
- Bela.io. "Bela & Bela Mini." Accessed 16.08.2024. <https://bela.io/products/bela-and-bela-mini/>.
- Bela.io. "Controlling Bela from a GUI." Accessed 16.08.2024.  
<https://learn.bela.io/tutorials/pure-data/communication/controlling-bela-from-a-gui/>.
- Bela.io. "CTAG multichannel audio board." Accessed 16.08.2024.  
<https://learn.bela.io/products/multichannel/ctag-multichannel-board/#ctag-multichannel-audio-board>.
- Bela.io. "Libpd." Accessed 16.08.2024. <https://learn.bela.io/using-bela/languages/pure-data/#libpd>.
- Bonardi, Alain. "abclib." Accessed 04.08.2024.  
<https://github.com/alainbonardi/abclib>.
- Candy, Linda. "Interaction in Art and Technology." Accessed 06.08.2024.  
<https://crossings.tcd.ie/issues/2.1/Candy/>.
- Cycling'74. "Introducing RNBO." Accessed 14.08.2024.  
<https://cycling74.com/products/rnbo>.
- Cycling'74. "Max." Accessed 14.08.2024. <https://cycling74.com/shop/max>.
- Cycling'74. "What is Max." Accessed 14.08.2024.  
<https://cycling74.com/products/max>.
- Dayton-Audio. "EXCITERS & TACTILE TRANSDUCERS 101." Accessed 10.07.2024. <https://www.daytonaudio.com/topic/excitersbuyerguide>.
- derivative. "Features." Accessed 09.08.2024. <https://derivative.ca/feature>.
- Freed, Adrian, and Andrew Schmeder. 2009. "Features and Future of Open Sound Control version 1.1 for NIME." NIME.
- Garrelfs, Iris. 2015. "From inputs to outputs: an investigation of process in sound art practice." University of the Arts London.

- Goudarzi, Visda, and Artemi-Maria Gioti. 2016. "Engagement and interaction in participatory sound art." *Proceedings of Sound and Music Computing*.
- Holmes, Thom. 2022. *Sound Art: Concepts and Practices*. Routledge.
- IEM. "[fudiformat]." Accessed 17.08.2024. <https://pd.iem.sh/objects/fudiformat/>.
- IEM. "[fudiparse]." Accessed 17.08.2024. <https://pd.iem.sh/objects/fudiparse/>.
- IEM. "[line~]." Accessed 17.08.2024. <https://pd.iem.sh/objects/line~/>.
- IEM. "[netreceive]." Accessed 17.08.2024. <https://pd.iem.sh/objects/netreceive/>.
- IEM. "[vline~]." Accessed 17.08.2024. <https://pd.iem.sh/objects/vline~/>.
- Institute-of-Electronic-Music-and-Acoustics. "Home." Accessed 12.08.2024. <https://puredata.info/>.
- Kadam, Pushkar, Gu Fang, and Ju Jia Zou. 2024. "Object Tracking Using Computer Vision: A Review." *Computers* 13 (6): 136.
- Kraus-[GbR], Brüll &. "About TouchDesigner." Accessed 09.08.2024. <https://thenodeinstitute.org/about-touchdesigner/>.
- McDonnell, M. J. 1981. "Box-filtering techniques." *Computer Graphics and Image Processing* 17 (1): 65-70. [https://doi.org/https://doi.org/10.1016/S0146-664X\(81\)80009-3](https://doi.org/https://doi.org/10.1016/S0146-664X(81)80009-3).
- MIPI-Alliance, -Inc. "DRAFT MIPI Alliance Specification for Camera Serial Interface 2 (CSI-2)." Accessed 24.07.2024. [https://caxapa.ru/thumbs/799244/MIPI Alliance Specification for Camera S.pdf](https://caxapa.ru/thumbs/799244/MIPI_Alliance_Specification_for_Camera_S.pdf).
- Möser, Michael. 2018. *Psychoakustische Messtechnik*. Springer.
- OpenCV. "Contours : Getting Started." Accessed 15.08.2024. [https://docs.opencv.org/4.x/d4/d73/tutorial\\_py\\_contours\\_begin.html](https://docs.opencv.org/4.x/d4/d73/tutorial_py_contours_begin.html).
- OpenCV. "Image Moments." Accessed 15.08.2024. [https://docs.opencv.org/3.4/d0/d49/tutorial\\_moments.html](https://docs.opencv.org/3.4/d0/d49/tutorial_moments.html).
- OpenCV. "Image Thresholding." Accessed 15.08.2024. [https://docs.opencv.org/4.x/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html).
- OpenCV. "Introduction." Accessed 15.08.2024. <https://docs.opencv.org/4.x/d1/dfb/intro.html>.
- OpenCV. "Smoothing Images." Accessed 15.08.2024. [https://docs.opencv.org/4.x/d4/d13/tutorial\\_py\\_filtering.html](https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html).
- Pisano, Giuseppe, and Per Magnus Lindborg. 2023. *Introducing the Open Ambisonics Toolkit*.
- Plenk, Valentin. 2024. *Angewandte Netzwerktechnik kompakt: Dateiformate, Übertragungsprotokolle und ihre Nutzung in Java-Applikationen*. Springer-Verlag.

- Pulkki, Ville. 2001. *Spatial sound generation and perception by amplitude panning techniques*. Helsinki University of Technology.
- Pulkki, Ville, and Matti Karjalainen. 2001. "Localization of amplitude-panned virtual sources I: stereophonic panning." *Journal of the Audio Engineering Society* 49 (9): 739-752.
- Raspberry-Pi. "About the Camera Modules." Accessed 28.07.2024. <https://www.raspberrypi.com/documentation/accessories/camera.html#hq-camera>.
- Raspberry-Pi. "Raspberry Pi hardware." Accessed 20.07.2024. <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>.
- Sarkar, Dipanjan, Raghav Bali, and Tushar Sharma. 2018. "Deep Learning for Computer Vision." In *Practical Machine Learning with Python: A Problem-Solver's Guide to Building Real-World Intelligent Systems*, edited by Dipanjan Sarkar, Raghav Bali and Tushar Sharma, 499-520. Berkeley, CA: Apress.
- Schmeder, Andrew, Adrian Freed, and David Wessel. 2010. "Best practices for open sound control." Linux Audio Conference.
- Schmidt, Ulrich. 2013. *Professionelle Videotechnik: Grundlagen, Filmtechnik, Fernsehtechnik, Geräte-und Studiotechnik in SD, HD, DI, 3D*. Springer-Verlag.
- Sergio-Canu. "Object tracking from scratch – OpenCV and python." Accessed 20.07.2024. <https://pysource.com/2021/10/05/object-tracking-from-scratch-opencv-and-python/>.
- Sertronics-GmbH. "senasoren-module." Accessed 13.08.2024. <https://www.berrybase.de/sensoren-module/>.
- Shoer, Ibrahim, Berkay Kopru, and Engin Erzin. 2022. "Role of Audio in Audio-Visual Video Summarization." *arXiv preprint arXiv:2212.01040*.
- solipd. "AudioLab." Accessed 22.08.2024. <https://github.com/solipd/AudioLab>.
- Süße, Herbert, and Erik Rodner. 2014. *Bildverarbeitung und Objekterkennung*. Springer.
- The Institute of Electrical and Electronics Engineers, Inc. . 2008. *IEEE Standard for Floating-Point Arithmetic*.
- Tinapple, David. "Max/MSP - Blob-tracking." Accessed 08.08.2024. <https://www.youtube.com/watch?v=cytx9NqSQNA>.
- Tinapple, David. "Tracking - Blob Tracking." Accessed 08.08.2024. <https://tinapple.notion.site/Tracking-Blob-Tracking-9b5b314087074429808b53bf4598e4de>.
- Weinzierl, Stefan. 2008. *Handbuch der Audiotechnik*. Springer Science & Business Media.



Wendzel, Steffen. 2018. *IT-Sicherheit Für TCP/IP-und IoT-Netzwerke*. Springer.

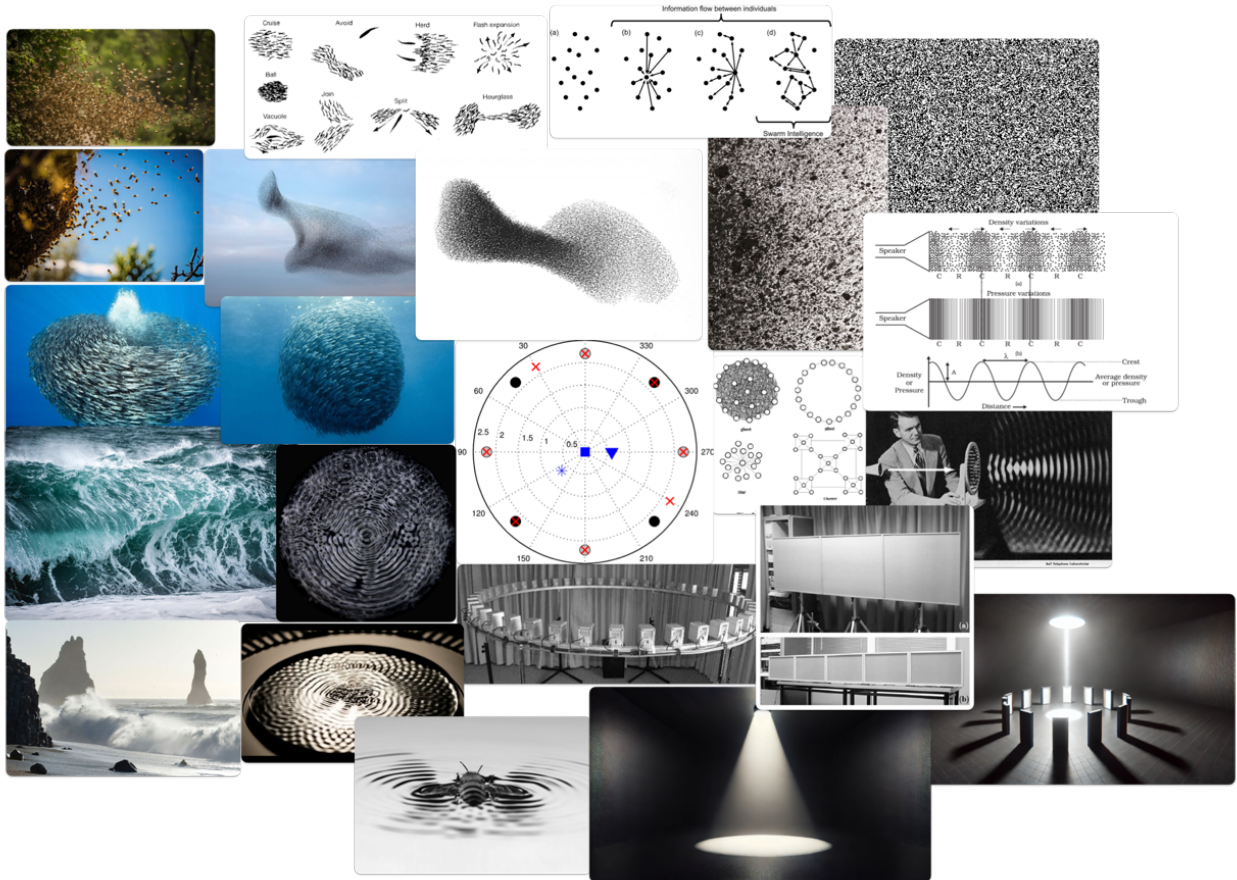
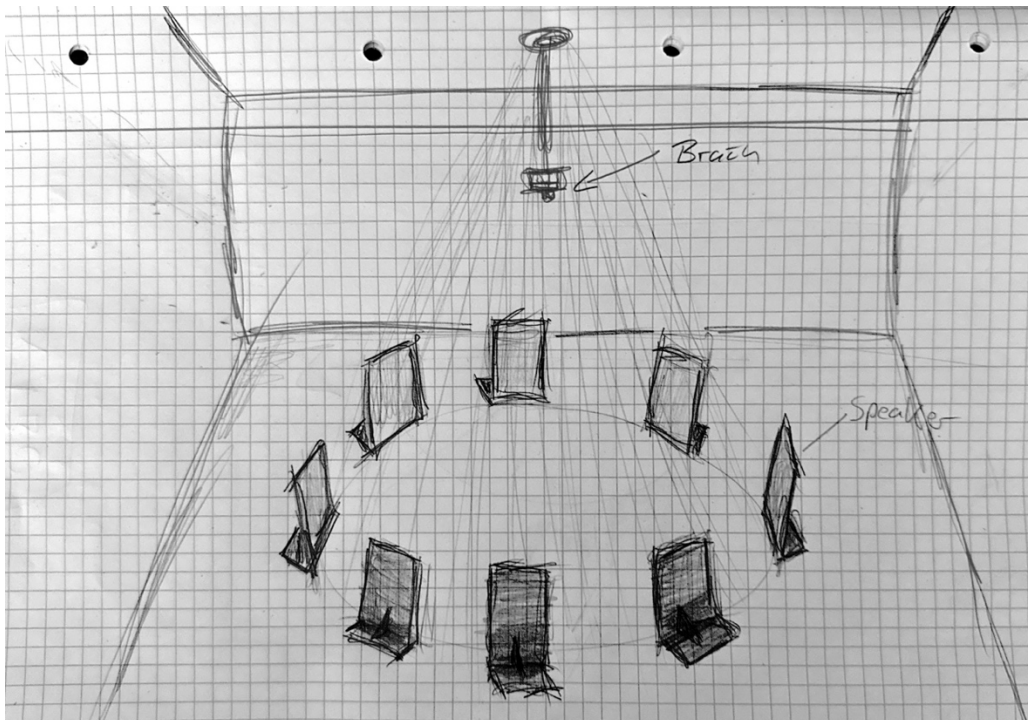
Werner, Martin. 2021. *Digitale Bildverarbeitung*. Springer.

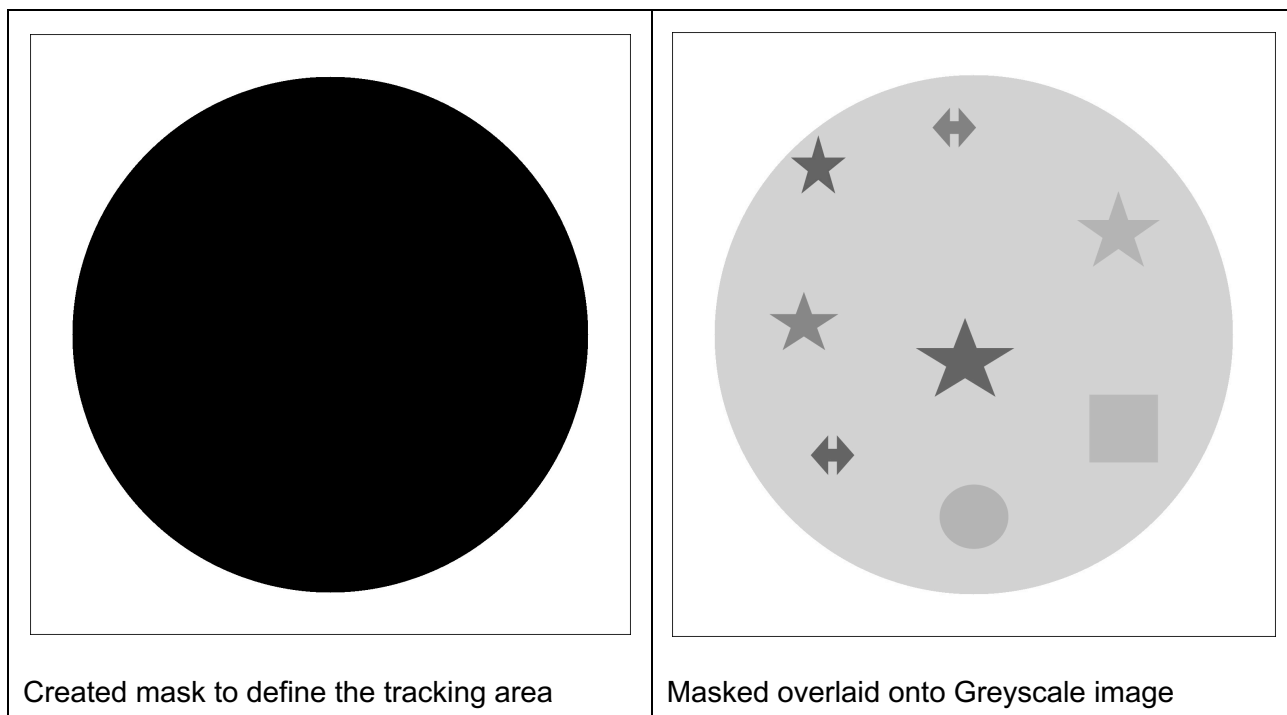
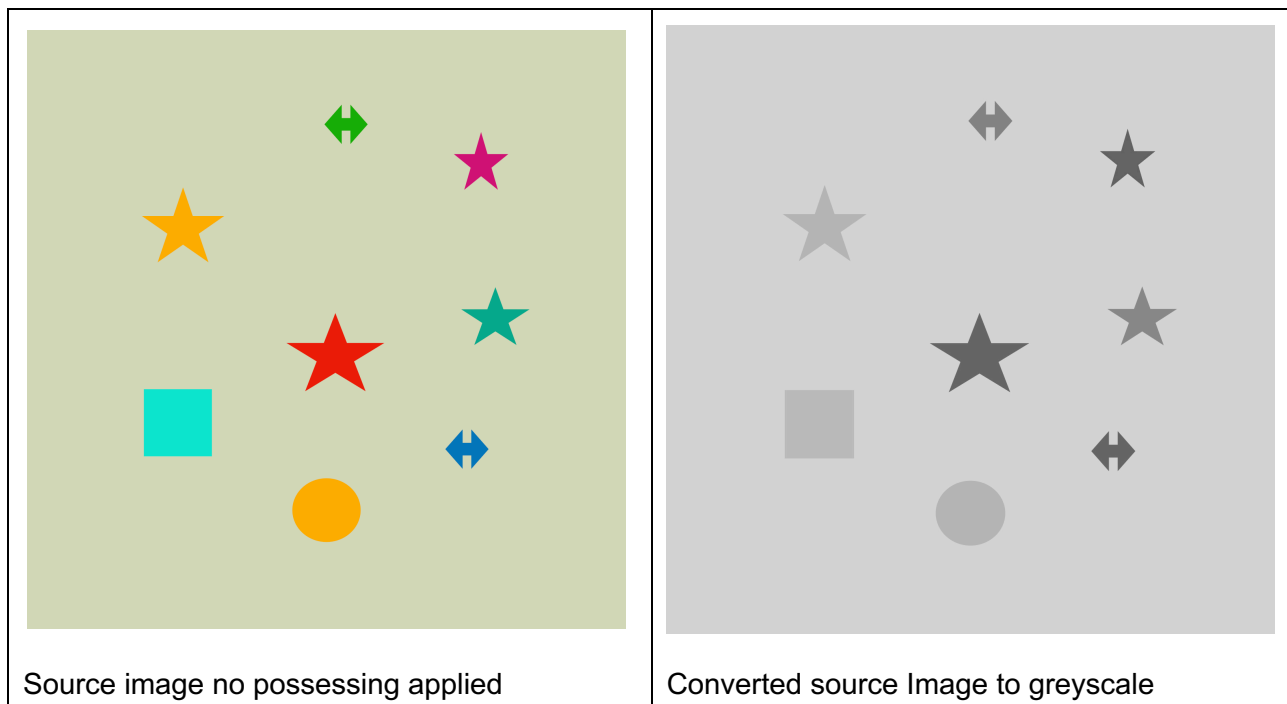
Wright, Matthew, and Adrian Freed. 1997. "Open SoundControl: A new protocol for communicating with sound synthesizers." ICMC.

Xu, Xiangyang, Shengzhou Xu, Lianghai Jin, and Enmin Song. 2011. "Characteristic analysis of Otsu threshold and its applications." *Pattern Recognition Letters* 32 (7): 956-961. <https://doi.org/https://doi.org/10.1016/j.patrec.2011.01.021>.

## VI. Appendix

Appendix 1 swarm of sound installation draft and mood board .....	XIII
Appendix 2 Visualization of the image processing steps using a test image .....	XIV
Appendix 3: Calibration web interface basic calibration example.....	XVI
Appendix 4: Data sheet Dayton Audio exciter .....	XVIII
Appendix 5: Safety certificate Styrodur 3035 .....	XIX
Appendix 6: Flow chart visualizing the entire processing chain .....	XX
Appendix 7: Second test setup, rigging camera unit, and speaker array .....	XXI



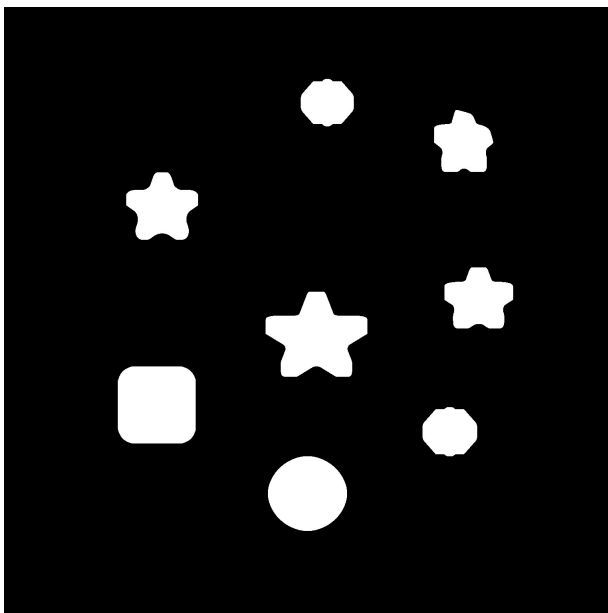




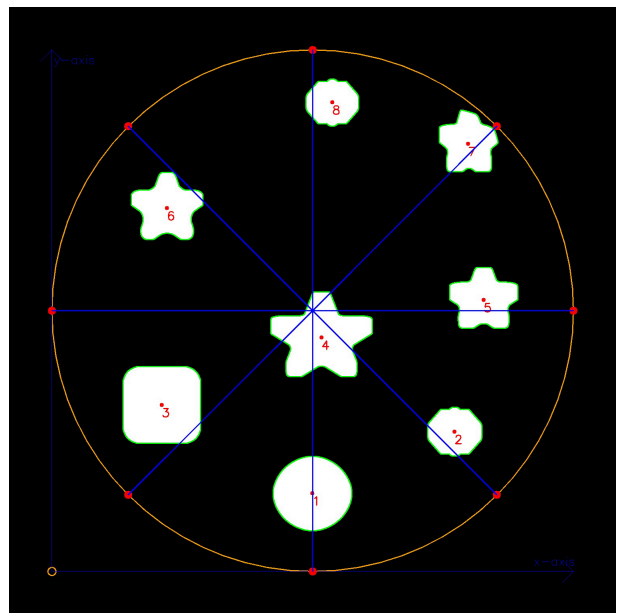
Inverting the Greyscale



Applied average filter

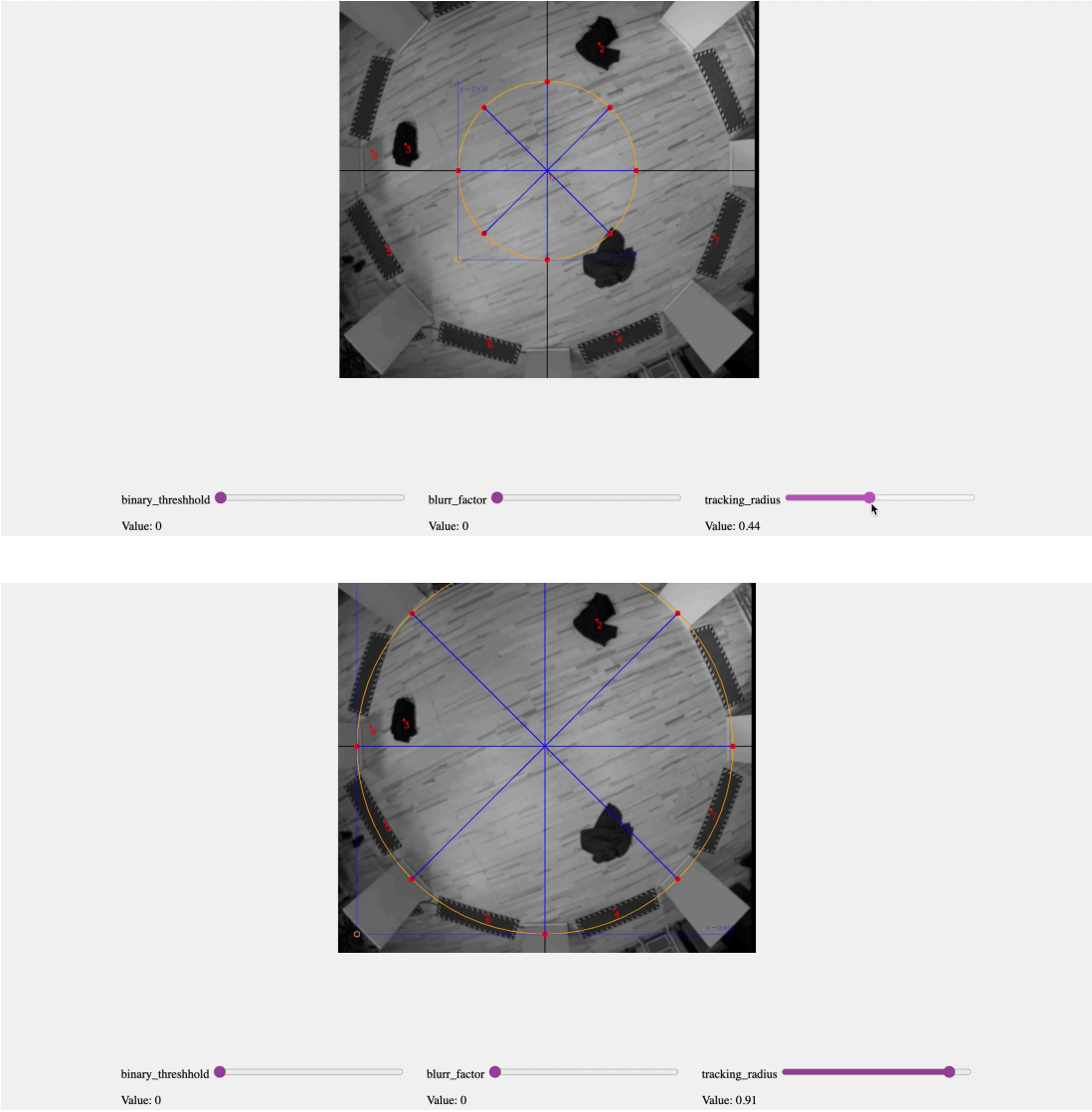


Thresholding for background separation



Contour detection and center point calculation

### Appendix 3: Calibration web interface basic calibration example

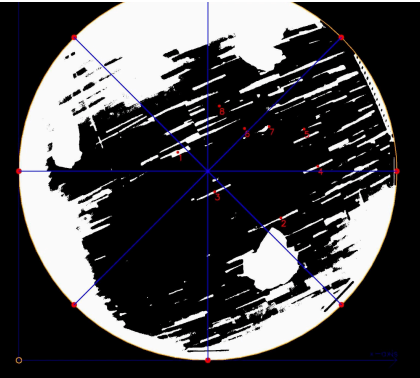


The image displays two screenshots of a calibration web interface, illustrating the process of adjusting the tracking area by setting the radius value for a circle.

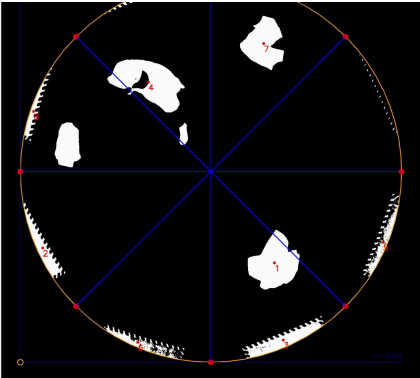
**Top Screenshot:** The interface shows a top-down view of a circular arena with a wooden floor and several black obstacles. A yellow circle is overlaid on the arena, representing the tracking area. Below the arena, there are three sliders: `binary_threshold` (Value: 0), `blurr_factor` (Value: 0), and `tracking_radius` (Value: 0.44). The `tracking_radius` slider is currently set to 0.44.

**Bottom Screenshot:** The same interface is shown, but the `tracking_radius` slider has been adjusted to 0.91. The yellow circle representing the tracking area is now significantly larger, covering more of the arena.

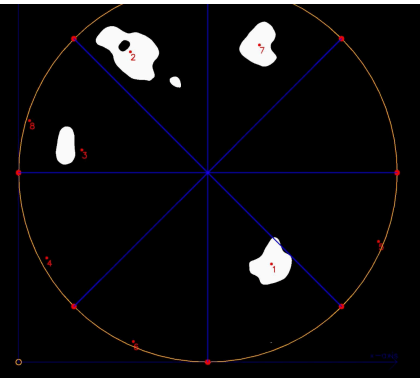
Adjusting tracking area by setting the radius value for the circle



binary\_threshold Value: 0.54    blurr\_factor Value: 0    tracking\_radius Value: 0.91

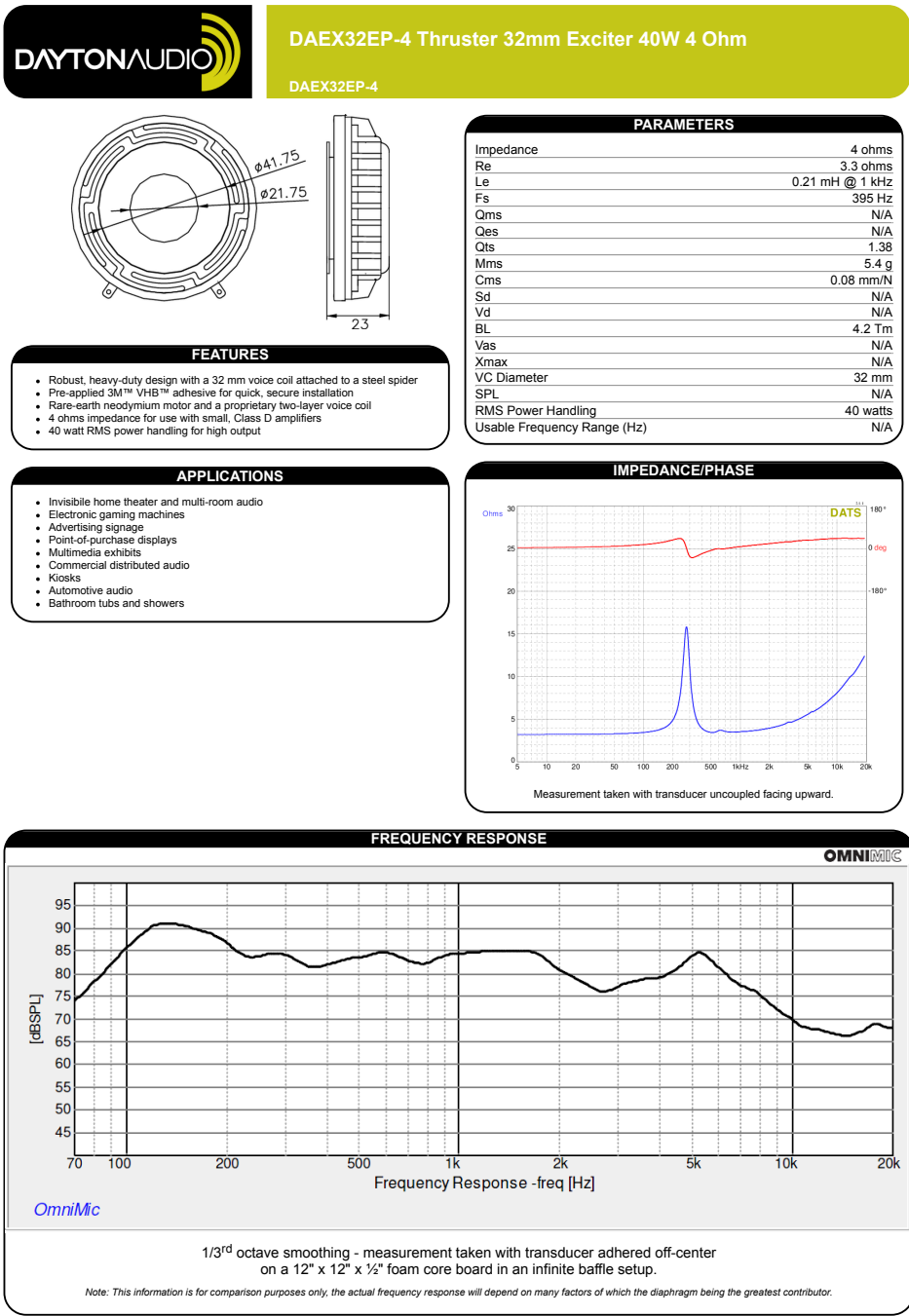


binary\_threshold Value: 0.82    blurr\_factor Value: 0    tracking\_radius Value: 0.91



binary\_threshold Value: 0.82    blurr\_factor Value: 0.032    tracking\_radius Value: 0.91

Adjusting the threshold and average filtering value to highlight the objects of interest







Falls Empfänger verzogen, bitte nachsenden. Anschriftenberichtigungskarte mit neuer Anschrift. Falls unzustellbar, zurück an Isover Dialog – Postfach 1240 – 68521 Ladenburg

An unsere Kunden

SAINT-GOBAIN ISOVER G+H AG  
Bürgermeister-Grünzweig-Straße 1  
67059 Ludwigshafen  
Tel. (06 21) 501-0

Ludwigshafen, 12. März 2009

---

**Begleitschreiben zum Sicherheitsdatenblatt von Styrodur® C**

Sehr geehrte Damen und Herren,

stellvertretend für alle von Isover vertriebenen Styrodur-Produkte erhalten Sie anliegend das Sicherheitsdatenblatt für Styrodur 3035 CS in 100 mm.

---

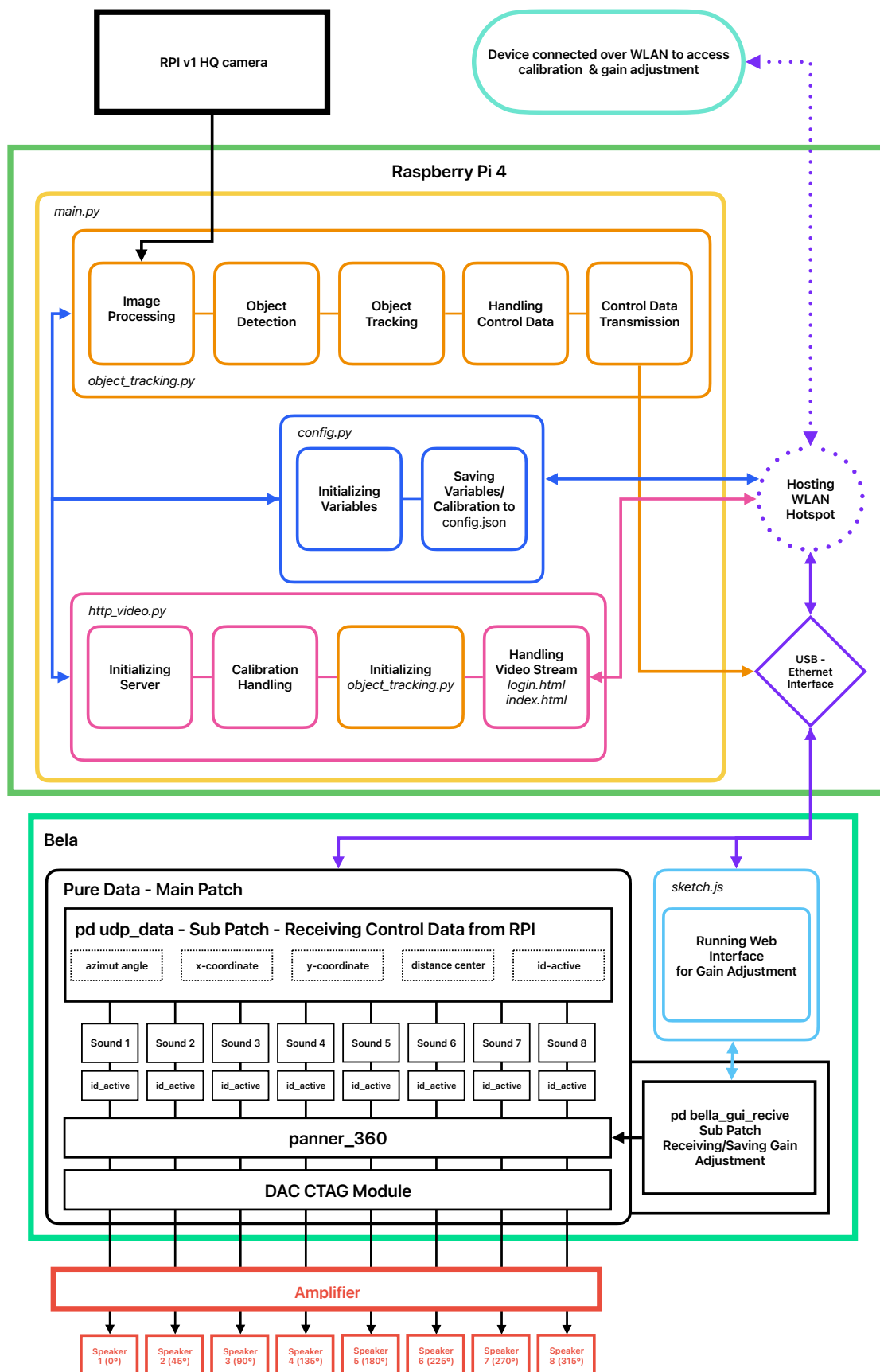
Die in diesem Sicherheitsdatenblatt stehenden Angaben zu Styrodur 3035 CS gelten uneingeschränkt auch für die anderen Styrodur-Sorten 2500 C, 2800 C, 3035 CN, 4000 CS und 5000 CS.

Mit freundlichen Grüßen  
SAINT-GOBAIN ISOVER G+H AG

---

Vorsitzender des Aufsichtsrats: Paul Neeteson, Vorstand: Michael Wörtler (Vorsitzender), Jürgen Hohmeier  
Sitz der Gesellschaft: Bürgermeister-Grünzweig-Straße 1 D-67059 Ludwigshafen, Amtsgericht Ludwigshafen am Rhein, HRB Nr. 3570  
HypoVereinsbank Ludwigshafen, BLZ 545 201 94, Konto 1171, Deutsche Bank Ludwigshafen, BLZ 545 700 94, Konto 0125 062

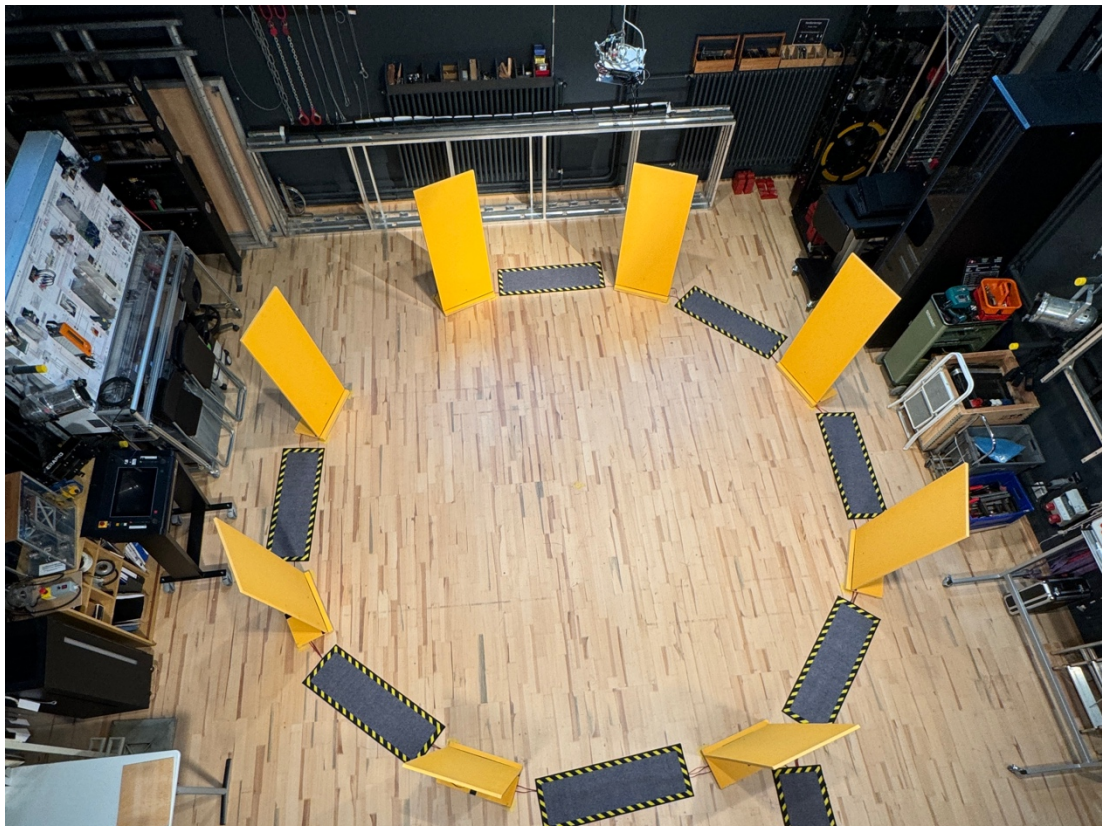
Appendix 6: Flow chart visualizing the entire processing chain



*Appendix 7: Second test setup, rigging camera unit, and speaker array*







## VII. Source code

Source code 1: Object tracking main.py handling mode select, calibration, or installation mode .....	XXIV
Source code 2: Object tracking object_tracking.py handling image processing, detection, tracking, control data handling, and transmission .....	XXV
Source code 3: Object tracking http_video.py handling web server and video stream .....	XXXIII
Source code 4: Object tracking config.py background data handling and storing between the Python scripts .....	XXXIV
Source code 5: Object tracking config.json saved data to initialize on program start .....	XXXV
Source code 6: Object tracking index.html calibration web interface and receiving video stream .....	XXXVI
Source code 7: Object tracking login.html login page .....	XXXVIII
Source code 8: sketch.js web interface script for output gain adjustment hosted on the Bela .....	XXXIX

## Source code 1: Object tracking main.py handling mode select, calibration, or installation mode

28/08/2024, 10:51

main.py

main.py

```
1 import object_tracking
2 import http_video
3 import config
4
5 frame1 = None
6
7 def main():
8     while True: # set mode
9         try:
10             mode_input = input('>>> Enter: set mode: 0 = installation file, 1 = stream on, [1]: ')
11             if mode_input != '':
12                 mode = int(mode_input)
13
14             if mode_input == '':
15                 mode = 0
16
17             break
18         except ValueError:
19             print('invalid input!')
20
21     if mode == 0:
22         config.mode = mode
23         config.save_config()
24         for frame1 in object_tracking.object_tracking1(config.config):
25             test = 0
26     elif mode == 1:
27         config.mode = mode
28         config.save_config()
29
30     try:
31         http_video.start_video_feed()
32     except Exception as e:
33         print(f"An error occurred while starting the video feed: {e}")
34     else:
35         print("Invalid mode selected. Please enter 0 or 1.")
36
37 if __name__ == "__main__":
38     main()
39
40 # swarm of sound Version 1.0
41 # Copyright (c) [2024] [Sebastian Obel]
42
43 # Permission is hereby granted, free of charge,
44 # to any person obtaining a copy of this software and associated documentation files (the "Software"),
45 # to deal in the Software without restriction, including without limitation the rights to use,
46 # copy, modify, merge, publish, distribute, sublicense,
47 # and/or sell copies of the Software
48
49 # This Software was inspired by:
50 # Sergio-Canu
51 # https://pysource.com/2021/10/05/object-tracking-from-scratch-opencv-and-python/
52 # and Sharon McCaman and David Tinapple
53 # Sharon McCaman and David Tinapple
54 # http://www.sharonmccaman.com
55 # http://www.sharonmccaman.com
56 # https://www.youtube.com/watch?v=cytx9NqSQNA
```

localhost:65427/c8fda01d-1149-4071-bf6a-f211748c6dd8/

1/1

Source code 2: Object tracking object\_tracking.py handling image processing, detection, tracking, control data handling, and transmission

28/08/2024, 10:50

object\_tracking.py

object\_tracking.py

```
1 import cv2
2 import numpy as np
3 import math
4 import operator
5 import socket
6 import sys
7 import select
8 import time
9 from flask import Flask, Response, render_template, request, jsonify, redirect, url_for, session
10 from functools import wraps
11 import config as cf
12
13 config = cf.config
14
15 def check_for_input():
16     # Check if there is input ready to be read
17     if select.select([sys.stdin], [], [], 0)[0]:
18         return sys.stdin.read(1)
19     return None
20
21 def apply_variable_blur(image_path, blur_factor):
22
23     # Read the image
24     image = cv2.imread(image_path)
25     height, width = image.shape[:3]
26
27     # Determine the maximum kernel size (must be an odd number)
28     max_kernel_size = min(height, width)
29     if max_kernel_size % 2 == 0:
30         max_kernel_size -= 1
31
32     # Calculate the kernel size based on the blur factor
33     kernel_size = int(blur_factor * max_kernel_size)
34     if kernel_size % 2 == 0:
35         kernel_size += 1
36
37     # Apply blur only if kernel_size is greater than 1
38     if kernel_size > 1:
39         blurred_image = cv2.blur(image, (kernel_size, kernel_size))
40     else:
41         blurred_image = image.copy()
42
43     return blurred_image
44
45
46 def object_tracking1(config): # Tracking Algorithmen Funktion
47     while True:
48         mode = cf.mode
49         print("mode", mode)
50
51         # Assigning values from the config to new variables
52         azimuth_1 = config['azimut_1']
53         #area = config['area']
54         pxc = config['pxc']
55         id = config['id']
56         #id_check_list = config['id_check_list']
57         #id_check_list_copy = config['id_check_list_copy']
58         azimut_dict = config['azimut_dict']
59         rnbo = config['rnbo']
60         resolution = config['resolution']
61         binary_thresh1 = config['binary_thresh1']
62         binary_thresh2 = config['binary_thresh2']
63         contour_size = config['contour_size']
64         tracking_distance = config['tracking_distance']
65         pick_img = config['pick_img']
66         show_img_local = config['show_img_local']
67         tcp_stream = config['tcp_stream']
68         video_source = config['video_source']
69         radius_circle = config['radius_circle']
70         ip_adress = config['ip_adress']
71         Picamera2= config['Picamera2']
72         pi_mode = config['pi_mode']
73
74
75         #Enter Settings or kalibration for user input
76
77         while True: # user menu enter settings or calibration menu
78             try:
```

localhost:65427/99c0381a-71b7-485f-8477-8d3f49a4f716/

1/8

```

79         enter_settings = input('>>> Enter: (s) for Settings, (c) for calibration, any other string continues with set
Values: ')
80         break
81     except ValueError:
82         print('Invalid input!')
83
84     if enter_settings == 's':
85
86         while True: # set resolution
87             try:
88                 res_input = input('>>> Enter: set resolution (e.g., 720, 480): ')
89                 if res_input != '':
90                     resolution = [int(x.strip()) for x in res_input.split(',')]
91                 break
92             except ValueError:
93                 print('invalid input!')
94
95         while True: # set radius for tracking circle
96             try:
97                 radius = input('>>> Enter: set tracking_circle_radius, [350] (0-350): ')
98                 if radius != '':
99                     radius_circle = int(radius)
100                 break
101             except ValueError:
102                 print('invalid input!')
103
104         while True: # set video source
105             try:
106                 source = input('>>> Enter: set video_source: 0 = video file, 1 = camera, [1]: ')
107                 if source != '':
108                     video_source = int(source)
109                 break
110             except ValueError:
111                 print('invalid input!')
112
113         while True: # set pi mode
114             try:
115                 mode = input('>>> Enter: pi mode: 0 = off, 1 = on, [1]: ')
116                 if mode != '':
117                     pi_mode = int(mode)
118                 break
119             except ValueError:
120                 print('invalid input!')
121
122         while True: # enter calibration menu or strat
123             try:
124                 enter_settings = input('>>> Enter: (c) for calibration or (e) to start program: ')
125                 break
126             except ValueError:
127                 print('invalid input!')
128
129     if enter_settings == 'c': # Enter calibration
130
131         while True: # set video or binary
132             try:
133                 pick = input('>>> Enter: set Visual to frame = 0, binary = 1: ')
134                 if pick != '':
135                     pick_img = int(pick)
136                 break
137             except ValueError:
138                 print('invalid input!')
139
140         while True: # set binary threshold
141             try:
142                 thresh = input('>>> Enter: set binary threshold, [80,210] (80-255): ')
143                 if thresh != '':
144                     binary_thresh1 = int(thresh)
145                 break
146             except ValueError:
147                 print('invalid input!')
148
149         while True: # set contour size
150             try:
151                 contour = input('>>> Enter a value to set contour size, [200] (10-400): ')
152                 if contour != '':
153                     contour_size = int(contour)
154                 break
155             except ValueError:
156                 print('invalid input!')
157
158         while True: # set tracking distance

```



```

159     try:
160         distance = input('>>> Enter a value to set tracking distance, [50] (0-300): ')
161         if distance != "":
162             tracking_distance = int(distance)
163         break
164     except ValueError:
165         print('invalid input!')
166
167
168
169     # Update config dictionary
170     # config['azimut_1'] = azimut_1
171     # config['area'] = area
172     # config['pxc'] = pxc
173     # config['id'] = id
174     # config['id_check_list'] = id_check_list
175     # config['id_check_list_copy'] = id_check_list_copy
176     # config['azimut_dict'] = azimut_dict
177     # config['rnbo'] = rnbo
178     config['resolution'] = resolution
179     config['binary_thresh1'] = binary_thresh1
180     config['binary_thresh2'] = binary_thresh2
181     config['contour_size'] = contour_size
182     config['tracking_distance'] = tracking_distance
183     config['pick_img'] = pick_img
184     config['show_img_local'] = show_img_local
185     config['tcp_stream'] = tcp_stream
186     config['video_source'] = video_source
187     config['radius_circle'] = radius_circle
188     config['ip_adress'] = ip_adress
189     config['Picamera2'] = Picamera2
190     config['pi_mode'] = pi_mode
191
192     cf.config = config
193
194     # Try to Import Picam library onli if pi_mode active
195     try:
196         if pi_mode == 1:
197             from picamera2 import Picamera2
198     except ImportError:
199         print('invalid input!')
200
201
202
203     #waiting after Settings to ensure Server Has rebooted and lost Prot
204     print ('.....loading.....:')
205     time.sleep(2)
206
207     #set variables for UDP-Socket for Controldata to Bella
208     UDP_IP1 = ip_adress    #Bella IP
209     UDP_PORT1 = 3001
210     UDP_IP = "127.0.0.1"  #localhost IP
211     UDP_PORT = 3001
212
213
214
215     # Getting Centerpoint of the Image
216     width_float = (resolution[0])
217     height_float = (resolution[1])
218     center_x = int(width_float/2)
219     center_y = int(height_float/2)
220     center_xy = (center_x, center_y)
221
222
223
224     # Define Funktions:
225     # get Key and Value from recursif Funktion
226     def recursive_items(dictionary):
227         for key, value in dictionary.items():
228             if type(value) is dict:
229                 yield from recursive_items(value)
230             else:
231                 yield (key, value)
232
233     if video_source == 1:
234         #Read Video PI
235         picam2 = Picamera2()
236         picam2.configure(picam2.create_preview_configuration(main={"format": "YUV420", "size": (resolution[0],
237         resolution[1])}, controls={"FrameRate": 30}))
237         picam2.start()
238         #print(1)

```

```

239
240     if video_source == 0:
241         #Read Video opencv
242         cap = cv2.VideoCapture('video.mp4')
243
244     # Initialize count and Tracking Variabels:
245     count = 0
246     img_counter = 0
247     center_points_prev_frame = []
248
249     tracking_objects = {}
250     track_id = 1
251
252     # start messuring to quantify possesing time
253     start_time = time.time()
254
255     # Defining insert Funtion for defiend empty spots
256     print(">>> Start Programm")
257     print('>>> Enter: (q) to pause and go back to menu')
258     while True: # capture loop
259
260         count += 1
261         center_points_cur_frame = []
262
263         # setting Vvalues from browser
264         binary_thresh1 = 255 * cf.var_1
265         #binary_thresh2 = 255 * var_2
266         #contour_size = 500 * var_3
267         #tracking_distance = 200 * var_4
268         radius_circle = int(resolution[1] * 0.5 * cf.var_3)
269         #calculate and Print Koordinaten Uhrsprung
270         pick_img = cf.var_6
271         x1 = center_x - radius_circle
272         y1 = center_y + radius_circle
273         encode_param = [int(cv2.IMWRITE_JPEG_QUALITY), int(cf.var_4)]
274
275         # chose sourcematerial for video
276         if video_source == 1:
277             frame = picam2.capture_array()
278             start_frame_time = time.time() # take start time of framecycle
279             height, width = frame.shape[0] * 2 // 3, frame.shape[1]
280             frame = frame[:height, :width]
281             #print(1)
282
283         if video_source == 0:
284             ret, frame = cap.read()
285             start_frame_time = time.time() # strat timer for framecycle
286             frame = cv2.resize(frame, (resolution[0], resolution[1]))
287             frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
288             height, width = frame.shape[0] * 2 // 3, frame.shape[1]
289
290         # Determine the maximum kernel size (must be an odd number)
291         max_kernel_size = min(height, width)
292         if max_kernel_size % 2 == 0:
293             max_kernel_size -= 1
294
295         # calculate the kernel size based on the blur factor
296         kernel_size = int(cf.var_2 * max_kernel_size)
297         if kernel_size % 2 == 0:
298             kernel_size += 1
299
300         # create a mask to define tracking area and overlay onto frame
301         mask_read = np.zeros_like(frame)
302         mask_read = 255 - mask_read
303         mask_read = cv2.circle(mask_read, center_xy, radius_circle, (0,0,0), -1)
304         frame_mask = cv2.bitwise_or(frame, mask_read)
305         #contrast adjustment maybe brings out shadows to much
306         #clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))
307         #frame_mask = clahe.apply(frame_mask)
308
309         # prosses frame invert blurr and convert to binary
310         img_inv = cv2.bitwise_not(frame_mask)
311         # apply blur (liniar filter - type box)
312         if kernel_size > 1:
313             img_blurred = cv2.blur(img_inv, (kernel_size, kernel_size))
314         else:
315             img_blurred = img_inv.copy()
316         # perform thresholding
317         ret, img_binary = cv2.threshold(img_blurred, binary_thresh1, binary_thresh2, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
318

```

```

319 #define later printed Image binary or Frame
320 print_img = img_binary
321 if pick_img == 1:
322     #print_img = img_binary
323     print_img = cv2.cvtColor(img_binary, cv2.COLOR_GRAY2RGB)
324 if pick_img == 0:
325     #print_img = frame
326     print_img = cv2.cvtColor(frame, cv2.COLOR_GRAY2RGB)
327
328
329 # find contours and find total area
330 cnts = cv2.findContours(img_binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
331 cnts = cnts[0] if len(cnts) == 2 else cnts[1]
332 for c in cnts: # find center of contour
333     pxc = np.count_nonzero(c)
334     if pxc > contour_size:
335         # compute the center of the contour
336         M = cv2.moments(c)
337         if (M["m00"] != 0):
338             cX = int(M["m10"] / M["m00"])
339             cY = int(M["m01"] / M["m00"])
340             center_points_cur_frame.append((cX, cY))
341         else:
342             cX, cY = 0, 0
343     #cv2.drawContours(print_img, [c], -1, (0, 255, 0), 2)
344
345 # Only at the beginning we compare previous and current frame
346 if count <= 2: # compare distance cur to prev frame
347     for pt in center_points_cur_frame:
348         for pt2 in center_points_prev_frame:
349             distance = math.hypot(pt[0] - pt2[0], pt[1] - pt2[1])
350
351             if distance < tracking_distance:
352                 tracking_objects[track_id] = pt
353                 track_id += 1
354 else:
355
356     tracking_objects_copy = tracking_objects.copy()
357     center_points_cur_frame_copy = center_points_cur_frame.copy()
358
359     for object_id, pt2 in tracking_objects_copy.items(): # update ID, remove old & add new
360         object_exists = False
361         for pt in center_points_cur_frame_copy:
362             distance = math.hypot(pt2[0] - pt[0], pt2[1] - pt[1])
363
364             # Update IDs position
365             if distance < tracking_distance:
366                 tracking_objects[object_id] = pt
367                 object_exists = True
368                 if pt in center_points_cur_frame:
369                     center_points_cur_frame.remove(pt)
370                 continue
371
372     # Remove IDs lost
373     if not object_exists:
374         tracking_objects.pop(object_id)
375
376     # Add new IDs found
377     for pt in center_points_cur_frame:
378         tracking_objects[track_id] = pt
379         track_id += 1
380
381 #Adding (nesting) tracking_objekts to rnbo slots:
382 id_check_list_copy = []
383 id_check_list = []
384
385 for id, z in tracking_objects.items(): # Finite ID 1-8
386     dct = {id: z}
387     dct2 = {0: 0}
388     #print (dct)
389
390     for k, v in rnbo.items():
391         vid = [*v]
392         vid = vid[0]
393
394     # Creating id_check_list to check later if object_id is in rnbo dictionary
395     for key, value in recursive_items(rnbo):
396         if key not in id_check_list:
397             id_check_list.append(key)
398

```

```

399
400     # check if objekt_id is not in rnbo
401     if id not in id_check_list:
402         if vid == -1:
403             #if v == {0: (0, 0)}: #####
404                 rnbo[k] = dct.copy()
405                 #print ('false')
406
407     # check if objekt_id is in rnbo
408     elif id in id_check_list and vid == id:
409         rnbo[k] = dct.copy()
410         #print ('true')
411
412
413     # Delete rnbo Value Items if old Objekt_id is not present anymore:
414     # Creating id_check_list_copy to check later if objekt_id is not in rnbo dictionary
415     for key, value in recursive_items(tracking_objects):
416         if key not in id_check_list_copy:
417             id_check_list_copy.append(key)
418
419
420     azimut_dict = rnbo.copy()
421     x_dict = rnbo.copy()
422     y_dict = rnbo.copy()
423     radius_dict = rnbo.copy()
424     id_active_dict = rnbo.copy()
425
426     #Check for old Items, Clculate Azimut Angel and Print ID on frame:
427     for k, v in rnbo.items():
428         vid = [*v]
429         vid = vid[0]
430         pt = (rnbo.get(k, {}).get(vid))
431         dct = {-1: pt}#####
432
433     #Delet old and insert Spaceholder Value
434     if vid not in id_check_list_copy:
435         rnbo[k] = dct.copy()
436         azimut_dict[k] = pt
437         x_dict[k] = pt
438         y_dict[k] = pt
439         radius_dict[k] = pt
440     #print ('PT:'+str(pt))
441
442     if pt != (0, 0) and pt != None:
443
444         #calculate relative xy koordinats to the center of Frame
445         angel_xy = tuple(map(operator.sub, pt, center_xy))
446         #calculate distance angle_xy to center_xy
447         angel_radius = np.sqrt((angel_xy[0]**2)+(angel_xy[1]**2))
448         # calculate the azimut angel of each position
449         if angel_xy[0] < 0:
450             if angel_xy[1] < 0:
451
452                 angel_devision= (angel_xy[1]/ angel_xy[0])
453                 #print(angel_devision)
454                 azimut = (math.degrees(math.atan(angel_devision)))
455                 azimut_1 = (90-azimut)
456
457             elif angel_xy[1] > 0:
458                 angel_devision= (angel_xy[1]/ angel_xy[0])
459                 #print(angel_devision)
460                 azimut = (math.degrees(math.atan(angel_devision)))
461                 azimut_1 = (-1*(azimut-90))
462
463         if angel_xy[0] > 0:
464             if angel_xy[1] > 0:
465                 angel_devision= (angel_xy[1]/ angel_xy[0])
466                 #print(angel_devision)
467                 azimut = (math.degrees(math.atan(angel_devision)))
468                 azimut_1 = (180+90-azimut)
469
470             elif angel_xy[1] < 0:
471
472                 angel_devision= (angel_xy[1]/ angel_xy[0])
473                 #print(angel_devision)
474                 azimut = (math.degrees(math.atan(angel_devision)))
475                 azimut_1 = (270+(-1*azimut))
476
477
478     #save Values for Pure Data X and Y are displayed in %factor as relative coordinat to circal diameter

```

```

559
560     sock = socket.socket(socket.AF_INET, # Internet
561                          socket.SOCK_DGRAM) # UDP
562     sock.sendto(MESSAGE.encode(), (UDP_IP1, UDP_PORT1))
563
564     #cv2.imshow('Contours', print_img)
565     #Encode the frame to JPEG
566     if mode == 1:
567         ret, buffer = cv2.imencode('.jpg', print_img, encode_param)
568         frame1 = buffer.tobytes()
569         yield (b'--frame1\r\n'
570              + b'Content-Type: image/jpeg\r\n\r\n' + frame1 + b'\r\n')
571     #Printing original Objekt ID to Frame (only for comparison, no real purpose)
572     #for object_id, pt in tracking_objects.items():
573
574         #cv2.circle(frame, pt, 5, (0, 0, 255), -1)
575         #cv2.putText(frame, str(object_id), (pt[0], pt[1] +30), 0, 1, (0, 0, 255), 2)
576
577
578     # Make a copy of the points
579     center_points_prev_frame = center_points_cur_frame.copy()
580
581     key = cv2.waitKey(1)
582     if key == 13:
583         break
584
585         # timer for Input check to quit
586     time.sleep(0.01)
587
588     # Check for user input
589     input_char = check_for_input()
590     if input_char == "q":
591         frame1 = mask_read
592         break
593
594     end_frame_time = time.time() # take end time of framecycle
595     frame_processing_time = end_frame_time - start_frame_time # calculate absolut time
596     #print(f"Frame {count}: {frame_processing_time:.4f} seconds")
597
598     print ("brake")
599     print ("-----")
600     cf.save_config()
601     end_time = time.time()
602     total_time = end_time - start_time
603     fps = count / total_time
604
605
606     print(f"Total frames: {count}")
607     print(f"Total time: {total_time:.2f} seconds")
608     print(f"Average FPS: {fps:.2f}")
609     print(rnbo)
610
611
612     if video_source == 1:
613         picam2.stop()
614         picam2.close()
615         cap.release()
616         print ('> client closing:')
617         print ('<<<<<<RESTART>>>>>>:')
618
619
620 # swarm of sound Version 1.0
621 # Copyright (c) [2024] [Sebastian Obel]
622
623 # Permission is hereby granted, free of charge,
624 # to any person obtaining a copy of this software and associated documentation files (the "Software"),
625 # to deal in the Software without restriction, including without limitation the rights to use,
626 # copy, modify, merge, publish, distribute, sublicense,
627 # and/or sell copies of the Software
628
629 # This Software was inspired by:
630 # Sergio-Canu
631 # https://pysource.com/2021/10/05/object-tracking-from-scratch-opencv-and-python/
632 # and Sharon McCaman and David Tinapple
633 # Sharon McCaman and David Tinapple
634 # http://www.sharonmccaman.com
635 # http://www.sharonmccaman.com
636 # https://www.youtube.com/watch?v=cytx9NqSQNA
637 # https://tinapple.notion.site/Tracking-Blob-Tracking-9b5b314087074429808b53bf4598e4de
638

```

```

479         azimuth_dict[k] = int(azimut_1)
480         x_dict[k] = "%.3f" % ((pt[0] - x1) / (2 * radius_circle))
481         y_dict[k] = "%.3f" % ((y1 - pt[1]) / (2 * radius_circle))
482         radius_dict[k] = "%.3f" % (angel_radius / radius_circle)
483         id_active_dict[k] = vid
484
485         #Print Visuals to Frame
486         cv2.circle(print_img, pt, 5, (0, 0, 255), -1)
487         cv2.putText(print_img, str(k), (pt[0], pt[1] + 30), 0, 1, (0, 0, 255), 2)
488         #cv2.line(print_img, pt, center_xy, (255, 0, 0), 2)
489         #cv2.putText(print_img, str(x_dict[k]), (pt[0], pt[1] + 60), 0, 1, (0, 0, 255), 2)
490
491         #print center of the Image and Tracking circle to Frame
492         cv2.circle(print_img, center_xy, radius_circle, (0, 165, 255), 2)
493         cv2.circle(print_img, center_xy, 3, (0, 165, 255), 2)
494         #cv2.line(frame, pt_xy, center_xy, (0, 0, 255), 2)
495         cv2.line(print_img, (0, ((resolution[1])//2)), (resolution[0], ((resolution[1])//2)), (0, 0, 0), 2)
496         cv2.line(print_img, (((resolution[0])//2), 0), (((resolution[0])//2), resolution[1]), (0, 0, 0), 2)
497
498         #Print Koordinaten Ursprung
499         cv2.circle(print_img, (x1, y1), 10, (0, 165, 255), 2)
500         #Print XY Axes
501         cv2.arrowedLine(print_img, (x1, y1), ((x1 + (2 * radius_circle)), y1), (255, 0, 0), 1, tipLength = 0.03)
502         cv2.arrowedLine(print_img, (x1, y1), (x1, y1 - (2 * radius_circle)), (255, 0, 0), 1, tipLength = 0.03)
503         cv2.putText(print_img, str('y-axis'), (x1 + 5, y1 - (2 * radius_circle) + 35), 0, 1, (255, 0, 0), 1)
504         cv2.putText(print_img, str('x-axis'), (x1 - 100 + (2 * radius_circle), y1 - 15), 0, 1, (255, 0, 0), 1)
505 #####
506         # Draw Speaker Positions
507         dot_radius = 10
508         dot_color = (0, 0, 255)
509         num_points = 8
510         angle_step = 360 / num_points
511
512         for i in range(num_points):
513             angle_deg = angle_step * i
514             angle_rad = np.deg2rad(angle_deg)
515
516             x = int(center_xy[0] + radius_circle * np.sin(angle_rad))
517             y = int(center_xy[1] - radius_circle * np.cos(angle_rad))
518
519             # Draw the dot
520             cv2.circle(print_img, (x, y), dot_radius, dot_color, -1)
521             cv2.line(print_img, (x, y), center_xy, (255, 0, 0), 2)
522 #####
523
524
525
526         print('IDAC:' + str(id_active_dict))
527         print('RNBO:' + str(rnbo))
528
529         #print('TROB:' + str(tracking_objects))
530
531         #print('IDCK:' + str(id_check_list))
532         #print('COPY:' + str(id_check_list_copy))
533
534         #transform dicts to lists, for prep for UDP
535         azimuth_list = list(azimut_dict.values())
536         x_list = list(x_dict.values())
537         y_list = list(y_dict.values())
538         radius_list = list(radius_dict.values())
539         id_active_list = list(id_active_dict.values())
540         #radius_list = [cf.var_4, cf.var_5]
541
542         #consolidate azimuth, x, y_list's into one list
543         PD_list = azimuth_list + x_list + y_list + radius_list + id_active_list
544
545         #replace Placeholder Values with 0
546         #for index, fruit in enumerate(PD_list): # inactive as old values are passed there
547             #if fruit == {0: (0, 0)}:
548                 #PD_list[index] = 0
549
550         # Sending Message to Bella/Pure Data over UDP Socket
551         MESSAGE = " ".join(str(x) for x in PD_list)
552
553         sock = socket.socket(socket.AF_INET, # Internet
554                             socket.SOCK_DGRAM) # UDP
555         sock.sendto(MESSAGE.encode(), (UDP_IP, UDP_PORT))
556
557         # Sending Message to Bella/Pure Data over UDP Socket
558         MESSAGE = " ".join(str(x) for x in PD_list)

```

### Source code 3: Object tracking `http_video.py` handling web server and video stream

28/08/2024, 10:52

`http_video.py`

`http_video.py`

```
1 from flask import Flask, Response, render_template, request, jsonify, redirect, url_for, session
2 from functools import wraps
3 import config
4 import object_tracking
5
6 app = Flask(__name__)
7 app.secret_key = config.SECRET_KEY
8
9 def check_auth(username, password):
10     return username == config.USERNAME and password == config.PASSWORD
11
12 def authenticate():
13     return Response(
14         'Could not verify your access level for that URL.\n'
15         'You have to login with proper credentials', 401,
16         {'WWW-Authenticate': 'Basic realm="Login Required"'})
17
18 def requires_auth(f):
19     @wraps(f)
20     def decorated(*args, **kwargs):
21         if 'logged_in' not in session:
22             return redirect(url_for('login'))
23         return f(*args, **kwargs)
24     return decorated
25
26 @app.route('/login', methods=['GET', 'POST'])
27 def login():
28     if request.method == 'POST':
29         username = request.form['username']
30         password = request.form['password']
31         if check_auth(username, password):
32             session['logged_in'] = True
33             return redirect(url_for('index'))
34         else:
35             return authenticate()
36     return render_template('login.html')
37
38 @app.route('/logout')
39 def logout():
40     session.pop('logged_in', None)
41     return redirect(url_for('login'))
42
43
44
45 @app.route('/set_var', methods=['POST'])
46 @requires_auth
47 def set_var():
48     var_index = int(request.form['var_index'])
49     value = float(request.form['value'])
50     if var_index == 1:
51         config.var_1 = value
52     elif var_index == 2:
53         config.var_2 = value
54     elif var_index == 3:
55         config.var_3 = value
56     elif var_index == 4:
57         config.var_4 = value
58     elif var_index == 5:
59         config.var_5 = value
60     elif var_index == 6:
61         config.var_6 = value
62     config.save_config() # Save configuration to file
63     return jsonify(status='success', value=value)
64
65 @app.route('/toggle_var6', methods=['POST'])
66 @requires_auth
67 def toggle_var6():
68     config.var_6 = 1 if config.var_6 == 0 else 0
69     config.save_config() # Save configuration to file
70     return jsonify(status='success', value=config.var_6)
71
72 @app.route('/video_feed')
73 @requires_auth
74 def video_feed():
75     return Response(object_tracking.object_tracking1(config.config), mimetype='multipart/x-mixed-replace; boundary=frame')
76
77
78 @app.route('/')
79 @requires_auth
80 def index():
81     return render_template('index.html')
82
83 def start_video_feed():
84     try:
85         app.run(host='0.0.0.0', port=5001, debug=True, use_reloader=False)
86     except Exception as e:
87         print(f"Failed to start video feed: {e}")
88
```

localhost:65427/1c6ddfbcf5d44c6b6d49830c8feab44/

1/1

#### Source code 4: Object tracking config.py background data handling and storing between the Python scripts

28/08/2024, 10:53

config.py

config.py

```
1 import os
2 import json
3
4 CONFIG_FILE = "config.json"
5
6 var_1 = 0.3
7 var_2 = 0.9
8 var_3 = 0.5
9 var_4 = 0.5
10 var_5 = 0.5
11 var_6 = 1
12
13 mode = 0
14
15
16
17 USERNAME = 'swarm_of_sound'
18 PASSWORD = 'dgfuy37d773gei83qgweuf7re36t4efgdanseiu'
19 SECRET_KEY = os.urandom(24)
20
21 # Configuration dictionary
22 config = {
23     'azimut_1': 0,
24     'area': 0,
25     'pxc': 0,
26     'id': 0,
27     'azimut_dict': {},
28     'rnbo': {1: {(0, 0)}, 2: {(0, 0)}, 3: {(0, 0)}, 4: {(0, 0)}, 5: {(0, 0)}, 6: {(0, 0)}, 7: {(0, 0)}, 8:
29     {(0, 0)}},
30     'resolution': (1520, 1520),
31     'binary_thresh1': 80,
32     'binary_thresh2': 255,
33     'contour_size': 150,
34     'tracking_distance': 40,
35     'pick_img': 0,
36     'show_img_local': 0,
37     'tcp_stream': 0,
38     'video_source': 0,
39     'radius_circle': 730,
40     'ip_adress': "192.168.7.2",
41     'Picamera2': None,
42     'pi_mode': 0
43 }
44
45 def save_config():
46     with open(CONFIG_FILE, 'w') as f:
47         json.dump({
48             'var_1': var_1,
49             'var_2': var_2,
50             'var_3': var_3,
51             'var_4': var_4,
52             'var_5': var_5,
53             'var_6': var_6,
54             'config': config,
55             'mode': mode
56         }, f)
57
58 def load_config():
59     global var_1, var_2, var_3, var_4, var_5, var_6, config, mode
60     if os.path.exists(CONFIG_FILE):
61         with open(CONFIG_FILE, 'r') as f:
62             data = json.load(f)
63             var_1 = data['var_1']
64             var_2 = data['var_2']
65             var_3 = data['var_3']
66             var_4 = data['var_4']
67             var_5 = data['var_5']
68             var_6 = data['var_6']
69             config = data['config']
70
71 # Load configuration on module import
72 load_config()
```

localhost:65427/6468fba2-3b24-41eb-bf9f-0ab91edaa5a1/

1/1



Source code 5: Object tracking config.json saved data to initialize on program start

28/08/2024, 10:52

config.json

config.json

```
1 {"var_1": 0.19, "var_2": 0.048, "var_3": 0.86, "var_4": 44.0, "var_5": 0.91, "var_6": 1.0, "config": {"azimut_1": 0, "area": 0,
  "pxc": 0, "id": 0, "azimut_dict": {}, "rnbo": {"1": {"18": [1157, 664]}, "2": {"19": [163, 912]}, "3": {"16": [984, 1096]}, "4":
  {"1": [1074, 207]}, "5": {"1": [959, 168]}, "6": {"1": [169, 601]}, "7": {"1": [840, 142]}, "8": {"1": [809, 238]}},
  "resolution": [1520, 1520], "binary_thresh1": 30, "binary_thresh2": 255, "contour_size": 100, "tracking_distance": 40, "pick_img":
  0, "show_img_local": 0, "tcp_stream": 0, "video_source": 0, "radius_circle": 720, "ip_adress": "192.168.7.2", "Picamera2": null,
  "pi_mode": 0}, "mode": 0}
```

## Source code 6: Object tracking index.html calibration web interface and receiving video stream

28/08/2024, 10:54

index.html

templates/index.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Video Stream</title>
5 <style>
6     body {
7         display: flex;
8         flex-direction: column;
9         align-items: center;
10        justify-content: center;
11        height: 100vh;
12        margin: 0;
13        background-color: #f0f0f0;
14    }
15    .video-container {
16        flex: 1;
17        display: flex;
18        justify-content: center;
19        align-items: center;
20        width: 100%;
21        max-height: 100%;
22        margin: 10px;
23        background-color: #f0f0f0;
24        position: relative;
25    }
26    .video-container img {
27        max-width: 70%;
28        max-height: 70%;
29        width: auto;
30        height: auto;
31        object-fit: contain;
32    }
33    .slider-container {
34        display: flex;
35        flex-direction: column;
36        align-items: center;
37        width: 100%;
38    }
39    .slider-group {
40        display: flex;
41        justify-content: space-around;
42        width: 80%;
43        margin: 20px 0;
44    }
45    input[type=range] {
46        width: 250px;
47    }
48 </style>
49 <script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
50 </head>
51 <body>
52 <div class="video-container">
53 
54 </div>
55 <div class="slider-container">
56 <div class="slider-group">
57 <div>
58 <label for="var1Slider">binary_threshhold</label>
59 <input type="range" id="var1Slider" min="0" max="1" step="0.01" value="0" oninput="updateVar(1, this.value)">
60 <p>Value: <span id="var1Value">0.5</span></p>
61 </div>
62 <div>
63 <label for="var2Slider">blurr_factor</label>
64 <input type="range" id="var2Slider" min="0" max="0.3" step="0.001" value="0" oninput="updateVar(2, this.value)">
65 <p>Value: <span id="var2Value">0.5</span></p>
66 </div>
67 <div>
68 <label for="var3Slider">tracking_radius</label>
69 <input type="range" id="var3Slider" min="0" max="1" step="0.01" value="90" oninput="updateVar(3, this.value)">
70 <p>Value: <span id="var3Value">0.5</span></p>
71 </div>
72 </div>
73 <div class="slider-group">
74 <div>
75 <label for="var4Slider">360_degree</label>
76 <input type="range" id="var4Slider" min="1" max="100" step="1" value="1" oninput="updateVar(4, this.value)">
77 <p>Value: <span id="var4Value">0.5</span></p>
78 </div>
```

localhost:65427/fbaa2eb4-7701-4b4b-8564-9457fbc01841/

1/2

```
79     <div>
80         <label for="var5Slider">var_5</label>
81         <input type="range" id="var5Slider" min="0" max="1" step="0.01" value="0.5" oninput="updateVar(5, this.value)">
82         <p>Value: <span id="var5Value">0.5</span></p>
83     </div>
84     <div>
85         <label for="var6Slider">pick_image</label>
86         <input type="range" id="var5Slider" min="0" max="1" step="1" value="1" oninput="updateVar(6, this.value)">
87         <p>Value: <span id="var6Value">0.5</span></p>
88     </div>
89 </div>
90 </div>
91 <script type="text/javascript">
92     function updateVar(varIndex, value) {
93         document.getElementById('var' + varIndex + 'Value').innerText = value;
94         $.post('/set_var', {var_index: varIndex, value: value});
95     }
96 </script>
97 </body>
98 </html>
99
100
```

## Source code 7: Object tracking login.html login page

28/08/2024, 10:54

login.html

templates/login.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4   <title>Login</title>
5 </head>
6 <body>
7   <h1>Login</h1>
8   <form method="post">
9     <label for="username">Username:</label>
10    <input type="text" id="username" name="username" required>
11    <br>
12    <label for="password">Password:</label>
13    <input type="password" id="password" name="password" required>
14    <br>
15    <button type="submit">Login</button>
16  </form>
17 </body>
18 </html>
19
```

localhost:65427/5185a676-8079-44cf-a0f6-67b7fb0ef1ac/

1/1

## Source code 8: sketch.js web interface script for output gain adjustment hosted on the Bela

28/08/2024, 10:55

sketch.js

sketch.js

```
1 let buffer = new Array(17).fill(0);
2 let result;
3 let counter = 0;
4 let save_state = 0;
5 let button;
6
7 function preload() {
8   console.log("Preloading file...");
9   values_load = loadStrings("/projects/SOS/values.txt", fileLoaded, fileLoadError);
10 }
11
12 function fileLoaded(data) {
13   console.log("File loaded successfully");
14   result = data;
15   console.log(values_load);
16 }
17
18 function fileLoadError(err) {
19   console.error("Error loading file:", err);
20 }
21
22 function setup() {
23   createCanvas(windowWidth, windowHeight);
24   setupElements();
25 }
26
27 function setupElements() {
28   background(200, 200, 200);
29
30   let values_string = values_load[0];
31   let values = splitTokens(values_string);
32
33   let v1 = parseFloat(values[0]);
34   let v2 = parseFloat(values[1]);
35   let v3 = parseFloat(values[2]);
36   let v4 = parseFloat(values[3]);
37   let v5 = parseFloat(values[4]);
38   let v6 = parseFloat(values[5]);
39   let v7 = parseFloat(values[6]);
40   let v8 = parseFloat(values[7]);
41   let v9 = parseFloat(values[8]);
42   let v10 = parseFloat(values[9]);
43   let v11 = parseFloat(values[10]);
44   let v12 = parseFloat(values[11]);
45   let v13 = parseFloat(values[12]);
46   let v14 = parseFloat(values[13]);
47   let v15 = parseFloat(values[14]);
48   let v16 = parseFloat(values[15]);
49
50   let n = windowWidth * 0.1;
51   let totalWidth = n * 8;
52   let x1 = (windowWidth - totalWidth) / 2;
53   let y1 = windowHeight * 0.2;
54
55   // Remove existing sliders and labels if they exist
56   for (let i = 1; i <= 16; i++) {
57     if (window['slider${i}']) window['slider${i}'].remove();
58     if (window['labelNames${i}']) window['labelNames${i}'].remove();
59     if (window['labelValues${i}']) window['labelValues${i}'].remove();
60   }
61
62   // Create sliders for snd1-snd8
63   for (let i = 1; i <= 8; i++) {
64     let slider = createSlider(0, 1, eval(`v${i}`), 0.01);
65     slider.style("transform", "rotate(-90deg)");
66     let sliderX = x1 + (i - 1) * n;
67     slider.position(sliderX, y1);
68     slider.size(windowHeight * 0.25);
69     eval(`slider${i} = slider`);
70
71     let labelName = createP(`snd${i}`);
72     labelName.position(sliderX + slider.width / 2 - 10, y1 - windowHeight * 0.17);
73     labelName.style('text-align', 'center');
74     eval(`labelNames${i} = labelName`);
75
76     let labelValue = createP('');
77     labelValue.position(sliderX + slider.width / 2 - 10, y1 + windowHeight * 0.12);
78     labelValue.style('text-align', 'center');
79     eval(`labelValues${i} = labelValue`);
```

localhost:65427/cf1f97e6-3779-4e0d-9e14-be77f4d59f78/

1/2

```

80   }
81
82   // Create sliders for vol1-vol8
83   for (let i = 9; i <= 16; i++) {
84     let slider = createSlider(0, 1, eval(`v${i}`), 0.01);
85     slider.style("transform", "rotate(-90deg)");
86     let sliderX = x1 + (i - 9) * n;
87     slider.position(sliderX, y1 + windowHeight * 0.45);
88     slider.size(windowHeight * 0.25);
89     eval(`slider${i} = slider`);
90
91     let labelName = createP(`vol${i - 8}`);
92     labelName.position(sliderX + slider.width / 2 - 10, y1 + windowHeight * 0.29);
93     labelName.style('text-align', 'center');
94     eval(`labelName${i} = labelName`);
95
96     let labelValue = createP("");
97     labelValue.position(sliderX + slider.width / 2 - 10, y1 + windowHeight * 0.57);
98     labelValue.style('text-align', 'center');
99     eval(`labelValue${i} = labelValue`);
100  }
101
102  if (button) button.remove();
103  button = createButton("save");
104  button.mouseClicked(save_state);
105  button.size(windowWidth * 0.08, windowHeight * 0.04);
106  button.position(windowWidth * 0.9, windowHeight * 0.9);
107 }
108
109 function press_save() {
110   console.log('Button pressed');
111   save_state = 1;
112   setTimeout(() => {
113     save_state = 0;
114     console.log('save_state set back to 0');
115   }, 1000);
116 }
117
118 function draw() {
119   buffer[0] = slider1.value();
120   buffer[1] = slider2.value();
121   buffer[2] = slider3.value();
122   buffer[3] = slider4.value();
123   buffer[4] = slider5.value();
124   buffer[5] = slider6.value();
125   buffer[6] = slider7.value();
126   buffer[7] = slider8.value();
127   buffer[8] = slider9.value();
128   buffer[9] = slider10.value();
129   buffer[10] = slider11.value();
130   buffer[11] = slider12.value();
131   buffer[12] = slider13.value();
132   buffer[13] = slider14.value();
133   buffer[14] = slider15.value();
134   buffer[15] = slider16.value();
135   buffer[16] = save_state;
136   Bela.data.sendBuffer(0, 'float', buffer);
137
138   labelValue1.html(slider1.value().toFixed(2));
139   labelValue2.html(slider2.value().toFixed(2));
140   labelValue3.html(slider3.value().toFixed(2));
141   labelValue4.html(slider4.value().toFixed(2));
142   labelValue5.html(slider5.value().toFixed(2));
143   labelValue6.html(slider6.value().toFixed(2));
144   labelValue7.html(slider7.value().toFixed(2));
145   labelValue8.html(slider8.value().toFixed(2));
146   labelValue9.html(slider9.value().toFixed(2));
147   labelValue10.html(slider10.value().toFixed(2));
148   labelValue11.html(slider11.value().toFixed(2));
149   labelValue12.html(slider12.value().toFixed(2));
150   labelValue13.html(slider13.value().toFixed(2));
151   labelValue14.html(slider14.value().toFixed(2));
152   labelValue15.html(slider15.value().toFixed(2));
153   labelValue16.html(slider16.value().toFixed(2));
154
155 }
156
157 function windowResized() {
158   resizeCanvas(windowWidth, windowHeight);
159   setupElements(); // Adjust the elements to the new window size
160 }
161

```

### **Declaration of Authenticity**

I declare that I completed the Bachelor thesis independently and used only these materials that are listed. All materials used, from published as well as unpublished sources, whether directly quoted or paraphrased, are duly reported. Furthermore, I declare that the Bachelor thesis, or any abridgment of it, was not used for any other degree seeking purpose.

Berlin 28.08.2024

A handwritten signature in black ink, appearing to be 'J. Sch.', is written over a horizontal line.

Signature