Determination Of Speaker Configuration For An Immersive Audio Content Creation System

Submitted in fulfilment
of the requirements of the degree of

Master of Science

of Rhodes University

Motebang Lebusa

Grahamstown, South Africa
July 2020

Abstract

Various spatialisation algorithms require the knowledge of speaker locations to accurately localise sound in 3D environments. The rendering process uses speaker coordinates to feed into their algorithms so that they can render the immersive audio content as intended by an artist. The need to measure the loudspeaker coordinates becomes necessary, especially in environments where the speaker layouts change frequently. Manually measuring the coordinates, however, tends to be a laborious task that is prone to errors.

This research provides an automated solution to the problem of speaker coordinates measurement. The solution system, SDIAS, is a client-server system that uses the capabilities provided by the Ethernet Audio Video Bridging standard to measure the 3D loudspeaker coordinates for immersive sound systems. SDIAS deploys commodity hardware and readily available software to implement the solution.

A server sends a short tone to each speaker in the speaker configuration, at equal intervals. A microphone attached to a mobile device picks up these transmitted tones on the client side, from different locations. The transmission and reception times from both components of the system are used to measure the time of flight for each tone sent to a loudspeaker. These are then used to determine the 3D coordinates of each loudspeaker in the available layout. Tests were performed to determine the accuracy of the determination algorithm for SDIAS, and were compared to the manually measured coordinates.

Acknowledgements

First and foremost, I would like to dedicate this work to my family and friends for their continued support throughout the seemingly endless journey for this research. Their love and support brought me to where I am today.

I would also like to thank all my research mates whom we travelled this journey with. Their interactions provided an atmosphere supportive of the challenges encountered throughout the research endeavours.

Finally, I would like to express my deepest gratitude to my supervisor, Professor Richard Foss, for the tireless work he has done in supervising me. His guidance and motivation kept me going, when I would have otherwise given up. Also, I deeply appreciate all the support he provided, not just on the research work, but also on securing funding to continue with my work, as well as accommodation when I had none. Without his tireless efforts, I would have not been able to reach this far. For all these, I will remain forever grateful.

This work was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from Telkom SA, Tellabs, Genband, Easttel, Bright Ideas 39, THRIP and NRF SA (TP13070820716). The authors acknowledge that opinions, findings and conclusions or recommendations expressed here are those of the author(s) and that none of the above mentioned sponsors accept liability whatsoever in this regard.

Contents

List of Figures

List of Tables

Abbreviations

2D	2-Dimensional
3D	3-Dimensional
API	Application Programming Interface
ASIO	Audio Stream Input Output
AVB	Audio Video Bridging
CBA	Channel-Based Audio
DBAP	Distance-Based Amplitude Panning
HOA	Higher Order Ambisonics
HTTP	Hypertext Transfer Protocol
IO	Input/Output
IEEE	Institute Of Electrical And Electronics Engineers
JSON	JavaScript Object Notation
JUCE	Jules Utility Class Extensions
LFE	Low-Frequency Effects
MLS	Maximum Length Sequence
OBA	Object-Based Audio
PC	Personal Computer
SBA	Scene-Based Audio
TOF	Time Of Flight
UI	User Interface
URL	Unique Resource Locator Interface
VBAP	Vector Base Amplitude Panning
WFS	Wave Field Synthesis
XML	Extensible Markup Language

Chapter 1

Introduction

Immersive sound has been making considerable progress in the entertainment industry in recent years. Its use cases, from production to reproduction, span a wide range - from studio productions, live sound systems, cinema, home theatre, art installations, live performances and virtual reality amongst others. Technological advances, such as the Internet of Things, are well-poised to further improve, as well as extend, the currently available use cases to provide new applications for immersive sound. For instance, audio could be embedded in remote objects via web technologies, such as the HTTP and web APIs, to connect the remote objects and transmit audio over geographically segregated locations.

Surround sound systems, such as 5.1 or 7.1, install loudspeakers in a horizontal plane surrounding a listener. For example, in a 5.1 system, the '5' represents the number of channels for mid to high frequencies, with each channel routed to a single loudspeaker. These loudspeakers form the base or surround sound layer. The '.1' represents a channel designated for low frequencies, typically played out on an LFE loudspeaker, also called a subwoofer (Holman, 2012). For creation of audio content for surround sound systems, audio signals are typically mixed to a predefined number of channels. These channels are rendered to specific (and geometrically regular) loudspeaker configurations. Audio mixed using this approach is called channel-based audio (CBA) (ITU-R, 2013). There are two problems with this CBA approach:

1. sound only emanates from the horizontal plane. This is in contrast to the real-world, where sound comes from any direction, and
2. the predefined number of channels and required room geometry are often not met during playback, as such, fidelity may be compromised.

1.1 Height Dimension in Audio

Immersive sound takes sound to a higher dimension. The goal of immersive sound is to encompass a listener in a full 360° sound field. During content creation, audio must be created with this capability of occupying any point in the immersive sound field. Spatialisation algorithms are used for creating immersive audio content. They create virtual sound images in the 3D space using various techniques, providing a mechanism to create a sound source possibly anywhere in the 3D space without the need to have a loudspeaker at that particular geo-point. These algorithms represent audio as objects or scenes in space, with associated metadata that is used later during reproduction. The resulting audio is called object-based audio (OBA) and scene-based audio (SBA) respectively. Example spatialisation algorithms include vector-based amplitude panning, distance-based amplitude panning and ambisonics (Kostadinov, Reiss, & Mladenov, 2010; Malham & Myatt, 1995; Pulkki, 1997).

OBA and SBA avoids problems associated with the CBA approach. Sound can be heard from any point in the immersive sound field, as provided by metadata in the case of OBA, and by encoded sound channels in the case of SBA. Also, these formats adapt to the available speaker configurations, as such, sound fidelity as created by an artist is retained.

For immersive sound reproduction, the surround sound configuration is augmented by loudspeakers at heights above the listener, called *height* loudspeakers, which make the configuration immerse the listener in the sound field delivered by the loudspeakers (Pohlmann, 2010). The different height levels of these additional loudspeakers reside in primarily two layers, namely the *height* and *top* layers (Barco, 2015). One approach is to arrange physical loudspeakers at varying heights across the room. Another approach is to place speakers such that they fire sound upwards to the ceiling, with the intention of having it bounce off the ceiling back to the listener (Dolby Laboratories, 2018b).

Another factor in the delivery of spatial audio is the ability of a renderer to decode such content. The renderer must be able to decode the 3D audio content for playback on the available speaker layout to preserve the spatial audio integrity. Such formats for which renderers must be equipped to decode 3D content include the immersive Dolby Atmos (Dolby Laboratories, 2012) and Auro-3D (Auro Technologies, 2019) formats.

1.2 Research Problem

An immersive audio content creation system - Immergo - has been developed at Rhodes University (Foss & Rouget, 2015). Immersive content creations system such as Immergo use spatialisation algorithms to localise sound sources in the 3D space. These algorithms depend on the positions of the loudspeakers to perform localisation. Various loudspeaker configurations may be employed by the 3D spatial audio content creation systems. The knowledge of loudspeakers positions is required in the production of immersive content by the localisation algorithms. Spatialisation algorithms incorporate the use of speaker positions in the 3D audio content workflow. This makes loudspeaker positions a key component for many localisation algorithms in various ways. As such, the need to accurately and quickly determine speaker positions in any given speaker layout becomes necessary, especially as the layouts change for various reasons, for instance, when a spatialised sound design is played back in various venues.

The determination of loudspeaker positions could greatly benefit from an automated approach instead of manually measuring the distances (e.g. with a laser distance measurer). This removes the need for the engineer to undergo a manual measurement process which tends to be laborious especially in large venues each time the configuration changes.

1.3 Research Question and Objectives

The Immergo system is a client-server system that uses mobile devices for the control of sound source spatialisation. This model enables a lot of positional flexibility on the part of sound engineers. They are able to move around a venue as they control the localisation of sound sources. With the ever-increasing ubiquity of mobile devices, there are opportunities for innovation and creativity using these devices. This thesis describes a project that was initiated with the hypothesis that a similar client-server configuration could be used to determine speaker positions. With this in mind, the specific question this research work sought to answer was the following:

"Can mobile devices be used in determining loudspeaker configurations for immersive audio content creation systems?"

Readily available hardware and software, such as consumer mobile devices and web APIs, were used to investigate the feasibility of developing such a mobile device-based solution

to the problem of speaker position measurement automation. To explore answers to this question, the research therefore deals with the specific objectives that seek to guide the exploration of solutions. The specific objectives of the research project were to:

1. review current literature and standards pertaining to immersive audio content, loudspeaker configurations and sound spatialisation,
2. explore various approaches that could be used to determine 3D loudspeaker coordinates, and
3. design and implement a system that will enable speaker configuration via a mobile device.

1.4 Thesis Layout

The rest of this thesis document is structured as follows:

- Chapter 2 describes various methods for localising immersive audio. The need to automate, hence expedite, the process of determining speaker positions used in the spatialisation algorithms is highlighted.
- Chapter 3 introduces a system proposed as a solution to automating and expediting the speaker measurement process - SDIAS. The client-server system uses a mobile device with an attached microphone to measure speaker positions for recording short impulses sent by the loudspeakers. A user, operating the mobile device, makes requests to a web server, which connects to an Ethernet AVB (IEEE 802.1BA, 2011) network, for streaming these short impulses. These requests carry out instructions necessary to complete the measurement process.
- Chapter 4 gives details of the underlying technology behind SDIAS. Various source type possibilities are explored, highlighting the final one of these sources selected for generating a waveform that could be used by the tones server. A process which synchronises the client and server components is also described. A discussion on how distances are extracted from the recorded audio is given in this chapter. These are accompanied by the details of how the distances are used in computing the 3D loudspeaker coordinates as a final result.
- Chapter 5 delves into the finer details of the design and implementation processes of the proposed system solution. A system design model used to specify and visualise the behaviour and structure of the system is explained in this chapter. The implementation details of SDIAS are described, from the Ethernet AVB network, the client, the web server through to the tones server.

- Chapter 6 presents various results obtained from the testing phase of SDIAS. A series of tests were performed to objectively compare the results obtained dynamically from SDIAS with those manually measured. A statistical analysis of these results is also given, to evaluate the accuracy of the system under the various test configurations used.
- Chapter 7 concludes the thesis project by reviewing the research objectives, giving a summary of the thesis and highlighting some achievements, limitations and possibilities for future work.

Chapter 2

Immersive Sound

Sound reproduction has evolved from mono, through stereo, to surround and recently to immersive soundscapes. Entertainment, especially in the film and game industries, is the main driver and consumer of the recent technological advances. Immersive sound enhances a sense of realism and immersion in the sound field. With immersive sound, a listener gets a perception that sound is coming from all directions - that the sound is truly in the air around them, not only from the horizontal plane provided by the surround sound. This is achieved with an addition of another dimension to the typical surround sound's 2D plane, yielding a perception of being immersed in the entire real-world surrounding soundscape (Pohlmann, 2010). The sound field now expands to various levels above the horizontal plane at the listener level. The encompassing 3D sound space is known as 3D sound, 3D spatial sound or immersive sound. Fundamentals of immersive sound environments are discussed in Section 2.1.

Various technologies are deployed along the 3D audio chain flow, that is, from production, through distribution to playback. Within the field of immersive audio, spatialisation algorithms are used for positioning sound sources in the 3D space. Depending on the techniques used for encoding the audio content, the algorithms are regarded as object-based, channel-based or scene-based. The algorithms incorporate a mix of a listener and speaker positions, amongst other factors, during content creation. This process is dependent on the knowledge of loudspeaker locations within a sound field. The various spatialisation algorithms require the knowledge of the coordinates of the speakers within a sound scene. Section 2.2 discusses the various algorithms used in the creation of immersive audio content.

3D spatial audio creators use various representations of a sound source or scene to encode

it into an audio content for distribution. In SBA, a sound scene is encoded into a set of audio signal mixes (channels) that can be rendered to available speaker layouts, while in OBA, audio objects are used to represent sound source elements and their associated metadata, such as a source's position in the 3D space, its size, shape and trajectory (Barco, 2015; Ludé, 2014). The ability of these approaches to adapt to available speaker layouts provides scalability and flexibility in the delivery of immersive sound workflow, from production to playback (Gasull Ruiz, Sladeczek, & Sporer, 2015). This is contrast to CBA, where audio signals are premixed into specific speaker configurations. For instance, this means that stereo content is encoded for audio playback on a stereo sound system, where each channel is distinctly meant to be routed to an independent loudspeaker. If the stereo content was to be played on a 9.1 surround sound system, it would not take full advantage of the presence of the additional seven loudspeakers (from 2.1 to 9.1), as these additional loudspeakers would have to play back exact replicas or mixes of the only available stereo channels, while other speakers might not even have any audio signals routed to them.

Section 2.3 discusses various speaker layouts for immersive sound reproduction. The number of loudspeakers in immersive sound system configurations keeps increasing. These numbers passed 300 as far back as the 1950s (Baalman, 2010; Lombardo et al., 2005; Malham & Myatt, 1995). Currently, large loudspeaker configurations can be found in professional cinemas. An example is the Pathé Massy Dolby 400-speaker cinema installation, which uses the Dolby Atmos immersive audio format that has as many as 128 streams that can be transmitted up to 64 individual speakers per mix (Dolby Laboratories, 2018a; Roberts, 2019). On the consumer side, typical 3D consumer configurations range from eight to as high as 34 loudspeakers, with various layouts such as 7.1.2, 9.1[1], 7.1.4, 11.1, and 24.1.10 (Dolby Laboratories, 2018b). Placement of the speakers is key to realising the creator's intent of the immersive sound. These are discussed in Section 2.3.

Section 2.4 provides an overview of a solution proposed for automating the loudspeaker measurements process. This is the focus of this research project, which deals with speaker configurations for immersive audio. Section 2.2 describes some of the common spatialisation algorithms in immersive sound, including the one used by Immergo - DBAP, and work geared at harmonising various immersive audio formats in the audio field. The next section gives a brief account of the fundamental concepts in the immersive sound field.

[1]The 9.1 and 11.1 layouts are the immersive layouts such that 9.1 = 7.1 surround = 4 height speakers.

2.1 Concepts in Immersive Sound Environments

Given a sound source in the 3D space, several factors constitute the perception of sound immersion that converges to a particular listening point (be it a microphone or a human ear) in the listening environment from this sound source. As one of the forms of energy, sound energy follows properties of energy, such as transmission in a waveform amongst others. The immersive sound that reaches the listening point is a combination of all incident sound waves emanating directly from the sound sources, together with the reflections of waves from the source, but which went to other directions and were reflected from the surrounding environment, inducing sound pressure at the point (Ahveninen, Kopčo, & Jääskeläinen, 2014).

Different factors affect the sound waves during propagation. One of them is reflectance capacity of the surround materials. This aspect is responsible for reflecting, and absorbing, sound waves which will ultimately reach the listener. Depending on especially the reflective capacity of the listening space components, sound reaching the listening point usually sounds different given different reflective materials. All these differences give sound a distinctive characteristic. In a free field, the reflected waves are less of an issue than in a diffuse sound field.

During sound production, it is normally an expectation of a sound engineer that the captured sound retains its originality, that is, during sound reproduction, the original characteristic features would be perceived as they would at the time of origin - providing all the sound which would reach the listener during production. When this is the case, authenticity of the sound is said to be preserved (Blauert, 1983), as all the 3D sound which existed in the original event is retained during reproduction. Closely related to the concept of authenticity is another concept called plausibility. It refers to a metric which compares the user's expectation against the perception of a given perceived acoustic event in relation to user experience (Möller & Raake, 2014), thereby fulfilling the user's expectation. A plausible acoustic event does not necessarily imply that the event is authentic (Möller & Raake, 2014).

It is common that processes involved both during production and reproduction of sound introduce, at varying degrees, elements which pollute sound quality as humans would hear it at the real acoustic event. Sound localisation is therefore characterised by several cues, namely the interaural time delay (ITD), interaural level delay (ILD), spectral and reverberation cues in reverberant environments (Shinn-Cunningham, 2000). These concepts are further discussed in subsequent sections.

2.2 Spatialisation Algorithms and Audio Formats

Panning is the process of moving sound in space. It moves an original sound source from one position by creating a *phantom sound image* at a target position. A sound source origin seems to be emanating at a different location from that of an actual loudspeaker. This affords an artist a possibility to place sound images at the desired locations which are different from the speaker locations. This is achieved with the help of panning and spatialisation techniques. Panning is typically performed through user interfaces of panning software applications. The underlying algorithms help achieve this by deploying various techniques which incorporate the positions of the various entities involved in the sound scene, such as the original sound source, the loudspeakers, the listener and the virtual sound source. The encoded audio needs to be packaged and stored for distribution for reproduction at a later stage. Various formats exist for storing immersive audio content. Below are descriptions of common spatialisation algorithms which are deployed by various formats during audio production.

2.2.1 Spatialisation Algorithms

Ambisonics

This technique creates sound fields, also called sound scenes, using spherical harmonic decomposition of the 3D space to encode the surrounding sound (Daniel et al., 2003). This scene-based spatialisation algorithm encodes the sound scene into a set of audio signals which can be decoded to any number of speaker feeds at playback. The audio signals carry the entire sound field information, hence independent of speaker layouts used in decoding them (Hollerweger, 2013).

There are various Ambisonics formats used along the audio chain flow. These include the A-format, B-format, C-format, D-format, E-format and G-format. For instance, the A-format is used for direct Ambisonics microphone signal capture whereas the C-format is a consumer format for audio reproduction over stereo systems. However, most of these formats are not in common use, except the B-format (Dehaan, 2018; Hodges, 2011).

Using the B-format, audio signal encoding is done in a multichannel format. The encoded audio channels are defined on the surface of, or within, a unit sphere. The format can encode audio using one channel for omnidirectional reproduction; additional two channels

for the left-right and front-back axes for pantophonic systems (2D encoding), and an additional channel for the height dimension for periphonic systems (3D encoding) (Malham & Myatt, 1995). This, therefore, means the algorithm can encode immersive audio starting from four channels. Ambisonics encodes in zeroth, first and onwards to higher orders. For instance, the zero order Ambisonics encodes an audio signal only in one channel in 1D; first order Ambisonics uses an additional two channels for 2D encoding or three channels for 3D encoding.

For speaker S_i, with a sound field defined by the direction (Φ_i, Θ_i), such that Φ_i spans the horizontal plane (azimuth) and Θ_i covers the vertical plane (elevation), encoding the B-format signals for S_i in a speaker layout of N speakers, is given by the following equations:

$$W = \frac{1}{N} \sum_{i=1}^{N} S_i \left[\frac{1}{\sqrt{2}} \right] \tag{2.1}$$

$$X = \frac{1}{N} \sum_{i=1}^{N} S_i \left[cos\Phi_i cos\Theta_i \right] \tag{2.2}$$

$$Y = \frac{1}{N} \sum_{i=1}^{N} S_i \left[sin\Phi_i cos\Theta_i \right] \tag{2.3}$$

$$Z = \frac{1}{N} \sum_{i=1}^{N} S_i \left[sin\Phi_i \right] \tag{2.4}$$

where:

W is the omnidirectional information - channel 0,
X is the x-directional information - channel 1,
Y is the y-directional information - channel 2 and
Z is the z-directional information - channel 3.

Figure 2.1 shows the spherical harmonics function visualisations up to 4^{th} order Ambisonics. The sequence of the individual directional first order Ambisonics channels 0 to 3, is in the AmbiX ordering convention, such that the four B-format channels are ordered as WYZX, whereas they would be ordered as WXYZ in FuMa ordering (Nachbar, Zotter, Deleflie, & Sontacchi, 2011).

The decoding process is done through various strategies, some which require regular speaker layouts, while others can decode in irregular layouts. For instance, the projection method decodes each speaker signal using the spherical harmonics depending on the speaker's position in the speaker layout, with requirements that the layout geometry

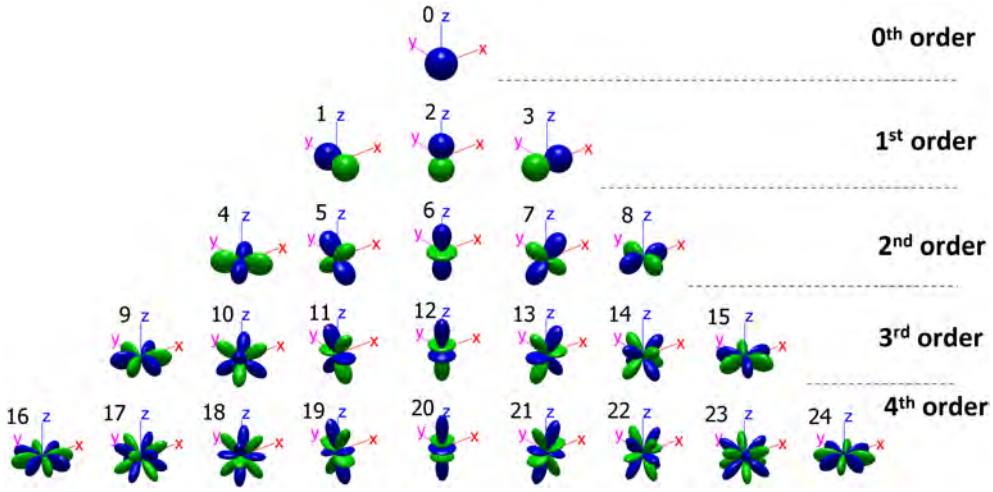


Figure 2.1: Fourth order spherical harmonics (Farina, Angelo, 2017)

is regular and has a minimum number of speakers at least equal to the number of Ambisonics channels. Figure 2.2 shows the platonic solids which could be used for speaker configurations for Ambisonics decoding, where a speaker is installed at each vertex.

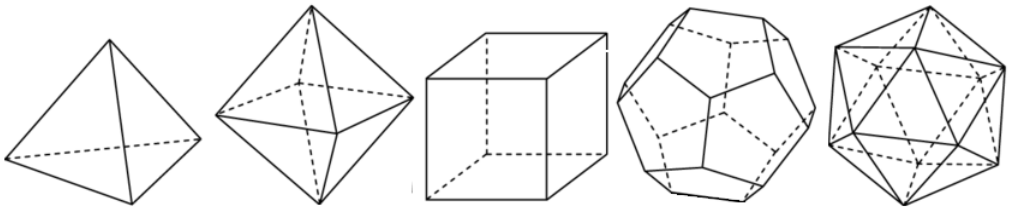


Figure 2.2: Platonic solids Ambisonics decoding speaker layout (Kaiser, 2011)

The feed signal, p_i, for speaker S_i, is given by the following equation:

$$p_i = \frac{1}{N}\left[W\frac{1}{\sqrt{2}} + (cos\Phi_i cos\Theta_i)X + (sin\Phi_i cos\Theta_i)Y + sin\Phi_i Z\right] \tag{2.5}$$

More importantly, the decoding process uses all four channels to determine the signal of a speaker, unlike in CBA systems where a channel corresponds to a speaker (Tarzan et al., 2017). Speaker positions are incorporated in the creation of a sound field, which must be equidistant in the regular geometry within the sphere of the sound scene.

The need for the information regarding speaker locations is crucial in this technique, as the encoding process relies on this locational information for encoding the sound scenes and the decoding process reproduces the sound fields using the speaker locations information in the 3D space. Because the Ambisonics technique uses speaker positions and requires

regular speaker layout geometry, it has a limited sweet spot. However, an advantage is that sound can be edited after the encoding process, such as by rotating it around any of the three axes (Malham & Myatt, 1995). The sweet spot is extensible using high order Ambisonics (HOA), though at a cost. Third order Ambisonics are in common use currently due to technological developments in the involved resources, such as the microphone arrays and panning capabilities (Farina, Angelo, 2017). HOA additionally enhances flexibility and scalability in the speaker configuration, and accuracy in sound localisation and increased listening spaces (Daniel et al., 2003; Hollerweger, 2013).

Vector Base Amplitude Panning (VBAP)

This is an OBA spatialisation technique. It is a popular technique that uses vectors to create virtual sound sources in 2D and 3D environments. The technique incorporates the listener position, together with loudspeaker coordinates at preferably the same distant from the listener to pan sound (Pulkki, 1997). The amplitude intensities of the loudspeaker signals are controlled and varied within planes bounded by the loudspeaker configuration. The gain-factor control creates an illusion of as distinct a phantom sound source as the available speaker layout can produce. To create a virtual sound source in any direction, for 2D VBAP, a pair-wise panning is applied to two adjacent loudspeakers, while a triplet-wise panning uses three loudspeakers which form a triangle for 3D VBAP (Pulkki, 1999).

For loudspeakers, S_1 to S_3, placed equidistant to the listener in a unit sphere, the unit vectors l_1 to l_3 define the directions for these speakers as shown in Figure 2.3. The virtual source, p, is expressed in terms of the following matrices:

$$p = g_1 l_1 + g_2 l_2 + g_3 l_3 \tag{2.6}$$

$$p^T = g L_{123} \tag{2.7}$$

where:

g_1 to g_3 are the gain factors for speakers S_1 to S_3, similar to the multipliers of the feed signal for the X, Y and Z from the Ambisonics in equation 2.5,

$g = [g_1 \quad g_2 \quad g_3]$, and

$L_{123} = [l_1 \quad l_2 \quad l_3]$.

The need to accurately determine speaker coordinates is also crucial in this algorithm as it uses the same gain factors, that require speaker positions, used in Ambisonics. Unlike the Ambisonics which creates a sound field, VBAP creates point sound sources - the created

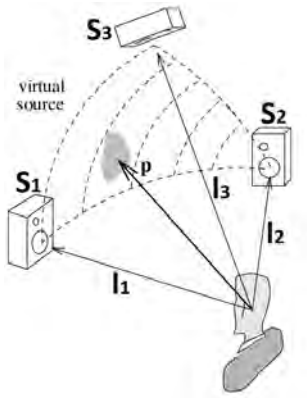


Figure 2.3: 3D VBAP sample configuration (Pulkki, 1997)

sound image has a distinct set of 2D or 3D coordinates. However, the dependence of this algorithm on the fixed listener position and speaker locations poses a constraint in terms of the area which provides the optimal listening experience, invariably leading to a limited sweet spot. Multiple-direction amplitude panning (MDAP) is a panning method which pans sound in multiple directions simultaneously (Pulkki, 1999). This algorithm seeks to solve the compromised listening experience associated with VBAP.

Wavefield synthesis (WFS)

This technique has similarities to the Ambisonics technique presented earlier in that it seeks to create sound images from sound waves. This technique is object-based as it creates the sound images in the 3D space as audio objects. Loudspeaker arrays are used to produce sound waves to provide a large listening spot (Daniel et al., 2003; Lossius, Baltazar, & de la Hogue, 2009). The theory underlying this algorithm is based on Kirchhoff-Helmholtz integral, hinged on the basis of Huygen's principle, which states that an initial source of a wave front can be substituted by a wave front that is considered secondary with the same acoustical properties of the primary source (Berkhout et al., 1993) as shown in Figure 2.4.

Following the Kirchhoff-Helmholtz integral, a sound field reproduction can be achieved, which is identical to the original sound source. This is done through the use of velocity and pressure microphone arrays during the recording stage for the capture of the original sound sources.

For reproduction, the velocity and pressure microphone arrays are substituted by an identical array of the monopole and dipole loudspeakers respectively, though the identical

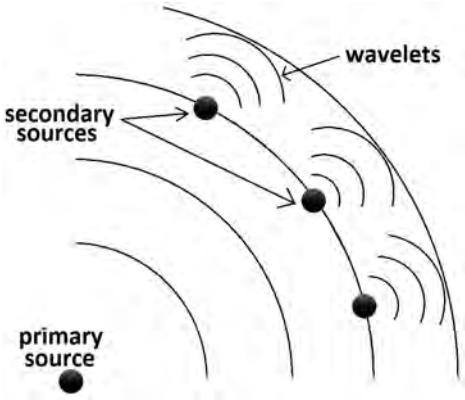


Figure 2.4: Huygen's principle (Daniel et al., 2003)

arrays constraint maybe avoided. Figure 2.5 shows a linear loudspeaker array configuration for WFS. The primary sound source creates the primary wave front, shown as the *actual wavefronts* in the figure. An array of loudspeakers is placed in a line perpendicular to the direction of the wave front towards the listener. This linear array is used, instead of planar array, as it retains the wave front shape along the linear array direction (Berkhout et al., 1993). Behind this array, the new wave fronts are formed, which have exact properties of those from the primary source. These are labelled *simulated wavefronts* in the figure.

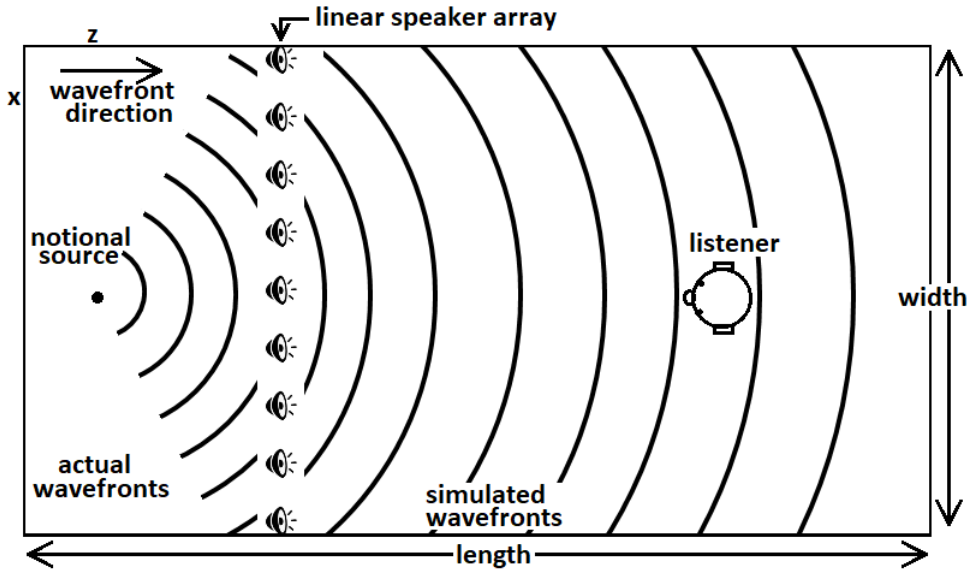


Figure 2.5: Linear loudspeakers array (Berkhout et al., 1993)

WFS, too, uses speaker locations as inputs to the equations used in the various steps involved in creating sound fields. Unlike VBAP and Ambisonics, the technique allows

for deployment in large listening areas, as the created sound field fills the entire area. However, the technique bears increased financial costs due to the increased number of loudspeakers required and computation power used for improved spatialisation (Baalman, 2010).

Distance Based Amplitude Panning (DBAP)

The technique uses loudspeaker coordinates to pan sound in space in order to place virtual sources in arbitrary locations in the 2D and 3D spaces (Kostadinov et al., 2010; Lossius et al., 2009). The algorithm is used for creation of OBA as it creates virtual sound sources as objects in the 2D and 3D spaces. The technique uses speaker locations, without prior assumptions on the listener positions and the speaker layout. This abstracts the listener position and speaker layout from the spatialisation process, and therefore increases the sweet spot by enlarging it beyond the limited specific region of sweet spot associated with other spatialisation algorithms like VBAP and Ambisonics. The listener position can still be incorporated into the algorithm, if known. DBAP uses the concept of equal intensity panning extended from a pair to an arbitrarily large array of speakers. Two assumptions are made for this technique:

1. the total intensity remains unchanged despite the new positions of the virtual sound sources, such that for a speaker S_i with gain g_i in a layout of N speakers,

$$\sum_{i=1}^{N} g_i^2 = 1 \tag{2.8}$$

2. all speakers are active at all the times, with a relation between each speaker gain factor to the distance of the virtual source given by:

$$g_i = \frac{1}{\sum_{j=1}^{N} \frac{{d_i}^2}{{d_j}^2}} \tag{2.9}$$

where d_i is a speaker distance from the virtual source, defined by the Cartesian coordinates (x_s, y_s, z_s), to each of the speakers, defined as:

$$d_i = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2 + (z_i - z_s)^2} \tag{2.10}$$

The algorithm also takes care of the undesirable consequences such as:

- the spatial blur, which occurs when the virtual source is superimposed onto a speaker, that is, when i = j in equation 2.9, causing a division by zero. This is

resolved by an addition of a spatial blur offset $r_s \geq 0$ in equation 2.10, such that the new equation becomes;

$$d_i = \sqrt{(x_i - x_s)^2 + (y_i - y_s)^2 + (z_i - z_s)^2 + r_s^2} \tag{2.11}$$

- when the virtual source is lies outside the boundaries of the given speaker layout. In Figure 2.6, the sound source, S_o, is projected onto the boundary of the convex hull of the loudspeaker layout, such that the virtual source, S_p, lies on the projection onto the boundary at the point with the shortest distance, d, to the source.

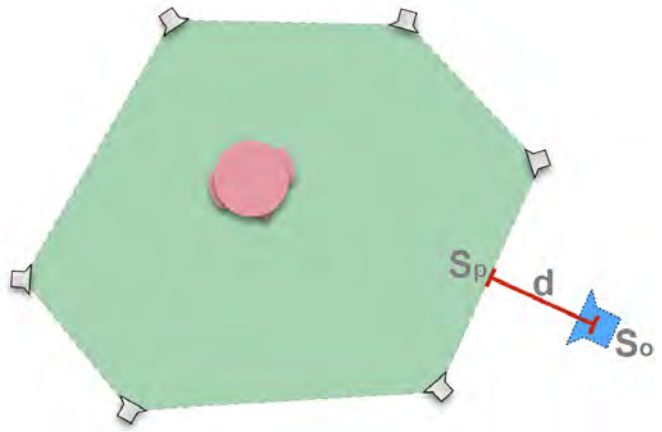


Figure 2.6: Virtual source projected on the convex hull of the loudspeaker (Tarzan et al., 2017)

The need to accurately determine the speaker locations therefore becomes crucial while using this algorithm, especially as they are the only factor used by this algorithm since no listener position nor speaker layout are required.

Binaural Rendering

Binaural rendering is another technique used in delivering immersive sound. This technique explores the human sound perception due to the influence of the human head shape and size, and the positions of the ears on the head. The technique involves recording of two distinct sound signals - one per human ear - which are meant for playback via equalised headphones (Breebaart et al., 2006; Møller, 1992). Each signal is recorded using information only related to that particular ear, relative to a specific point of sound source and a specific point of recording (Cheng & Wakefield, 1999). This is achieved through the use of HRTFs (Breebaart et al., 2006). HRTFs use several characteristics

of the 3D sound space to capture the sound signal. The cues include the ILDs and the ITDs. Known as the duplex theory, the ILDs are sensitive at the high frequencies (over 1.5kHz), while the ITDs are more sensitive below the 1.5kHz threshold, though still useful in higher frequencies (Möller & Raake, 2014). These help localise sound with respect to direction and distance.

A human or mannequin head is used for recording sound in a free field or loudspeakers. The head is attached with probe microphones at the eardrum or in the ear canal, with the intention of capturing sound as it arrives to the human ear from all directions as they need not obscure sound pressure within the human hearing system (the ear canal, eardrum, pinna and near the head) (Blauert, 1983). The information preserved includes binaural spatial cues which contains details of the essence of the 3D environment.

The sound could be recorded from free field from a live or real event with direct sources such as musical instruments, or it could be generated synthetically through computer system for the purpose of binaurally rendering it. Production and reproduction from real events is referred to as binaural recording, whereas that from artificial events is known as binaural synthesis (Minnaar, Olesen, Christensen, & Møller, 2001). Headphones are typically used for playback as they deliver the signal directly into the ear, which avoids the possibility of crosstalk between the two signals and reflections from the surroundings (Möller & Raake, 2014). However, the playback can be transmitted through a set of stereo loudspeakers, where crosstalk cancelling techniques would need to be deployed.

The next section is a description of an immersive sound system which uses some of the spatialisation algorithms described above.

Immergo - an Example Immersive Sound System

The automatic speaker configuration described in this thesis was intended to be incorporated into an immersive sound system called Immergo (Foss & Rouget, 2015). Immergo is a client-server system used for control and rendering of immersive sound over an Ethernet AVB network. Audio transmission is done by performing mixing of audio on an external device. The audio mixing could be done by the remote speakers such as the SPK-4Ps loudspeakers (miniDSP, 2020), or the UltraLite AVB interface (MOTU, 2019). The UltraLite AVB interface is used in the case of this project research[2].

[2]The SPK-4Ps loudspeakers were not available for continuous use.

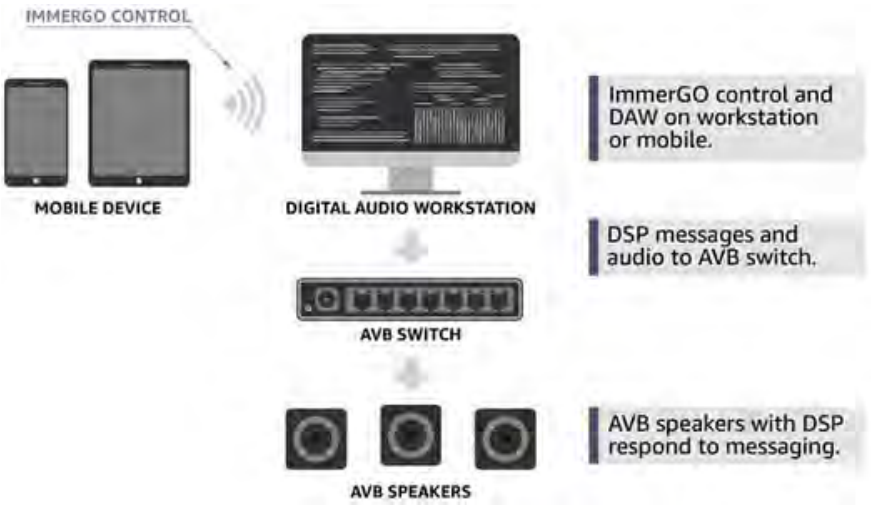


Figure 2.7: Immergo configuration (immersiveDSP, 2019)

The configuration for Immergo is shown in Figure 2.7, while that of the proposed system is shown in Figure 3.6. The Immergo configuration consists of a mobile device communicating wirelessly with a computer workstation connected to an Ethernet AVB switch. The Ethernet AVB switch further connects to various AVB endpoints. Digital signal processing and Ethernet AVB messages exchange occurs between the Ethernet AVB and the workstation. Loudspeakers are connected to the Ethernet AVB endpoints and are extensible to 90 loudspeakers. The 3D coordinates for these loudspeakers are used to enable the system to spatialise sound appropriately.

The server, built on the Node.js[3] cross-platform framework for web applications, delivers control code to the client over a web browser running on a mobile device upon the client request to the Immergo server. Spatialisation algorithms such as VBAP are used by the system to spatialise sound in the 3D space. These algorithms require the knowledge of loudspeaker coordinates to accurately determine the mix levels for the loudspeakers to pan sound in space. These levels are then sent across to the Ethernet AVB network to perform the desired mixing based on the loudspeaker configuration.

The loudspeaker locations information is stored in an XML file. The XML configuration contains information about the various MAC addresses of the endpoint devices participating in the Ethernet AVB network, together with each speaker's 3D coordinates namely the x-, y- and z-coordinates. This information is used, together with the track information, to set the appropriate mix levels. These coordinates need to be measured and entered

[3] `https://nodejs.org/`

into this XML file prior to executing the Immergo system. This research project worked on automating the measurement process involved for this step.

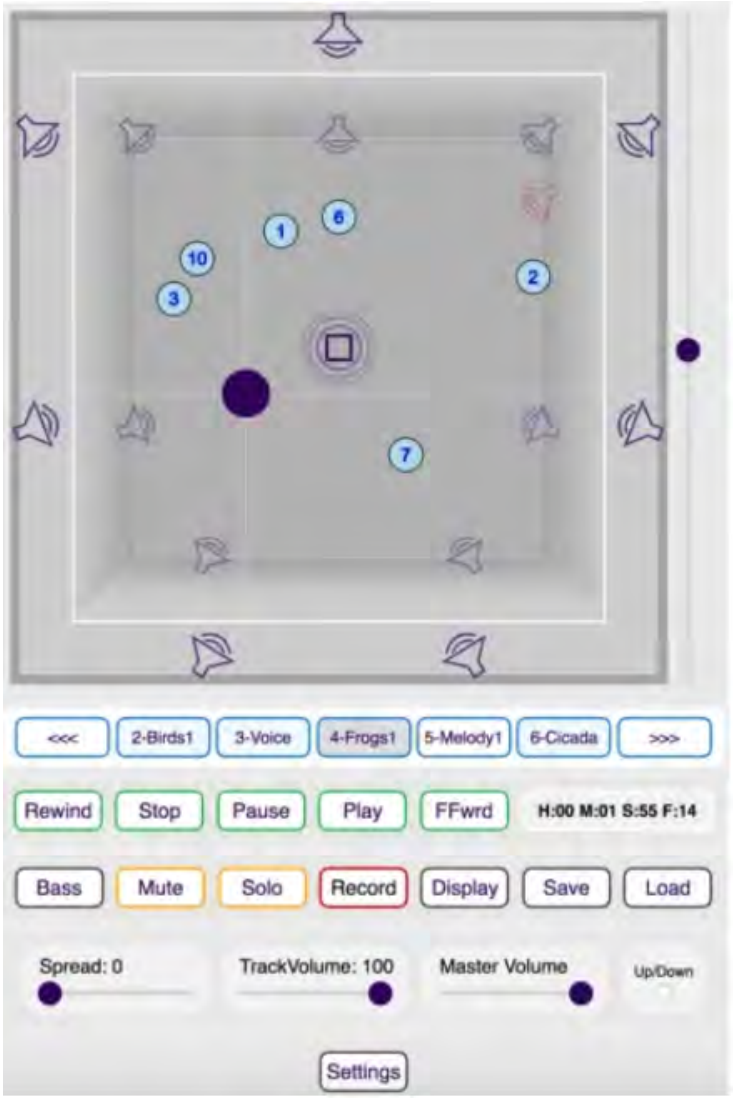


Figure 2.8: Immergo client UI (immersiveDSP, 2019)

OBA mixing control is performed via the client on the mobile device (or a PC) with a user-friendly control interface. Figure 2.8 shows a client user interface for the system. A user performs the following tasks via the client UI:

- visualise and interact with audio tracks (shown as the numbered circles in Figure 2.8) and speaker objects (shown by the typical speaker icons for speakers on the sides of the rooms, while those on the ceiling are shown by the squares surround by the three circles) within the room via the mobile device's tap and touch gestures. The various tracks are shown at various height levels within the room;

- track panning via touch and/or tilt features of the mobile device. This is done by moving the big dot within the room boundaries;
- audio track selection. This is done by tapping a combination of the tracks shown below the room layout;
- local or remote digital audio workstation (DAW) audio transport control. These include features such as loading, displaying the selected tracks and muting of tracks;
- recording, saving and playback of one or many audio tracks can also be performed.

2.2.2 Towards a Common Immersive Audio Format

Collaborative efforts aimed at the standardisation of the processes and outputs of the various steps of the immersive audio flow, that is, from production, through delivery chain to reproduction, have been made by organisations such as the Audio Engineering Society[4] (AES), International Telecommunications Union[5] (ITU), Society for Motion Picture and Television Engineers[6] (SMPTE), the Moving Picture Experts Group[7] (MPEG) and European Broadcasting Union[8] (EBU), together with vendors in the sound industry. This is aimed at improving interoperability of the exchange audio material along the production chain and minimising market fragmentation and confusion to consumers (Marston, 2013).

The various specification and standard documents from these organisations are:

- Recommendation ITU-R BS.2076-2 (ITU-R, 2013): ITU, in collaboration with EBU, has a recommendation specification for an audio definition model (ADM) that uses metadata with a full description of the audio. The model metadata is initially specified in XML but could be specified in other languages like JSON as well. It distinguishes between the audio format and content, as a central requirement for audio distribution through any platform. It is expected to be the backbone for the next generation audio streams.
- Recommendation ITU-R BS.2127-0 (ITU-R, 2019): The ADM renderer specification is also published by ITU. This gives a description of the rendering processes of the ITU ADM.
- ST 2098-1 (SMPTE, 2018b): This is similar to the ITU ADM metadata specification and is published by SMPTE for metadata for use in immersive sound for cinema.

[4] http://www.aes.org/
[5] https://www.itu.int/
[6] https://www.smpte.org/
[7] https://mpeg.chiariglione.org/
[8] https://www.ebu.ch/home

- ST 2098-2 (SMPTE, 2018a): The specification defines an encoding that describes a sound scene and the associated metadata for immersive sound reproduction.
- MPEG-H 3D Audio (Herre, Hilpert, Kuntz, & Plogsties, 2015): This is an immersive sound standard published by MPEG, meant for use with OBA, CBA and SBA for audio production and reproduction on various loudspeaker configurations.

Proprietary immersive sound formats include the Dolby Atmos (Dolby Laboratories, 2012), DTS:X (Digital Theater Systems, 2016), Auro-3D (Auro Technologies, 2015) and AuroMax (Barco, 2015). The first two formats are OBA formats, while the last two are CBA. The formats are available for home and cinema installations. Rendering immersive audio depends on the receiver being able to decode the 3D audio content, such as any of these formats. Also, a 3D loudspeaker configuration must be available.

2.3 Loudspeaker Layouts for Playback

Loudspeakers are the final devices in the audio chain. They are used in all sound systems - from surround sound stereo systems, surround and immersive sound home theatre installations to professional and cinematic immersive sound systems. They convert an audio signal in the form of electrical energy to acoustical energy into the transmitting medium such as air (Blauert, 1983). To deliver a fulfilling sound experience, loudspeaker placement needs to match the intent of an artist. A knowledge of a sound scene demands that the loudspeakers be placed such that an audience can enjoy it as created. Different layouts deliver different listening experiences. The layouts range from just one loudspeaker, through stereo, to surround and currently to the immersive layout types.

For improved sound quality, a distinct channel dedicated for delivery of limited-bandwidth low-frequency-extension (LFE) and bass signals exists (Holman, 2012; ITU-R, 1994). The upper limit of the LFE channel is recommended to be up to 120Hz (Rumsey et al., 2001). This limit forms the lower limit for the mid to high frequency channels. The LFE channel is typically played by low-frequency loudspeakers called subwoofers, which could be placed anywhere in the listening environment since the LFE channel does not contain any directionality. This is due to the long wavelengths formed from the low frequencies, which diffract more efficiently around various obstacles typically present in the home theatre environments. The mid to high frequency channels are routed to the loudspeakers called main or satellite speakers. These loudspeakers, unlike the subwoofers, have specific location recommendations which require a clear line of sight within the speaker layout,

especially for CBA, whereas the SBA and OBA contents can adapt to available speaker configuration geometry.

2.3.1 The Sweet Spot

Fundamental to the delivery of a fulfilling listener experience is the concept of a sweet spot, also known as a reference listening point (ITU-R, 1994). It is a location in the sound field where an envisaged listener (or audience) would best sound experience as intended by an artist. An ideal area for the sweet spot of a stereo is shown in Figure 2.9, with the listener head, H, and left (L), right (R) and low-frequency (LFE) loudspeakers. The listener ear level at a seated position is considered a reference position and is taken to be approximately at height 1.2m from the floor. The left and right loudspeakers form an equilateral triangle with the reference position. The sweet spot is a center of the circle whose circumference touches the left and right loudspeakers and has the centre H, with the loudspeakers lying at equal angles (recommended to approximately be 30°) from either side of the reference position as shown in the figure. Various spatialisation algorithms yield different reference listening points (and areas) in terms of size and extensibility to suit listening environments as explained in the algorithms' respective descriptions in Section 2.2.1. For instance, the limited sweet spot provided by Ambisonics is extensible by HOA, while that provided by VBAP is limited and can be extended by MDAP.

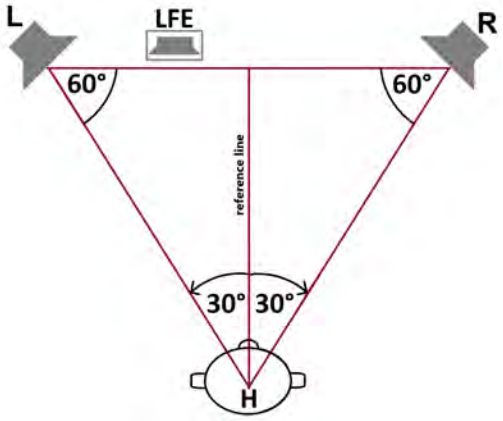


Figure 2.9: Sweet spot

2.3.2 Content to Speaker Configuration

Up-mixing and Down-mixing for CBA

Typically, for CBA, the number of loudspeakers corresponds to the number of channels available. However, this is not always the case. When there is a mismatch between the two numbers, up- or down-mixing usually occurs based on the mixing equations to provide upward or downward compatibility respectively (ITU-R, 1994). The two mixings respectively refer to a conversion of an audio stream from a smaller to higher number of channels and vice versa (Adenot & Toy, 2018). For example, if an audio stream has only one channel, and it is played out to a stereo system, the one channel will be up-mixed to a stereo output and played out at each of the two available satellite loudspeakers. The one channel input will be converted to the two channels of the stereo output system. The left and right channels are configured such that the sound image is perceived to emanate at the centre, that is, midway between the left and right speakers. Also, the sound output is attenuated by half the input power at each speaker to maintain a combined result of the original power output to avoid distorting the original sound image.

Seamless Rendering for OBA and SBA

The remixing of audio to various configurations may sacrifice the content creator's intent as some signals are combined or lost, possibly resulting in loss of fidelity in the reproduced sound. Additionally, to render the CBA content, loudspeaker setups require regular geometry from the perspective of distances and angles amongst the speaker for accurate sound reproduction (Tarzan et al., 2017). OBA avoids this problem as it encodes audio in objects, with metadata associated to the objects. This metadata helps seamlessly render the audio content to suit the available speaker layout, thereby preserving the content creator's intent. SBA also avoids the sound fidelity loss problem associated with CBA as the encoded audio signals are decoded to the available speaker layout.

2.3.3 Standardisation of Speaker Configurations

As with the immersive audio format standardisation, the international organisations and vendors working in or closely related to the audio industry make agreements and recommendations on the configurations they deem appropriate for optimal sound experience.

An audio guiding body, AES, together with a telecommunications body, ITU, make recommendations and/or specifications on the placement of the loudspeakers for optimal listening experience. Such recommendations for 5.1 systems are described in AES Technical Document AESTD1001 (Rumsey et al., 2001) and ITU Recommendation ITU-R BS. 775 (ITU-R, 1994). The recommendations are however extensible to larger speaker systems such as the 7.1 and 9.1 configurations. A reference model is recommended by the two documents. The model is a 5.1 multichannel surround system with an option for an LFE channel and motion picture. The recommended optimal listening experience, however, is not always possible to achieve in other loudspeaker layouts due to room physical constraints, hence the sweet spot is not always achievable.

2.3.4 Surround Sound Types

As mentioned in the introductory section of this chapter, surround sound forms the basis for immersive sound. As such, the speaker layouts for 2D sound form the basis for those of the 3D sound. For this reason, the horizontal plane layouts are briefly discussed in this section as a foundation for the 3D spatial sound layouts. Surround sound layouts are typically denoted by two numbers separated by a point '.', for instance, 2.1 or 5.1. The first number, for instance, the '2' in a 2.1 sound system, specifies the number of main loudspeakers and the '.1' specifies the number of subwoofers. Typical configurations range from 2.0 up to 11.2 layouts. Layouts with side or rear loudspeakers are sometimes denoted as X/Y/Z, where X, Y and Z represent the numbers of front, side or rear and LFE loudspeakers respectively. For example, the 4.1 can be denoted as 2/2/1 where the first 2 represents two front speakers, the second 2 represents two side speakers and the 1 represents the subwoofer. This level of loudspeakers at the listener's head is called the surround or base layer.

A 5.1 reference loudspeaker configuration from Recommendation ITU-R BS. 775 (ITU-R, 1994) is shown in Figure 2.10, with an additional loudspeaker labelled S. It is a useful reference layout and consequently used in the descriptions of the rest of the loudspeaker configurations.

The 2.0 and 2.1 layouts have the left and right loudspeakers fed by the left and right channels of a stereo content respectively. Headphones or desktop PC loudspeakers are normally designed for these stereo systems. The 2.0 layouts have no subwoofer due to the absence of the LFE channel, while the 2.1 speaker configurations have it. The listening position is crucial as it influences the sound quality that reaches the listening. For the

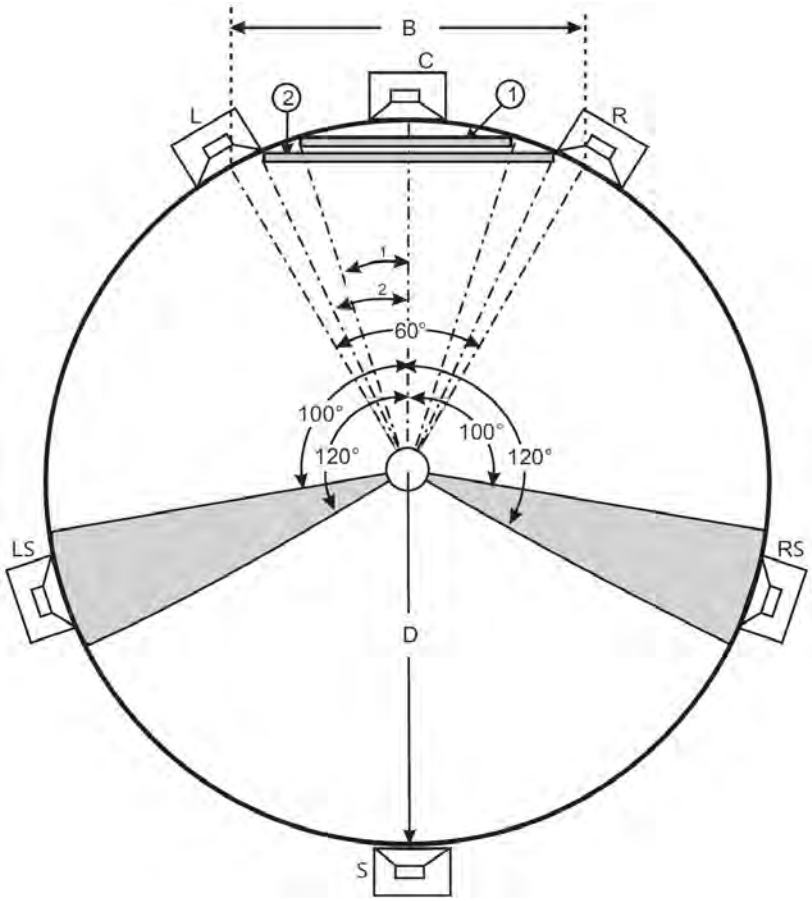


Figure 2.10: Reference loudspeaker layout adapted from ITU-R (1994)

stereo configurations, the listening position is recommended as shown in Figure 2.9, at roughly 30° azimuth from the reference line (Rumsey et al., 2001), and 0° elevation. The horizontal angle from the reference line is referred to as the azimuth angle and ranges from 0° at the reference position clockwise to 180° directly behind the listener (Kostadinov et al., 2010), while the vertical angle ranges from 0° at the head level, and 90° directly above the head, and -90° directly opposite the head. The 2.0 surround sound layout is represented by L and R loudspeakers. The listener position is at the centre of the circle shown in Figure 2.10.

The common 5.1 configuration is the basis for the ITU Recommendation ITU-R BS. 775 (ITU-R, 1994) mentioned previously. It has three front channels and two side channels. In Figure 2.10, the setup of a 5.1 configuration is represented loudspeakers labelled by:

1. C - the center loudspeaker, located at the front and meant for the centre channel,
2. L - the front left loudspeaker for the left channel,
3. R - the front right loudspeaker for the right channel,

4. LS - the left surround loudspeaker, located at the side of the listener and meant for the left surround channel, and
5. RS - the right surround loudspeaker, located at the side of the listener and meant for the right surround channel

The 5.1 layout became a popular setup for surround sound and many players in the audio industry created various audio formats dedicated to the 5.1 surround sound. Examples of the 5.1 surround sound audio formats include Dolby Digital, Pro Logic and Digital Surround (Andersen et al., 2004; Digital Theater Systems, 2018; Dolby Laboratories, 2003c).

The 7.1 layout is an evolution of the 5.1 layout, with two additional loudspeakers placed at the back. The range of the horizontal angle from the listener to the rear loudspeaker is from 135° to 150° (Dolby Laboratories, 2018b). The configuration of the 7.1[9] surround sound loudspeaker placement is shown in Figure 2.11 (a). Other variants of the 7.1 configuration exist. For instance, a 7.2 configuration has two subwoofers instead of one - each for the left and right sides of the loudspeaker layout. Each of the two subwoofers, however, plays out the same LFE channel since the audio format creators normally create one LFE channel.

The 9.1 configuration evolved from the 7.1 layout, with two additional loudspeakers placed between the front and surround loudspeakers. The additional loudspeakers' azimuth varies from 50° to 70° and can also be inclined vertically within the 15° from the relative position. These loudspeakers are sometimes called front wide loudspeakers, hence the front wide left (FWL) and front wide right (FWR) loudspeakers. Figure 2.11 (b) shows a 9.1[10] surround sound configuration.

Other configurations exists but they never gained popularity, for instance, the 3.1, 4.1 and 6.1 configurations. The layouts from 2.0 through 9.2, up to 13.2, fall in the surround sound category. This is because of their ability to reproduce sound only in the horizontal plane relative to the listener position. All loudspeakers are expected to be at the same height or slight angle inclinations to the listener's ear, though room conditions may not always permit the recommended loudspeaker locations. The layouts are dominant in the home theatre entertainment systems and consequently form the large portion of the consumer systems. Table 2.1 summarises the combined recommendations of the surround sound loudspeakers placement.

[9] 7.1 Speaker Placement - `https://www.dolby.com/us/en/guide/surround-sound-speaker-setup/7-1-setup.html`

[10] 9.1 Speaker Placement - `https://www.dolby.com/us/en/guide/surround-sound-speaker-setup/9-1-setup.html`

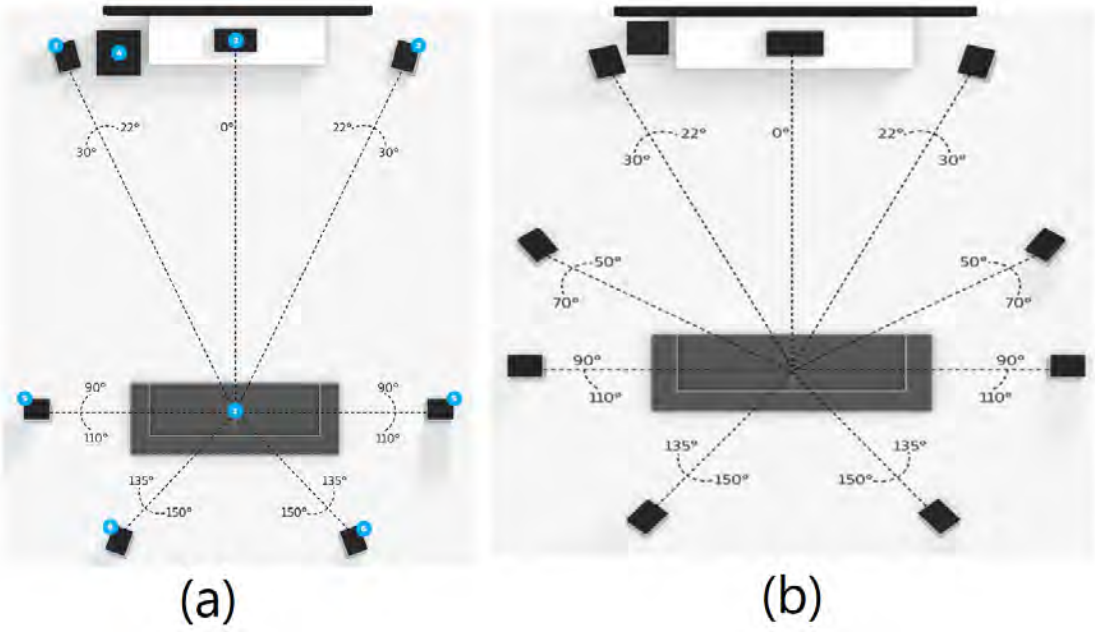


Figure 2.11: Dolby surround layouts: (a) 7.1 (Dolby Laboratories, 2003a) (b) 9.2 (Dolby Laboratories, 2003b)

Table 2.1: Loudspeaker configuration specification

loudspeaker	Horizontal angle	Vertical angle	Height (m)
C	0	0	1.2
L, R	22 - 30	0	1.2
LS, RS	100 - 120	0 - 15 down	$\geq$ 1.2
LB, RB	135 - 150	0 - 15 down	$\geq$ 1.2
S	180	0 - 15 down	$\geq$ 1.2
FWL, FWR	50 - 70	0 - 15 down	$\geq$ 1.2

2.3.5 Immersive Sound Types

There are at least two popular loudspeaker configurations for home installations within the immersive sound field. These are defined for Dolby Atmos (Dolby Laboratories, 2012) and Auro-3D (Auro Technologies, 2019) immersive sound formats. Other immersive sound formats use configurations based on these two configurations. For instance, DTS:X uses layouts similar to those of Dolby Atmos (Digital Theater Systems, 2019; Sasidharan, 2018). Dolby Atmos configurations use the X.Y.Z nomenclature. This notation builds on top of the surround sound standard notation (such as 5.1 and 7.1), with the third number added at the end to introduce the height dimension (Dolby Laboratories, 2018b). For instance, in a 5.1.2 immersive sound speaker configuration, the 5.1 is the standard 5.1 surround sound configuration, while the '.2' is the number of speakers for the height

dimension. The Auro-3D configurations use a notation similar to the surround sound speaker configurations in that it only uses two numbers, as X.Y. However, with this notation, the total number of the surround sound and height speakers is specified by the X, while the Y specifies the number of the LFE speakers. With this nomenclature, it is not readily known how the surround sound and height speakers are spread within the configuration, as it is with the X.Y.Z notation.

2.3.6 Dolby Atmos Configurations

The height dimension in these configurations augments the surround sound configurations to provide the overhead sound effects. The height dimension provision is done at two levels, namely the top and height speaker levels.

The Top Speaker Level

This is the level which delivers sound emanating from the top - at the ceiling. The azimuth angle must be 90° for the middle top overhead speakers. Audio encoded for height is sent to these speakers during playback. This level has two strategies for achieving overhead sound.

1. Overhead speakers: these are speakers mounted directly at the ceiling. They direct sound downwards towards the listener. Figure 2.12 (a) shows a 5.1.2 configuration which is made up of:
   - the standard 5.1 surround sound layout defined in Section 2.3.4, and
   - the left and right middle overhead speakers to complete the immersive sound field.
2. Dolby Atmos enabled speakers (Dolby Laboratories, 2016): these speakers direct sound to the ceiling and have it bounce off the ceiling downwards to the listener as though it comes from above. They are placed at the surround sound layer. Two types of speakers exist for this option, namely the add-on speaker modules and the integrated Dolby Atmos enabled speakers. The add-on speaker modules are physically placed on top of the front left and right speakers to radiate sound towards the ceiling, while the integrated Dolby Atmos enabled speakers have both the standard (left and front) front speakers and include the up-firing speakers within one cabinet. Optimal ceiling requirements are needed to yield optimal sound quality, and this may not always be the case. Figure 2.12 (b) shows a 5.1.2 configuration which consists of:

- the standard 5.1 surround sound layout, and
- the integrated Dolby Atmos enabled speakers placed at the front (both left and right).

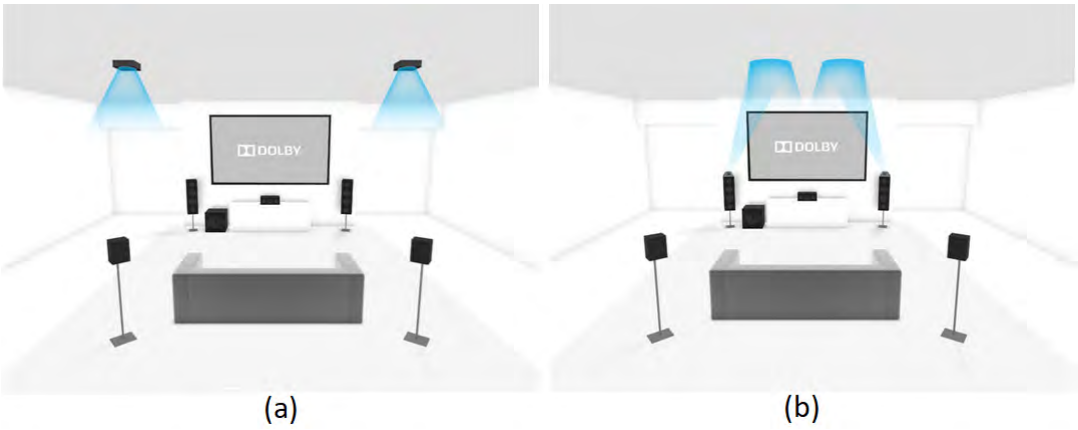


Figure 2.12: Dolby Atmos 5.1.2 loudspeaker layouts (Dolby Laboratories, 2018b)

Depending on various circumstances, hybrid setups can be achieved. This is when both the overhead and the Dolby Atmos enabled speakers are used to deliver the overhead sound. This occurs in configurations with at least four speakers for the height dimension, for instance in a 5.1.4 or 7.1.6 configuration. This is because similar speaker types are recommended to be placed in pairs, that is, either use left and right overhead speakers or left and right Dolby Atmos enabled speakers, not a mix of these types for respective left and right speakers. A hybrid configuration is shown in Figure 2.13 (a). The configuration consists of:

- the standard 7.1 surround sound configuration,
- six speakers in the height dimension, consisting of:
  - four integrated Dolby Atmos enabled speakers - the left and right Dolby Atmos enabled speakers for the front and rear speaker positions, and
  - two top speakers (left and right top middle overhead speakers)

The Height Speaker Level

This is a level of loudspeakers between the top and surround layers. This is found in configurations with at least six speakers for the height dimension, such as the 7.1.6 and 11.1.8. The height loudspeakers elevation angle range from 30° to 45° with respect to the listener position. Figure 2.13 (b) shows the Dolby Atmos 7.1.6 layout. This consists of:

- the standard 7.1 surround sound configuration,

- four height speakers (left and right height front speakers and left and right rear height speakers), and
- two top speakers (left and right top middle overhead speakers).

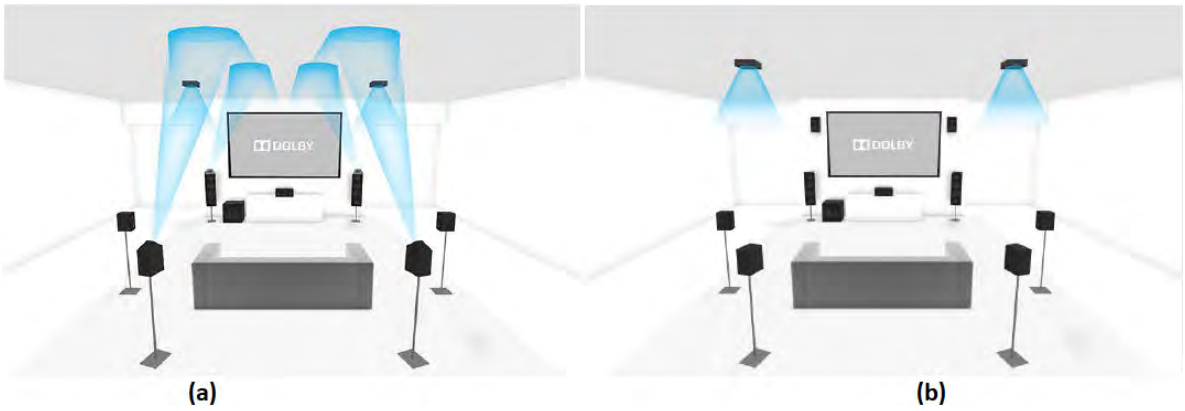


Figure 2.13: Dolby Atmos 7.1.6 loudspeaker layouts (Dolby Laboratories, 2018b)

Dolby Atmos creates only one LFE signal, as such, its layouts are in the form X.1.Z. Standard Dolby Atmos configurations start from 2.1.2 through to 11.1.8 for home installations. These layouts can be extended to 34 loudspeakers, for a 24.1.10 layout. The speaker positioning within these configurations follow the ITU Recommendation ITU-R BS. 775ITU-R (1994). Additional recommendations are made for positions which do not appear in the ITU Recommendation by the vendor.

2.3.7 Auro-3D and AuroMax Configurations

As previously mentioned, Auro-3D and AuroMax are channel-based immersive audio formats. Configurations for these formats use three layers to deliver immersive sound. These are the surround, the height and the top layers. The surround layer is the standard surround sound configuration, that is, a layer at the listener ear level. The height layer places speakers around the listener often in a layout similar to that of the surround layer, but at some height above the surround layer with an elevation of 30° to the listener position. The top layer provides the height dimension in sound directly from above the listener, at 90°. Figure 2.14 shows the three-layer concept for Auro-3D speaker configurations.

Detailed speaker positioning is based on the ITU Recommendation ITU-R BS. 775ITU-R (1994). The Auro-3D configurations start from 8.0 setups through to 13.1 for home installations, with common configurations being the 9.1, 10.1, 11.1 and 13.1. AuroMax

layouts for large environments include the 20.1, 22.1 and 26.1 configurations. These are compatible with the 11.1, 7.1 and 5.1 Auro-3D layouts.

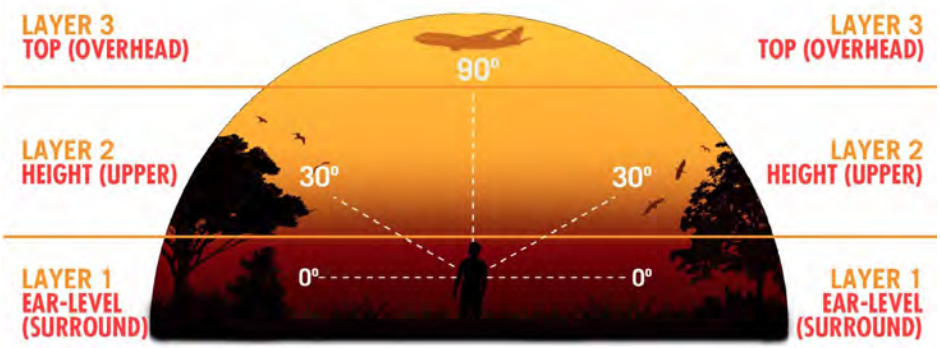


Figure 2.14: Auro-3D's unique three-layer concept (Auro Technologies, 2015)

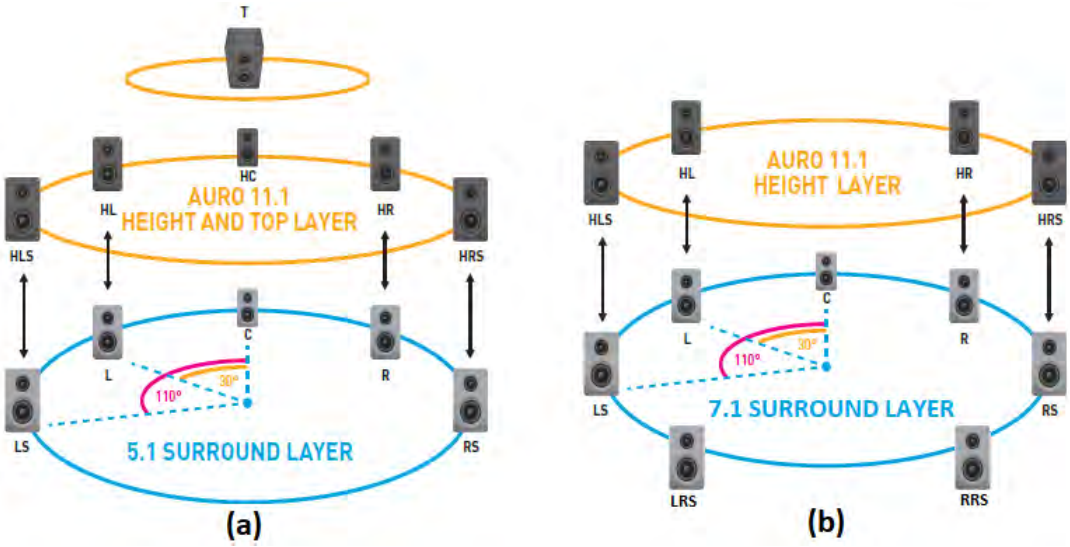


Figure 2.15: Auro-3D 11.1 loudspeaker layouts (Auro Technologies, 2015)

Figure 2.15 shows two 11.1 configurations. Figure 2.15 (a) is a three-layered configuration which consists of:

1. Layer 1: the surround sound layer which is the 5.1 surround sound layout
2. Layer 2: the height layer consisting of the speakers placed in a 5.1 surround sound layout such that each loudspeaker is directly above a corresponding speaker of layer 1, and
3. Layer 3: the top layer made up of just one top loudspeaker, also known as the Voice-Of-God.

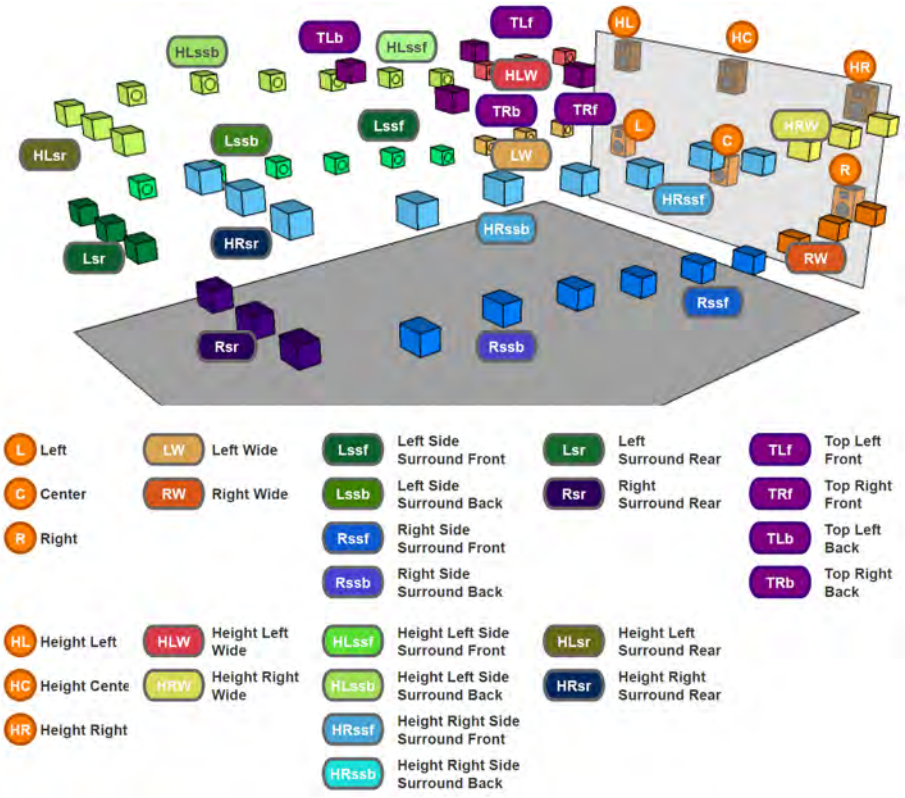


Figure 2.16: AuroMax 26.1 layout (Barco, 2015)

Figure 2.15 (b) is a two-layered configuration consisting of:

1. Layer 1: the surround layer made up of the standard 7.1 surround sound configuration, and
2. Layer 2: the height layer consisting of four loudspeakers.

The 26.1 AuroMax cinema layout is shown in Figure 2.16. This uses smaller groups of loudspeakers, called zones, to divide the large environments such that these zones are compatible with the 11.1 configurations.

The combined synergy of spatialisation algorithms, audio formats and loudspeaker configurations provides a means for 3D audio content production by content creators and reproduction by consumers. The algorithms use the knowledge of loudspeaker coordinates to accurately spatialise sound in the 3D space. For Immergo, the client-specified loudspeaker coordinates are entered into an XML file. They are hard coded into the system for localisation algorithms to consume. When the loudspeaker layout changes, which is common, the hard-coded coordinates accordingly must be re-measured and re-coded into the file. This follows a process of measuring the coordinates, which tends to be tedious manual labour. The procedure takes a long time, and often, accuracy is compromised. A

less tedious and less time-consuming alternative is therefore proposed and introduced in the next section.

2.4 The Proposed Solution

A semi-automated solution termed SDIAS[11] is proposed for loudspeaker configuration. This deals with speaker configurations for immersive audio content creation systems. The solution system uses a mobile device as a client and a PC as a server. The server connects to an Ethernet AVB network, which in turn connects to loudspeakers to deliver impulses at equal intervals clocked at sample-accurate granularity. Most tests for SDIAS were performed on a 6.0.2 immersive sound loudspeaker configuration. The six loudspeakers were created from a standard 5.1 surround sound configuration, without any LFE channel. The setup additionally had a rear centre loudspeaker and the '.2' were left and right front height loudspeakers. The transmitted impulses are received by a client-attached microphone. Some manual measurements are still required, hence the semi-automated solution. This is due to some initialisation and configuration parameters which require human interaction.

The automated part comprises the computation of the loudspeaker coordinates once the distances have been extracted from the audio. The manual phase involves measuring the distances for placement of the microphone, and a distance that will be used as a calibration distance, which is measured from a selected loudspeaker - the calibration loudspeaker. Automation is envisaged to increase accuracy and reduce the time it takes for determination of loudspeaker coordinates in comparison to an entirely manual process.

All calculations involved in determining the 3D speaker coordinates follow a step-by-step geometric algorithm which relies on a specific alignment of the microphone locations and the loudspeakers as sound sources. The algorithm is expressed using mathematics.

2.5 Chapter Summary

This chapter set out to introduce the background to immersive sound. It explained immersive sound and its evolution from surround sound. It also introduced an immersive

[11] This is abbreviated from *Speaker Determination from an Immersive Audio System*

audio content creation system called Immergo. Some of Immergo's core features were described. These include the client-server architecture, its use of the Ethernet AVB standard for audio transport and control messages, the user-friendly UI which provides control using a mobile device, the use of spatialisation for creation of immersive sound content and finally the use an XML file, which contains 3D loudspeaker coordinates which are hard-coded into this file.

Common spatialisation algorithms were introduced, namely the VBAP, DBAP, WFS and Ambisonics. For each of these algorithms, a brief explanation was given. This included short discussions on how the encoding of audio is performed, which indicates the audio representations used by the algorithm, and therefore resulting in the algorithm as being channel-based, object-based or scene-based. The use of loudspeaker configurations and positions was also explained for each algorithm. That each algorithm uses speaker positions in some way pointed to the need for automating and expediting the determination of the speaker positions measurement process.

Various loudspeaker layouts based on immersive sound formats were described. These descriptions showed how three immersive sound formats, namely the Auro-3D, AuroMax and Dolby Atmos, have specified loudspeaker configurations for improved immersive sound delivery. Home installations reach up to 14 speakers (13.1 where the LFE channel is used for a full frequency range speaker) for Auro-3D and up to 35 speakers for the 24.1.10 Dolby Atmos, while for large environments, installations reach up to 64 speakers for Dolby Atmos in the professional cinema setups, while AuroMax format provides up to 27 loudspeakers.

Finally, SDIAS was proposed as a solution for automation of loudspeaker configuration. The next chapter gives a full account of the proposed solution.

Chapter 3

SDIAS – an Automatic Speaker Configuration System

SDIAS was built with the intention of providing an automated solution to the problem of speaker configuration. Immersive audio systems use speaker locations to localise sound sources. Speaker locations in the 3D space can be described using Cartesian or spherical coordinates. Immergo uses the 3D Cartesian coordinates - (x,y,z). SDIAS, like Immergo, is a client-server system architecture. The client and server components work hand in hand to enable the configuration process. There are other components which exist within SDIAS. These include the Jules Utility Class Extensions (JUCE) application, which is the tones server for the system, and the AVB network which is used for the delivery of timely audio streams as well as connection management and control. The user's primary point of interaction with SDIAS is through a mobile device, which also serves as a system controller. The client sends requests to the server, and the server fulfils these requests either by giving responses directly back to the client or through relaying requests to the JUCE application. The following sections give details of how these components interact, together with their interaction with the user.

3.1 Conceptualisation, Visualisation and Verification

A 3D physical model was built to aid visualisation of the sound source and microphone positions in the 3D space. The process also formed part of the design phase. This enabled visual confirmation that, given a set of three or four orthogonal microphone positions, it

would be possible to compute the 3D coordinates of the supposed sound source. Figure 3.1 shows the model. The figure shows the components of the 3D space from which the coordinates of the sound sources are to be computed. These features are:

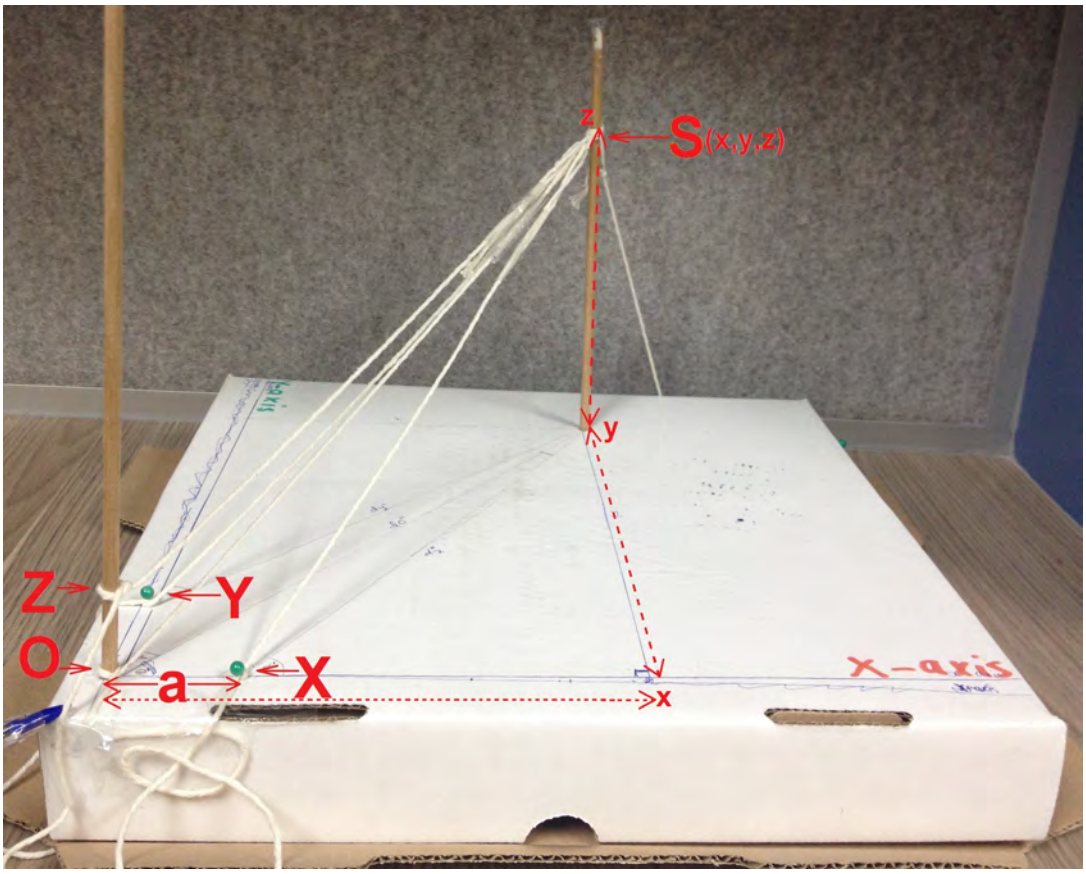


Figure 3.1: Physical model of the microphone positions and the speaker

- the 2D xy-plane: represented by x and y axes on the box,
- the z-axis: represented by the stick extending from the xy-plane upwards from the point labelled O, to give the third dimension of the 3D Cartesian coordinate system,
- the sound source: represented by the point at which all the strings meet, marked S. This is the speaker which sends the impulses for the microphone to record, on request by the user,
- the four microphone positions: shown by the green pin heads on the x and y axes, and as O and Z for the z-axis. These are the four points at which the microphone is placed to record the sound impulses. These are used later to obtain the distance of the speaker to the microphone at that particular position and ultimately the coordinates along the respective axes. The four distances were manually measured for use in the geometric algorithm which computes the coordinates[1],

[1]A complete JavaScript program which implements this algorithm is shown in Listing A.1

- microphone-to-speaker distances: represented by the strings extending from the four microphone positions, namely OS, XS, YS and ZS. The distances are obtained from the time conversions of the durations of the sound impulses from the speaker to the microphone points,
- the orthogonal distance: marked as 'a', represented by the distance from the origin O, to either of the three axes, for example, OX. The distance is equal from O to each of the X, Y and Z microphone positions. These distances were also measured manually for use in the computations of the coordinates, and lastly,
- the 3D coordinates: labelled as (x,y,z) next to the speaker S, represent the required speaker's coordinates. These are equal to the values marked with small letters x, y and z along the x-axis, y-axis and z-axis respectively. These distances were also measured manually, to compare against those that would be obtained from the algorithm that computes the coordinates using the microphone distances and the sound source.

The manually measured microphone positions and orthogonal distances were used as input to the code that computes the coordinates. The code uses trigonometric rules and functions to compute the coordinates. The results of the calculations were then compared with manual coordinates of the speaker S. The model, together with the results respectively provided a visual and computational assurance that the distances measured from the origin and three other orthogonal points on the axes would yield a 3D point from which sound emanates. The algorithm was programmatically tested with JavaScript in all the eight octants of the 3D space using the physical model as a reference.

A verification process was performed to investigate the accuracy and validity of calculations obtained from the model testing. The process was necessary since the calculations form a basis for the entire speaker position measurement process. The process sought to provide an assurance that given some orthogonal distance and the various microphone-to-speaker distances, the coordinates of the speaker can always be calculated for any point in the 3D space. The JavaScript program was therefore designed to verify the triangulation process of the geometric algorithm. A screenshot of the program output is shown in Figure 3.2. The program performs five iterations. For each iteration and each of the eight octants of the 3D space, the program:

- creates a random 3D point: this is the supposed 3D point of the sound source. This point is shown as S(x,y,z) in Figure 3.1. It is denoted by 'Old Point' in the program output shown in Figure 3.2. From the program output, the created point is (929,911,833). This 3D point lies in the first octant as denoted by all the positive

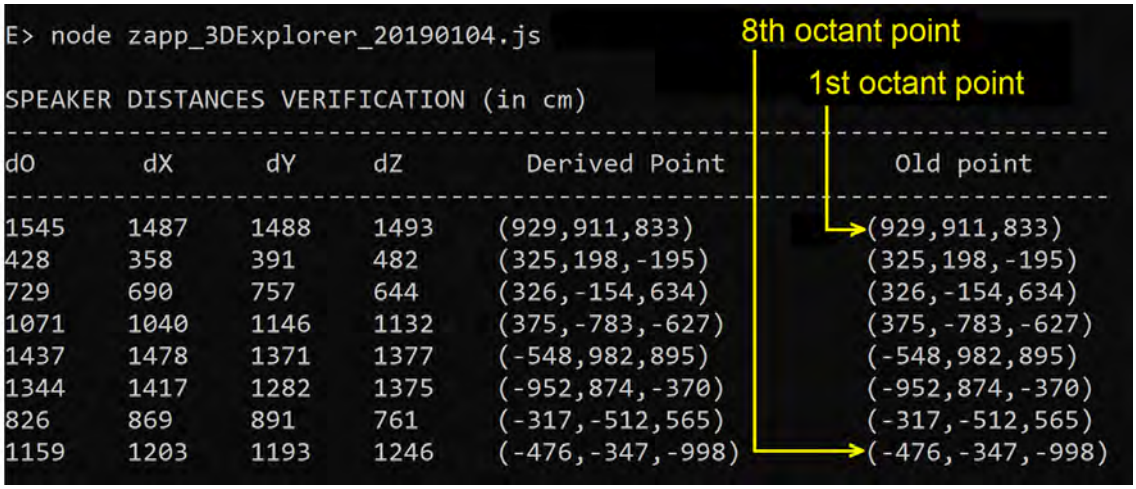


Figure 3.2: Triangulation verification

coordinates. A point that lies in the eighth octant would have each of its coordinates negative, for example (-476,-347,-998).

- using trigonometry, calculates the four microphone distances that would result in the created 3D point of the sound source. The calculations are based on the knowledge that there is always a reference point referred to as an origin, from which all coordinates are based. This point is also one of the four positions. It is where the distances are then measured to the speaker, for each of the four microphone-to-speaker positions. From Figure 3.1, the four distances are OS, XS, YS and ZS. The distances are shown in the output as dO, dX, dY and dZ, corresponding to each of the microphone positions at the origin O, along x-axis, y-axis and z-axis respectively. The distances are based on the orthogonal distances of 100cm, that is, each of the microphone positions is 100cm along its respective axis from O;
- additionally, the three angles formed by lines between each point on the axis to the speaker and the origin and the speaker are calculated. From the physical model shown in Figure 3.1, for instance, the angle θ_x along the x-axis is formed between the lines OX and OS. The angle is calculated using the distances of those two lines, together with that of the orthogonal distance, (a = 100cm). Equation 3.1 shows a formula which yields the value of this angle:

$$\theta_x = \cos^{-1}\left(\frac{a^2 + dO^2 - dX^2}{2 \times a \times dO}\right) \tag{3.1}$$

The y and z angles are similarly calculated for the y and z axes using respective distances. θ_y is obtained from equation 3.1 by replacing dX with dY, while θ_z is obtained by replacing dX with dZ. These computations form the core feature

necessary to calculate the coordinates;

- for each of the three axes, the origin distance, O, is used with an axis distance and angle to calculate a corresponding coordinate using the cosine law. Equation 3.2 shows the calculation of the x-coordinate using θ_x from equation 3.1:

$$x = dO \times cos\theta_x \tag{3.2}$$

  The Y and Z coordinates are similarly calculated by replacing θ_x in equation 3.2 with each axis' respective angle. These yield the 3D coordinate (x,y,z), which forms a new 3D point. These computations, too, form the core feature of this project. The new point is expected to be the same 3D point created in the first step. The new point is shown as the 'Derived Point' in the program output;
- a comparison between each of the new and old coordinates is performed and reported if mismatches are encountered.

3.2 The Geometric Algorithmic Mapping to SDIAS Processes

The feasibility study described in the previous section lays a foundation for a geometric algorithm to be used for coordinates calculations using trigonometry. The core function of SDIAS is to yield 3D coordinates for each loudspeaker in the immersive sound system. For each loudspeaker, S, using the physical model in Figure 3.1 as a reference, a set of five distances is required to yield the speaker's 3D coordinates (x,y,z). These distances are shown in the physical model as OS, XS, YS, ZS and OX. The physical space must be prepared to enable the measurement process to proceed. The microphone positions must be set prior to launching the SDIAS application. The user must ensure that the distances are accurately aligned and measured, as described in the following points:

3.2.1 The Orthogonal Distance

The distance, a (= OX), is measured manually and set at the beginning of the configuration process, during the *calibration* phase of SDIAS. It is set such that all the three distances from O to each microphone position along each axis are equal in length and orthogonal to each other. These conditions are mathematically expressed as:

1. a = OX = OY = OZ, and

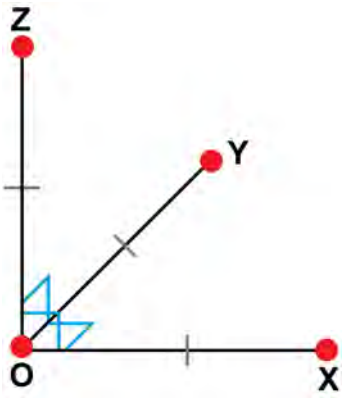


Figure 3.3: Microphone positions orthogonality

2. OX $\perp$ OY $\perp$ OZ (or equivalently, that angle XOY = XOZ = ZOY = 90°)

These conditions form a geometric shape with the distances and angles shown in Figure 3.3, which lays a requirement for accurate microphone positions alignment. Section 3.4.2 gives a description of how the conditions are achieved, together with additional details necessary for the speaker determination process to calculate the coordinates, while Section 4.6 further fully describes the rationale for these requirements.

3.2.2 Microphone-to-Speaker Distances

The distances OS, XS, YS and ZS are extracted from an audio data recorded from the impulses sent by the various loudspeakers, with a microphone at the various locations O, X, Y and Z. The various locations construct a geometric shape which is used to extract various geometric quantities such as the distances and angles, which are required to calculate the final coordinates for each sound source. The distances are used in calculating the angles which are used in calculating the coordinates. At the end of this process, the distances extracted form a geometric shape shown in Figure 3.4. This process occurs for each speaker, S at some random height h from the floor, in the immersive sound system, such that each of the distances from the microphone positions to the loudspeaker can be calculated. This step occurs at the *streaming* phase of the application. Section 3.4.3 explains how this process is performed, while the details of extraction of distances from the recorded audio data are given in Section 4.7.

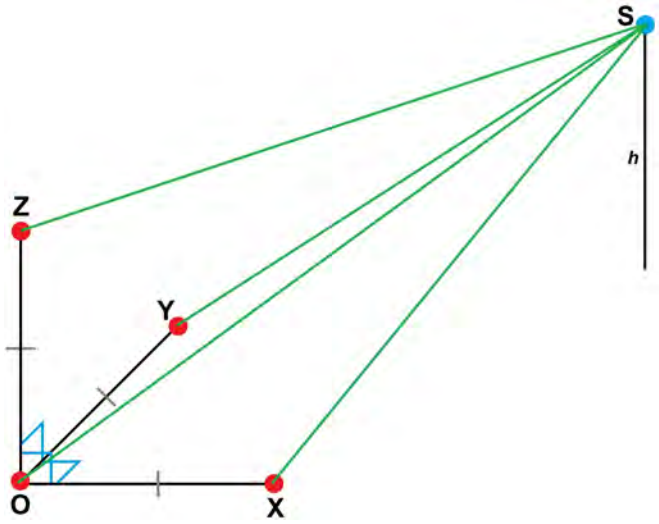


Figure 3.4: Microphone-to-speaker distances

3.2.3 Angles and Coordinates

The distances in the previous section are used to calculate the angles for each axis per speaker. Figure 3.5 shows the angle α along the x-axis, calculated from the triangle SOX since the lengths of all sides of this triangle are known after extracting the distances OS and XS in the previous step. This angle is also equal to angle TOX, where T is a projection of S onto the xy-plane. The angle is in turn used to calculate the x-coordinate of the corresponding speaker. The x-coordinate is given by the point x, where a perpendicular line to the x-axis, subtending from T, cuts the x-axis. This makes the angle OxT a right-angle as indicated in the figure. The angles for other axes are calculated and further used to calculate the axes' respective coordinates, similarly to the x-axis' angle and coordinate to yield the 3D coordinates (x,y,z) for speaker S.

Section 3.4.4 gives a description of the steps undertaken to achieve these calculations. A detailed, step-by-step and analytical description of the geometric algorithm used in calculating the angles and ultimately the 3D coordinates is given in Section 4.8. These calculations occur at the *configuration* phase of the SDIAS application.

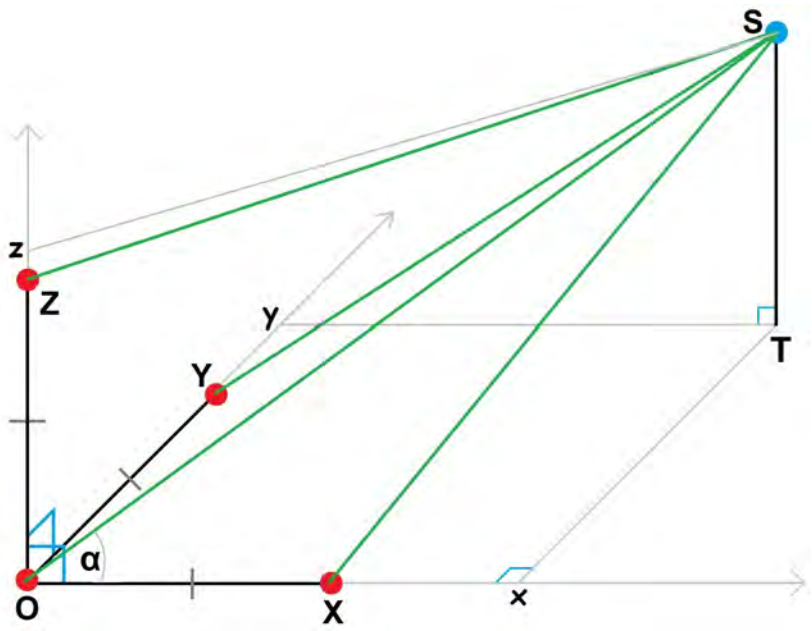


Figure 3.5: Angles and coordinates

3.3 Client-server Architecture

The Immergo system has a client-server architecture as described in Chapter 2. An assumption is that a user will control the system using a mobile device, which has a built-in or attached microphone. The capability of the attached microphone, together with the ability to place the mobile device at different positions were used to create an automatic speaker configuration system. An overview of the configuration system is shown in Figure 3.6. The diagram shows the components that make up the SDIAS system, namely a mobile device with an attached microphone, runs a client web application; and a host PC running a web server which attaches to an AVB network and loudspeakers. The following sections describe these system components in detail.

3.3.1 The Client

The client is a web application hosted on browser running on a mobile device. The mobile device could be a smartphone or tablet. The web application is the client component of the system and it functions as the primary controller. It is also the main component for user interaction. Communication between the mobile device and the Windows PC hosting the server application is done via wireless connection. The client and server applications

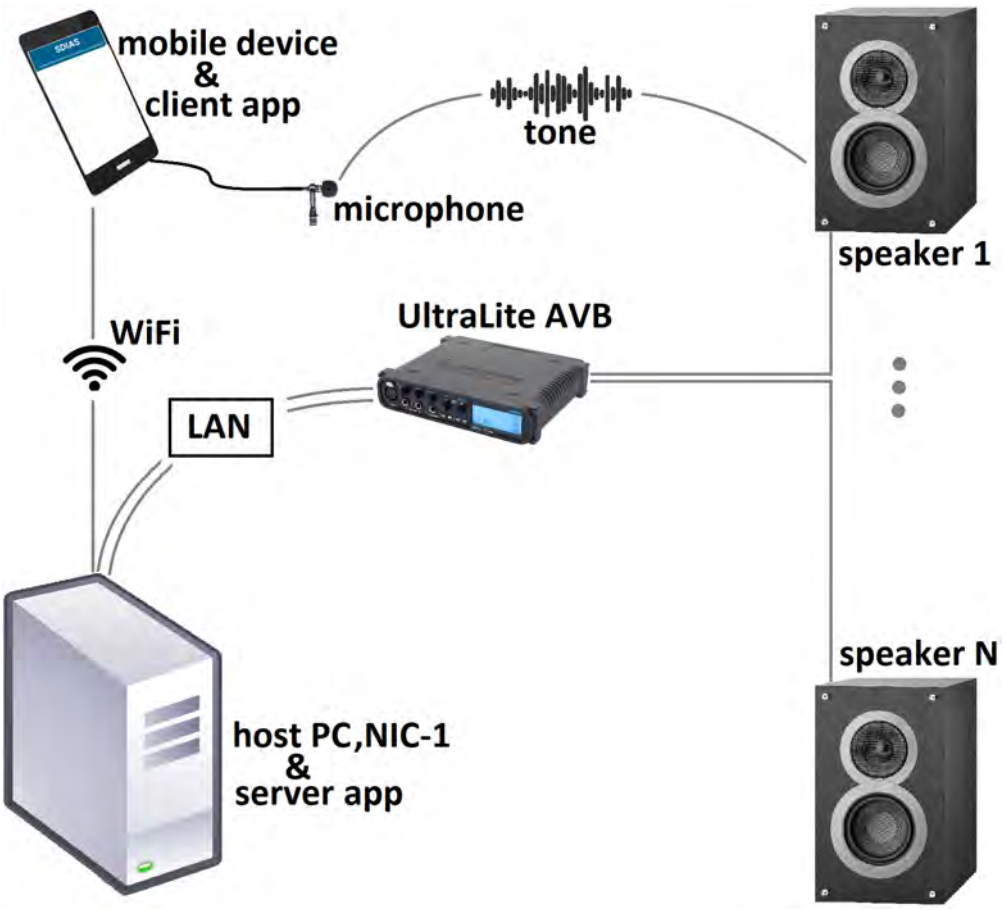


Figure 3.6: System diagram

communicate via web sockets. The client places requests for control and receives responses from the server. The microphone attached to the mobile device is used for recording audio impulses from the loudspeakers. These impulses are generated by the tones server and sent to the loudspeakers via the AVB network in a timeous manner.

3.3.2 The Server

The server is a Node.js server that responds to requests made by the client. It performs most of the computations, such as extraction of the tones from the audio data sent by the client and conversion of each tone's sample position to the time of flight (TOF) each tone takes to reach the microphone from the loudspeaker. It converts these TOFs into speaker distances and computes speaker coordinates from these distances. Additionally, it relays requests and responses designated for the client and the tones server that is embedded within the server.

Embedded within the Node.js server is a tones server. This is built on the JUCE cross-platform application framework which incorporates an API for the ASIO audio device (JUCE, 2019). ASIO provides an API to audio input and output devices, and the low latency required for the time-sensitive delivery of audio (Steinberg, 2019). JUCE was therefore appropriate for use within SDIAS. The tones server has two primary functions, namely to:

1. deliver the tones at regular intervals as required by the client and
2. provide the timing mechanisms required by the tone generation feature.

Both the Node.js and tones servers are hosted on a Windows PC.

3.3.3 The Ethernet AVB Network

Ethernet AVB is a set of Ethernet protocols that handle real-time, time-critical audio tasks such as synchronisation, stream audio, bandwidth reservation, control and connection management of endpoints using wired connection for all endpoints in the network (IEEE 802.1BA, 2011). The components making up this network in SDIAS include:

- an AVB Echo Streamware network interface card (Echo, 2019a), also referred to as NIC-1: This device, installed on the Windows PC via a PCI Express slot, acts as an ABV-capable sound card and a network adapter. The device connects the Windows PC with other AVB devices primarily using a wired Ethernet connection. NIC-1 acts as the talker in SDIAS.
- an UltraLite AVB (MOTU, 2019) for Ethernet AVB networking. This device uses an Ethernet wired connection to the NIC-1 for communication, and
- AVB endpoints which act as AVB listeners and talkers on the AVB network. SDIAS only uses the UltraLite AVB device endpoints as listeners.

The Ethernet AVB protocol set enables some critical functions, such as connection management, discovery of participating devices and transport of audio across the network. The server relays client requests for audio streaming to the tones server via the AVB network. The AVB network streams audio from the NIC-1, which acts as the talker, to the UltraLite AVB device, which acts as a listener.

3.3.4 The Loudspeakers

The loudspeakers are connected to analogue output streams of the UltraLite AVB device to deliver tones according to server commands. These are marked ‘speaker 1’ to ‘speaker

N' in Figure 3.6. Their function is to play back the tones as requested by the client so that the client records them through the microphone for later determining the distances of each loudspeaker from a specific position, and subsequently the 3D coordinates for the speakers. Obtaining accurate 3D coordinates of the speakers is the main goal of this research project. It was important to first visualise the process before designing the system and this visualisation is discussed in the next section.

3.4 User Interaction with the System

The user must go through a series of steps to complete the configuration process. These steps are

- start-up,
- calibration,
- streaming and
- configuration.

To execute these steps, the user interacts with the system via a web browser application accessible through the mobile device over a Wi-Fi connection. The web browser application communicates with the server, issuing requests for control commands to be sent on the AVB network and receives responses from the server.

3.4.1 Startup

The user must set up the various components constituting the system environment. These include the mobile device, the host PC and the Ethernet AVB network components. The user must ensure that all components are connected physically and ready for communication. The user then needs to start the server, which in turn launches the tones server so that both are ready for the streaming process at the request of the client. On the mobile device, the user then types the URL of the server on a browser to navigate to the server. A connection is set up between the mobile device and the server. Upon successful connection to the server, the server sends the client JavaScript application code to the client. This file contains configuration information about the tones server and the speaker system, such as the number of speakers available in the immersive speaker system under test.

One of the first tasks the JavaScript application performs is to request for media access. This feature is implemented by all major browser vendors such as Mozilla's Firefox[2], Google's Chrome[3], or Microsoft's Edge[4] browsers. It serves to protect user privacy, specifically to prevent malicious web applications from secretly using various media such the microphone and camera for undesirable purpose (Adenot & Toy, 2018). Figure 3.7 shows a portion of the client UI requesting media resources permissions. The user must give permission for media resources by tapping the 'Share' option on the UI. Doing so allows gives the JavaScript application permission to use the microphone for recording, and takes the process to the next step - calibration.

Figure 3.7: Client Startup UI Portion for Media Access Request

3.4.2 Calibration

This step sets up the system client and server components so that their clocks are synchronised and syntonised to allow for delivery of audio samples between the two components (Watkinson, 2001). Syntonisation is a process of synchronising clock sample rates , that is, it is synchronisation in the frequency domain (Wolter, Reinecke, & Mittermaier, 2011). Both these steps are crucial for accurate timestamps as the components have separate distinct clocks which will not be ticking at the same rate and do not reference the same global clock. That these components have separate clocks poses a well-known problem in networked environments (Kurose & Ross, 2013) - that no two clocks ever tick at the same rate.

Granting the client application media resources takes the UI to the calibration window.

[2]Available at: `https://www.mozilla.org/en-US/firefox/new/`
[3]Available at: `https://cloud.google.com/chrome-enterprise/browser/download/`
[4]See: `https://www.microsoft.com/en-us/windows/microsoft-edge`

This is shown in Figure 3.8. Several variables need to be set at this step. Section 4.6 describes the purpose of the calibration phase in detail. The variables are:

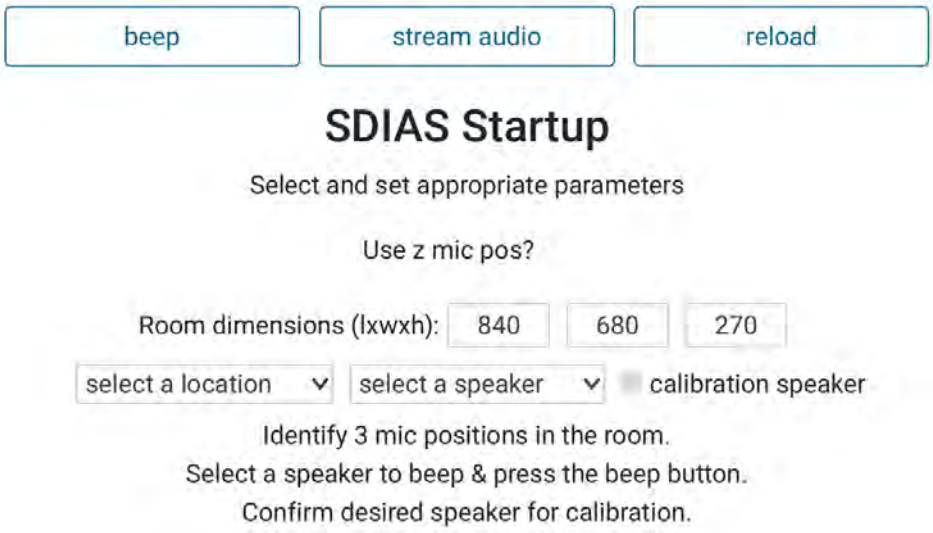


Figure 3.8: Client Calibration UI

- z-axis microphone position data usage: the user must optionally decide to use audio data recorded with the microphone position at the z axis. By default, and preferably, the option is to not use this position. The user must leave the checkbox next to the 'Use z mic pos?' text unchecked to not have to use audio data recorded by the microphone at the z-axis. The user can check the box to use data from this position. The details for this feature are explained in Section 4.8.
- room dimensions: these are the height, width and length of the room from which the measurement process is undertaken. The user must measure the room (or estimate these values[5]) and type them into the UI. These are used to verify the validity of the final coordinates, to ensure they do not exceed the given room or space dimensions. This step is only used for giving descriptive user feedback if the results are inaccurate. The dimensions do not affect any of the calculations related to the coordinates as the algorithmic calculations do not depend on the loudspeaker physical layout boundaries.
- calibration speaker number: the user must select a speaker to identify and ultimately mark as the calibration speaker from the UI. This is done by tapping the dropdown from UI, and selecting any desired speaker, for instance, 'speaker_01' (as shown in Figure B.2 in Appendix B). Once the speaker is selected, the user can tap the beep button to auditorily and visually identify the speaker's physical location in the

[5]This may pose problems with speakers located very close to the testing room's bounding box boundaries. Estimation is preferable if all loudspeakers lie fairly far away from the room walls and ceiling.

room. The user can go through all the speakers in the list to identify their physical locations. For convenience, the calibration speaker needs to be the closest of all speakers to the origin relative to the rest of the speakers in the system so that its resultant TOF is shortest relative to all other speakers' TOFs.

- calibration speaker confirmation: the user will mark the closest speaker as the calibration speaker by tapping the checkbox for the calibration speaker. Calibration speaker confirmation creates input boxes for the user to type the calibration and orthogonal distances (shown in Figure B.3 in Appendix B).
- calibration speaker distance: the user must manually measure the distance of the calibration speaker. This is the distance from the origin to the calibration distance. The user must then type the obtained distance into the respective input text, in centimetres.
- orthogonal distance: similarly, the user must manually measure the orthogonal distance, described in Section 3.1. The user must then type the obtained distance into the respective input text, in centimetres.

Once all the required variables are set, the streaming steps then follow.

3.4.3 Streaming

Streaming involves sending tones from the loudspeakers at the client request and a subsequent transmission of the tones audio data from the client to the server. The process comprises a number of steps:

1. the first step is to select the location from the locations dropdown list. The user must select the origin location to start the streaming process (shown in Figure B.4 in Appendix B).

2. the user places the microphone attached to the mobile device at the origin location, then requests tones to be streamed to this location by tapping the stream button on the UI.

3. the client sends a stream request to the server and instructs the microphone to starts recording. A spinner is displayed for the duration of the tones transmission. Also, a text guiding the user about which position to next place the microphone is displayed as shown in Figure 3.9.

4. the server relays a stream request to the tones server.

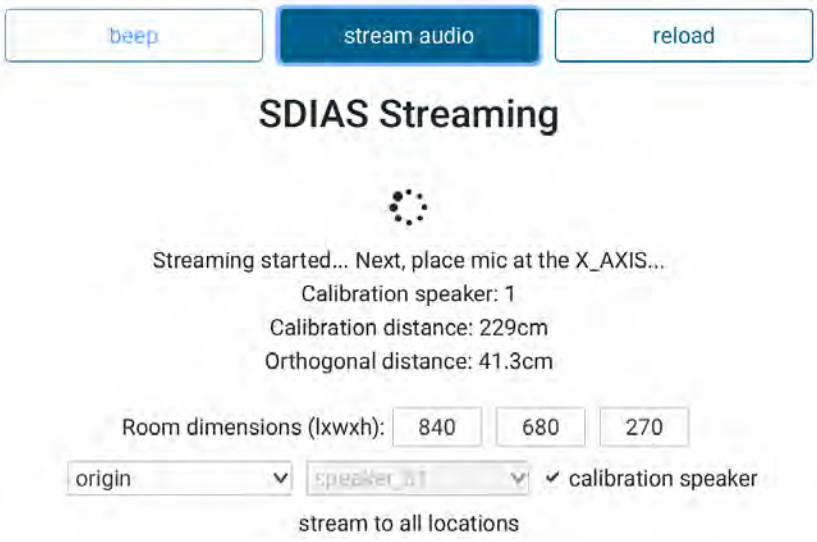


Figure 3.9: Client UI - Streaming

5. the tones server starts streaming tones, via the NIC-1 through to the UltraLite AVB interface for playback by the loudspeakers. A tone is delivered by the tones server, destined to the UltraLite AVB output stream connecting to each speaker in sequence, at the origin location, O, as shown in Figure 3.10.

6. once streaming stops, the user then places the microphone at the next location, X, and requests streaming to commence. Streaming occurs at the rest of the locations Y and optionally Z as well as the origin again. If the z-axis microphone position usage has not been selected, the user need not place the microphone at this position, he or she must just place the microphone back at the origin for the second time.

7. the tones recording stops on the mobile device.

8. the mobile device identifies the first tone and slices the recorded audio data from that point onwards to the last tones. This is done to avoid sending a large file over the Wi-Fi connection to the server. An in-depth description of the slicing process is given in Section 5.3.11.

3.4.4 Configuration

Configuration is the final step in the speaker determination process. The server performs most of the intensive computations necessary to complete the measurement process. The client therefore becomes idle during this process, as it awaits results from the server. The

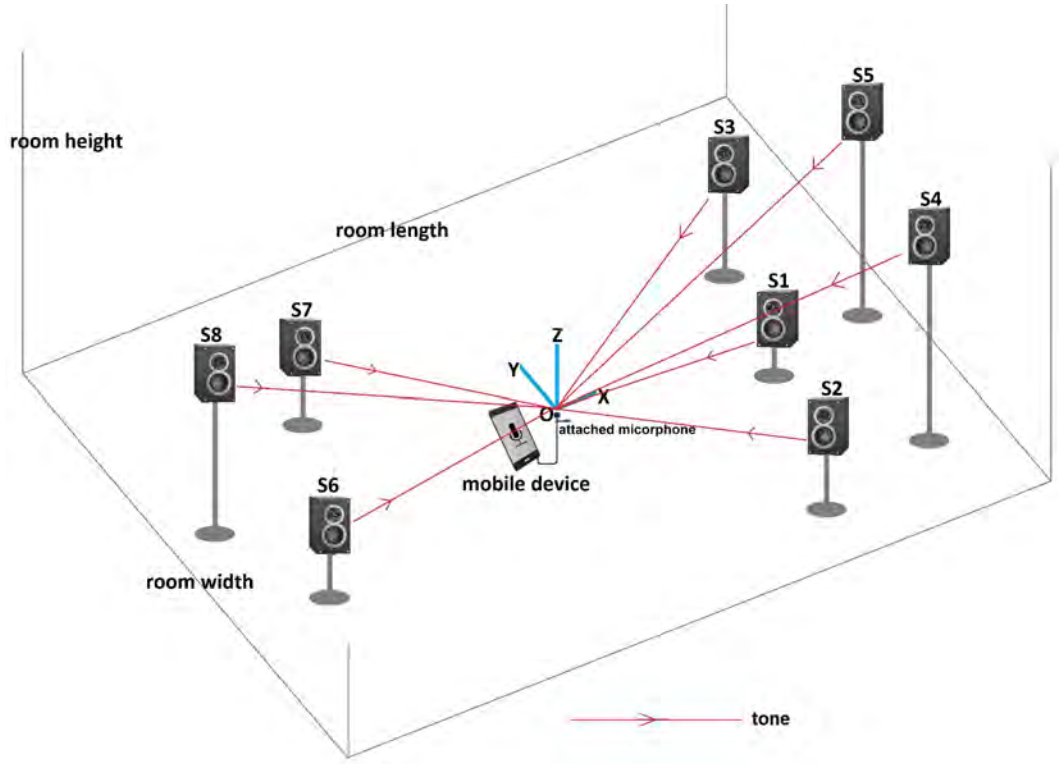


Figure 3.10: Audio streaming at origin location

server receives the recorded audio samples from the client, processes the audio samples and the final 3D coordinates as an end result on the server. Important data for the coordinate determination are the occurrence of each tone transmission from the server and the corresponding time of reception at the client's microphone. These two times determine the TOF of the tone from the loudspeaker to the microphone, and it is from this TOF that the distance of the speaker to the microphone is derived. All speakers' TOFs are obtained at this step and are converted to distances. Once distances are obtained for each speaker at each location, triangulation is used to compute each speaker's 3D coordinates. These coordinates are then sent to the client.

The client then displays the results for interpretation. These include the information related to the SDIAS run just completed, distances, coordinates and a visualisation. The following sub-sections describe the results, with portions of the client UI for each piece of the results shown.

SDIAS Run Information

Immediately below the header is a portion of the client UI display of a summary of various variables related to the current measurement process. These include the timestamp, some previously set quantities such as the room dimensions, calibration speaker position and distance, orthogonal distance and new quantities such as the sample rate differential (explained in detail in Section 4.6.2) and the value of speed of sound used, amongst others. Figure 3.11 shows a portion of the client UI with a complete list of this information.

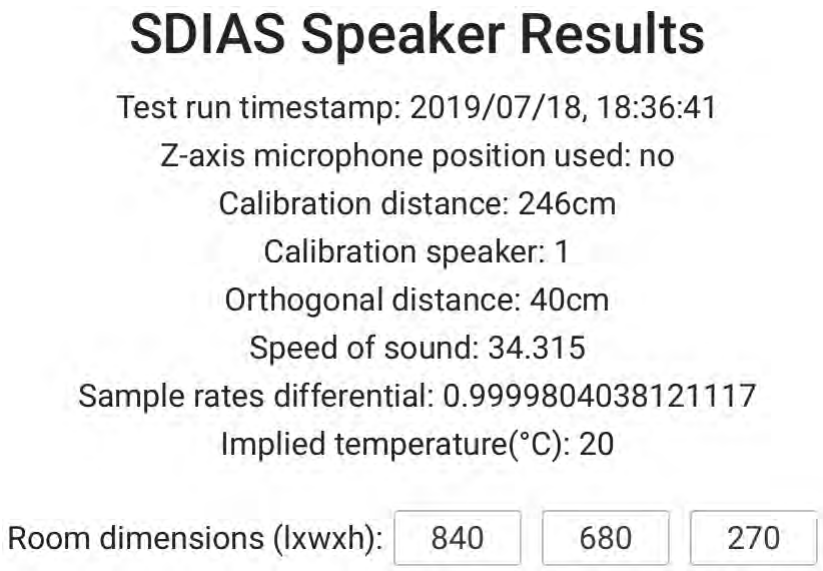


Figure 3.11: Results display -current run information

Distances

These are distances obtained from each location for each speaker as shown in Figure 3.12. These are displayed below the information relating to the current run. The interpretation for these values is as follows, using Speaker 8 as an example in Figure 3.12, the speaker was found to be:

- 379.2cm from the microphone positioned at the origin (O1),
- 360.6cm from the microphone positioned at X - a point 40cm from the origin as the orthogonal distance as the user would have set during the calibration phase (as shown in Figure 3.11), along the x-axis,
- 354.9cm from the microphone positioned at Y, a point 40cm from the origin and orthogonal to x and z axes, along the y-axis (Y),

- 380.1cm from the microphone positioned at Z, a point 40cm from the origin and orthogonal to x and y axes, along the z-axis (Z). This value need not be valid if the microphone position at the z-axis was not used,
- 379.9cm from the microphone positioned at the origin for the second time (O2). This value is expected to be equal, or very close, to the value obtained from O1 as the microphone is at the same location for both O1 and O2, but just at different times during the process.

DISTANCES (O1, X, Y, Z, O2)

Speaker 1 :	246.0	235.2	211.7	246.1	246.0
Speaker 2 :	228.8	235.9	192.3	226.7	227.3
Speaker 3 :	304.5	280.1	275.9	304.6	304.5
Speaker 4 :	222.2	241.5	257.2	222.3	222.2
Speaker 5 :	301.5	273.6	327.2	301.6	301.5
Speaker 6 :	223.5	210.6	257.8	224.3	224.2
Speaker 7 :	324.2	329.2	359.3	325.1	325.0
Speaker 8 :	379.2	360.6	354.9	380.1	379.9

Figure 3.12: Results Display - Distances

Coordinates

The 3D coordinates are displayed immediately below the distances. These are obtained from the server are presented in two formats, namely the:

- original coordinates: for each speaker, a triplet of (x,y,z) coordinates are displayed, given with respect to the origin O, and
- translated coordinates: for each speaker, a triplet of (x,y,z) coordinates are displayed, given with respect to a reference speaker. This eliminates the microphone positions from the results, leaving only on the relations within loudspeakers layout. The use of a reference speaker is explained in Section 6.2.2. The interpretation from these results is as follows, with Figure 3.13 used as reference for directions in the interpretation:
  - For Speaker 1, the first 0.0 means the speaker is zero centimetres along the x-axis, that is, the right-left direction. Similarly for the second 0.0, the speaker is zero centimetres along the y-axis, that is, the front-back direction. For the last value, 83.8, the interpretation is that the speaker is 83.8cm above the floor.

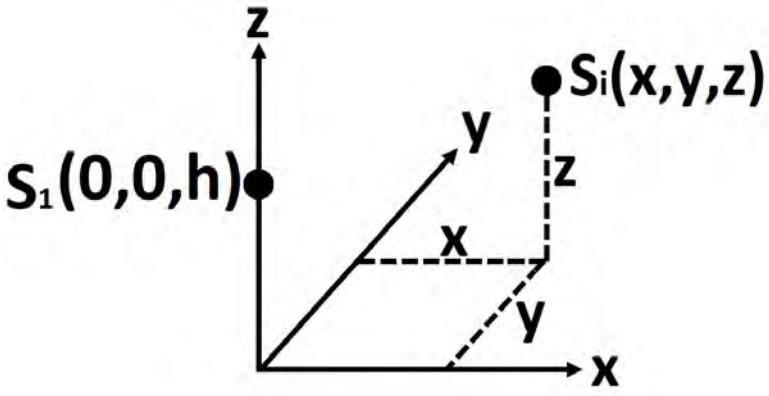


Figure 3.13: 3D Coordinates Directions

These three collectively define the location of Speaker 1 as (0.0,0.0,83.8) as the speaker's 3D coordinates. Speaker 1 was selected as the calibration speaker for the indicated results, as indicated by the coordinates given by $S_1(0, 0, h)$ as explained in Section 6.2.2 and as shown in Figure 3.13.

– For Speaker 2, the -106.0 for the x-coordinate means the speaker is 106.0cm to the left of the reference speaker - Speaker 1. The -4.4 means the speaker is 4.4cm to the back of the reference speaker. Lastly, the 86.6 for the z-coordinate means the speaker is 86.6cm above the floor - the z-coordinate is given independent of the reference speaker's z-coordinate.

– The rest of the speaker coordinates are interpreted in a similar manner to those for Speaker 2. The positive directions for the 3D space are given by the indicated directions of labelled axes, such the opposite directions would define the negative portions of the 3D space. For instance, some speaker $S_i(x, y, z)$, as shown in Figure 3.13, lies in the octant with all positive coordinates.

Visualisation

Lastly, a view of the sliced audio (more details on the data slicing which reduces data transmission across the system components are given in Section 5.3.11) recorded by microphone at the four locations is drawn on a canvas. This is displayed below the coordinates. The visualisation assists in determining the accuracy of the obtained coordinates, especially if there are major errors in the sliced audio data. The visualisation, shown in Figure 3.15, shows a compressed waveform of the audio that represents five locations, each with N (= 8 for instance) tones from N loudspeakers in the speaker layout. An uncompressed

COORDINATES (X, Y, Z)

	original			→	translated		
Speaker 1 :	84.9	216.3	80.8	→	0.0	0.0	83.8
Speaker 2 :	-21.1	211.9	83.6	→	-106.0	-4.4	86.6
Speaker 3 :	198.1	227.4	41.8	→	113.2	11.1	44.8
Speaker 4 :	-91.5	-190.0	70.2	→	-176.4	-406.3	73.2
Speaker 5 :	220.9	-182.3	94.3	→	136.0	-398.6	97.3
Speaker 6 :	90.2	-186.5	84.0	→	5.3	-402.8	87.0
Speaker 7 :	-20.3	-279.3	163.4	→	-105.2	-495.6	166.4
Speaker 8 :	192.5	243.0	218.4	→	107.6	26.7	221.4

Figure 3.14: Results Display - Coordinates

version of the waveform which distinctly depicts the five locations together with their respective eight impulses is shown in Figure B.5. If for any reason data slicing was inaccurate, parts of the graph could show straight lines which indicate silence instead of the expected impulsive tone wave. The silence possibly suggests that data was sliced at the wrong region where nothing was streaming from the speakers if it was indeed streamed and therefore leading to unsatisfactory results. More examples of sources of the errors which might lead to unsatisfactory results are given in Section 5.3.13.

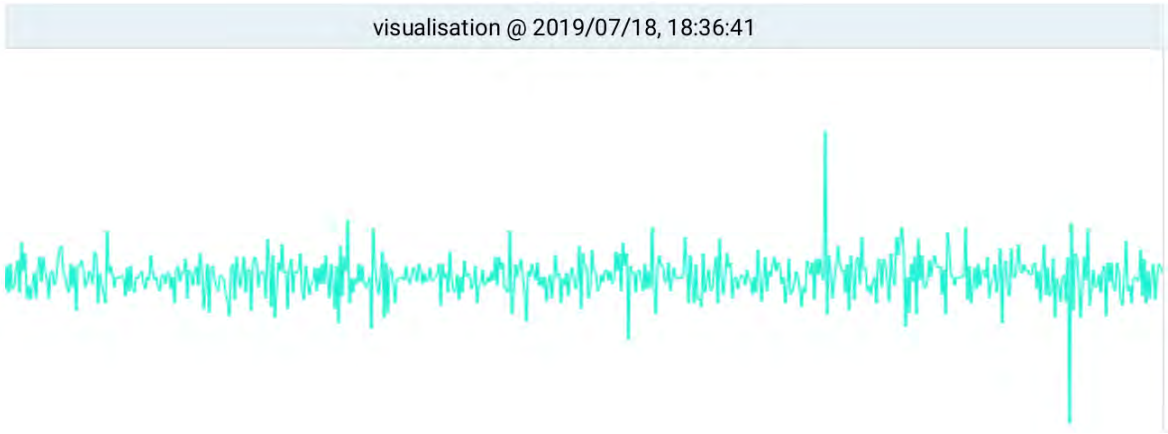


Figure 3.15: Results Display - Visualisation

A description of the unsatisfactory results would appear above the visualisation and below the coordinates, as shown in Figure B.6 of the appendix, which shows that there was a problem with the distance and ultimately the coordinate values obtained for the z axis for Speaker 3. This would require that the process is restarted, which is done by tapping

the reload button on the UI, otherwise this marks the end of the configuration process. The coordinates are ready for use in localisation algorithms in immersive audio systems such as Immergo system.

3.5 Chapter Summary

This chapter provided an overview of the speaker position measurement system created for this research - SDIAS. An account of the physical model which aided in visualisation of the entire process of speaker position measurement was given, highlighting its significance in conceptualising various quantities such as the orthogonal distances, associated angles and microphone-to-speaker distances required for the success of the process. This was further verified with a JavaScript program to assess the feasibility of trigonometry in computing the 3D coordinates.

Also, the chapter gave an account of the client-server architecture of the system. It showed how the user, through the client application running on a mobile device, drives the whole process and how the server responds to requests from the client. It also explained the Node.js server and the JUCE application and how these play a crucial role of serving the client requests.

Finally, a description of the system user interaction was given, together with the details of the steps from startup through to configuration, with details on the presentation of results obtained from the server. The next chapter is an account of the technology used in SDIAS.

Chapter 4

The Technology Associated with SDIAS

This chapter gives an account of various techniques that were explored and some ultimately deployed in the speaker positions measurement system. In order to localise sound for each speaker in the speaker layout, a tone had to be played out by each speaker and time measured for how long the tone travelled from the speaker to the microphone of the measuring device - the mobile device. It was crucial to use a tone that could be distinguishable from other unwanted sound, for example background noise, for the purpose of measuring time of flight from the speaker to the microphone.

A digital recording of a tone is composed of many audio samples. A specific point within the tone has to be used as a distinctive point to uniquely identify the tone within the many audio samples recorded by the mobile device. This point is referred to as the detection point. The point sought to identify a single unique sample within a tone, for every recorded tone in the entire array of audio samples. More details are given in Section 4.1 on the usefulness of having this point in determining speaker positions. There were many types of tones that could be used. Various aspects of the investigated tones, such as the waveform's shape property, are described in Section 4.2, together with how patterns in the various waveforms could be of use in SDIAS. There were some features from the investigated wave types which could be combined and used in tone detection. Section 4.3 describes how these features were designed to create a waveform that could be used for speaker position measurements.

Inherent in natural environments are factors such as ambient noise during audio recording and latency during audio processing and transmission. Section 4.4 gives an account of how

ambient noise was handled so as to minimise its undesirable effects. Latency is inherent in many, if not all computing processes (Kurose & Ross, 2013). In any computing process where time is critical, latency becomes a hurdle. SDIAS being a time-critical networked system was no different. It became apparent early in the design process that latency would have to be handled carefully if the system was to successfully and accurately measure speaker distances and ultimately compute the expected 3D coordinates. Details of the challenges brought about by latency are discussed in Section 4.5. Closely related to the latency issue was a problem emanating from the existence of different clocks in the tone generation and tone reception components of SDIAS. Each component had its own clock, and this posed a challenge for the measurement of the time which tones travelled from the speakers to the microphone. As a result, calibration became necessary to handle these differences. Section 4.6 gives details on how the challenges posed by the differences were overcome, together with those posed by latency.

After providing the comparative and compensatory measures to counteract the existence of more than one clock, the distances of each speaker need to be determined. These distances are determined with respect to a reference speaker, called the calibration speaker. They are extracted via a detection algorithm from the recorded audio samples obtained at the four positions of the microphone as tones are streamed from each location. A detailed account of the steps involved in obtaining all the required distances is given in Section 4.7. Finally, Section 4.8 gives a description of the computation of the 3D speaker coordinates from the distances that were measured. The section also describes challenges associated with microphone placement along the z-axis, together with a solution to mitigate these challenges.

4.1 The Detection Point

For each tone transmitted by the server through a loudspeaker, a point of detection within the tone was required to uniquely identify the sent tone. The detection had to accurately resolve to a precise point in time at which the transmitted tone was received on the microphone. The precision of the identification point would best be achieved by a sample-accurate detection mechanism, that is, the granularity of the detection mechanism should be able to distinguish one sample from the next at the very least. As this depends on the sampling rate, it meant the higher the sampling rate, the better the granularity. A higher sample rate would have been preferred, but there was a limitation on the available resources, namely the Streamware Controller application used by the Echo Streamware

NIC-1 card since the application only supports the 48,000Hz sample rate (Echo, 2019a, 2019b) and the mobile devices used. For this reason, a sample rate of 48,000Hz is used in this research. One sample translates to a time duration of one $48{,}000^{th}$ of a second, equating to $20.8\mu s$ and equivalent to $7.1mm$ when using the value of the speed of sound in air approximately as $343ms^{-1}$. The 7.1mm length then became the margin of error for the speaker positions. The need for the tone detection point to occur at the same point in the duration of the tone at each playback has an implication that the same sample position should ideally be detectable for each set of audio samples constituting the tone emanating from a loudspeaker.

4.2 Common Sound Sources

Sound occurs naturally or gets generated artificially from unnatural sources such as mathematical functions. Various sources of sounds were explored for use in SDIAS. These included the naturally occurring types such as a hand clap, to other types such as those mathematically generated, like the sine wave. In order to determine the most suitable sound source, several tests were performed for the various sound sources, and microphone, to determine the most suitable for use in the system. The tested sound source types, together with their key properties, in the time-domain are described below.

4.2.1 Sine Wave

A sine wave is a wave created from the trigonometric sine function. Sound generated from this type of source will be a tone of constant pitch as its frequency remains constant for the entire duration of the tone. The sine wave has some attractive properties. The properties include the distinctive and cyclic wave pattern it generates as it plays out, a distinguishable start and an end of a cycle, uniform peaks of the wave and the number of cycles found within each wave. All these features earmarked the sine wave for the potential as a sound source, since they could be harnessed in finding a detection point within the audio recording.

The sine wave is a mathematical graph that follows a curve of a smooth periodic oscillation, whose formula is given by the equation below:

$$y = A\sin(ax + b) \tag{4.1}$$

where:

A = the amplitude of the wave (that is, the deviation of the function from zero),

a = the angular frequency (that is, the number of cycles per second), and

b = the phase

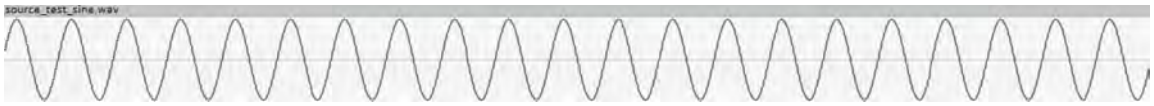


Figure 4.1: A full sine wave visualisation

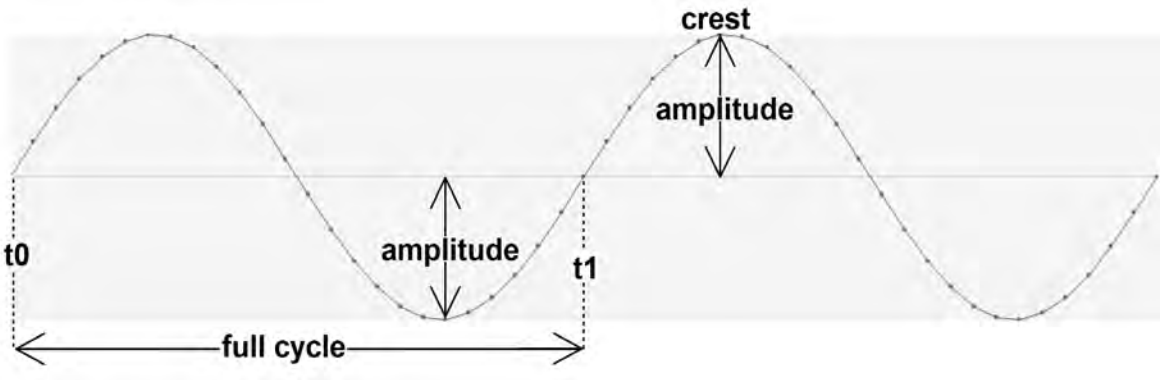


Figure 4.2: Properties of the sine wave

A sine wave tone audio file was created in a waveform audio file format (.wav) (Weber, 2019) for investigating its possibility for use as a sound source. Its graph is shown in Figure 4.1[1]. This was synthesised from a 512-samples sine tone at a frequency of 1,920Hz with a sampling rate of 48,000Hz as used in SDIAS. The frequency had to lie within the human hearing frequency range - 16Hz to 20kHz (Blauert, 1983). It was also selected to provide sufficient energy to the microphone by avoiding low frequencies which yield low power at the microphone. It also had to avoid subjecting the investigator to extended unpleasantness from frequencies at the extreme ends of the hearing range, hence aimed for the midrange frequencies. The frequency also gives an integer period.

For consistency, this value is maintained throughout the rest of this thesis (for instance, it is used further in the tones used for SDIAS during implementation and testing, described in Section 4.3). The 1,920Hz midrange frequency value was therefore used. The graphs yield twenty complete cycles, generated from the first 500 samples, while the twelve last

[1]This is a portion of a screenshot from Reaper which highlights features of the sine wave. The rest of the UI has been clipped off due to limited space.

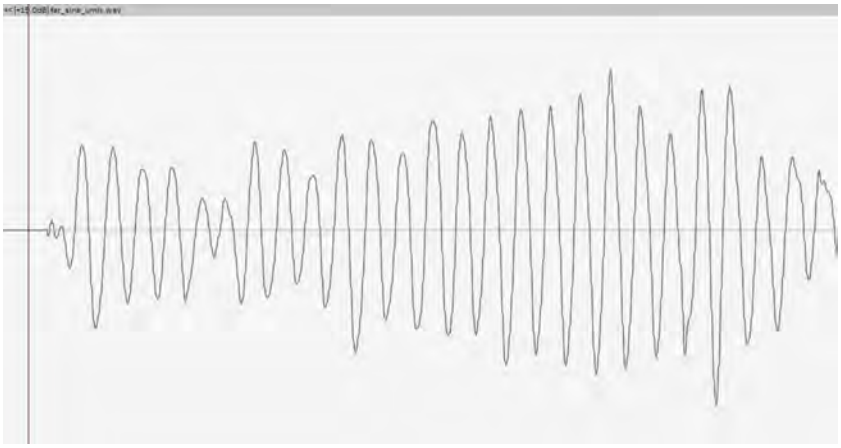


Figure 4.3: Sine wave response

samples do not form a complete cycle. A zoomed in display is shown in Figure 4.2 to highlight some features of interest. The figure shows two cycles, two crests and an amplitude of the sine wave. The period is the time it takes to complete one cycle, shown as the duration $|t1 - t0|$ in the figure. These properties form part of the various attributes exploited in the identification of a sound source used in the determination of speaker positions.

Figure 4.3 shows a sine wave response for the UMIK-1 microphone (miniDSP, 2019) attached to a PC. Multiple microphones were used, and their responses analysed (including via visualisation) for appropriateness. Using the first peak of the sine wave as a detection point was not possible. This is due to absorption part of the energy from the first few samples of the recorded tone as a result of energy transfer from sound to kinetic energy which occurs when the microphone's diaphragm vibrates (Watkinson, 2001). Also, the evenness of the cycle amplitudes is lost, making it difficult to distinguish the sine from other audio such as noise.

4.2.2 Sine Sweep

This is a variation of the sine wave described in the previous section and it is also known as a *chirp*. The difference comes in the change of frequency over time. The tone starts off at a certain frequency and changes over time. The tone could start at a lower frequency and end at a higher frequency (see Figure 4.4), or vice versa. It could also start off at a lower frequency, get to its highest frequency towards the middle of the duration and die out again to a lower frequency as the tone ends (see Figure 4.5), and vice versa. The

oscillations of the sine wave are densest at the regions of highest frequency, and sparse at the regions of lowest frequency. These patterns could also be used as an identifying feature to distinguish the tone from ambient noise, hence earmarking this type of tone as a potential sound source.

However, this could not be used as the clarity of the waves was distorted at the region towards the high-frequency cycles. This was possibly due to indistinct region from low-frequency to high-frequency cycles. Also, it did not depict a uniquely identifying feature as shown in Figure 4.6 from the sweep microphone response.

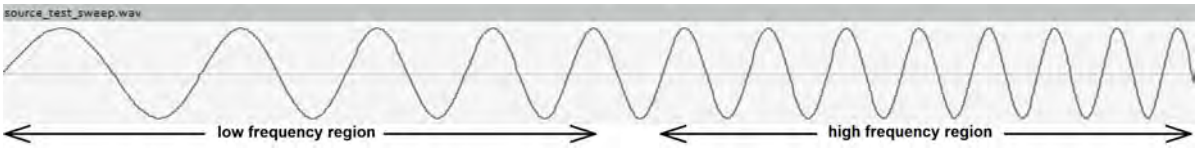


Figure 4.4: The sine sweep - low to high frequency

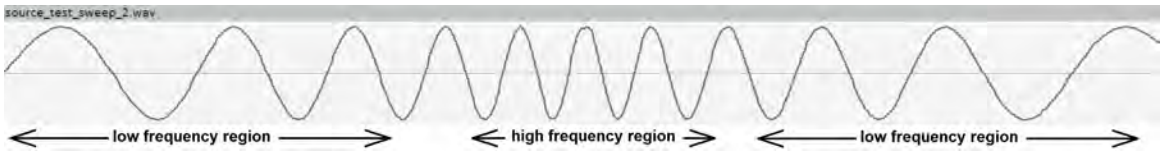


Figure 4.5: The sine sweep - low to high to low frequency

Figure 4.6: Sine sweep response

4.2.3 The Hand Clap

Another source of sound investigated was a hand clap. Figure 4.7 shows a 42ms (approximately 2100 samples) long hand clap as captured with Reaper. Its outburst of energy

also provides an interesting pattern for exploitation in identifying a specific tone as shown in Figure 4.8, which is a zoomed screen capture from the hand clap. The amplitude suddenly rises to a few distinct peaks (*peak 1* and *2* in Figure 4.8) as a phenomenon known as a *shock wave* which occurs due to increases in pressure which lead to distortion in the sound wave (Watkinson, 2001) and dies out gradually. However, the shock wave caused by distortion in the wave propagation was not consistently distorted and as predictable as desired under different test configurations. Figure 4.9 shows a response for the clap recorded via the UMIK-1 microphone through Reaper.

Figure 4.7: Hand clap

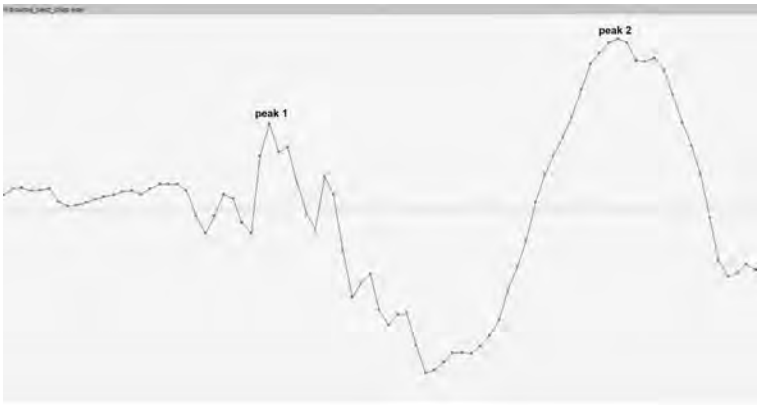


Figure 4.8: Clap's signature region

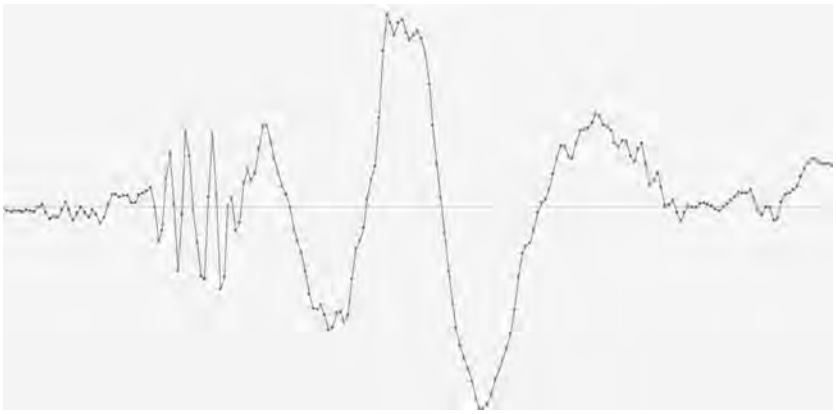


Figure 4.9: The clap response

4.2.4 Square Wave

A further tone type investigated as a potential sound source was a square wave. This type of wave simply has its amplitude changing between two extreme values - a maximum amplitude and a minimum amplitude - at periodic intervals. Depending on the frequency, several samples have a high positive value, and then an equal number of samples have a high negative value, but with the same absolute value. The pattern for this wave was also investigated, since it is clearly distinct and should be distinguishable from background noise. Figure 4.10 shows a portion of a 512-sample square wave with a frequency of 1,920Hz.

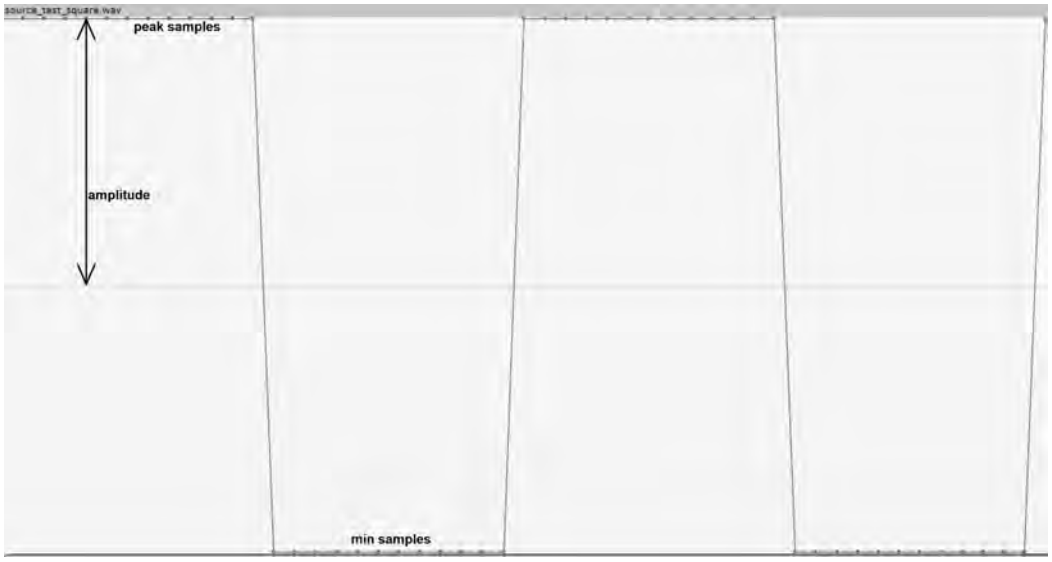


Figure 4.10: Properties of a square wave

Figure 4.11 shows the square wave response recorded with a Panda microphone (Purple Panda, 2019). This tone type, despite producing a clear wave, could not be used as the wave did not maintain the distinct squares, possibly due to how the square wave could be represented by a sum of infinite harmonics (MathWorks, 2020).

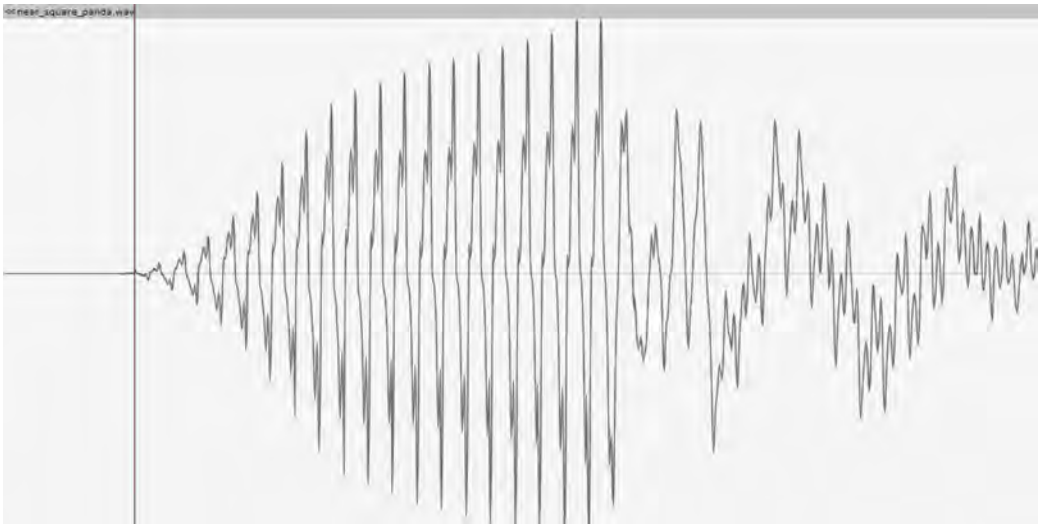


Figure 4.11: Square wave response

4.2.5 White Noise

Finally, noise was also investigated. There are many types and sources of noise. Examples of types include the white, pink, brown and impulsive noise, while example sources are acoustic, electromagnetic and electronic in nature (Vaseghi, 2008). One type of noise investigated was white noise. The identifying feature for this type of sound is its aperiodicity and randomness in all frequencies over time. This property could also be distinct in the identification of tones from the speakers, since different sounds produce defined wave patterns while white noise produces a randomly varying waveform, from one sample to the next. That no pattern is discernible in white noise could be a pattern on its own. Figure 4.12 shows a 512-sample long noise while Figure 4.13 shows randomness in a few samples of the zoomed in noise wave. This source type could not be used due to its inability to determine one sample that could be used as the identifying feature for the tone as shown from noise microphone response in Figure 4.14.

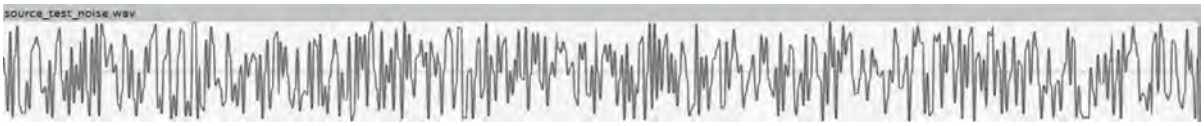


Figure 4.12: White noise

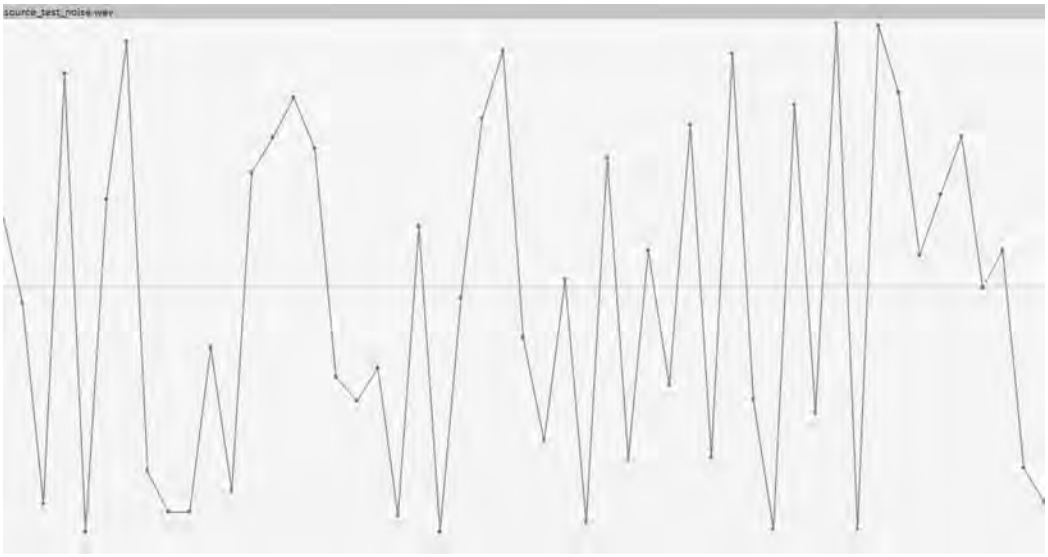


Figure 4.13: Zoomed in noise

Figure 4.14: Noise zoom response

Finding an appropriate sound source type was a large part of this thesis. Some of the features of the sound source types described in this section were combined to provide sufficient basis to yield the required results as described in the next section. Therefore, other types of sound source signals such as the maximum length sequence (MLS) were not investigated. MLS, for instance, depicts features similar to that of white noise (Stan, Embrechts, & Archambeau, 2002), and as such, would not be appropriate for use in SDIAS.

4.3 The Nature of the Detected Waveform

A combination of two properties of the sound sources discussed in the previous section were used to create a suitable wave that could function as a discernible sound source. The

predictability of the sine wave and the sharp peaks of the hand clap were features used to create the desired tone that would be sent to a speaker. Also, from the microphone responses of the various sound sources, the problematic characteristics had to be avoided, such energy distortion at the start of a tone. A certain number of low amplitude cycles were generated at the same frequency to get the microphone diaphragm vibrating, and then a high amplitude sine wave cycle was generated while retaining the frequency. This combination was envisaged as ideal for uniquely identifying a tone. A tone sent to a speaker could therefore be identifiable through the number of cycles available within the duration of the tone. A total of nineteen low amplitude cycles were formed by a 1,920Hz sine tone at 48,000Hz sampling rate. The higher frequency provided higher energy. The amplitude of the 20th cycle was substantially higher. This was to provide a unique region within the tone and minimise the sharpness that comes with, for example, a hand clap that would result in undesirable distortion.

The resultant sound source became a 512-sample long sine wave with nineteen low amplitude cycles and the last one cycle peaking to maximum amplitude, at 1,920Hz frequency. This is shown in Figure 4.15. The peak sample in the graph gives the desired detection point - a unique point for identification of the sound source as shown in Figure 4.16. This combination provided an adequate mechanism to resonate the microphone at the start of the tone and allow for a distinct peak to occur later in the tone. It is a compromise of the Dirac impulse and the sinusoid. Figure 4.18 shows the microphone response of this tone.

The tone described thus far was suitable for small testing environments through experimentation. Tests were done using the tone in a small test room (see Section 6.1.1). The tests gave accurate results (see Section 6.2.3), however, there were problems in larger testing environments. The problems arose with the accuracy in identifying the peak cycle. This is due to reflections of early cycles from the room walls and ceiling distorting the shape of the peak cycle at the end, hence, making it difficult to identify this peak. For this reason, a variation of this tone was created. The variation moved the peak amplitude from the last cycle, to the third. It was not moved to the first cycle to avoid problems of energy absorption highlighted with the cycles at the beginning of the tone. Moving the peak cycle to an earlier position in the tone solved the problem. Results from the larger testing environment (described in Section 6.1.2) are given in Section 6.2.4. The graph of this variation is shown in Figure 4.17. Its microphone response is also similar to that shown in Figure 4.18, except the peak occurs much earlier. In ideal recording environments there would be no other sounds except the tones from the speakers, but this would not always be the case. As such, noise from various sources needed attention and isolation as described in the following section.

Figure 4.15: Final graph with detection point

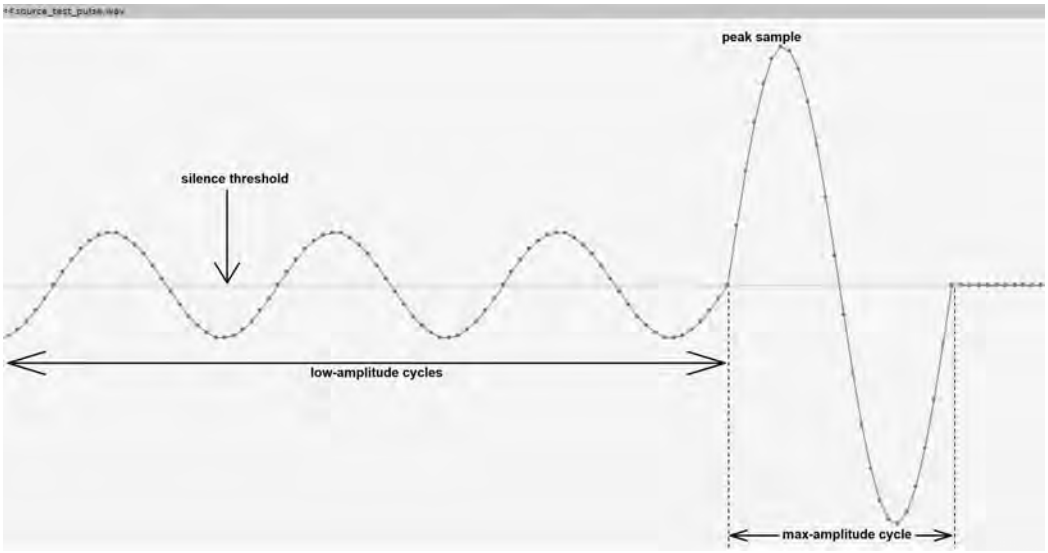


Figure 4.16: Zoom of unique region

4.4 Ambient Noise

Under ideal conditions, the types of sound sources described in the previous sections would yield waveforms similar to their original sources, but due to unwanted noise, distortion often occurs (Vaseghi, 2008). It was necessary to be aware of the impact unwanted noise would have on the accuracy of tone detection. Examples of unwanted noise include acoustic reverberation, microphone equivalent noise level and background noise from the testing environments. Any of these would create extraneous waveforms in the audio recording, and hence distort the expected waveform, which would in turn lead to inaccurate tone identification. To counter the negative effects of such noise, a method was devised which explored the amplitude of the sine wave and the crests which composed the waveform. The method considered the threshold and the interval between tones as described in the following sections.

The tone detection algorithm searched for tones in the recorded audio samples from the beginning to the end of the recording. The threshold determined the level of noise above which the algorithm would consider searching for useful features of a sine wave above

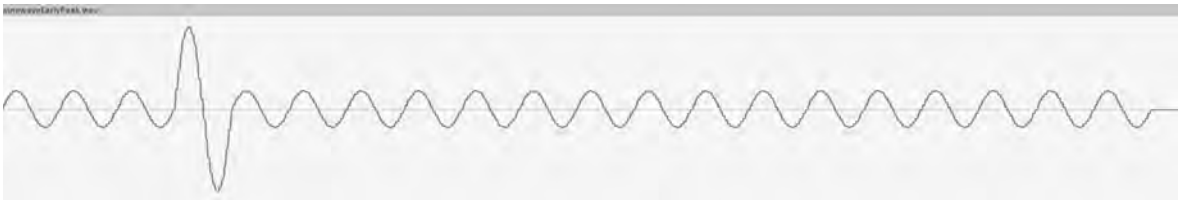


Figure 4.17: Early peak sine wave

the silence threshold. This feature exploited the amplitude of the sine function. The amplitude of silence, that is, when no sound source is present, is zero (or values with small deviations from zero) for the duration of the silence period (see Figure 4.15). External sound sources cause a noticeable deviation from zero amplitude. The pattern of deviation is known for the expected tone and could be searched for and recognised. A deviation of unknown pattern would hence be noise.

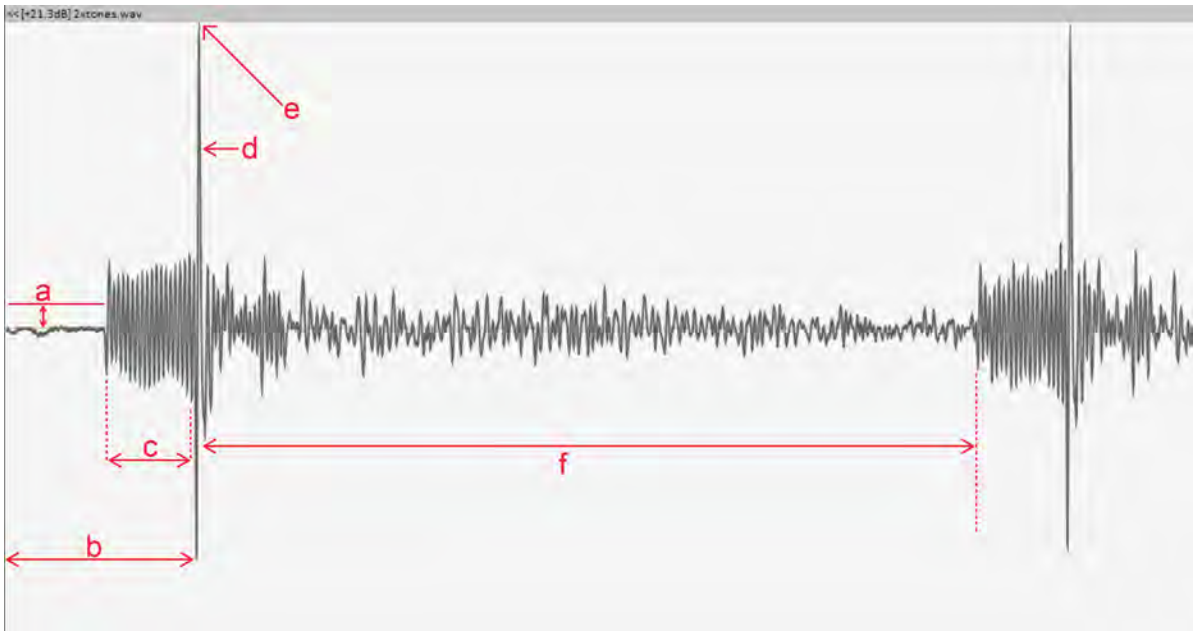


Figure 4.18: Tone played successively at a speaker

To further counteract the negative effects of ambient noise there is an interval of silence between tones. The tones are generated at equally spaced intervals each time. This means that whenever the user requires tones to be streamed at a particular location, the tone generator sends a tone to *speaker* 1, waits a specified number of samples, then sends to *speaker* 2, waits a further equally specified number of samples, and so on, until a tone is sent to each of the eight speakers. This means that the intervals between the tones are of constant duration from the generator's point of view. The expectation is that a single

tone should be detected for each period, which consists of the duration of the tone itself and the waiting interval.

Reverberation typically accompanies playback of sound in diffuse sound fields such as the ones used in SDIAS, also adding to the total sound experience of the listening environment (Shinn-Cunningham, 2000). It was important to allow sufficient time for reverberation to decay after each tone's playback. Through experimentation, the waiting interval was varied to identify the best one that could avoid reverberation. Examples of such intervals were 64ms, 96ms, 128ms and 160ms. The 96ms interval was found to be adequate to yield the most accurate results without too much delay, so it was chosen as the interval between tones. Withing this interval, sound would travel approximately 33 metres. Figure 4.18 shows a graph of a sound source sent to a speaker at the said interval. This is the same sound source described in Figures 4.15 and 4.16. Figure 4.18 highlights some attributes whose joint capabilities served to minimise the effects of ambient noise, potentially enabling the detection of only the transmitted waveforms. The attributes are:

(a) - the threshold

(b) - the tone duration

(c) - the low amplitude cycles

(d) - the high amplitude cycle

(e) - the detection point

(f) - the interval between tones

Another critical consideration, besides noise, was given to time delays existing within and across the various components of SDIAS. These are described in the next section.

4.5 Latency Issues

Latency is inherent in all network applications. It thus required consideration in the time sensitive SDIAS for accurate time measurements to be acquired. Different links and processes created delays as messages and audio were propagated within and to different components of the system. Figure 4.19 shows the various delays which occurred within the

system. If mishandled or unhandled, they could have a negative impact on the accuracy of the acquired measurements, hence the need to trace and handle them at various points of occurrence during the configuration process.

Delays have previously proven to be a difficult problem to solve, especially in asymmetric networks (Kurose & Ross, 2013) such as that found in SDIAS. The approach used for correcting for the latency had to take into consideration the delays which were indeterminate and those that were deterministic and treat them accordingly. An example of an indeterminate delay included the delay occurring in the Wi-Fi connection responsible for communication between the mobile device and the Windows PC hosting the server. Every transmission to the PC has a different latency that cannot be predetermined (De Vito, Rapuano, & Tomaciello, 2008). Consequently, this type of delay must be isolated such that its effects do not negatively influence the overall time measurements for the identified tones.

Speaker coordinate measurements in SDIAS depend on the accuracy of TOF measurements for sound transmission from speakers to the microphone. At each point in the transmission, time transpired due to the execution of messages and routing of audio from one point to the next. This occurs from the time the user presses a button on the mobile device UI requesting audio streaming, until audio is available on the mobile device after it has been recorded via the microphone. Delays occurring at each point or link needed tracking, isolation, and subsequently treatment. Delays are introduced at the following various places and processes of the full audio routing path:

1. Audio stream requests within the mobile device. The delay caused by the request is not of interest in the system but poses uncertainties if mishandled or not isolated from the measured times. It is indeterminate in nature, so the best way is to isolate it.

2. the routing of audio stream requests via the Wi-Fi connection onwards to the server. This is another indeterminate delay which needs isolation.

3. the execution of the request response by the server prior to propagation of the generated tones to the JUCE application. This is a further indeterminate delay.

4. the transmission of the 512-samples audio buffer tones from the JUCE application to the to the NIC-1.

5. the buffered tone is encapsulated in an AVB stream packet by the NIC-1. It is at this point that the AVB audio stream packets are marked with a presentation time

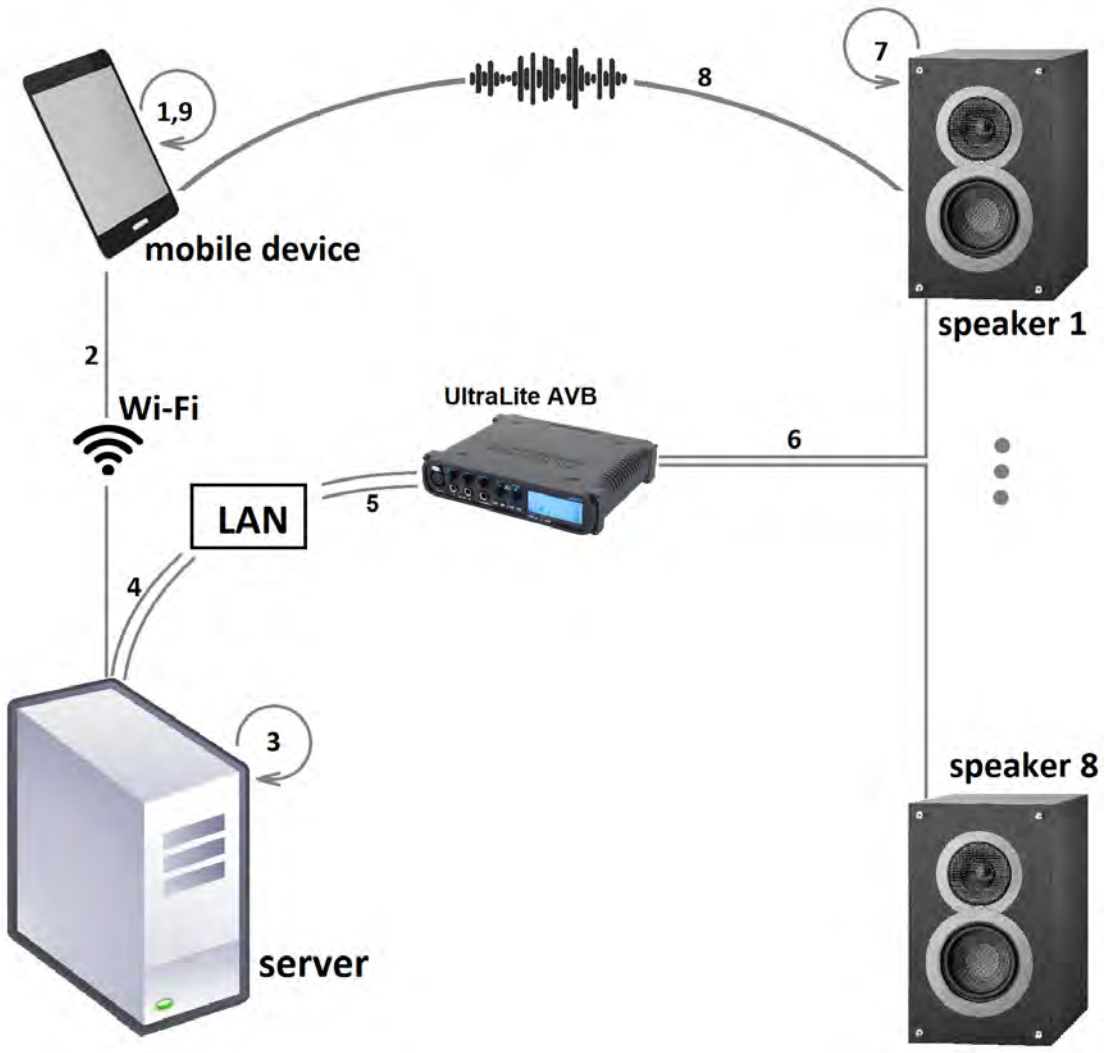


Figure 4.19: Delays in the audio path

to provide for a deterministic play out at the endpoints found on the UltraLite AVB device (IEEE P1722.1, 2013; MOTU, 2019).

6. Within the AVB network as the tones are propagated to the UltraLite AVB. This one delay is bounded, hence deterministic.

7. Digital to analogue conversion takes place in the UltraLite AVB device.

8. the actual sound is transmitted into the air (to be recorded by the microphone). This is ideally the only delay of interest in SDIAS - the only one whose duration should be measured. A means was thus required to obtain this delay as accurately as possible. Calibration is used to determine the latency, as explained in the following

section.

9. Analogue-to-digital conversion, where the analogue signal is sampled to its digital equivalence in the mobile device's microphone. This is a determinate delay.

As previously mentioned, if these delays are mishandled or not handled at all, it would be near impossible to obtain accurate time measurements that would yield accurate 3D speaker coordinates. As such, it became necessary to properly handle them, and this is described in the next section.

4.6 Calibration within SDIAS

A calibration speaker is employed to overcome the various latency issues described in the previous section. This section will describe the speaker measurement process that employs a calibration speaker. Firstly, a microphone attached to the mobile device is placed at some known position, for instance 203cm, from a speaker identified as the nearest of all the available speakers in the layout. This nearest speaker is called the calibration speaker and the measured 203cm distance is called the calibration distance. The calibration distance is measured from the origin, O, as shown in Figure 4.20. The microphone positions must be predetermined such that the microphone can be placed at each of the four locations along the three axes x, y and z with line of sight to all loudspeakers in the layout. The microphone positions are shown by the red dot in the figure.

Another important measurement is the distance from the origin O, equal for all the axes, to each microphone position along the three axes. This distance is represented by OX (= OY = OZ) in the figure. This distance is called the orthogonal distance since it must satisfy the requirement that each position must be orthogonal to the other two positions on the other axes. Both the calibration and orthogonal distances provide a mechanism to measure the time of flight for each speaker tone in SDIAS. The user must measure the two distances, both from the origin, and set their values on the user interface at the start-up process. These distances play a critical role in the accuracy of the coordinates. As such, precision is required in positioning the microphone to yield accurate measurements.

Once the setup of the positions is predetermined, the user can initiate the streaming of audio. Streaming starts off at the origin O, followed by the x, y, z axes and then back to the origin position. The following steps describe the process in detail:

1. the user places the microphone at the origin and then initiates streaming.

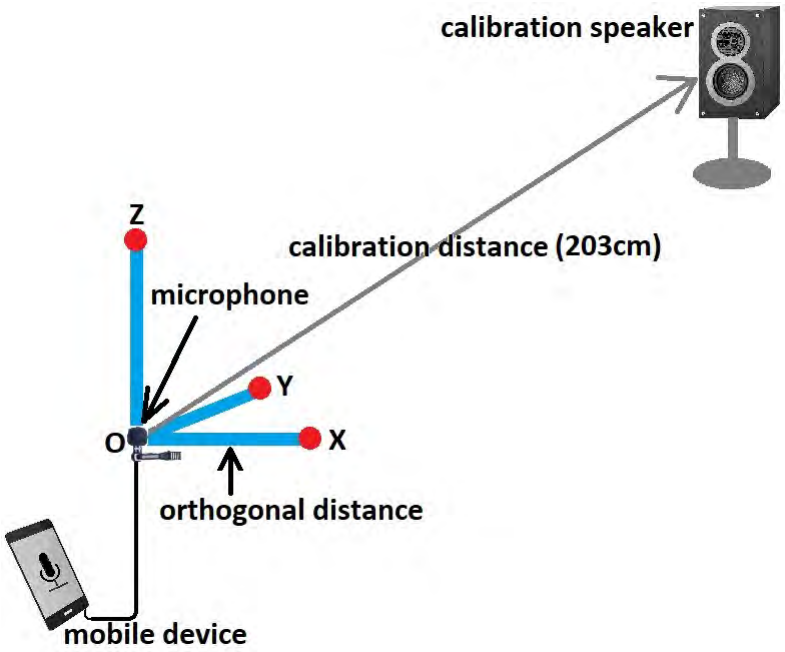


Figure 4.20: Microphone and speaker positions

2. A request for audio streaming is sent by the client to initiate audio transmission by each speaker connected to the UltraLite AVB interface endpoints. The request is sent to the server.
3. the mobile device starts recording instantly at this point. This enables the capture of the tones generated by the speakers for analysis later in the configuration process.
4. the server sends an audio stream request to the JUCE application via a udp socket.
5. The JUCE application sends a tone to the NIC-1 via an ASIO driver destined to the first speaker.
6. NIC-1 packets the audio into smaller AVB frames and sends them to the UltraLite AVB interface (IEEE P1722.1, 2013).
7. The Ethernet AVB protocol, in particular AVDECC Discovery Protocol (IEEE P1722.1, 2013) and Stream Reservation Protocol (IEEE 802.1Qat, 2010) ensure synchronised audio streaming. This ensures that UltraLite AVB endpoint buffers audio stream packets until presentation time (IEEE 802.1AS, 2011) as defined by the network clock master source. Both the NIC-1 and the UltraLite AVB are capable of being the clock master, so any of them is selected during initialisation.
8. The UltraLite AVB interface routes the packets for the tone to the first AVB output stream. This further sends the audio to the connected loudspeaker.
9. the speaker plays out the tone.
10. A certain predefined amount of time passes before the tone is played out by the

second UltraLite audio output streaming as determined by the JUCE application. This periodic tone streaming repeats for all speakers at the current location.

11. the mobile device picks up these tones via the microphone at intervals that are not necessarily as equally spaced as they were generated. This is due to the various distances travelled by the tones emanating from the different speakers. Speakers are at varying distances from the microphone, and as such, the times taken to reach the microphone vary. The effects of these varying distances will be discussed further in the coming sections, specifically Sections 4.6.1 and 4.6.2, while Section 4.7 discusses the details of how the tone distances are extracted from the recorded audio samples and converted to the 3D speaker distances in preparation for 3D coordinate computation.
12. When each speaker has played out a tone, the microphone is moved to the next location, X, on the x-axis as shown in Figure 4.20. The mobile device remains in recording mode as relocation of the microphone takes place. If it were to stop, this approach to the calibration would require a different method, as the start point of the tone generation component would not be able to be synchronised to a single point in the tones reception component.
13. the process repeats for Y, Z and O.

The constant intervals between tone transmission are used later during the configuration process. It is this constant interval feature that provides the opportunity to compute the additional delay incurred by each tone's flight duration at the various positions that are further than the calibration speaker. The condition that the calibration speaker be the closest in the speaker layout is convenient, since the rest of the speakers are further than the calibration speaker. Hence tone transmission times from these other speakers to the microphone will be longer. The TOF of the tone transmitted by the calibration speaker is used as a benchmark against all other TOFs for the tones recorded from other speakers.

The placement of the microphone back at the origin position marks the completion of the streaming and calibration processes. Speaker tones from the calibration speaker at the first and last microphone positions are then used for the next steps in the calibration process, namely synchronisation and syntonisation.

4.6.1 Synchronisation

The two components - the tone generation and tone reception components - involved in measuring time use different audio clocks. As such, it is important to have a comparison

mechanism between the two clocks, such that any clock reading of one component can be mapped or compared to a clock reading in the other component. This is an especially important feature for timing events transmitted between these components.

To determine the clock differences, tones transmitted from the calibration speaker are used. Figure 4.21 illustrates the timing process for determining additional speaker tones' TOFs. This figure shows the tone event times at the server and the client. Speaker 2 is the calibration speaker for the illustration in the following explanation. There are four places where the calibration speaker's tone timestamps are used. On the server and the client, the occurrences are at the first and last origin positions.

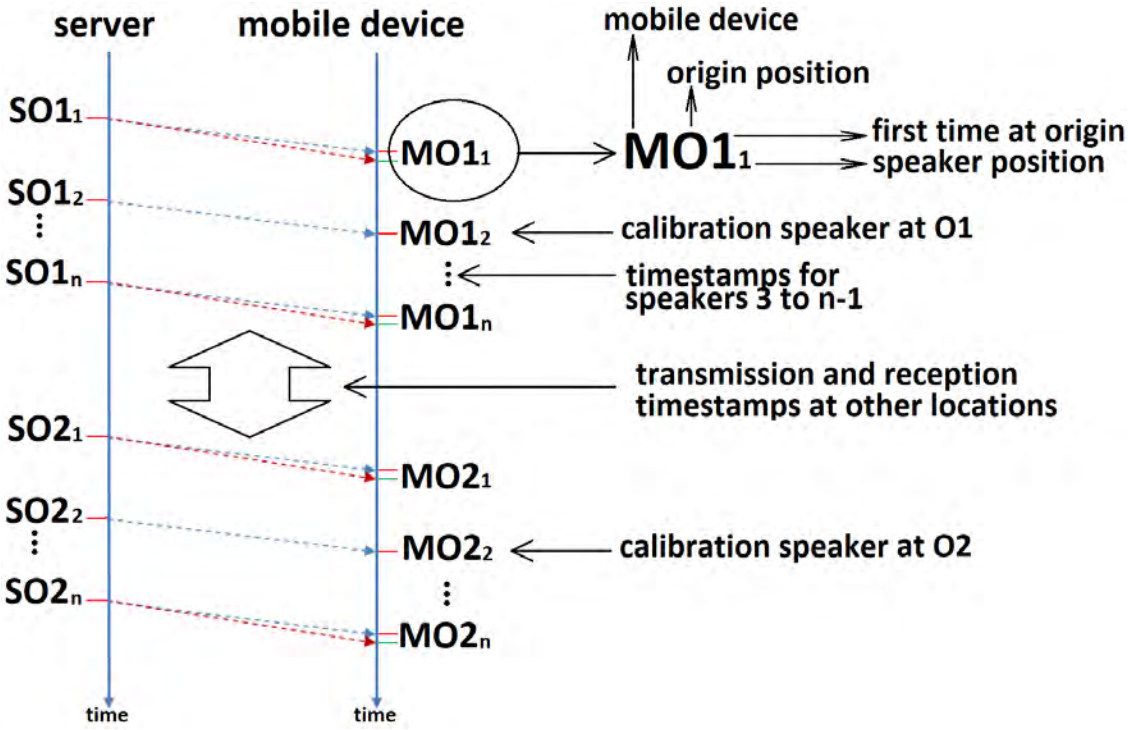


Figure 4.21: Transmission and reception times for the tones

In Figure 4.21, the calibration tone transmission times from the selected calibration speaker to the microphone at the first and last origin positions, via the server, are denoted $SO1_2$ and $SO2_2$ respectively. The corresponding tones received on the mobile device are denoted $MO1_2$ and $MO2_2$. A tone is sent from the server to the first speaker at time $SO1_1$ and is recorded by the mobile device at time $MO1_1$. The second tone is sent from the second speaker at time $SO1_2$ and is recorded by the mobile device at time $MO1_2$. This repeats for the n speakers in the audio system. The tones on the server side are sent at regular intervals and their transmission times are recorded. This results in regularly

spaced timestamps continuing after $SO1_2$ through $SO1_n$. The pattern carries on for the rest of the positions as (SX_1, ... , SX_n), (SY_1, ... , SY_n), (SZ_1, ... , SZ_n) and back to the origin with timestamps $SO2_1$ through $SO2_n$ as shown in Figure 4.21. Similarly, timestamps on the mobile device are generated as tones are received, however, the timestamps do not necessarily occur at the equally spaced intervals as they do on the server for all speakers. This is because of the varying distances of the speakers.

The timestamps from the calibration speaker are used as the reference against which all other speakers' timestamps are measured. The red lines in Figure 4.21 indicate speakers' tone reception timestamps, while the blue lines indicate the times that would be expected if the speakers were at the same distance as the calibration speaker. Speakers at further distances are expected to take longer for the sounds to travel to the microphone than the calibration speaker.

The resultant TOFs from the other speakers would occur slightly later in the timeline than if the speakers were at the same distance as the calibration speaker. The required additional delays were those occurring between each speaker's reference calibration time (time in blue) and the actual time (in red). This would yield the additional distance by which each speaker was further than the calibration speaker. The core of this research rested on the accuracy in determining these distances.

The technique described above synchronises the two clocks on the client and the server. However, another clock problem of clock drifting exists. The next section describes clock drifting and proposes a solution to mitigate the challenge.

4.6.2 Syntonisation

For any two or more clocks at the same nominal clock sampling rate, there are tiny variations in the actual times clocked at each sample due to hardware implementations which result in sample drifts over time (Wolter et al., 2011). However, this was also observed to be true even within a single clock as shown in Figure 4.22. The figure shows two successive image captures[2] of the sample rate measured from NIC-1. The measured sample rate from this figure is highlighted as Reading 1 being 48000.95Hz in the first image and Reading 2 as 48000.77Hz in the other image. These variations are insignificant in general applications but become significant in time-critical applications such as SDIAS.

[2]These were extracted from a video capturing the NIC-1 as its clock drifts, via the Streamware Controller application. A complete screenshot is available in Appendix B, Figure B.7

This is especially true when recordings of audio samples extend over lengthy periods, as is the case while positioning the microphone across all the four locations. The time it takes to position the microphone at these four locations, whilst the microphone is recording, results in the sample rates differential being noticeable. A compensation mechanism was therefore required for this sample rates differential.

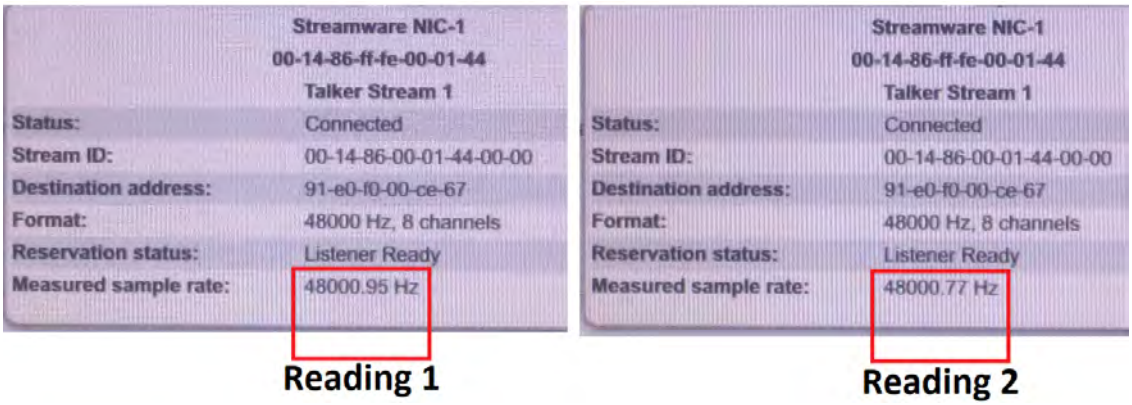


Figure 4.22: Clock drift within NIC-1

To calculate the differential, timestamps of tones from the calibration speaker at the first and last positions of the microphone at the origin position needed to be determined on the server and client. These timestamps are shown in Figure 4.21. $SO1_2$ and $SO2_2$ are the timestamps on the server side for the calibration speaker. Similarly, $MO1_2$ and $MO2_2$ are the corresponding timestamps on the client. The first and last timestamps of the server provide a benchmark for compensating for the drift in the samples rates by finding the server difference, SDiff, between the two timestamps as follows:

$$SDiff = SO2_2 - SO1_2 \tag{4.2}$$

The first and last timestamps of the client measure the corresponding mobile device difference, MDiff, between the two timestamps as follows:

$$MDiff = MO2_2 - MO1_2 \tag{4.3}$$

The resultant ratio of the server and mobile device sample rates is then given by the server and mobile device differences from the two previous equations, and it is termed the sample rates differential (SRD). This is used in later steps, specifically in scaling the values obtained for the distances of each microphone-to-speaker for all the locations. SRD

is given by the following equation:

$$SRD = \frac{SDiff}{MDiff} \tag{4.4}$$

The calibration speaker is used as a reference point in the configuration process. It provides for synchronisation in the time domain, while syntonisation via the SRD, provides synchronisation in the frequency domain. At this point, further processing for speaker distance computations was ready.

4.7 3D Speaker Distances Calculation

The timestamps used in the previous section were recorded in terms of sample positions, that is, the audio clocks used were in terms of samples, as opposed to milliseconds. This meant the readings $SO1_i$ and $MO1_i$ were sample positions corresponding to the samples when a tone was sent to a speaker by the server and received by the mobile device respectively. The first tone is sent and timestamped as the zeroth sample, then after N samples - equal to the duration of samples represented by the interval between tones - the next tone is sent to the second speaker, and so on until a tone is sent out to all speakers in one locations. Then the same thing occurs at the next location. The timestamps need conversion to their equivalent distances to prepare them for use in the coordinate computations.

The sample-to-distance conversion is calculated using the speed of sound in air and the sample rate used in SDIAS. The value of the sample rate used is fixed at 48,000Hz. The value of the speed of sound in air however is dependent on the temperature of the air, as such, it varies with temperature (Watkinson, 2001). This means sound travels faster in higher temperatures than in cooler ones. Consequently, the value of the speed of sound is dependent on the temperature prevailing during SDIAS execution. At a room temperature of 20°C (293.15 absolute temperature), the speed of sound is $343.14 ms^{-1}$, obtained from the equation below (Watkinson, 2001):

$$v = \sqrt{\frac{1.4 \times 8.31\ T}{2.896 \times 10^{-2}}} \tag{4.5}$$

where:

v is the speed of sound given in ms^{-1},

1.4 is the adiabatic constant for air,
8.31 is the gas constant,
T is the absolute temperature, and
2.896×10^{-2} is the molecular weight of air.

This means sound travels 343.14m every second. The two quantities (frequency and sound speed) translate to a ratio of 48,000 samples to 343.14 metres, or 48,000 samples to 34,314 centimetres, simplifying to $48 : 34.314$ as the ratio of samples to centimetres. For example, if a mobile device sent an instruction to the speaker to play a tone, and the additional samples beyond those of the calibration speaker as in Figure 4.21 were 24 samples, it would imply that the speaker is 17.2cm further than the calibration speaker, resulting in 220.2cm as the actual speaker distance after adding the calibration distance of 203cm.

The computation of each speaker's distance from the microphone at each of the four location O, X, Y and Z is required to calculate the speaker coordinates. For an interval of 6,144 samples between tones at the sample rate of 48,000Hz, example start of tone transmission times for the server for the tones in the four locations are as shown in Table 4.1. The interval between each speaker's transmission time at a specific location 6,144 samples. The 6,144 samples interval corresponds to an interval of 128ms, which was sometimes used. The lapse between the transmission time of one location's last speaker position (for instance 43,218 samples for Speaker 8 in location O1) and the next location's first position (for instance 589,824 samples for Speaker 1 in location X) is due to the time taken for the user to relocate the microphone from one location to the next (from O1 to X in this case). The transmission times for each of the speakers at a particular location are an exact multiple of 6,144 samples, since this process takes place in the tone generation component of SDIAS.

Table 4.1: Server sample positions

	O1	X	Y	Z	O2
Speaker 1	0	589,824	1,026,048	2,230,272	2,598,912
Speaker 2	6,144	595,968	1,032,192	2,236,416	2,605,056
Speaker 3	12,288	602,112	1,038,336	2,242,560	2,611,200
Speaker 4	18,432	608,256	1,044,480	2,248,704	2,617,344
Speaker 5	24,576	614,400	1,050,624	2,254,848	2,623,488
Speaker 6	30,720	620,544	1,056,768	2,260,992	2,629,632
Speaker 7	36,864	626,688	1,062,912	2,267,136	2,635,776
Speaker 8	43,008	632,832	1,069,056	2,273,280	2,641,920

For the client tone detection times, all speakers except the calibration speaker (Speaker 2) are further away from the 203cm calibration distance. As such, their corresponding

detection times occur later than the calibration speaker's detection times. Consequently, their distances are going to be longer than the calibration speaker distances. Table 4.2 shows the client tone detection timestamp values corresponding to the server values given in Table 4.1. The values in these two tables represent the transmission and reception timestamps as shown in Figure 4.21.

Table 4.2: Client sample positions

	O1	X	Y	Z	O2
Speaker 1	0	589,874	1,026,058	2,230,318	2,598,965
Speaker 2	6,137	596,013	1,032,219	2,236,458	2,605,103
Speaker 3	12,389	602,253	1,038,428	2,242,704	2,611,354
Speaker 4	18,438	608,242	1,044,533	2,248,759	2,617,404
Speaker 5	24,700	614,514	1,050,736	2,255,014	2,623,664
Speaker 6	30,856	620,652	1,056,919	2,261,173	2,629,819
Speaker 7	37,037	626,911	1,063,111	2,267,341	2,636,002
Speaker 8	43,218	633,089	1,069,271	2,273,523	2,642,250

Using the timestamps from both the server and the client, the sample rates differential can then be calculated using equations 4.2 through 4.4. The ratio is calculated with the timestamps for the calibration speaker (Speaker 2) from Table 4.1 and Table 4.2. From these tables, the server timestamps at the first and last origin locations are 6,144 and 2,605,056. Using these values in equation 4.2 gives the server difference as 2,598,912. Similarly, the corresponding client timestamps at the similar locations are 6,137 and 2,605,103. These give the client difference as 2,598,966, from equation 4.3. The two differences are then used in equation 4.4. The resultant SRD is 0.9999792225061813. This SRD is then applied to the client values in Table 4.2, simply by multiplying each value by the SRD. The values resulting from this operation are given in Table 4.3.

Table 4.3: SRD-rated client sample positions

	O1	X	Y	Z	O2
Speaker 1	0	589,861.744	1,026,036.681	2,230,271.66	2,598,911
Speaker 2	6,136.872	596,000.616	1,032,197.553	2,236,411.532	2,605,048.872
Speaker 3	12,388.743	602,240.487	1,038,406.424	2,242,657.402	2,611,299.743
Speaker 4	18,437.617	608,229.362	1,044,511.297	2,248,712.276	2,617,349.617
Speaker 5	24,699.487	614,501.232	1,050,714.168	2,254,967.146	2,623,609.487
Speaker 6	30,855.359	620,639.104	1,056,897.04	2,261,126.018	2,629,764.359
Speaker 7	37,036.23	626,897.974	1,063,088.911	2,267,293.89	2,635,947.23
Speaker 8	43,217.102	633,075.846	1,069,248.783	2,273,475.762	2,642,195.101

The additional times corresponding to the additional distances travelled by the speaker tones at further distances are obtained by calculating the difference of the client SRD-

rated and server times. For instance, Speaker 1 gets 0; Speaker 2 gets -7.128 from 6136.872 minus 6144; Speaker 3 gets 100.743 from 12388.743 minus 12288. The operation repeats for all the speakers in each location. Additionally, each result is adjusted with respect to the calibration speaker (Speaker 2) such that this speaker has zero additional time. The adjustment is done by subtracting the calibration speaker result across the values. For instance, Speaker 1's additional times are obtained as 7.128 (from 0 - -7.128); Speaker 2 has zero additional time as the calibration speaker; Speaker 3's is 107.87 (from 100.743 - -7.128). Similarly, the operation repeats for all the speakers in all locations. Table 4.4 shows a complete set of the additional times for the speakers in all the locations.

Table 4.4: Additional speaker times

	O1	X	Y	Z	O2
Speaker 1	7.128	44.871	-4.191	6.787	6.128
Speaker 2	0	39.744	12.681	2.66	0
Speaker 3	107.87	135.614	77.552	104.53	106.87
Speaker 4	12.744	-19.51	38.425	15.404	12.744
Speaker 5	130.614	108.359	97.296	126.274	128.614
Speaker 6	142.486	102.232	136.167	141.146	139.486
Speaker 7	179.358	217.102	184.039	165.018	178.358
Speaker 8	216.23	250.974	199.911	202.889	282.228

The additional distances are then calculated from values in Table 4.4. The conversion from the sample position times to distances uses the ratio of the sample rate to the speed of sound. This is 48,000Hz : 343.14m/s, and reduces to 48 : 34.314. For each speaker, the additional samples are converted to distances as follows:

$$D_{Li} = \frac{S_{Li} \times 34.314}{48} + D_c \tag{4.6}$$

where:

D_{Li} is the speaker distance at location L for the i^{th} speaker, in centimetres,
S_{Li} is the additional sample position at location L for the i^{th} speaker and
D_c is the calibration distance of the calibration speaker.

Applying equation 4.6 to the additional sample values in Table 4.4 and assuming D_c is 203 gives the speaker distances shown in Table 4.5. Using the speaker distances in Table 4.5, the 3D speaker coordinates can now be calculated. For example, for Speaker 1, the

calculation for the distance to be used to compute the coordinates becomes:

$$208.095 = \frac{7.128 \times 34.314}{48} + 203 \tag{4.7}$$

Table 4.5: Speaker distances

	O1	X	Y	Z	O2
Speaker 1	208.095	235.078	200.004	207.852	207.38
Speaker 2	203	231.412	212.065	204.901	203
Speaker 3	280.114	299.948	258.44	277.726	279.399
Speaker 4	212.111	189.053	230.469	214.012	212.111
Speaker 5	296.374	280.464	272.555	293.271	294.944
Speaker 6	304.861	276.084	300.343	303.903	302.716
Speaker 7	331.22	358.202	334.566	320.968	330.505
Speaker 8	357.578	382.416	345.912	348.042	404.76

4.8 3D Speaker Coordinates Calculation

The speaker-to-microphone positions are used in the final step of computing the 3D speaker coordinates. Trigonometry rules are used to determine the xyz coordinates of each speaker. One such rule is the law of cosines. This law states that for a given triangle, T shown in Figure 4.23(a), with the sides a, b and c and an angle, θ, facing opposite to c, their relationship is given by the equation (Ryan, 1986):

$$c^2 = a^2 + b^2 - 2abcos\theta \tag{4.8}$$

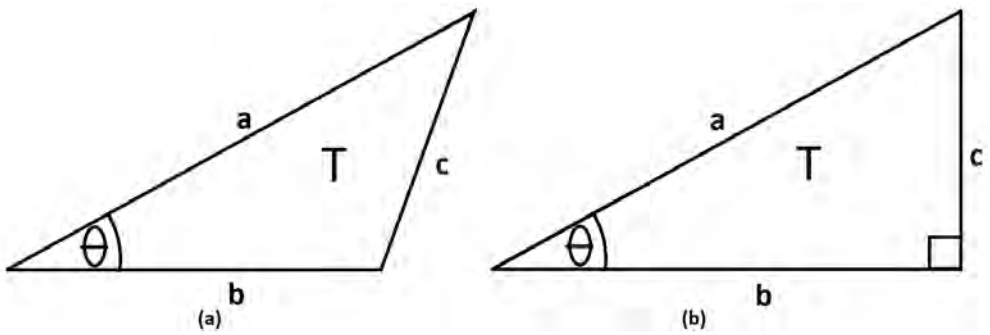


Figure 4.23: Triangle; (b) Right-angled triangle

For the special case where triangle T is a right-angle as shown in 4.23(b), the cosine is given by:

$$cos\theta = \frac{a}{b} \tag{4.9}$$

The two equations 4.8 and 4.9 are used to compute the coordinates for each speaker in SDIAS. The equations could also be manipulated depending on which quantities are known and which are required.

4.8.1 Using All Microphone Positions

Placement of the microphone in the different positions sought to derive triangles similar to the ones shown in Figure 4.23. For instance, the distances of a speaker to the microphone at the origin position and at the position on the x-axis, created a triangle shown in Figure 4.24 with vertices ORS, where O is the origin position, R is the orthogonal distance along the x-axis and S is the speaker position. This figure is a diagram of the model shown in Figure 3.1. After running SDIAS, the involved lengths OS, OR and RS are known quantities, as distances of the speaker from the origin (OS), the orthogonal distance OR and the speaker distance from the x-axis RS.

The triangle ORS is used to calculate the angle θ_x between the lengths OS and OR using following equation:

$$\theta_x = \cos^{-1}\left(\frac{OR^2 + OS^2 - RS^2}{2 \times OR \times OS}\right) \tag{4.10}$$

The angle θ_x from equation 4.10 is obtained from equation 4.8 after solving for θ. The angle θ_x is used later in calculating the value of the x-coordinate, X. X is the point where a perpendicular straight line from the speaker intersects with the x-axis. It is the required x-coordinate of the speaker S and its value is calculated, from equation 4.9, as:

$$X = OS \times cos\theta_x \tag{4.11}$$

θ_x is substituted from equation 4.10 into 4.11 to yield the value of X. The values of X could be negative or positive depending on the position of the speaker relative to the

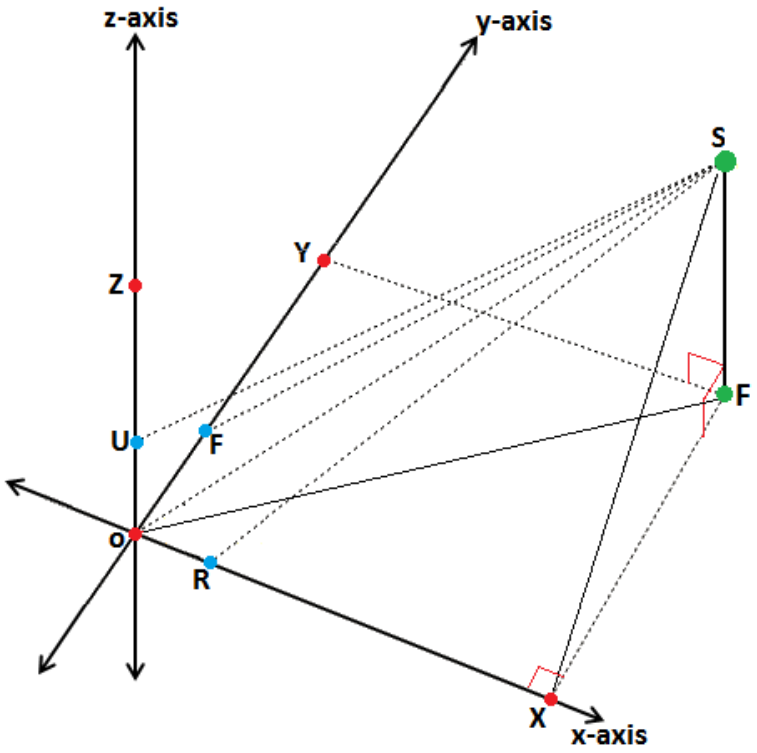


Figure 4.24: Microphone and speaker distances

direction of movement of the microphone along the respective x-axis. If the speaker lies on the same side as the direction of the microphone from the origin to the orthogonal distance, the value becomes positive, else it becomes negative.

A similar configuration on the y-axis is used to compute Y as follows:

$$Y = OS \times cos\theta_y \tag{4.12}$$

using angle, θ_y, between OS and OF calculated as:

$$\theta_y = \cos^{-1}\left(\frac{OF^2 + OS^2 - FS^2}{2 \times OF \times OS}\right) \tag{4.13}$$

Similarly, Z is computed using angle, θ_z, between OS and OU as:

$$\theta_z = \cos^{-1}\left(\frac{OU^2 + OS^2 - US^2}{2 \times OU \times OS}\right) \tag{4.14}$$

and using the calculated angle to compute Z as follows:

$$Z = OS \times cos\theta_z \tag{4.15}$$

4.8.2 Without the Z Microphone Position

The z-coordinate can optionally be calculated using only the x- and y-coordinate values obtained from equations 4.11 and 4.12 respectively. This is done by further exploiting trigonometry. Using Figure 4.24, another right-angled triangle OXF is formed, with vertex X. The length OX is equivalent to the value of X, while XF is equivalent to Y. The length OF, as the hypotenuse of OXF, is given by the sum of the squares of X and Y as follows:

$$OF^2 = X^2 + Y^2 \tag{4.16}$$

This length forms a right-angled triangle OFS, with vertex F and OS as the hypotenuse. F is a projection of S onto the xy-plane. The length FS is equal to Z. Given that OS is known (obtained from the audio recorded from the microphone position) and OF is also known from equation 4.16, the length SF, equals Z, can then be calculated as follows:

$$FS^2 = Z^2 = OS^2 - OF^2 \tag{4.17}$$

which, after substituting equation 4.16 into equation 4.17, reduces to

$$FS = Z = \pm\sqrt{OS^2 - X^2 - Y^2} \tag{4.18}$$

The negative solution from equation 4.18 is discarded as the measurements are taken from the ground, as 0cm, going upwards in the positive direction. This means the z-coordinate can be computed without using audio data from the z-axis microphone position. The option of leaving out the z-axis microphone position is preferred. This is due to the factors which make it difficult to accurately position the microphone at the required position along the z-axis, namely, the inherent inaccuracy of positioning the microphone:

1. at the precise location in the air above the origin,

2. at the required distance of the orthogonal distance, and
3. orthogonal to the other two x and y microphone positions.

Microphone placement on the xy-plane is not susceptible to this problem since the floor provides the required plane, attention is only directed towards the required orthogonal distance. This option is implemented as the default in SDIAS as a means of enhancing user experience because this offloads some burden off the user to strive for placement of the microphone at the z position, which is seemingly difficult to achieve. The user, therefore, does not need to stream (and record) audio with the microphone placed at this location. The option contributes to improved usability of the system, as it would be easier to use the system than when through the alternative.

The Cartesian coordinates for the speaker S are thus, determined in the 3D space as (X,Y,Z) given, in centimetres, from equations 4.11, 4.12 and 4.15 or 4.18. The process repeated for all speakers in the immersive audio system configuration to obtain the coordinates (X1,Y1,Z1), (X2,Y2,Z2) through (Xn,Yn,Zn) defining the positions of the speakers S1, S2 through Sn respectively, where n is the number of speakers. Using distances from Table 4.5, the coordinates are obtained as shown in Table 4.6. An XML file is then created showing each speaker number, x-coordinate, y-coordinate and z-coordinate. The file contents are listed in Listing 4.1. This can be used by the Immergo system (Foss & Rouget, 2015) for sound localisation as described in Section 2.2.1.

Table 4.6: Speaker coordinates

	X	Y	Z
Speaker 1	-184.3	70	16.7
Speaker 2	-190.7	-47.7	2.1
Speaker 3	-176.7	209.5	37.2
Speaker 4	169.2	-120.4	1.5
Speaker 5	168	240.9	45.5
Speaker 6	293.6	60.6	24.7
Speaker 7	-295	-22.1	126.4
Speaker 8	-291.3	151.8	127.2

Listing 4.1: Example XML speaker layout file

```
1 <?xml version='1.0' encoding='UTF-8'?>
2 <config>
3     <speaker number = "1" xpos = "-184.3" ypos = "70" zpos = "16.7"></speaker>
4     <speaker number = "2" xpos = "-190.7" ypos = "-47.7" zpos = "2.1"></speaker>
5     <speaker number = "3" xpos = "-176.7" ypos = "209.5" zpos = "37.2"></speaker>
6     <speaker number = "4" xpos = "169.2" ypos = "-120.4" zpos = "1.5"></speaker>
7     <speaker number = "5" xpos = "168" ypos = "240.9" zpos = "45.5"></speaker>
```

```
8       <speaker number = "6" xpos = "293.6" ypos = "60.6" zpos = "24.7"></speaker>
9       <speaker number = "7" xpos = "-295" ypos = "-22.1" zpos = "126.4"></speaker>
10      <speaker number = "8" xpos = "-291.3" ypos = "151.8" zpos = "127.2"></speaker>
11  </config>
```

4.9 Chapter Summary

This chapter gave descriptions of the various aspects of the technology associated with the SDIAS system. A detailed description of the need for a detection point within a sound source transmitted to the speakers was given, with key aspects being the uniqueness of the point, so that it distinguishable from other parts of the audio data. That led to a discussion of choice of tone from a variety of tested sound sources. The tests led to the choice of a tone to be used, and its structure was described.

Challenges associated with audio recording were presented, particularly ambient noise and latency, which had negative effects on the precision of detection of tones in the audio data. Calibration was described as a solution to the latency problem. Microphone positions were used for streaming audio to the loudspeaker at equal intervals, while the microphone recorded the tones. This enabled synchronisation of the clock times on the server and client components of SDIAS. Syntonisation was explained as corrective technique to overcome the presence of audio clock drifts in networked environments. The technique calculated the drift ratio, termed sample rates differential.

The calculation of speaker distances was also laid out in this chapter. This showed how the sample rates differential obtained during calibration was used to improve accuracy of results. Finally, a description was given, of the computations of the 3D coordinates that used trigonometry, with illustrations on how each coordinate is obtained from the microphone-to-speaker distances at the various microphone locations. The chapter also presented problems associated with the z-axis microphone position, and provided a solution to overcome them. The next chapter describes the design and implementation aspects of the SDIAS system.

Chapter 5

Design and Implementation

This chapter gives a detailed description of the design and implementation processes of the proposed system. As mentioned in Chapter 2, the implemented solution system derives its architectural style from the client-server nature of the Immergo system (Foss & Rouget, 2015). The design and implementation details therefore focus on the client and server components of the implemented system. The initial section focusses on the design aspects, explaining various phases undertaken to fully realise a complete system design model. SDIAS was built mostly on a collection of JavaScript-based technologies. JavaScript was therefore used as the default programming language. The latter sections focus on the implementation, giving explanations of each sub-component of SDIAS and the associated technologies used. Details of some of the challenges encountered during implementation are also described.

5.1 System Design

System behaviour and structure are crucial to visualise and model prior to committing resources for implementation. An object-oriented approach to design was adopted. Requirements for the system were identified and specified, especially drawing from the Immergo system (Foss & Rouget, 2015). These are given in Section 5.1.1. To model the system, the Unified Modelling Language (UML) [1] was used. An unregistered version of StarUML, version 3.2.1, modelling software (MKLabs, 2019) was used to create the various parts of the system model. Usability as a key aspect in design and development

[1] `http://staruml.io/`

of systems also played a role in this system. It was kept at the forefront of the thought process during all the design and development phases of the system. This is because the system must be engaging during user interaction, and it must have intuitive controls and gestures such that the user does not encounter frustration from confusing interaction styles (Sharp, Rogers, & Preece, 2015). As such, a minimalistic approach design would best suitable, where only useful UI is provided by the system.

The behaviour of the system is described using use cases. These define the goals of the system. Use cases are described in Section 5.1.2. Class diagrams describe a static structure of the system. They are derived from the nouns and verbs identified from the system requirements. The classes are instantiated with objects, whose structures and behaviour respectively represent the object's states or attributes and actions they can perform, or which can be performed on them. Class diagrams are described in Section 5.1.3. Sequence diagrams describe the interaction between objects within the system, with emphasis on the temporal ordering of those interactions. These are described in Section 5.1.4.

The implementation discussion focuses on the details of the system components, with source code listings and UI screenshots given to highlight important aspects. The components are the client, Node.js server, the tones server which utilises the capabilities provided by the Ethernet AVB network. Section 5.2 gives the details related to the Ethernet AVB network. The Node.js server, JUCE tones server and client implementations are laid out in Section 5.3.

5.1.1 System Requirements

System requirements were gathered from the Immergo system (Foss & Rouget, 2015). These were used to create the system design model and ultimately the system implementation. The main system functions and/or features of the proposed solution system are:

1. mobile client device: one of the basic requirements for the system was that the client had to be a mobile device. A mobile device such as a smartphone, tablet or iPad would suffice. Mobility is necessary for flexibility and convenience during the microphone placement at the various positions for streaming as it would not hamper movement.
2. user-friendliness: as the main controller of the speaker positioning process, the client device must be easy to operate, use and configure. As such, a simple and

user-friendly UI is necessary. This would allow for intuitive operation of the device, for instance, during the calibration phase of the process, where quantities used by the Node.js server, such as the distance for the calibration speaker and orthogonal distance are set.

3. sound transmission: the system must be able to stream tones via the loudspeakers. This will allow the timing mechanism to take place on the server side.
4. sound recording: similarly to the sound transmission, there must be a mechanism for recording sound played out from the speakers. This contributes to the timing mechanism feature, but on the sound reception side of the system. This requires the capability of the client to record via a microphone.
5. 3D speaker configuration: the system must be able to produce a 3D speaker configuration as a final product. This is done through the calculation of 3D coordinates from the tones recorded by the microphone. The process entails extraction of important details from the recorded audio. These include times of transmission and reception of individual speaker tones, extraction of tone peak positions using these times, through to the conversion of the peak positions to speaker distances and the final coordinates which make up the speaker layout. The 3D speaker configuration is saved in an XML format. The layout can be used by the Immergo system.

The preceding system requirements were grouped into four use cases. The use cases are described in the next section.

5.1.2 Use Cases

Four use cases were created for the system. These encompass all the steps necessary to completely perform the speaker positions measurement process. The use cases are startup, calibrate, stream and configure. Figure 5.1 and 5.2 show the use case diagrams for the client and the server components respectively.

Startup

The *startup* use case entails a series of initialisation steps. The user sets up the client, Node.js server and Ethernet AVB environments so that they can start communication necessary for the speaker positions measurement process. The user must first set up the Ethernet AVB environment. This is done by launching the Streamware Workbench Controller software which manages the timing and synchronisation tasks, as well as transport

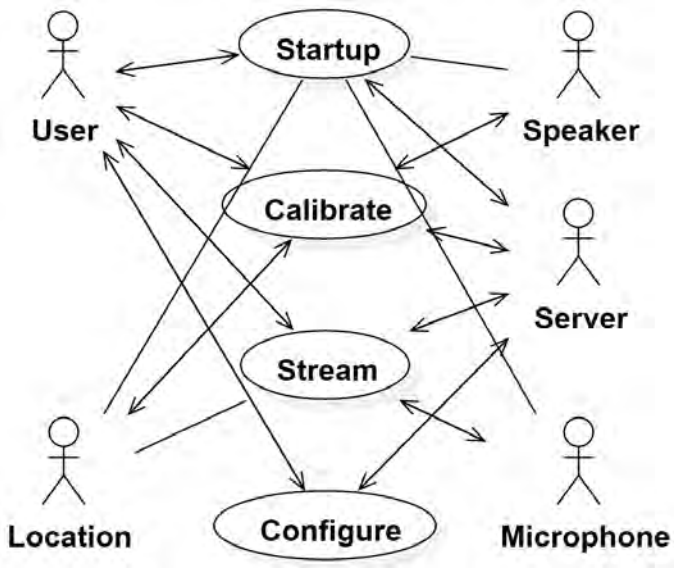


Figure 5.1: Client use case diagram

of audio for the Echo Streamware NIC-1 card (Echo, 2019a). The Controller application performs the necessary tasks for ensuring Ethernet AVB traffic can be transmitted during the *stream* use case later. Also, the user must ensure the AVB UltraLite (MOTU, 2019) interface audio IO streams are routed appropriately, through the AVB UltraLite device web control application. The AVB UltraLite device connects the Ethernet AVB endpoints together.

On the server side, the Node.js server application must first be launched. The Node.js server in turn launches the tones server. The Node.js server establishes a connection between itself and the tones server and awaits connection from the web client.

When the Node.js server is ready, the web client can then connect to the Node.js server. This is done via a browser on the client device. The user types the URL of the Node.js server and the server responds by rendering an HTML page with an embedded JavaScript application to the browser. The JavaScript application contains source code which controls the rest of tasks the user must perform to complete the speaker measurement process, such as acquiring microphone access for recording during the *stream* use case later.

Calibrate

The second use case is *calibrate*. In this use case, configuration of some variables necessary for computations later in the process is performed. Such variables include the calibration

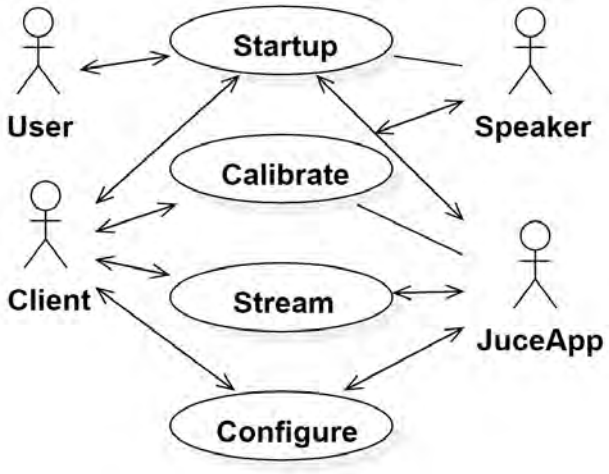


Figure 5.2: Server use case diagram

distance, orthogonal distance and calibration speaker selection amongst others. The calibration distance is the distance from the microphone to the nearest speaker selected as the calibration speaker. The orthogonal distance is the distance from the origin O equidistant to each of the positions along the xyz axes. These distances are shown in Figure 4.20. The values of these quantities are set on the client device UI and saved by the client. They are transmitted to the Node.js server for use in the computations of the speaker distances and the resultant 3D coordinates later in the *configure* use case. The user can select a speaker to beep and send a beep request to the selected speaker to identify it for confirmation as the calibration speaker.

Stream

The next use case is *stream*. It starts once the calibration is completed. This entails transmission of short impulses to the speakers. The user requests for impulse transmission at each of the locations. At the same time, the client device starts recording via an attached microphone. A high resolution timing mechanism to capture impulse transmission and reception times is deployed in this phase as the Ethernet AVB network streams audio. The user can be assured of this task when impulses are heard from the loudspeakers. Once the client has recorded audio from all the locations, especially at the origin for the second time, streaming stops.

Configure

The final use case is *configure*. Processing of the recorded audio data into the final 3D coordinates is performed in this phase. The following several steps provide an overview of the process. The client:

- saves the recorded audio,
- searches for the first tone to use for trimming. Trimming is done to reduce the amount data to be transferred across the Wi-Fi network,
- trims the audio, using additional configuration information sent by the Node.js server,
- sends the trimmed audio to the server, along with configuration information, for instance, the calibration distance and orthogonal distances, for the server to complete the remaining processes.

When the server receives data from the client, it:

- extracts speakers' tone peak positions at each location,
- calculates speaker distances from the peak positions,
- computes coordinates from the distances,
- creates speaker layout using the coordinates in an XML format and
- sends the layout and the distances to the client.

The client displays the results at the end of this process, which comprises the distances and 3D coordinates received from the server. This marks the end of the *configure* use case. Depending on the accuracy of the measurements, the user can then use the speaker layout for instance, for spatialisation in the Immersive audio content creation systems, such as the Immergo system, or other coordinates consuming applications. Otherwise the use can restart the process if unsatisfactory results were obtained.

5.1.3 Class Diagrams

Class diagrams creation followed after the use cases were completed. These were created through the identification of the various object interaction messages within the system. The UML class diagram contains the classes from which these objects are instantiated, as well as their relationships. Class methods define actions performed on or by the class instances (objects in the system), while the class attributes define some aspect of an object. Some example client classes, and their associated methods and attributes include:

- *Window*: This class instantiates the global browser *window* object automatically when a web browser page is opened. The *window* contains a Document Object Model (DOM) document (MDN, 2019b). Amongst its numerous events, its *load* event is used to ensure the client application is only launched when the entire web page has completed loading all its resources (MDN, 2019c).
- *Application*: this is the main class. It is instantiated when the *window* fires the *load* event, detected by its *addEventListener()* method. The *Application* instance controls instances of all other classes. This class has its only member function, the *start()* method, which is used to start and control the rest of the processes involved in the measurement of speaker positions. It has *audioData* as one of its attributes, which holds the audio samples for the tones recorded by the microphone.
- *CalibrationWindow*, a dependent class of the *Application*, has various sub-views for setting various variables as explained in the use cases. The *CalibrationWindow* has the properties *calibrationSpeaker* and *orthogonalDistance*. The *calibrationSpeaker* has properties *speakerNumber* and *speakerDistance*.
- *Locations*, a property of the *Application* instance, holds the names of the four locations where the microphone captures sound.
- *Speakers*, this is a property of the *Application* instance. It holds the positions of the loudspeakers available in the speaker configuration.
- *Microphone*, this is a property of the *Application* instance. It captures the sound played out by the loudspeakers. Its methods include *startRecording()* and *stopRecording()*.

Figure 5.3 shows various objects and methods for the client class diagram.

Similarly, server classes, together with their methods and attributes were also created. These are:

- *Server*: this is main application class. Its methods include the *computeCoordinates()*, *calculateSampleRateDiffential()*, *createXMLConfiguration()* amongst others, while some of its attributes are *roomDimensions* and *juceMessages* which hold the test room dimensions and messages destined for the tones server - *JuceApp*, respectively.
- *HttpServer*: this is the web server class. Its attributes include the *response* and *request*, and it has a *listen()* method, as one of its methods.
- *JuceApp*: this is the tones server application class. It has attributes such as the *numberOfSpeakers*, the *sineTone*, the *sampleRate* and the *bufferSize* amongst others. Its methods include the *prepareToPlay()* and *getNextAudioBlock()*.

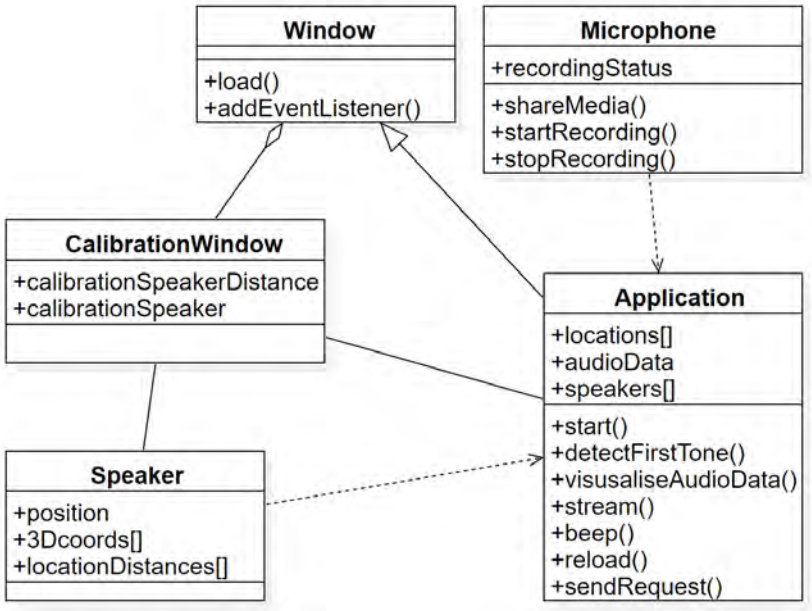


Figure 5.3: Client class diagram

- *Speaker*: this is a class which defines a speaker object similar to the client speaker class.
- *AVBNetwork*: this is the Ethernet AVB network class. Its attributes include an *interfaceCard*, a *switch* and a network master clock attribute - *grandMasterClock*.

The objects and their attributes and methods are shown in Figure 5.4.

5.1.4 Sequence Diagrams

The final step in the design process was the specification of a system behaviour. This gives a description of what the system does, showing an exchange of messages in a chronological order. From the message exchanges of the objects in the class diagrams, sequence diagrams were created. This was done by extracting messages and requests from each use case. Their responses were then established. This means a single use case can result in multiple messages, requests and responses being exchanged between the involved objects. The final result of this process is the client and server sequence diagrams.

The client as the main system controller, initiates most of the tasks. These can be either through user interactions, such as beeping a speaker, or a completion of some task initiated by the user, such as the creation of speaker objects initiated by a discovery

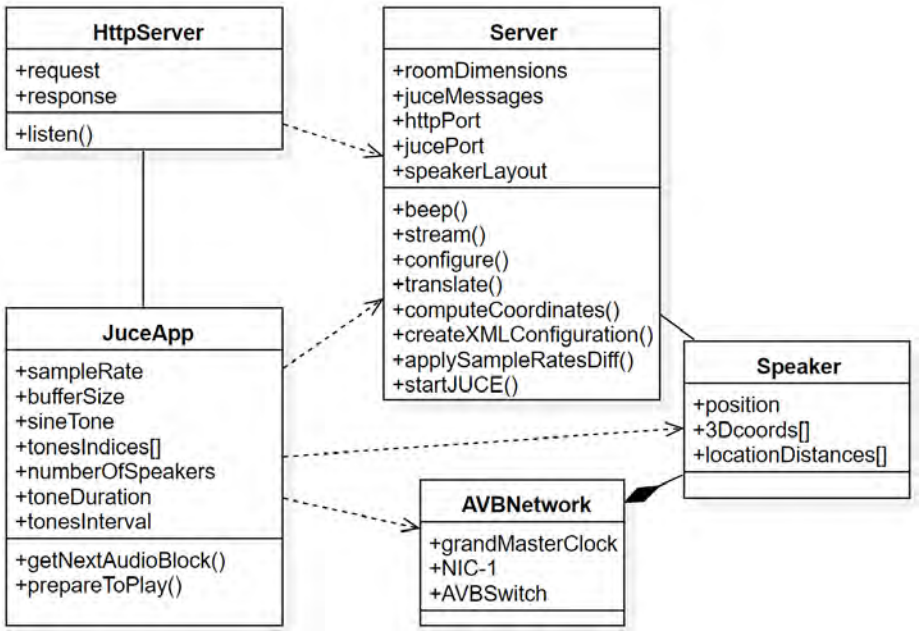


Figure 5.4: Server class diagram

message response received by the client. The user tasks trigger interactions amongst various objects in the system. As an illustration, an exchange of messages between the server and the client for a startup processes are shown in Figures 5.5 and 5.6.

The server startup messages are as follows:

- the user launches the web server program.
- the web server then launches the tones server application.
- the tones server establishes connection with the audio device - the AVB Streamware NIC-1 card.
- the NIC-1 card grants access to the tones server as an audio device.
- the web server sends a request for tones server information to the tones server.
- the tones server responds with the tones server information.
- the web server saves the tones server information.
- the server finally starts listening to the client connections.

Once the server startup processes are successfully completed, the client startup messages can be executed as follows:

- the user types the server URL into the browser.
- the browser sends a request to the typed URL's server.
- the server responds by supplying the HTML web page, with an embedded JavaScript application.

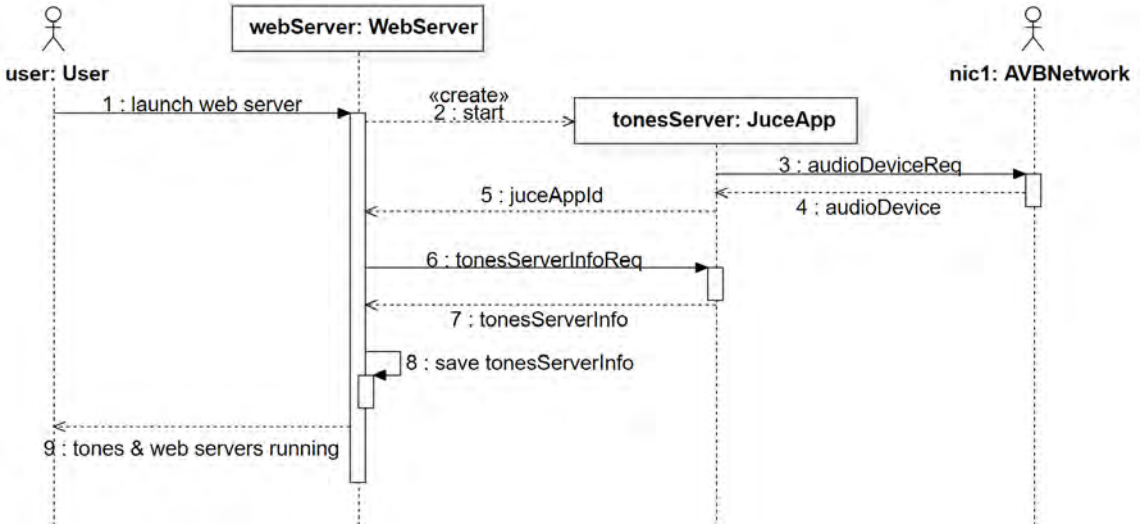


Figure 5.5: Server startup sequence diagram

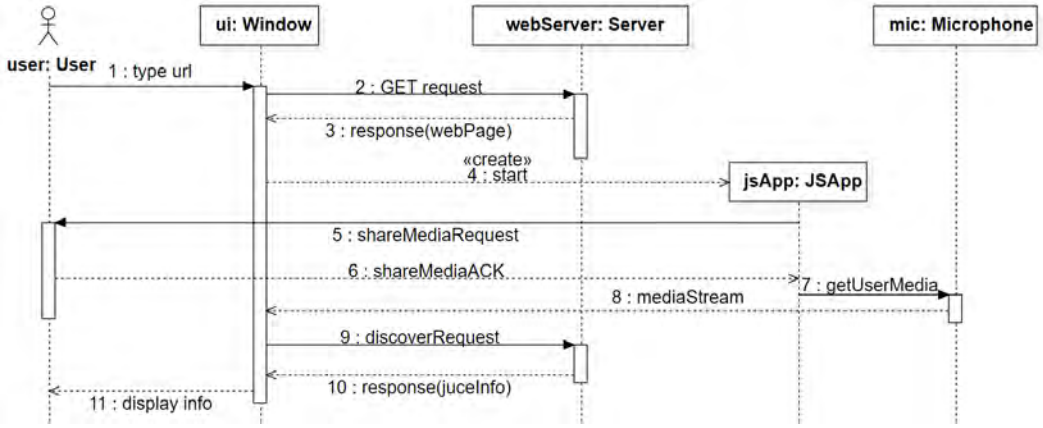


Figure 5.6: Client startup sequence diagram

- the client application requests for media access permissions from the user.
- the user grants media resources to the client application.
- the client application gets access to the microphone.
- the microphone grants the client application access to stream.
- the application sends a discovery request to the server.
- the web server responds by supplying tones server information.
- the browser displays the information to the user.

The completion of the sequence diagrams marks the end of the design phase. Implementation of the system, based on the design model, starts with the Ethernet AVB network implementation described in the next section.

5.2 Ethernet AVB Network Configuration

As briefly introduced in Section 3.3.3, the Ethernet AVB standard is used in SDIAS to meet the critical time requirements needed for the measurement process and to emulate the Immergo system network. The AVB Echo Streamware NIC-1 card (Echo, 2019a) and UltraLite AVB device (MOTU, 2019) were used to form the AVB network. The Streamware NIC-1 card doubles as a network interface card and a sound processor. The card has only one Ethernet jack, as a result, an AVB switch is required to extend the network so that more network devices can be connected. Such a device switch is the UltraLite AVB device. Connections amongst the hardware devices (the Windows host PC, the Echo Streamware card, the UltraLite AVB and the Genelec speakers) form the physical links of the network. Immergo uses the SPK-4Ps speakers (miniDSP, 2020) as the Ethernet AVB endpoints. These were not used in this research project as they were not continuously available, hence the use of the described configuration as the next best alternative.

Figure 5.7: The Ethernet AVB Echo Streamware NIC-1 card (Echo, 2019a)

The Streamware NIC-1 card is shown in Figure 5.7. This is used, in particular, to:

- transmit and receive the Audio Video Transport Protocol (AVTP) (IEEE 1722, 2016) streams,
- act as the Ethernet AVB network general Precision Time Protocol (gPTP) (IEEE 802.1AS, 2011) grandmaster clock, and
- act as sound card with ASIO support.

The Echo Streamware Workbench Controller (Release 2.6.16) software for the Streamware NIC-1 interface is used to set up the gPTP configuration as well as the Talker and Listener audio streams (Echo, 2019b). The streams are formatted according to the AVTP. The Controller software transmits and receives audio via an ASIO driver. The ASIO driver will

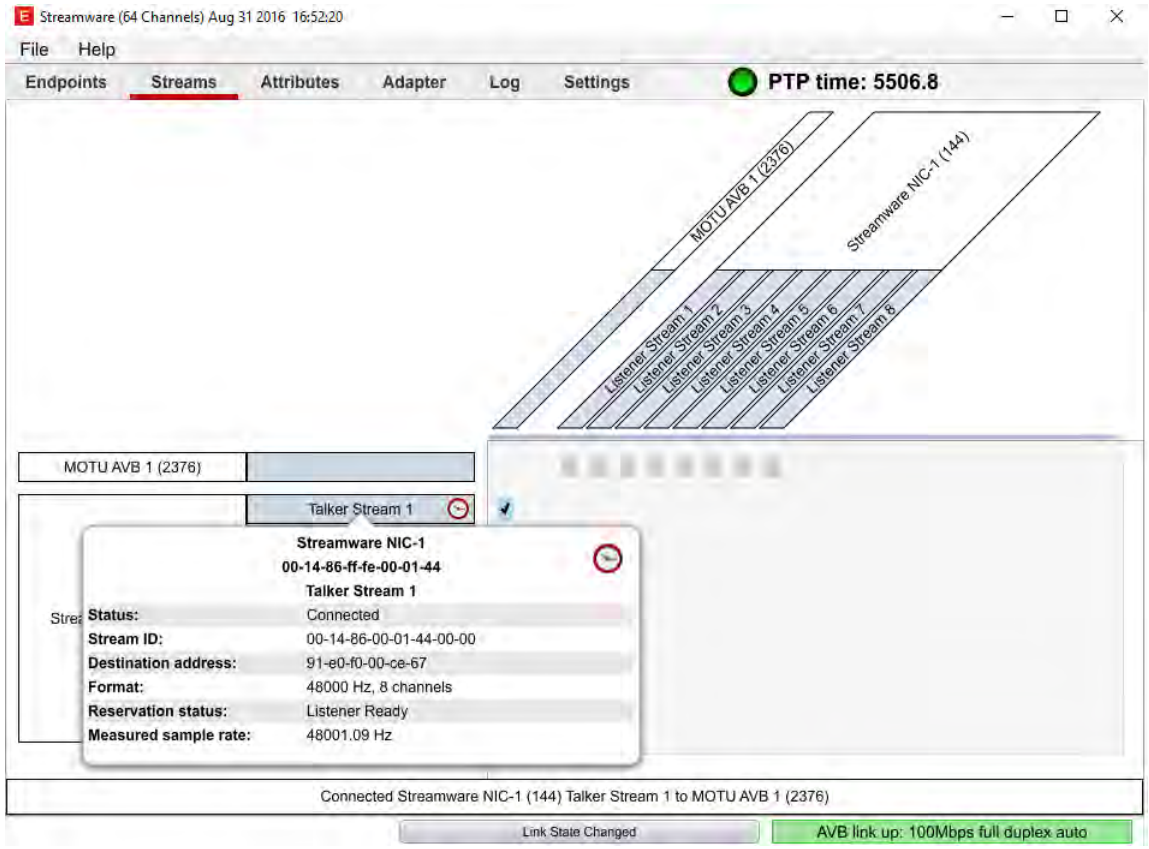


Figure 5.8: Echo Streamware Controller UI

interface with the Streamware NIC-1 card, packaging the audio channels within AVTP streams. Figure 5.8 shows the Workbench Controller application Streams tab UI. The UI shows some of the following Ethernet AVB key features:

- the Talker: this is the Streamware NIC-1 card - an audio source,
- the Listener: this is the MOTU AVB 1 (the UltraLite AVB device) - an audio sink, and
- the gPTP grandmaster clock: this is provided by the Talker.

The Ethernet AVB input streams need to be routed to the analogue outputs. The Workbench Controller enables a user to connect the Talker streams associated with the Streamware NIC-1 card to the Listener streams associated with the UltraLite AVB device.

The UltraLite device is shown in Figure 5.9, front (A) and rear (B) panels. The device has, amongst other ports, up to eight analogue output ports, enough to connect eight loudspeakers for use in SDIAS. The liquid crystal display on the UltraLite AVB device, annotated as 1 in the figure, displays various information, for example, the sample rate and level meters for analogue IO endpoints. The Streamware NIC-1 card Ethernet port

connects to the UltraLite AVB device with a standard Ethernet cable on the AVB Ethernet port (annotated as 2) to create an AVB audio network.

Figure 5.9: UltraLite AVB (MOTU, 2019)

Eight loudspeakers are physically connected to the UltraLite AVB output ports - two in the *MAIN OUT* stereo analogue output ports (shown as 4), while the other six are connected to *ANALOG OUT* output jacks (shown as 3 in the figure). These eight endpoints connect to the 1029A Studio Monitor Genelec (Genelec, 2019) speakers to produce sound.

The UltraLite AVB device has a control web application. The control web application provides audio IO routing via a matrix on the UI. This routing matrix must be configured correctly to ensure that AVB input streams are appropriately routed to the expected AVB output streams. Figure 5.10 shows the web controller application routing tab. The figure shows the routing matrix. In the figure, eight NIC-1 input AVB streams are routed to the eight UltraLite AVB device output AVB streams, such that each input AVB stream is routed to the output AVB stream according to the following mapping:

- 'AVB Stream 1 1' to 'Main L'
- 'AVB Stream 1 2' to 'Main R',
- 'AVB Stream 1 3' to 'Analog 1'
- 'AVB Stream 1 4' to 'Analog 2'
- 'AVB Stream 1 5' to 'Analog 3'
- 'AVB Stream 1 6' to 'Analog 4'
- 'AVB Stream 1 7' to 'Analog 5'
- 'AVB Stream 1 8' to 'Analog 6'

The next section discusses the implementation of the various components of SDIAS, which utilise the capability of the Ethernet AVB standard to determine the loudspeakers coordinates.

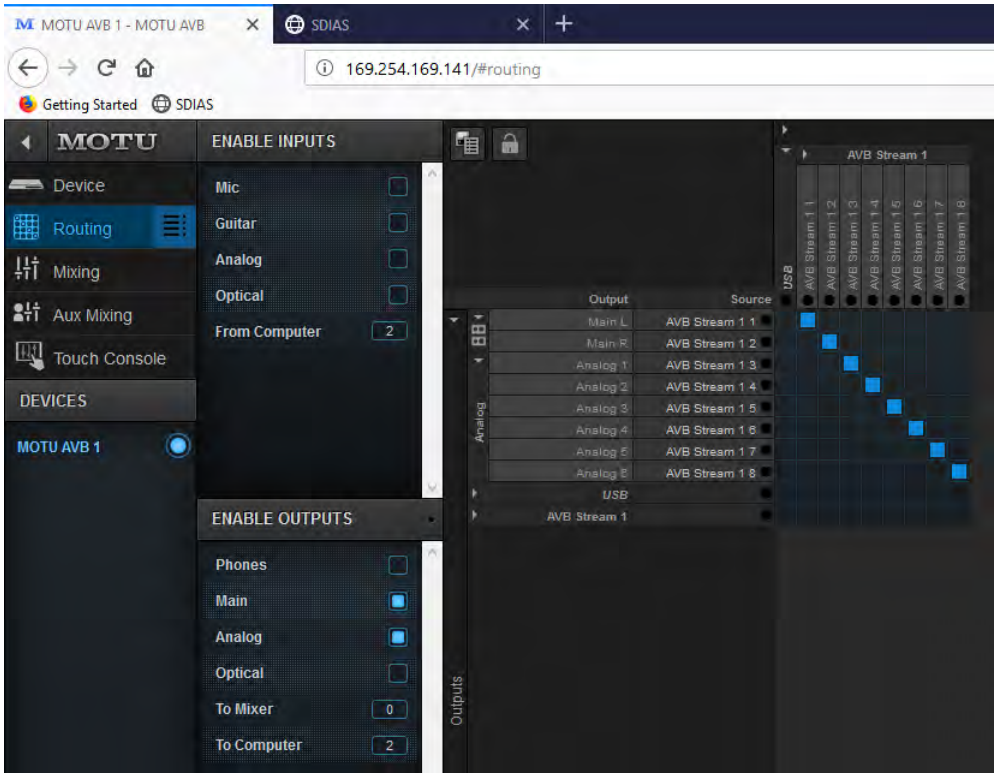


Figure 5.10: Audio IO routing matrix in UltraLite AVB web control application.

5.3 Implementation of SDIAS Use Cases

To fulfil the tasks of the use cases, the components of SDIAS, namely the Node.js server, the tone server and the web client, communicate by sharing messages amongst themselves throughout the process of speaker positions measurement. These messages are in the form of requests from the clients and the responses sent by the servers. For instance, when the web client places a stream request to the web server, the web server relays this request to the tones server, becoming a client placing the request in this instance. The tones server responds by streaming audio the loudspeakers. The messages are discussed in the following sections.

5.3.1 The Server Environment

The server environment consists of some hardware and software. The hardware component consists of the following:

- PC: a Windows-based PC, running version 10 (Microsoft, 2019). This was used throughout the project.

- Echo Streamware NIC-1 card: an audio IO device was required for processing audio. This was introduced in Section 5.2.

The software components primarily consist of JavaScript technologies based on the Node.js platform. JavaScript is used for coding the server application. The server application was built using Node.js[2]. Node.js is an asynchronous event-driven runtime environment built on top of the Google Chrome V8 JavaScript engine and designed to build scalable network applications (The Node.js Foundation, 2019b). The Node.js environment comes with a default command line interface at installation. It can run simple to complex JavaScript programs which traditionally only ran on the various browsers such as Google Chrome, Mozilla Firefox or Microsoft's Internet Explorer.

The JavaScript libraries, called *modules*, are contained in *packages* which are managed via a command line program called *npm*. The *npm* program ships together with Node.js during installation. The program fetches Node.js packages externally from a repository of third-party Node.js packages, itself called *npm*[3]. External packages can also be from a developer's code base, that is, code written to be imported as modules for other applications. A module call is made through the Node.js *require()* function.

Some of the built-in modules include the *HTTP* module. The module is used in the SDIAS server to enable the HTTP functionality. This behaviour is required to handle client HTTP requests and to issue HTTP responses back to the client. For instance, to enable this functionality, the module is included and accessed via the *http* variable throughout the application, as follows:

Listing 5.1: Enabling the HTTP functionality

```
1    let http = require('http');
```

Express[4] is an open-source web framework based on Node.js. It is designed to simplify tasks for Node.js during development, therefore providing a fast, flexible and easier approach to building websites, web applications and web APIs (The Node.js Foundation, 2019a).

The server functionality is contained in a file *sdias_server.js*. The server related functions are described later where the descriptions of the four use cases are given. A basic functionality of the *sdias_server.js* source code file is shown in Listing 5.2[5].

[2]https://nodejs.org
[3]https://www.npmjs.com/
[4]https://expressjs.com/
[5]Code lines correspond to those from the source code file.

The coding design pattern used for the server (and the client) is the JavaScript namespacing pattern. It is used together with object literal notation, to encapsulate the object behaviour of the server. Namespacing helps declutter the global environment, and potentially avoid collisions.

The server object is accessed via the *SDIASServer* variable. The object function is further created as an *immediately invoked expression function* (IIFE) (MDN, 2020a). As the name suggests, the object function is immediately executed, eliminating the need to explicitly call in later in the source code.

Listing 5.2: Basic SDIAS server functionality

```
16  var SDIASServer = SDIASServer === undefined ? {} : SDIASServer;
17  SDIASServer = ( function () {  // Nodejs server - this project's system server
29      var express = require('express'),    // express web framework module
30          app = express(),    // create the app
31          http = require('http').Server( app ),    // create the HTTP server
32          io = require('socket.io')( http ), // mount socket.io on nodejs http server
39          dgram = require('dgram'),
40          udpClient = dgram.createSocket('udp4'),
53          httpPort = 4400 ,   // port for http connections
54          udpPort = 4401,
104     // directory for static files
105     app.use ( express.static ('clientApp'));
106     app.use ( express.static ('utilities'));
107
108     // route for home
109     app.get('/', function(req, res) {
110         res.sendFile(__dirname + '/clientApp/index.html');
111     });
112
113     // LISTEN FOR SOCKETS
114     io.on('connection', function ( socket ) {
191     });
192
193     // LISTEN FOR JUCE RESPONSES
194     udpClient.on('message', function ( buf, remote ) {
254     });
1135    http.listen( httpPort, function ( err ) {
1136        if ( err ) {
1137            console.log("SDIAS server couldn't start");
1138            return console.error(err);
1139        }
1150        var juceExeId = startJUCE(dur), // start juce
1157                discoverAVBNetwork(Object.keys(udpMessages)[udpMessages.discover]);
1161    });
1162 } ());// end of IIFE - the server
```

5.3.2 The JUCE Framework

Due to the critical timing requirements for SDIAS, a high-resolution timing mechanism was required both at the audio transmission and reception components. A sub-sample resolution was required to precisely timestamp each tone transmission on the server. A similar mechanism was also required on the client for each tone received by the microphone. None of the available JavaScript client APIs nor *npm* packages could provide such a mechanism which provide precision and accuracy required by SDIAS. An external framework was therefore used for this purpose. Such a framework is called *Jules Utility C++ Extension (JUCE)*[6]. JUCE provides cross-platform capabilities for audio processing across many audio formats in C++ (JUCE, 2019). It uses low-level platform features to manipulate and process audio, therefore providing the low latency required for SDIAS. An Audio Stream Input/Output (ASIO) API is available within JUCE for the provision of low-level audio processing (Steinberg, 2019). ASIO functionality must enabled prior to its usage. This means (downloading and) installing the drivers for ASIO. ASIO SDK2.3[7] drivers were used for SDIAS.

The JUCE tones server functionality is implemented in the *MainComponent.cpp* C++ file[8], implementing four methods of the audio processing class object, called the *MainComponent* in the JUCE context. Three of these, *MainComponent()*, *prepareToPlay()* and *getNextAudioBlock()*, are the *MainComponent*'s instance virtual methods. One of them, *listenForConnections()*, was newly created. These methods are discussed in the relevant sections, in relation to the use cases and where each is used.

5.3.3 The Client Environment

The SDIAS client was built using a collection of web-based technologies as a single-page-application (SPA). These technologies are HTML5[9], CSS3[10], Bootstrap[11] and vanilla JavaScript latest version - ECMAScript version 6 (ES6)[12]. Additionally, a set of web APIs worked in concert to fulfil some of the required functions. For instance, web audio (Adenot & Toy, 2018), web real-time communication (Burnett et al., 2018) and media capture and

[6]https://juce.com
[7]https://www.steinberg.net/asiosdk
[8]This is generated during audio project creation. Details can be found in Appendix D.3
[9]https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5
[10]https://developer.mozilla.org/en-US/docs/Archive/CSS3
[11]https://getbootstrap.com/
[12]http://es6-features.org/

streams (Burnett et al., 2019) APIs were used to access media resources such as the client microphone and other audio processing tasks. Further details are given in later sub-sections when such feature implementations are described.

Host Device and Attached Microphone

Several mobile devices (smartphones and tablets), including PCs mostly during the development phase, were used as client host devices. However, one was extensively used, especially during the testing phase. The mobile device extensively used was the *Samsung Galaxy Tab S2*[13]. The tablet operates an *Android OS version 7.0*, codenamed *Nougat*[14].

Several microphones were tested, and an analysis was done to determine how (well or badly) they performed with the various sound sources in Section 4.2 and 4.3. The microphones included the UMIK-1 microphone[15], the PMIK-1 microphone[16], Panda microphone[17] and the embedded microphone within the Samsung Galaxy Tab S2. All of these microphones were omnidirectional, which makes them the right candidates for acoustical measurement tests since this microphone directivity type is recommended for room acoustic measurements (Hak, van Haaren, Wenmaekers, & van Luxemburg, 2009). However, the embedded microphone within the tablet performed poorly during the microphone tests. It suffered challenges such as line of sight obscurity by the tablet body shape, and as such, performed least favourably than the other three microphones. The other three were externally connected to the host devices via USB adapters.

Based on the results of the microphone tests, the Panda microphone performed best and was therefore selected for use in carrying out the experiments extensively during the testing phase of SDIAS. The microphone is omnidirectional, plug-and-play and extensible to about three metres (3m) (Purple Panda, 2019). These features allow for flexibility and mobility and contribute to the easy-of-use and user-friendliness features of the system during microphone placement at the various locations during the measurement process. Microphones with other types of directivity, such as cardioid, ambisonic or bidirectional, are generally not recommended for acoustical measurements, and as such, they must generally be avoided. Nonetheless, cardioid microphones could potentially be used due to their good sound pickup from the front and sides while deploying the SDIAS mechanism

[13]https://www.samsung.com/ph/tablets/galaxy-tab-s2-9-7-t815/
[14]https://www.android.com/versions/nougat-7-0/
[15]https://www.minidsp.com/products/acoustic-measurement/umik-1
[16]https://www.minidsp.com/products/acoustic-measurement/pmik-1-detail
[17]https://www.purplepandastore.com/products/purple-panda-lavalier-microphone-kit

since the placement of the microphone on the floor essentially demarcates the sound source landscape as the sides and front.

5.3.4 The Server Startup

The server performs several tasks. These are responses to client requests (client as the main controller) and those related to the JUCE application. Most of the intensive processing tasks, such extracting tone peaks, also occur on the server.

From the perspective of the server processes, the first step is to launch the server application. A simple and basic method of running a Node.js application is through the Windows command prompt by issuing the following command:

> node sdias_server.js

The command launches the application in the default *development* mode[18]. There are several tasks common performed when the server is launched. The server

- sets up various variables required for a successful execution of the measurement process. For instance, these include variables to hold the number of speakers, sample rates for both JUCE tones server and the SDIAS client and recorded audio data. Additionally, modules required for various functionality are also included. Such include:
  - the *Express* module (loaded from an external package in Line 29 of Listing 5.2). The Express application can then be used to add middleware as required, as shown in Lines 105-106, which respectively define the directories for the client root folder and utilities.
  - the *HTTP* module. This object is used to create a server (Line 31) with the Express application using the method chaining approach, which allows as many methods as required to be invoked on an object via the *dot notation* common in many object-oriented programming languages.
  - the *socket.io* (in Line 32) and *dgram* (in Line 39) external modules for client and JUCE real-time communication channels respectively.
- defines the only endpoint for the Express application for SDIAS to which clients can route requests. This is performed in the Lines 109-111. The *get* method requires, as arguments, a URL and a callback which specifies the response to send back to the client placing the request.

[18] Alternative approaches to running the Node.js server can be found in Appendix D.2

- establishes two real-time connection channels: One of the established *Internet Protocol* communication methods is through the use of the *transmission control protocol* (TCP) for traffic. The TCP-oriented *socket.io*[19] module is used for relaying messages between the SDIAS server and its client(s) in real-time. The other connection used in SDIAS is the *user datagram protocol* (UDP) connection. This is used for transmission of messages between the SDIAS server and the JUCE tones server, via the *dgram* module.

  The *socket.io* and *dgram* use the Node.js's *event-driven* behaviour to raise and handle *events* for their respective objects called *emitters*. The event-driven behaviour uses the *publisher-subscriber* design pattern. The *on()* function of an *event emitter* object is used to *register or subscribe listeners* to a named event. Event emitters also trigger, raise or publish events via the emitter's *emit()* function. Events are triggered later in the process.
- starts listening for connections expected from the client. The *http* server invokes its *listen()* function to start listening to connections from the clients, as shown in Lines 1135-1161.
- starts the JUCE application through a custom function. This is done via a call to the custom function *startJUCE()* in Line 1150. The function executes the JUCE executable. The JUCE application is launched as a child process of the server process, which conveniently dies when the server shuts down. An external module, *child_process*, is used to enable the functionality related to child processes.
- sends a discovery request to the JUCE application for the necessary variables set by the JUCE application, in Line 1157. The required information includes the tone duration, length of the interval specified between each tone playback, number of speakers and the set buffer size for the audio device. The quantities requested from the JUCE application are saved in their respective variables on the Node.js server. The discovery task is executed only when the JUCE application has started successfully.

5.3.5 The JUCE Startup

The JUCE tones server application performs several tasks in order to get the tones streaming process ready for client requests. Many methods of the *MainComponent* object perform these tasks, but the tasks of interest are performed by the following three methods:

[19]https://socket.io

- *MainComponent()*: the audio device's IO channels are set within this method. One input channel was set, while eight output channels were set for each loudspeaker in the SDIAS layout. The *listenForConnections()* is also invoked in this method.
- *listenForConnections()*: this establishes a communication channel between the JUCE tones server and the Node.js server. It then starts listening for incoming requests from the Node.js server. These expected messages are the:
  - discover request: it informs the tones server to send information related to itself. This include the sample rate, buffer size, number of speakers and tone duration.
  - beep request: it informs the tones server to beep a specified loudspeaker.
  - stream request: it tells the tones server to start streaming tones to the loudspeakers.

  The handling of these requests is discussed later at the use case where they occur.
- *prepareToPlay()*: this method prepares the audio application for playing. Initialisation of the application variables occurs in this method. For instance, the ASIO functionality setup is done in this method, as well as initialisation of the variables used within the application. The audio sample buffer variable, *myTone*, is filled with the sine waveform as part of the initialisation. Listing 5.3 shows a portion of this method that creates the sine wave audio sample buffer. The buffer is filled with low-amplitude cycles, only the fourth one is a high-amplitude cycle[20]. This is the sine wave with an early high amplitude cycle, described in Section 4.3 and shown in Figure 4.17.

Listing 5.3: Filling the buffer with the sine waveform within *prepareToPlay()*

```
61      for (int sample = 0; sample < 500; ++sample) {
62          //========== early impulse ==========//
63          if (sample < 75)
64              myTone->setSample(0, sample, (float)0.2 * (float)std::sin(sample * delta));
65          // make fourth cycle high low amplitude
66          if ((sample >= 75) && (sample < 100))
67              myTone->setSample(0, sample, (float)0.9 * (float)std::sin(sample * delta));
68          // make the rest even lower
69          if ((sample >= 100) && (sample < 500))
70              myTone->setSample(0, sample, (float)0.2 * (float)std::sin(sample * delta));
71      }
```

Getting the Node.js and JUCE tones servers started sets up the necessary environment for the SDIAS client to start interaction with these servers.

[20] At a frequency of 1,920Hz used for this sine wave and the sample rate of 48kHz, 1 cycle is completed by 25 samples (1 cycle x 48,000 samples / 1,920 cycles).

5.3.6 The Client Startup

The user must launch the client to start the speaker positions measurement process. The user types the Node.js server's URL on the mobile device's browser. The URL is comprised of the IP address of the server, together with port number, for example, the user would type '146.123.231.88:4400', where 146.123.231.88 is the IP address of the Wi-Fi adapter on the server host computer, and 4400 is the port the server application is bound to. The server responds by sending the only HTML page for SDIAS clients.

The HTML Webpage

The HTML page, *index.html*, has three sections commonly found in standard webpages. The web page sections are as follows:

- the header section: displays the title of the application - **SDIAS** as an abbreviation, and the full title (**Speaker Determination for Immersive Audio Systems**) below it.
- the main content section: this is where all the UI pertaining to the measurement processing is displayed. Figure 5.11 shows a portion of the UI for the main content (a complete UI is shown in Figure B.1 of Appendix B). The UI comprises the following:
  - three buttons, which are:
    - beep: used for beeping a selected speaker for auditory identification during the calibration phase
    - stream audio: this sends a stream audio request to the server so that audio is streamed to the speakers at a location.
    - reload: restarts the measurement process at the discretion of a user.
  - current step: this shows the name of the current step in the process. This is shown as *SDIAS Startup* in Figure 5.11. This changes according to various steps throughout the process.
  - instructions: instructions are used to guide the user at each intermediary step of the process. Alternatively, it shows results at the end of the process. Also, some configuration information is provided. This is the rest of the content below the *current step* element in Figure 5.11.
  - recorded audio visualisation: the page has a HTML5 canvas element for displaying the recorded audio visualisation. The visualisation is drawn on the

canvas after recording audio. An example visualisation is shown in Figure 3.15.

- the footer section: this is the copyright information - year and the Rhodes University Distributed Audio Networks Group link.

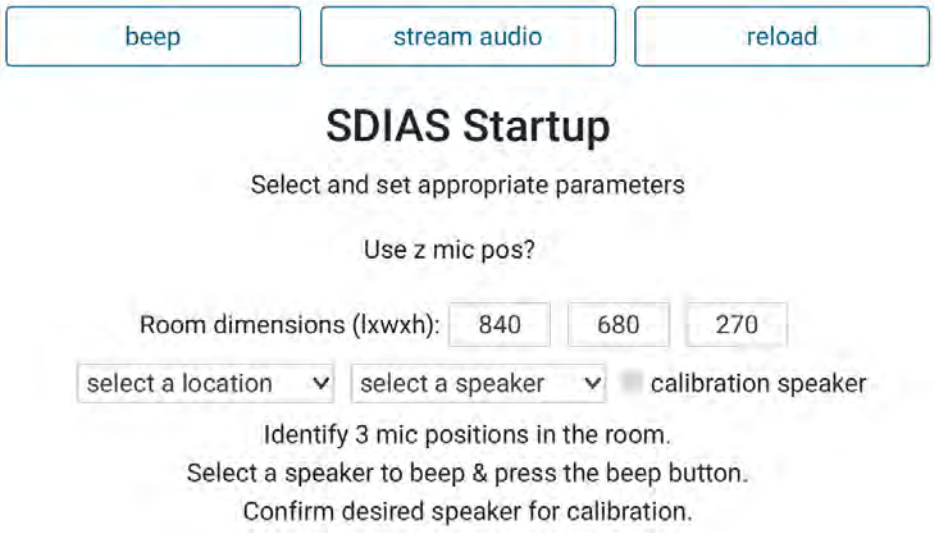


Figure 5.11: Client Main Content UI portion

Embedded in this page is the JavaScript client application source code which controls all the functionality of the client.

The JavaScript Application

The controller behaviour of the SDIAS client is defined in the JavaScript file called *app.js*. The application behaviour defines all of the functionality required to fulfil the system requirements described in Section 5.1.1. The client JavaScript application is also built with the same design patterns as the server application. These are the JavaScript namespacing, using the object literal notation and as an IIFE. Listing 5.4 shows a skeleton of the client application. The client object, *SDIASClient*, defines a function, *start()*, which implements the entire functionality of the client. This function is exposed to the global environment (Line 13) so that it can be accessed the by the browser global *window* object. The functionality of *start()* is only required when all the entire web page has loaded. This is shown by line 928 of Listing 5.4.

Listing 5.4: Skeleton of the SDIAS client application

```
9  var SDIASClient = SDIASClient === undefined ? {} : SDIASClient;
```

```
10
11 SDIASClient = ( function () {  // this project's system client
12     // expose member functions & variables to the public
13     return {
14         start : function() {
61                 appMediaStreamConstraints =  { audio : true, video : false }; // media ↩
                       constraints for this app (only audio is required, no video is ↩
                       required)
920             // get media device access
921             navigator.mediaDevices.getUserMedia ( appMediaStreamConstraints )
922                 .then ( mediaSuccess )
923                 .catch ( mediaError );
924         } // ends start function
925     }; // ends public exposure
926 } ());// end of IIFE - the client
927 // start the application when the page has loaded
928 window.addEventListener("load", SDIASClient.start);
```

The SDIAS client needs to record the tones played out by the loudspeakers, as such, it needs to use a microphone. Several web APIs are used to provide this capability. These APIs conveniently provide a mechanism for web applications to capture sound using various media resources, such as the microphone. These web APIs are the:

- *web audio*: the API allows for processing of audio at low level, therefore allowing capturing and manipulation of raw audio (Adenot & Toy, 2018),
- *media capture and streams*: the API sets up an environment for requesting media input devices such as microphone for audio and web camera for video (Burnett et al., 2019). It allows a web client to place requests for permission for accessing available media hardware resources such as the audio recording device, video recording and/or screen sharing service, and
- *web real-time communication (webrtc)*: webrtc provides mechanisms for data or information sharing amongst web clients (Burnett et al., 2019).

To enable the use of the microphone, the client must first require media access permissions. This functionality is enabled by using the *media capture and streams* API. The API works together with the *webrtc* API to stream media over browsers (Burnett et al., 2019). The *media capture and streams* API requests for the permission to use the microphone with the *navigator.mediaDevices* interface's *getUserMedia()* method. This is shown in Line 921 of Listing 5.4. The microphone access permission is granted by tapping the *Share* option on the UI, as shown in Figure 3.7.

The *getUserMedia()* method implements one of ES6's new features known as *promises* (MDN, 2020b, 2020c). A promise allows asynchronous behaviour to be handled as if in a synchronous manner. The *promise* is said to *resolve* if completed successfully or it is

otherwise *rejected*. So, when the user grants the application access to media resources, *getUserMedia()* resolves to an object of type *MediaStream* - a media content stream. This is executed by Line 922 of Listing 5.4. If the user denies the application permission, the promise is rejected with an error (Burnett et al., 2019). This is achieved by Line 923. The *MediaStream* object contains information regarding video or audio tracks, depending on the media constraints passed to *getUserMedia()*. For SDIAS, only audio is specified as no other types of media streams are required. This is done by passing the media constraints object (shown in Line 61) which only needs audio to the *getUserMedia()* method. The tracks are used for capturing (or recording) related media, or for generating media content.

In order to record the tones played by the loudspeakers with the microphone, the *web audio API* is used. The behaviour of SDIAS client for capturing audio is defined in the *mediaSuccess()* method. The method is implemented as a success or completion function, that is, it executes tasks if the user grants permission for media access. Listing 5.5 shows some key aspects of the *mediaSuccess()* method.

Listing 5.5: The client's *mediaSuccess* function

```
760             mediaSuccess = function( mediaStream ) {
761                 // cross browser AudioContext creation
762                 var AudioContext = window.AudioContext || window.webkitAudioContext;
763                 audioContext = new AudioContext();
768                 clientBufferSize = 512; // smaller buffer size reduces latency in audio↩
                        processing/rendering
774                 var audioInput = audioContext.createMediaStreamSource ( mediaStream ),
775                     noOfInChans = 1,    // input channels
776                     noOfOutChans = 1,   // output channels
779                     recorder = audioContext.createScriptProcessor ( clientBufferSize, ↩
                            noOfInChans, noOfOutChans ), // recorder
780                     channel = []; // store audio channel data ie audio samples
781
782                 recorder.onaudioprocess = function ( audioProcessingEvent ) {
783                     if ( !recording ) { // return if recording is false
784                         return;
785                     }
786                     channel = audioProcessingEvent.inputBuffer.getChannelData ( 0 );  ↩
                            // get current data
787                     audioData.push(channel); // push data into array
788                 };
806             }, // ends mediaSuccess callback
```

The web audio API provides an *AudioContext* interface which represents an audio routing graph. This is an audio processing environment from which various audio nodes (that is, audio inputs, outputs, processors such as filters, and control such as gain) are created and linked together, as well as processing of audio. The *web audio* API is able to capture the raw audio signals using a *ScriptProcessorNode* interface. In the *mediaSuccess()* method,

a script processor, *recorder*, is created with a buffer size of 512, one input channel and one output channel. This script processor provides an interface directly to the raw stream data from the microphone. This is the data which needs to be saved and analysed for tone peaks. The buffer size determines the size of the input and output channel buffers. It also controls the frequency at which the input channel buffer is filled with audio samples.

The audio data accessed through the script processor nodes is in a linear 32-bit float pulse code modulation format. This format is represented as *Float32Array* object in JavaScript. Whenever the script processor's input buffer is filled with data, an *audioprocess* event is triggered to the event handler, *onaudioprocess*, of the script processor. Within the event handler, the 512-element input buffer data is copied to the *channel* array variable. This only occurs when the client is in recording mode. The *channel* data is then pushed into the *audioData* array to save the recorded data. The *audioData* array holds all the audio data of the tones recorded as the microphone is placed at each of the locations. This means each element of *audioData* variable is a 512-element array of 32-bit floats, making the variable a two-dimensional array.

On the other hand, *mediaError()* implements the *getUserMedia()* promise's rejection, that is, what must happen in the case of a user denying access to media resources. The *mediaError()* method performs this by only returning a *null* object.

The user proceeds to the next step, which is calibration, once the microphone access is granted.

5.3.7 The Client Calibration

Calibration requires the user to set up some necessary variables via the UI shown at startup (Figure 5.11). These are:

- the option to use audio data recorded at the z position: this is an option which the user can choose, though it is not preferable due to the inherent and increased inaccuracies involved with the use of data from the z position. This is discussed in Section 4.8.2.
- room dimensions: the length, width and height of the testing room are used in the verification of the obtained results. It is therefore necessary for the user to set them.
- the location: at any point during the measurement process, the user needs to specify the location for streaming (in the order origin, x-axis, y-axis and z-axis and origin for the second time). At the beginning, the required option is the origin as the initial streaming location for the process.

- speaker: the user needs to iteratively select and beep speakers from the drop down list, to determine which of the speakers will be used as the calibration speaker.
- calibration speaker confirmation: this confirms the selected speaker as the calibration speaker. The user must select this option before continuing.
- calibration distance: once the user has decided (via an explicit confirmation by checking the confirmation checkbox) on the calibration speaker, preferably the nearest speaker to the microphone positions, the user must measure this distance and enter it in the calibration input.
- orthogonal distance: the user must also measure the orthogonal distance and type it in the corresponding input box.

All the information configured on the client during this step is not sent to server until later after audio recording. For the *calibration* use case, the Node.js server does not perform any significant actions besides relaying a beep request for the calibration speaker identification. The next phase is to start the streaming process.

5.3.8 The Client Streaming

The audio streaming process starts only when the microphone is at the origin. Audio needs to be captured and saved as the microphone is moved to each of the locations. The end of streaming is signalled when the user has completed recording at the origin for the second time.

The user presses the *stream audio* button on the UI at each of the locations, starting at the origin, so that the loudspeakers can play out the tones. The button click invokes the *stream()* function that implements the streaming functionality. Listing 5.6 shows some features of the *stream()* function.

Listing 5.6: The client's *stream* function

```
230         stream = function() {
234             var streamingLocation = locationsDropDown.selectedIndex,
235                 reqName = "stream",
244                 if ( streamingLocation === 1) { // @ origin
245                     if ( !recording ) { // first time @ origin
246                         audioData = []; // clear audio array
247                         startRecording();
248                     } else { // second time @ origin
249                         audioType = "data";
250                         streamingLocation = 5;
251                         stopRecording();
252                     }
```

```
253                     }
254                     // send the stream request to the server
255                     sendRequest(reqName, streamingLocation - 1);
325             },
```

The microphone is set to a capturing mode by invoking the *startRecording()* (Line 247) when the *stream()* function is called for the first time at the origin. From this point onwards, the *audioData* array starts being filled with the audio data (Lines 786 and 787 of Listing 5.5). A *stream* request is sent to the server (Line 255 of Listing 5.6). This request specifies the name of the request and the current location of the microphone. The name of this request is used by the Node.js server's *socket.io* object to distinguish a request for streaming from other types of requests. The *socket.io* object is listening (or expecting) for this event. This is the event-driven behaviour discussed in the Node.js server's startup section earlier.

Subsequent calls to *stream()* will only send a request to the server and not alter the microphone's capturing mode. When the *stream()* function is invoked for the second time at the origin, it will execute the *stopRecording()* function which will stop the audio capture. The location information sent in the request is used by the tones server as a signal to stop transmitting tones. At this point, *audioData* array is filled with all the necessary audio data to start the configuration process to determine the 3D coordinates.

5.3.9 The Server Streaming

The Node.js server receives various requests from the web client. These include the stream messages. When the server receives a stream message from the client, it needs to relay it to the JUCE tones server. The Node.js server listens for the incoming messages from the web client as shown in Line 114 of Listing 5.2. The received message is accessed through the *socket* argument parameter object of the callback function available for inbound connections. The server determines the message type by inspecting the *socket* custom event and the associated data object.

Custom events destined for the JUCE tones server are defined by the Node.js server in a 5-key/value object variable *udpMessages*. The five keys are *discover*, *stream*, *calibrate*, *configure* and *beep* and their corresponding values are *0* through *4*.

The outbound messages to the JUCE tones server are sent by the Node.js server's *sendUDPMessage()* method. This method uses the *send()* method of the *udpClient* object (created in

Line 40 of Listing 5.2). Once the server determines that the incoming messages is a *stream* audio request, it invokes a *streamAudio()* method. This method uses *sendUDPMessage()* to send the data received from the *stream* event. The data specifies the location of the client microphone at the time of request. Additionally, *streamAudio()* invokes another helper method, *sendID()*, which sends the microphone location in the case of a stream request.

A *beep* request is handled similarly to the *stream* request. A number corresponding to the speaker position sought for beeping is sent through the *sendID()* method in the case of a *beep* request. The eight loudspeaker numbers start from *0* for the first loudspeaker, through *7* for the last loudspeaker in the SDIAS loudspeaker configuration.

5.3.10 The JUCE Streaming

The *getNextAudioBlock()* method performs the critical timing tasks. Figure 5.12 shows how audio data is processed by this function. The method is automatically repeatedly called by the ASIO driver, each time getting the next block of audio data as the audio device hardware requires. The frequency of these invocations is dependent on the buffer size and sample rate of the audio device. The buffer size determines the number of samples the *getNextAudioBlock()* method fetches at each call. The timed calls also help with a sample-accurate timing of events as the calls are made to meet the specified (or default) audio device sample rate. At the specified sample rate of 48kHz and the buffer size of 512 audio samples, the method is called approximately every 10.667ms (512 samples x 1000ms/48000 samples). This is the upper time limit for which all data processing must be complete before the next method invocation.

Listing 5.7 shows how the *getNextAudioBlock()* method performs audio processing.

Listing 5.7: Audio processing with *getNextAudioBlock()*

```
78 void MainComponent::getNextAudioBlock (const AudioSourceChannelInfo& bufferToFill)
79 {
80     float * const buffer = bufferToFill.buffer->getWritePointer(currentSpeaker, ↩
           bufferToFill.startSample);
81     const float *myToneBuffer = myTone->getReadPointer(0);
82     if (numSamplesSinceStart + sdiasBufferSize < startOfNextTone) {
83         bufferToFill.clearActiveBufferRegion();
84     }
85     else {
86         FloatVectorOperations::copy(buffer, myToneBuffer, sdiasBufferSize);
87         bufferToFill.buffer->applyGain(gain);
88         startOfNextTone += samplesBetweenTones;
```

```
89          tonesCount++;
90
91          switch (statusCurrent) {
92          case statusIdle:
93              bufferToFill.buffer->applyGain(0.0f); // mute when not beeping or streaming
94              break;
95          case statusStreaming:
96              bufferToFill.buffer->applyGain(gain);
97              currentSpeaker = (currentSpeaker + 1) % (numberOfSpeakers);
98              if (currentSpeaker == muteSpeaker) {
99                  statusCurrent = statusIdle;
100             }
101             break;
102         case statusBeeping:
103             bufferToFill.buffer->applyGain(gain);
104             if (tonesCount == stopBeepingIndex) {
105                 currentSpeaker = muteSpeaker;
106                 statusCurrent = statusIdle;
107             }
108             break;
109         }
110     }
111     numSamplesSinceStart += sdiasBufferSize;
112 }
```

The buffer, *bufferToFill*, dispatched at each call to *getNextAudioBlock()* contains eight channels - one per speaker. This buffer must be filled with audio data for output. All the channels are either filled with zeroes (silence) or only one of them is filled with the sine wave data. These are shown in Figure 5.12. The figure shows a buffer filled with silence - also referred to as the silence buffer, as well as a buffer with one of its channels filled with the sine wave data - also referred to as the sine buffer, represented by the orange and green boxes respectively. Additionally, a channel filled with silence and another filled with the sine wave data are shown. A count of calls to the *getNextAudioBlock()* method is shown, which occurs every 512 samples. Also, a count for the occurrences of the sine buffers, *tonesCount*, is shown, which occurs every 4,608 samples.

The choice of when to use the sine wave is determined by the interval at which the tones must be played out by the loudspeakers. The choice for the channel to fill with the sine wave data depends on the loudspeaker to steam to, which depends on the type of request received by the tones server. The beep request specifies a loudspeaker position, while the stream request requires that the tones server must always start streaming to the first loudspeaker.

For the first call to *getNextAudioBlock()*, silence will be chosen[21], as it is not yet time for

[21] The program branches into the *if* block in Line 82 as $numSamplesSinceStart + sdiasBufferSize = 0 + 512 < startOfNextTone = 4608$.

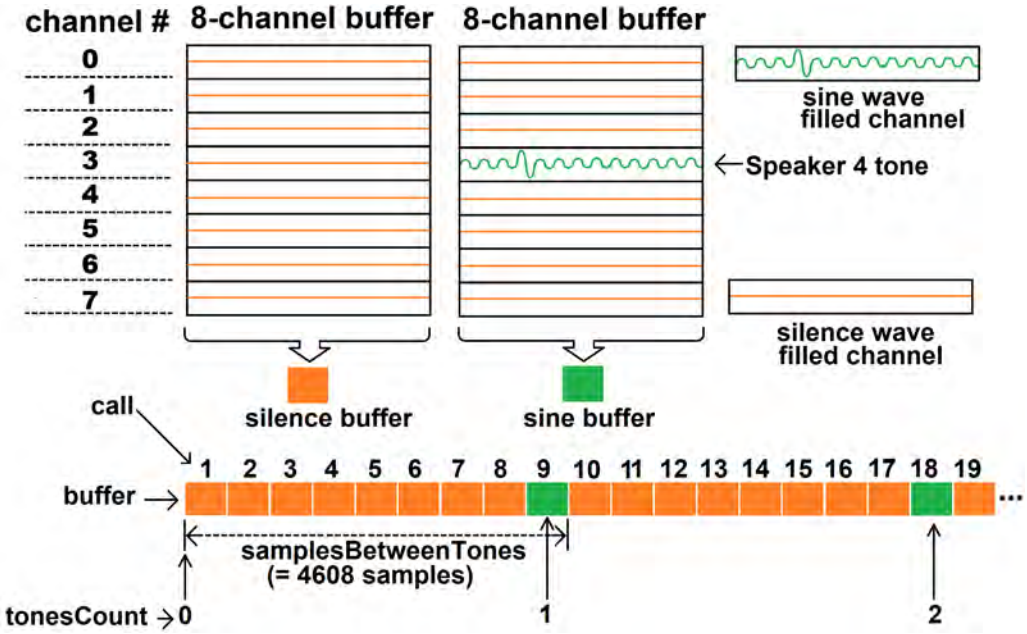


Figure 5.12: Audio processing in JUCE

a tone. The *getNextAudioBlock()* method clears the buffer, that is, fills zeroes for all eight channels. The interval between tones is specified by *samplesBetweenTones*, initialised to 4,608 samples at startup. This controls when next to fill the buffer with the sine wave data. For the 512-sample long (specified by *sdiasBufferSize*) buffer and 4,608 interval, eight calls to *getNextAudioBlock()* will be made before it is time to fill the buffer with the sine wave data, that is, eight buffers will be filled with zeroes. These are shown in Figure 5.12 as a series of eight silence buffers. The sine wave data is filled at the ninth buffer (code branches into *else* block in Line 85; the blue box in the figure). The tones server tracks the occurrence of each sine buffer (the buffer filled with the sine wave data) with the *tonesCount* integer variable, which becomes one at the ninth call to the *getNextAudioBlock()* method, two at the 18^{th} call, three at the 27^{th} call and so on.

When the sine wave data is filled, the *getNextAudioBlock()* method, with the help of *listenForConnections()* method, uses three states to decide how to handle the sine buffer. These states are the:

1. idle state: this occurs when no request is received from the Node.js server or when the status needs to be changed to idle from the beeping or streaming states described below. A gain of zero is applied to the buffer (Line 93), effectively silencing it.
2. beeping state: this is triggered by a beep request from the Node.js server. A beep must be played from a loudspeaker specified by the user. A non-zero gain is applied to the sine buffer (Line 103). The number of beeps is defined by *numOfBeeps*, set

to 1 at initialisation. The channel is determined by *currentSpeaker*, obtained along with beep request. This determines the *bufferToFill*'s channel number for writing the sine wave data (Line 80). The status is set back to idle when enough sine buffers have been filled (tracked by the *stopBeepingIndex* variable).

3. streaming state: this is triggered by the stream request from the Node.js server. The first speaker is set to play the tone (Line 97) as this state is preceded by the idle state from the previous eight silence buffers. This means the buffer will fill the first channel with the sine wave audio data. The stream request comes along with the location position (0 through 4). This is used to set the corresponding element of the *toneIndices[]* array with the value of the *tonesCount*. At the first stream request, the location position is 0, so *toneIndices[0]* is set to some *tonesCount* t_{o1}, which is the value of *tonesCount* when the sine buffer is streamed at location O1. Eight silence buffers are dispatched before reaching this state for the second time, which is when the sine buffer must be dispatched to the second speaker. This repeats until eight sine buffers, one per loudspeaker, are dispatched to the output device. The state is then set to idle. This will be for the first location, O1.

The stream request at subsequent locations, X, Y, Z and O2 will be processed in a similar manner. At each location, *toneIndices[]* element corresponding to the location position specified by the request is assigned a *tonesCount* at the time of filling a buffer for the first speaker for that location. At the end, the *toneIndices[]* array contains five elements, for instance, as [342, 416, 490, 581, 609]. Using Figure 4.21 for a comparison, *toneIndices*[0] = 342 corresponds to $SO1_1$ while *toneIndices*[4] = 609 corresponds to $SO2_1$.

The end of the streaming phase is determined by a message which requires streaming, but for the second time at the origin location. This is used as a signal to stream audio to the loudspeakers for the last time, and then stop streaming. At end of the streaming, the *toneIndices[]* array is sent to the Node.js server. This is a final step required for the JUCE tones server. The *tonesIndices[]* information will be used for synchronisation in the Node.js server. The next tasks are handled by the Node.js server and the web client as already explained in their respective sections.

5.3.11 The Client Configuration

The configuration process starts immediately when streaming stops. Audio processing is performed by the *processData()* function. Listing 5.8 shows parts of the *processData()* function which perform important actions. One of the design intents was to minimise

the size of data moving between the web client and Node.js server. Given the potentially huge size of the recorded data by the end of the streaming, a slicing process was used to reduce the data to be transmitted to the server.

The *processData()* first identifies a buffer which contains the first tone within the *audioData* array. It invokes the *detectFirstTone()* function to achieve this. The function is shown in Listing 5.9.

Listing 5.8: Parts of *processData()* which perform key actions

```
154             var bufsPerTone = samplesBetweenTones / clientBufferSize; // number of client arrays in the interval
157             var tones = detectFirstTone( audioData, samplesBetweenTones, threshold ); // nb: just get samples
161             var firstToneIndex = Math.floor( tones[0] / clientBufferSize );
167             var dataOfInterest = [];
168                 dataOfInterest = tonesIndices.map(function(juceLocIndices, j){
169                     var start = 0, end = 0, cutoff = 0;
170
171                     if ( juceLocIndices[0] >= 0) { // juce sends -1 for unstreamed locations, so pick those with 0 or more
172                         start = juceLocIndices[0] * bufsPerTone + firstToneIndex - 1;   // set start index
173                         end = ( start >= 0 ) ? start + ( bufsPerTone * numOfSpeakers ) : start; // end index
174                         cutoff = start * clientBufferSize; // number of samples cut off
175                     }
177                     cutoffSamples.push( cutoff );
180                     curData = audioData.slice( start, end );
192                     // sliced data
193                     return curData;
194                 });
201             var reducedData = {
202                 audioSamples : dataOfInterest,
211             }
215             sendRequest( audioType, reducedData );
223             if(dataOfInterest[0].length > 0){
224                 $('#display').addClass('visible');
225                 visualiseAudioData(dataOfInterest, 0);
226             }
```

Listing 5.9: Detection of the first tone

```
588         detectFirstTone = function( samplesArr, interval = samplesBetweenTones, threshold = 1*0.8) {
605             while( end < samplesArr.length) {
607                 searchSamples = shared.flattenArray( samplesArr.slice( start, end ));
610                 for (sample = 0; sample < searchSamples.length; ++sample ) {
611                     if (searchSamples[sample] > threshold)  {
```

```
612                             crestValue = searchSamples[sample];
613                             shiftedToneStart = sample + start * bufferSize;
614                             // add values to arrays
615                             tones.push( shiftedToneStart );
618                         }
619                     }
630                 return tones;
631             },
```

To identify the first tone peak position within the audio data, the function goes over the *audioData* array. It:

- takes several elements of the 2D *audioData* at a time and flattens them into a one dimensional array,
- goes over the flattened smaller array since *audioData* is a 2D array,
- searches for an audio sample with an amplitude over a certain threshold within the smaller array. When it finds it, it saves the position of this audio sample into a variable. This audio sample is assumed to be the peak of the tone described in Section 4.3.

The *processData()* function uses the identified peak position, together with the tone indices from the tones server to determine a region which contains the data from the loudspeakers for each microphone position. Figure 5.13 is used for illustrating how this is achieved. A single buffer is a Float32Array of 512 audio samples (the orange box in the figure).

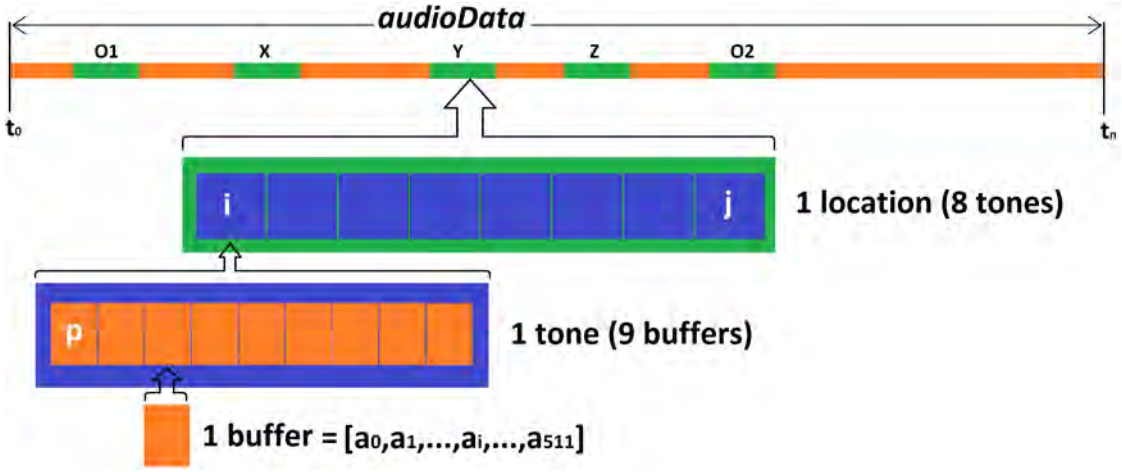


Figure 5.13: Data detection and splicing within *audioData*

For an interval of 96ms, a region which contains a tone is 4608 samples long (equivalence of 96ms). This is made up of nine buffers, shown in blue in the figure. For eight tones,

one per loudspeaker in the speaker layout, 36,864 (= 4608 x 8) audio samples are required to contain audio for one location. This region is shown by green box. For each of the five microphone positions, the tones are contained in 184,320 (= 4608 x 8 x 5) samples. These are indicated by the patches of green in the top plot, for the microphone at origin (O1), x-axis (X), y-axis (Y), z-axis (Z) and at the origin for the second time (O2). This is the only audio data which needs to be analysed for determination of the speaker coordinates. The required audio data length is slightly less than four seconds (184,320 / 48,000), at the sample rate of 48kHz. However, the time taken to move the microphone from one location to the next (shown as the interleaving orange patches) is normally arbitrarily long, especially in comparison to the time required for recording the loudspeakers tones. This time extends into minutes for the entire duration of recording. This significantly increases the amount of data the microphone ultimately records. To reduce this data, the position of the first detected tone is used in conjunction with the tone indices received from the tones server.

The first tone peak position determined by *detectFirstTone()* would be an audio sample, a_i, contained in some buffer as shown in the figure. The index of this buffer, in relation to the entire audio data array is shown as p within the tone, and it is the *firstToneIndex* in Line 161 of Listing 5.8. For each location, *firstToneIndex* is used together with the first index of indices from the tones server, *juceLocIndices[0]*, to identify the positions i and j in the figure. These values correspond to the *start* and *end* in Listing 5.8. The positions are used to splice the large data array in Line 181. The spliced data is saved into another array in Line 193. At the end of this process, a smaller 2D array contains only the regions of the microphone position. It contains five elements, one per location, such that each element is an array of 72 (8 speakers x 9 buffer per speaker) 512-float32 buffers. This is shown in Figure 5.14. The trimmed audio data is sent to the server for processing (Line 215).

5.3.12 The Server Configuration

The bulk of processing is performed by the *configure* use case. The server uses the audio data sent by the client, together with additional information sent by the JUCE application, to start the configuration process. The process follows a series of steps to create the final 3D loudspeaker coordinates. The implementation steps are described below.

Handling JUCE tone indices: One of the initial tasks performed by *configure()* is to handle the tone indices sent by the tones server. The JUCE tones server sent five values that

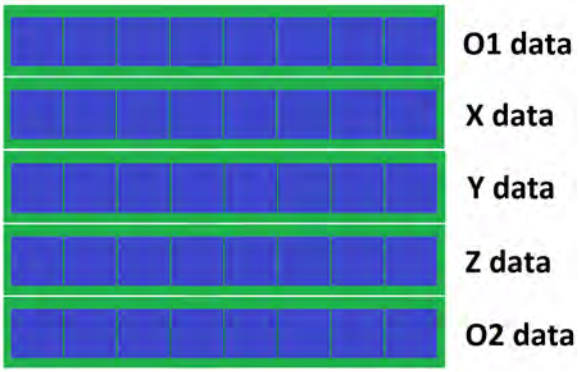


Figure 5.14: 2D array of reduced locations tones data

correspond to the tone sent to the first loudspeaker at each location. The Node.js server needs to regenerate the rest of the locations' indices for use in the subsequent steps. Using Figure 5.12 as a reference, the subsequent indices of the tones at a location are successive increments of one per speaker. For example, if the indices from the JUCE tones are as shown at the end of Section 5.3.10, that is, [342, 416, 490, 581, 609], subsequent indices for location O1 become 343 for speaker 2, 344 for speaker 3, continuing all the way to 349 for speaker 8. Other locations' indices are generated in the same way. A total of 40 indices are generated.

These are then reset using the first index, 342, by making it zero and adjusting the rest by the initial value of the first index. These are the tone indices used by the client to perform data reduction. Each index is an integer multiple of the interval used - 4,608 samples. This information is used to convert the indices to the sample positions, by multiplying each index with 4,608. Figure 5.15 shows the indices and their corresponding sample positions. These are ready to be used with the peak positions from the client to complete the speaker measurement process.

Peaks extraction: As explained in Section 4.3, the tone created for tone transmission has a series of low amplitude cycles and one high amplitude cycle following this series in one case (small test room) or preceding it in another case (large test room). To identify each tone, the peak of the high amplitude cycle must be located within the audio recording to a granularity of a sample. A successfully identified loudspeaker tone is represented as a sample position within the audio data array. The server method which performs the configuration process is *configure()*.

Listing 5.10 shows a portion of the *configure()* code which extracts peaks from the audio data. The results of this function is a 40-element array of the positions of the peaks

JUCE tone indices

	1	2	3	4	5	6	7	8
O1	0	1	2	3	4	5	6	7
X	74	75	76	77	78	79	80	81
Y	148	149	150	151	152	153	154	155
Z	239	240	241	242	243	244	245	246
O2	267	268	269	270	271	272	273	274

JUCE sample positions

O1	0	4,608	9,216	13,824	18,432	23,040	27,648	32,256
X	340,992	345,600	350,208	354,816	359,424	364,032	368,640	373,248
Y	681,984	686,592	691,200	695,808	700,416	705,024	709,632	714,240
Z	1,101,312	1,105,920	1,110,528	1,115,136	1,119,744	1,124,352	1,128,960	1,133,568
O2	1,230,336	1,234,944	1,239,552	1,244,160	1,248,768	1,253,376	1,257,984	1,262,592

Figure 5.15: JUCE indices and sample positions

within the audio data array *audioSamples*. These are reset using the first peak position by setting it to zero and adjusting the rest with the value of the first peak. The results are shown in Figure 5.16.

Listing 5.10: Extracting tone peaks

```
328         // This procedure assumes that there is a single sine wave larger than the rest
329         // Go through each set of tone samples and find the max for each
330         for (var i=0; i<5; i++) {
331             for (var j=0; j<8; j++) {
332                 slicedSamples = [];
333                 bottomSampleNo = intervalBetweenTones*(i*8+j);
334                 topSampleNo = bottomSampleNo + (3*toneDuration);
335                 // go over a search space 3 times the tone duration
336                 slicedSamples = audioSamples.slice(bottomSampleNo, topSampleNo);
337                 maxSample = Math.max(...slicedSamples); // Find the peak of the last ↩
                        high sine wave
338                 tonesMaxs.push(maxSample);
339                 // We know where the slice started - now add the position of the peak
340                 actualTonesArr[i*8+j] = bottomSampleNo + slicedSamples.indexOf(↩
                        maxSample);
341                 ampsArr[i*8+j] = maxSample;
342             }
343         }
```

These positions are from the trimmed data, which is a much short array compared to the original array the client had. The positions of this smaller array are shorter than they would be if the original array was used. This gives an impression that the tones occurred earlier in time, which would imply shorter distances than the actual distances. To solve this, the detected peak positions are adjusted to the original positions in the originally recorded audio data. The positions of the buffers used for splicing the original audio data

array by the client are saved and sent along with the audio data. These are incorporated in this step of adjusting the positions with respect to the original audio data. This results in the peaks that would have been detected from the original audio data, as shown in Figure 5.16.

Client's peak positions

	1	2	3	4	5	6	7	8
O1	0	4,618	9,292	13,806	18,490	23,102	27,852	32,478
X	340,997	345,626	350,278	354,818	359,467	364,098	368,858	373,468
Y	681,971	686,590	691,268	695,828	700,500	705,127	709,824	714,453
Z	1,101,334	1,105,951	1,110,626	1,115,140	1,119,824	1,124,436	1,129,186	1,133,812
O2	1,230,360	1,234,978	1,239,653	1,244,167	1,248,850	1,253,463	1,258,213	1,262,839

Figure 5.16: Client sample positions

The sample rate differential calculation: Prior to the conversion of the sample positions to their equivalent distances, a compensation for the clock drifts in the JUCE tones server and the client needs to be made. Detailed information on how this occurs are given in Section 4.6. As previously discussed from that section, the calibration speaker's sample positions on both the client and tones server are used for this step.

Using equations 4.2 through 4.4 of Section 4.6, together with the tone positions information in Figures 5.15 and 5.16, the positions of the first and last sample positions of the calibration speaker (speaker 1) for the tones server are 0 and 1,230,336 respectively, while the corresponding client sample positions are 0 and 1,230,360 respectively. The tones server difference, SDiff, calculated using equation 4.2, becomes:

$$SDiff = 1,230,336 - 0 = 1,230,336$$

The client difference, MDiff, using equation 4.3 becomes:

$$MDiff = 1,230,360 - 0 = 1,230,360$$

Finally, the sample rates differential, calculated using equation 4.4, becomes:

$$SRD = \frac{1,230,336}{1,230,360} = 0.9999804935140935$$

The value of the sample rates differential, *0.9999804935140935*, is then used to scale the client's sample positions. This is done multiplying each tone position, shown in Figure 5.16, by this value.

The results of the scaling step are used in conjunction with JUCE tone positions to calculate the tones' *samples of flight*, this is an equivalence of a tone's *time of flight, TOF*, which is the time the tone takes from source to destination. A tone's *samples of flight* is the same concept, in samples. This is obtained by taking the difference between a tone's positions on the client and on the tones server. For instance, for speaker 2 at location O1, this difference is obtained from (0.9999804935140935 x 4618) - 4608 = 9.90991904808379. The bracketed value is the result from scaling (multiplying a client tone position in Figure 5.16 by *SRD*). The samples of flight are additionally adjusted so that those corresponding to the calibration speaker are zero[22]. The distances are computed from the results of this step.

Distances conversion: The samples of flight obtained in the previous step are converted to their equivalent distances in this step. Equation 4.6 is used for this conversion. Each result (samples of flight) from the previous step is plugged into this equation to obtain the corresponding distance. For instance, using speaker 2, the distance, D_{s2}, becomes:

$$D_{s2} = \frac{9.90991904808379 \times 34.314}{48} + 229 = 236.08455983614573$$

All the tone distances are obtained in a similar manner. The resulting distances are ready for coordinates calculations.

Coordinates calculations: The distances obtained in the previous step are used to calculate the 3D coordinates for each loudspeaker. A function, *computeCoordinates()*, is used to compute these coordinates. Listing 5.11 shows the *computeCoordinates()* function. Along with the speaker distances, it uses an orthogonal distance to calculate the coordinates. For the z-coordinate, it uses one of the options discussed in Section 4.8, depending on whether the user wants to use data recorded at the z microphone location. The output of this is a set of 3D coordinates for the SDIAS speaker layout, as shown in Figure 5.17.

Listing 5.11: 3D coordinates calculations

```
9     computeCoordinates = function(
10        locationDistancesArray,     // array of distances sort by speaker
11        orthogonalDistance,
12        zMicPosUsed = 0)            // client must specify this flag & send it over to ↩
              the server
13        {
14
```

[22]This is helpful when the calibration speaker is not the first speaker, otherwise it has no effect as the samples of flight are already zero for speaker 1.

```
15          let _do = locationDistancesArray[0],
16              _dx = locationDistancesArray[1],
17              _dy = locationDistancesArray[2],
18              _dz, vcoordX, coordY, coordZ, coordZforComparison, arr =[];
19
20          // calculating the coordinates
21          coordX = _do * Math.cos(Math.acos((orthogonalDistance * orthogonalDistance +
                _do * _do - _dx * _dx) / (2 * orthogonalDistance * _do))),
22          coordY = _do * Math.cos(Math.acos((orthogonalDistance * orthogonalDistance +
                _do * _do - _dy * _dy) / (2 * orthogonalDistance * _do)));
23          arr.push(coordX);
24          arr.push(coordY);
25
26          if(!zMicPosUsed)  {
27              /* METHOD 2 */
28              coordZ = Math.sqrt(_do*_do - (coordX * coordX + coordY * coordY));
29              coordZ = Math.sqrt(_do * _do - (coordX*coordX + coordY*coordY));
30              arr.push(coordZ);
31          } else {
32              /* METHOD 1 */
33              _dz = locationDistancesArray[3];
34              coordZ = _do * Math.cos(Math.acos((orthogonalDistance * orthogonalDistance
                    + _do * _do - _dz * _dz) / (2 * orthogonalDistance * _do)));
35              arr.push(coordZ);
36          }
37          return arr;
38      };
```

SDIAS 3D speaker coordinates

	X	Y	Z
Speaker 1	23.5	216.5	70.9
Speaker 2	-70	215.3	66.8
Speaker 3	136	219.9	115.6
Speaker 4	-95.3	-188.4	45.2
Speaker 5	213.1	-114.7	120.1
Speaker 6	35.8	-270.1	17.6
Speaker 7	-89	340.5	127.9
Speaker 8	128.8	312.4	189.2

Figure 5.17: SDIAS 3D speaker coordinates

The XML speaker configuration: To finalise the speaker measurement process, the speaker configuration is created. This is specified in XML. For each speaker, the configuration contains the speaker position and the 3D coordinates obtained from the previous step, namely the x-coordinate, the y-coordinate and the z-coordinate. This is done with the *createXMLConfiguration()* method. The XML configuration is sent to the client and also

saved to disk in an XML file. An XML configuration for the speaker coordinates obtained from the previous step is shown in Listing 5.12. This configuration can be accessed by other systems which need information for the speaker positions. For example, Immergo can use this file for use with its various spatialisation algorithms.

Listing 5.12: The XML speaker configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <speaker number = "1" xpos = "23.5" ypos = "216.5 " zpos = "70.9"></speaker>
    <speaker number = "2" xpos = "-70" ypos = "215.3 " zpos = "66.8"></speaker>
    <speaker number = "3" xpos = "136" ypos = "219.9 " zpos = "115.6"></speaker>
    <speaker number = "4" xpos = "-95.3" ypos = "-188.4 " zpos = "45.2"></speaker>
    <speaker number = "5" xpos = "213.1" ypos = "-114.7 " zpos = "120.1"></speaker>
    <speaker number = "6" xpos = "35.8" ypos = "-270.1 " zpos = "17.6"></speaker>
    <speaker number = "7" xpos = "-89" ypos = "340.5 " zpos = "127.9"></speaker>
    <speaker number = "8" xpos = "128.8" ypos = "312.4 " zpos = "189.2"></speaker>
</config>
```

5.3.13 The Client Results

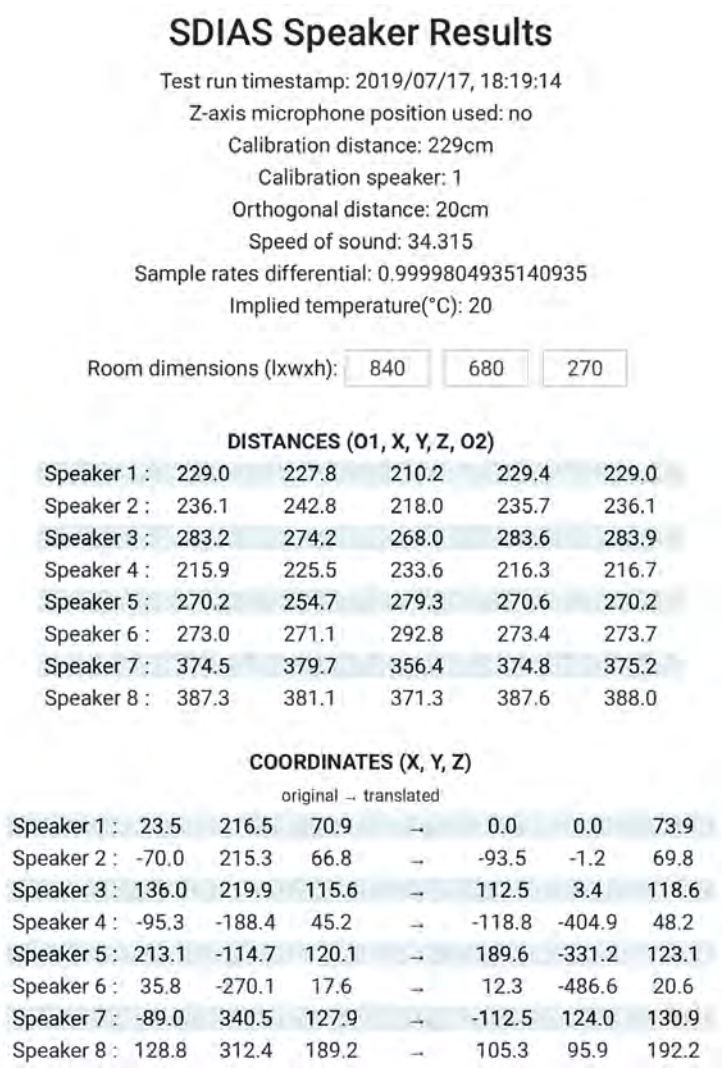
The results transmitted from the server to the client are the distances and coordinates for the loudspeakers. The coordinates are formatted such that each of the eight speakers has an x-, a y- and a z-coordinate value, while the distances are formatted such that each speaker has the distance calculated at each of the streamed locations. Figure 5.18 shows a UI portion which displays the results received from the server.

The client UI furthermore displays a visualization of the sampled audio data. The visualisation is displayed via the HTML5 canvas elementA visualisation which shows distinct eight impulses, one per speaker, for each of the five locations is shown in Figure B.5.. An example visualisation is shown in Figure 3.15. The user can immediately detect from the visualisation whether the process could yield results as expected. For instance, there must be:

- variability throughout the plot, which suggests that the data trimming step was potentially accurate (while a flat plot would indicate that the trimming occurred where there was silence)
- potential spikes within the plot, which coincide with the peak cycles of the speaker tones.

A visualisation[23]hich does not show these features would potentially not yield accurate

[23] w



SDIAS Speaker Results

Test run timestamp: 2019/07/17, 18:19:14
Z-axis microphone position used: no
Calibration distance: 229cm
Calibration speaker: 1
Orthogonal distance: 20cm
Speed of sound: 34.315
Sample rates differential: 0.9999804935140935
Implied temperature(°C): 20

Room dimensions (lxwxh): 840 680 270

DISTANCES (O1, X, Y, Z, O2)

Speaker 1 :	229.0	227.8	210.2	229.4	229.0
Speaker 2 :	236.1	242.8	218.0	235.7	236.1
Speaker 3 :	283.2	274.2	268.0	283.6	283.9
Speaker 4 :	215.9	225.5	233.6	216.3	216.7
Speaker 5 :	270.2	254.7	279.3	270.6	270.2
Speaker 6 :	273.0	271.1	292.8	273.4	273.7
Speaker 7 :	374.5	379.7	356.4	374.8	375.2
Speaker 8 :	387.3	381.1	371.3	387.6	388.0

COORDINATES (X, Y, Z)

	original			→	translated		
Speaker 1 :	23.5	216.5	70.9	→	0.0	0.0	73.9
Speaker 2 :	-70.0	215.3	66.8	→	-93.5	-1.2	69.8
Speaker 3 :	136.0	219.9	115.6	→	112.5	3.4	118.6
Speaker 4 :	-95.3	-188.4	45.2	→	-118.8	-404.9	48.2
Speaker 5 :	213.1	-114.7	120.1	→	189.6	-331.2	123.1
Speaker 6 :	35.8	-270.1	17.6	→	12.3	-486.6	20.6
Speaker 7 :	-89.0	340.5	127.9	→	-112.5	124.0	130.9
Speaker 8 :	128.8	312.4	189.2	→	105.3	95.9	192.2

Figure 5.18: Client UI portion for the 3D speaker coordinates

results. Along with the errors emanating from absence of the described features, other errors are possible. The visualisation of the inaccurate results would be accompanied by a description of how many coordinates are (possibly) inaccurate, and therefore not satisfactory.

Suppose, from Figure 5.18, the geometric algorithm that computes the coordinates inaccurately detected Speaker 8's tone at the x-axis microphone position such that the distance is further than as shown in figure, as 395cm instead of 381.1cm. This would result in the x-coordinate of -140.9cm, which is erroneous as it implies that the speaker lies in the opposite direction to the expected one as evidenced by the negative sign. This is an error a user must discern as it would not appear as an error in the system results. Furthermore, if only the xy-plane data use option was selected, the erroneous value would propagate to

the z-coordinate since its value is calculated using the x and y coordinates. The onus is therefore on the user to discern these types of errors.

However, if the same Speaker 8's distance was detected as 405cm, the x-coordinate would yield an erroneous value of -340.0cm, whose error must be discernible to the user since the system would not yield an error. However, in the case of the z-coordinate, a value would be displayed on the system since the trigonometric calculations would not yield a valid value. Based on these types of observations, the user would have to restart the process.

5.4 Chapter Summary

This chapter detailed all processes undertaken to design and implement SDIAS. Firstly, a design approach was described, which produced system design models necessary to implement the system. The various SDIAS component implementations were then described.

Ethernet AVB functionality was discussed, with key functions necessary in the transmission of time sensitive audio streams. The capabilities of the Ethernet AVB standard provided a fundamental platform for meeting the critical timing required by the SDIAS.

Details of the server were then given, explaining in detail, how Node.js as the web server framework was used for building the server application. Critical server functions were also described, including, but not limited to, the initialisation steps such as launching the JUCE application, establishing real-time communication for both the JUCE and the web client application, processing of the audio data recorded by the client, together with the necessary steps for converting this data to the required result - the 3D speaker coordinates.

The web client component was also discussed in detail. Various sub-components constituting the client, such as the mobile device with an attached microphone, together with the software sub-components such as the HTML web page, the JavaScript application and the web APIs were described. The use of the web APIs to provide the ability to achieve tasks such as recording of tones sent by the loudspeakers were described in detail. This was an especially crucial capability required of the web client. The completion of the implementation of SDIAS rendered the system usable and testable. The next chapter gives a description of the testing phase of SDIAS.

Chapter 6

System Testing

This chapter describes the testing of some aspects of the system which are fundamental to achieving the research objectives. System testing is key to ensuring that the system does what it was designed to do (Sharp et al., 2015). Tests were performed to evaluate the system accuracy. These were compared objectively with the manual results obtained from physical measurements. The accuracy tests were achieved by comparing the results obtained during the experiments with SDIAS with those obtained from the manual measurements. The system needed to be able to accurately yield 3D speaker coordinates for the layout available in the test trials. As indicated in Chapter 4, the final coordinates are obtained via a series of stages which involve computations designed to obtain specific values and quantities necessary to calculate the coordinates. Errors could arise at any of these multiple stages, leading to invalid values in subsequent steps. This meant that at each stage, it became necessary to inspect and test its output for validity before moving to the next stage. The following sections describe the testing procedure and results from the experiments. Finally, an analysis and discussion of the results are also provided.

6.1 Test Configurations and Procedure

The speaker coordinates are calculated from the distances between microphone positions and speakers. These distances are in turn obtained from the speaker tone sample positions extracted from the audio recorded from the client device, typically a mobile device such as a tablet. To run the tests, a user ensured that the testing environment was setup and ready to start the speaker configuration process. A comprehensive procedure for the steps involved is given in Section 3.4. In summary, the steps involve:

- server setup: ensure that both the Node.js server and tones server are running,
- client setup: check that the client device can record sound, especially over a web browser,
- connection between all entities: verify that a connection is established between the mobile device and the server hosting PC; also between the PC and the AVB network and between the AVB network and the loudspeakers,
- speaker layout identification: confirm that the position of every speaker in the room can be identified, via a beep from the speaker,
- microphone positions identification: ensure a clear line of sight from each of the microphone positions to every speaker,
- calibration: ensure that the system variables such as the orthogonal distance, calibration speaker position and distance are properly set on the client,
- audio streaming: ensure audio streams from the JUCE application whilst the client records it.

The system setup for the test environments was as described in Section 3.3. It comprised a PC running the server application and the tone generator, while the client application ran on a Samsung Galaxy Tab S2 (Samsung Electronics, 2019) tablet. A Panda microphone (Purple Panda, 2019) was attached to the mobile device. The AVB network consisted of an Echo Streamware sound and network card (Echo, 2019a), connected to the MOTU UltraLite AVB interface (MOTU, 2019). The UltraLite AVB interface has eight balanced outputs, which were each connected to an active Genelec monitor loudspeaker (Genelec, 2019). The placement of the loudspeakers loosely followed typical layouts for immersive sound. This was not a problem since the goal of the system was to accurately determine 3D speaker coordinates for any given layout, rather than for typical immersive layouts. This meant an even more haphazard layout is desirable than a standard layout. The available speakers were laid out as follows:

- a centre speaker,
- front-right-surround and front-left-surround speakers,
- height-left-surround and height-right-surround speakers,
- rear-right-surround and rear-left-surround speakers, and
- a height-rear-centre speaker.

Tests were performed both manually using a tape measure and automatically using SDIAS. Steps involved in the manual determination involved two phases:

- measuring the individual distances from the four microphone positions, and
- using those distances in a Node.js program created expressly for computing the coordinates.

The manual measurements were taken so that a comparison with SDIAS results could be made. This is taken as a measure of the system accuracy. The accuracy of SDIAS therefore increased with increasing proximity of its results to the manual results. Tests performed with SDIAS integrate all the manual steps into one continuous step, from measuring the distances, through computing the coordinates and presentation to the user on the client device. The only manual process is the placement of the microphone at the four positions and measurements of the calibration and orthogonal distances. Tests were performed in two environments; one set of tests in a small room and another set in a large room. The tests were performed in far and diffuse sound fields, though a free sound field was simulated in the large room.

6.1.1 Small Test Room

The setup for this environment consisted of:

- a room with dimensions approximately 500cm long, 270cm high and 300cm wide,
- a region for microphone positions,
- a chair with a right angle between seat and back, used as a support for the placement of the microphone at the four locations, aimed at improving the orthogonality of the axes,
- a calibration distance of 83cm, measured from the nearest speaker, and
- an orthogonal distance of 33cm, dependent on the dimensions of the chair.

For each test, the order of the microphone positions was O, X, Y, Z and O. O is taken as the origin with the coordinates (0,0,0). X, Y and Z are the positions of the microphone at each of the axes, at a length equal to the orthogonal distance, 33cm for example. Consequently, all the distances are measured from O and all the coordinates are given relative to O. The height from the floor to the O position was 47cm, as such, the 0 in the O's z-coordinate could be substituted for 47 for tests involving this setup.

6.1.2 Large Test Room

This test environment setup consisted of:

- a large room of dimensions approximately 820cm long, 270cm high and 680cm wide,
- two reflectivity capacities for the room walls (one reflective, that is, a room with natural drywall; another with damping applied via padded chairs placed behind speakers to act as noise absorbers),

- various regions for the microphone positions,
- various calibration distances using different calibration speakers,
- three orthogonal distances of 20cm, 40cm and 80cm.

Tests from this configuration were categorised according to the reflectivity property of the test room walls and the orthogonal distances. For instance, a series of tests for the reflective (i.e. natural walls without any damping) room, with the microphone positions at 40cm orthogonal distances was performed; another series of tests from the damped room and the 80cm orthogonal distance for microphone positions.

6.2 Results and Discussion

Measurements of the distances from the four microphone positions were taken manually. Due to the difficulty of manually measuring the coordinates of a speaker in 3D space, the coordinates of the speakers were not measured manually, instead, they were fed to the same algorithm used in SDIAS, to yield the coordinates. A Node.js code listing which performs this task is given in Listing A.2. The coordinates resulting from these manually measured distances were used for comparison against those obtained dynamically from SDIAS. The manual coordinate results are discussed in Sections 6.2.1 and 6.2.2 for the small and large test rooms respectively. The coordinates determined by SDIAS are physical measurements of the speakers locations in the 3D space. These comprised the dynamically calculated distances from O, and along each of the x, y and z axes. The 3D coordinates results for these distances are discussed in Sections 6.2.3 and 6.2.4 for the small and large test rooms respectively.

The goal was to investigate the accuracy of SDIAS in determining a coordinate of a speaker. The 3D space meant that there were three coordinates which needed to be investigated. The results from the eight speakers within the immersive speaker layout were analogous to having eight speaker results investigated at a go. To achieve this goal, multiple tests were run. The results were gathered from various output produced from each test of SDIAS. These included 3D coordinates data saved to the disk in various outputs, the XML format, the audio sample data saved as plain text in the default raw 32-bit float web audio API (Adenot & Toy, 2018) number format and the screenshots of both the server and client UI.

For SDIAS results, some descriptive statistics is used to discuss the speakers' respective coordinates for various tests. The minimum and maximum coordinate values are listed

for each coordinate of each speaker. The difference between these two values provides a range value. The smaller the range, the more consistently the SDIAS algorithm is in determining the respective coordinate of an axis, the larger the range, the less consistently it does so. The mean and standard deviation (column *s.d.*) are also given to indicate the centre and spread of the datasets from the average, respectively. Smaller values of the standard deviation indicate consistency in the results, which implies accuracy, especially if the values are close to the mean. The manual coordinates are also given (column *man**) for comparing the ranges and the mean.

6.2.1 Small Test Room Manual Measurements

The manually measured distances from four microphone positions for the small test room setup are shown in Table 6.1. As an example, the microphone distances for Speaker 6 were measured to be 83cm from O, 112cm from X, 76cm from Y and 85cm from Z. The distances must obviously be within the bounds of the room dimensions listed in the environment setup in Section 6.1. This is a physical condition met by the distances and as such, it is expected to be true for all the measurements, even the dynamic ones from SDIAS. As an example, Figure 6.1 shows the distances of the four microphone positions from Speaker 6[1].

Table 6.1: Small test room manual distances

Speaker #	O	X	Y	Z
1	202	226	179	192
2	112	140	140	114
3	144	163	111	145
4	153	132	130	155
5	96	73	125	98
6	83	112	76	85
7	168	142	177	152
8	187	215	207	175

The coordinate results from the Node.js program that computes the coordinates using the distances in Table 6.1 are given in Table 6.2. The speaker coordinates conform to the octants in 3D Euclidean space as defined by the placement of the microphone along the axes. For example, Speaker 1 with coordinates (-139.1, 149.3, 76.2) lies in the fifth[2]

[1]Other speakers's distances are not shown to maintain clarity as many lines would clutter the display.

[2]An order adopted is given by (±x,±y,±z) such that the 1st octant is (+++), 2nd is (++-) and 8th is (- - -). However, in this case, the speakers can be in one of the four octants because z-coordinate is always positive.

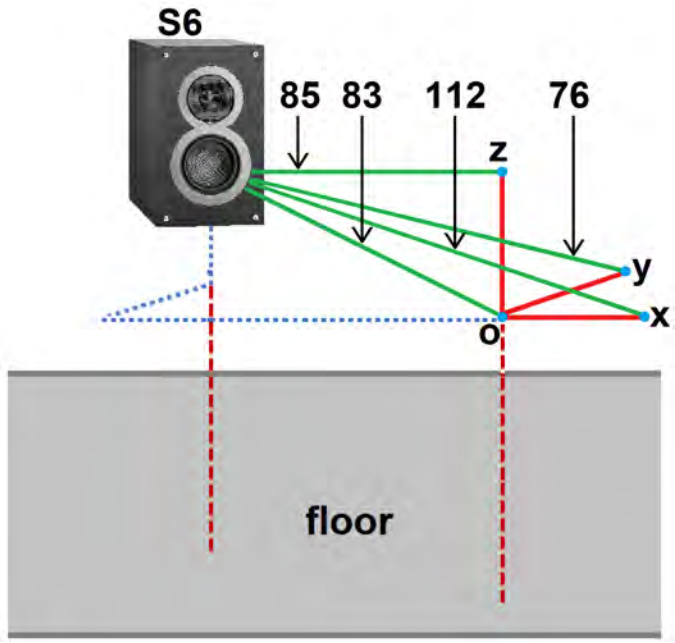


Figure 6.1: Speaker distances

octant as defined by the microphone positions. A layout of the coordinates is shown in Figure 6.2. In the figure,

- a blue dot represents the origin O,
- a red solid arrow indicates a positive direction of the axis from O, at X, Y and Z,
- a red dotted line indicates a height of the origin from the floor and is taken as 0. The z-coordinate for each speaker is measured from the top of these lines (which were at 47cm from the floor),
- a blue dotted line indicates the path to the speaker's coordinate along an axis,
- a speaker is labelled with its position, and
- a room layout is also shown.

Table 6.2: Small test room coordinates from manual distances

Speaker #	X	Y	Z
1	-139.1	149.3	76.2
2	-90.4	-90.4	9.7
3	-71.9	144	12.1
4	107.2	115.1	7.2
5	75.4	-80.6	10.6
6	-69.2	33.4	11.4
7	138.6	-30.5	94.1
8	-154	-102.9	82.3

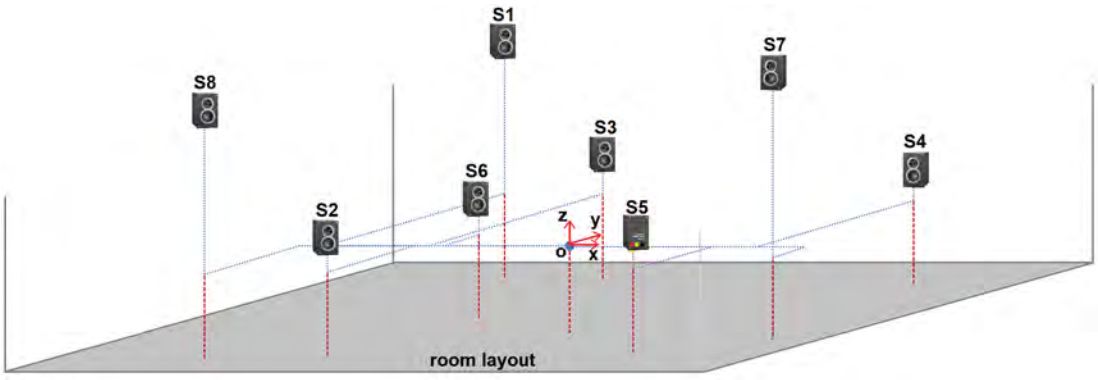


Figure 6.2: Small test room speaker layout sketch

6.2.2 Large Test Room Manual Measurements

Similarly for this test setup, the measured distances from the four microphone positions are shown in Table 6.3. The origin, O, for the four microphone positions is the spot on the floor marked with a blue dot in Figure 6.3. The distances for this configuration are larger, since the speakers are spread out in a larger room. Also, the bounding conditions are the room dimensions. Some aspects remain similar to those from the previous configuration, to allow for comparison. For instance, the manual distances from the four microphone positions were measured, then used in the Node.js program to yield the coordinates.

Table 6.3: Large test room manual distances

Speaker #	O	X	Y	Z
1	229	229	192	216.5
2	236.5	254	201	225
3	283.5	266	252	273
4	217.5	238	252	206
5	274	239	292	260.5
6	275	271	313.5	264
7	376	389	341	361
8	390	379	358	370

Table 6.4 shows the 3D coordinate results from the output of the Node.js program that computes the coordinates from these distances. The presentation of the results for this configuration differs slightly from that of the small test room. The x- and y-coordinates are given in columns X and Y respectively like in the small test room. However, unlike in the small test room, the z-coordinate from the manual measurement of the z-distance is represented in column Z1 (rather than Z). The z-coordinate was calculated using the results obtained from the first two x- and y-coordinates as explained in Section 4.8. The

result of this z-coordinate calculation is given in column Z2. ΔZ shows the difference of the z-coordinates obtained from Z1 and Z2.

Coordinate translation was used. This was achieved with the use of a reference speaker. The reference speaker was selected as a reference point within the room. Its coordinates were chosen as (0,0,h), where h represents the speaker's height from the room floor. This therefore means the origin (0,0,0) of the room was selected as the point directly below Speaker 1 on the floor, shown by a green dot in Figure 6.3. All other speaker coordinates were then expressed relative to the reference speaker. It was conveniently chosen as Speaker 1 in this instance due to the speaker's proximity to the microphone positions during the trials. The translation serves to remove the microphone position reference from the final 3D coordinate results, so that only the speaker relations remain in the layout. The translation is helpful when multiple sets of microphone positions are used, as was the case with tests for the large test room. For instance, if two test trials were performed at different times for the same speaker layout, such that, for instance, the coordinates for one trial for two speakers are $Si(60, y_{i1}, z_{i1})$ and $Sj(110, y_{j1}, z_{j1})$, while for another trial the coordinates are $Si(-10, y_{i2}, z_{i2})$ and $Sj(40, y_{j2}, z_{j2})$. The speakers are 50cm apart along the x-axis in each trial, which is key for the spatialisation algorithms. The two sets of results would not apparently reflect this, and if the results are used in the localisation algorithms or in analysis, each speaker in the second trial would be a different speaker from that in the first trial due to the different 3D coordinates.

Table 6.4: Large test room coordinates from manual distances

Speaker #	X	Y	Z1	Z2	ΔZ
1	0	0	91.1	93.8	2.7
2	-104	-0.5	87.9	76.8	-11.1
3	116.4	15.7	94.4	108	13.6
4	-113.1	-384.6	82.6	92.4	9.8
5	217.3	-311.9	111	91.7	-19.3
6	26.4	-462.9	95.4	98.2	2.8
7	-120.4	115.3	157.5	164.7	7.2
8	102.4	101.2	207.7	204.5	-3.2

The results in Table 6.4 indicate that the trigonometric computations using the distances are close to the manually measured coordinates, with variations in the calculated heights of speakers. This could be attributed to the problems associated with placement of the microphone at the z-axis as explained in Section 4.8.2. For instance, Speakers 1 to 6 were all at the same height in the room. The height of the speakers from the floor to the test speakers' reference axis was 87cm. This measurement is their z-coordinate. Table 6.4

shows that the z-coordinate of these speakers ranges from 82.6cm to 111cm. The layout produced from the coordinates is shown in Figure 6.3.

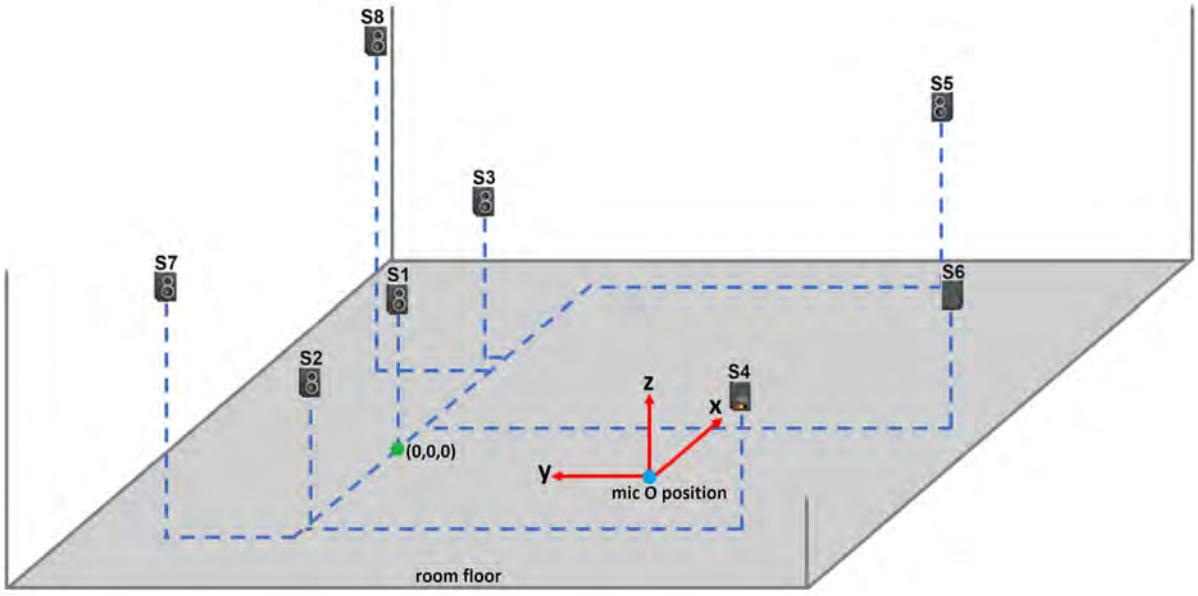


Figure 6.3: Large test room speaker layout sketch

6.2.3 Small Room Test SDIAS Measurements

The results of the speaker coordinate measurements for this configuration are shown in Table C.1 of Appendix C. The table shows a series of 26 sets of coordinates. The first 25 rows are results from SDIAS tests, while the last row results are for the manual measurements. The table shows the 3D coordinates listed for each test within the columns x, y and z for each of the eight speakers. The statistics of these results is used to analyse the results, per axis.

Table 6.5: Small test room x-coordinate statistics

Speaker No.	max	min	range	mean	man*	s.d.
1	-145.9	-135.6	10.3	-140.9	-139.1	2.7
2	-90.6	-85.3	5.3	-88.3	-90.4	1.5
3	-75.2	-67.4	7.8	-71.5	-71.9	2
4	95.7	111.1	15.4	103.7	107.2	4.2
5	78.2	87.9	9.7	80.8	75.4	2.4
6	-83	-82.6	0.3	-82.9	-69.2	0.1
7	128.5	138.8	10.3	132.6	138.6	2.5
8	-148.4	-134.1	14.3	-141.7	-154	3.8

Table 6.5 shows some statistics for the x-coordinate for the test results. The ranges of the speaker coordinates lie between 0.3cm for Speaker 6 and 15.4cm for Speaker 4. This means that for all the trials, the consistency of the coordinates varied within this range for the x-coordinates. The dimensions of the 1029A Studio Monitor Genelec speakers used were 15.1cm in width, which is along the x-axis, 19.1cm in depth, which runs along the y-axis and 24.7cm in height, which runs along the z-axis (Genelec, 2019). The deviation margin of the speaker coordinates therefore lies within the width, depth and height dimensions of the speakers for the x, y and z-coordinates respectively. The standard deviation of the coordinate values per speaker are in the range 0.1cm to 4.2cm for Speakers 6 and 4 respectively. The relatively small range and standard deviation for Speaker 6 indicate that its x-coordinate was best consistently determined from these tests, the large values for Speaker 4 indicate that its x-coordinate was worst consistently determined.

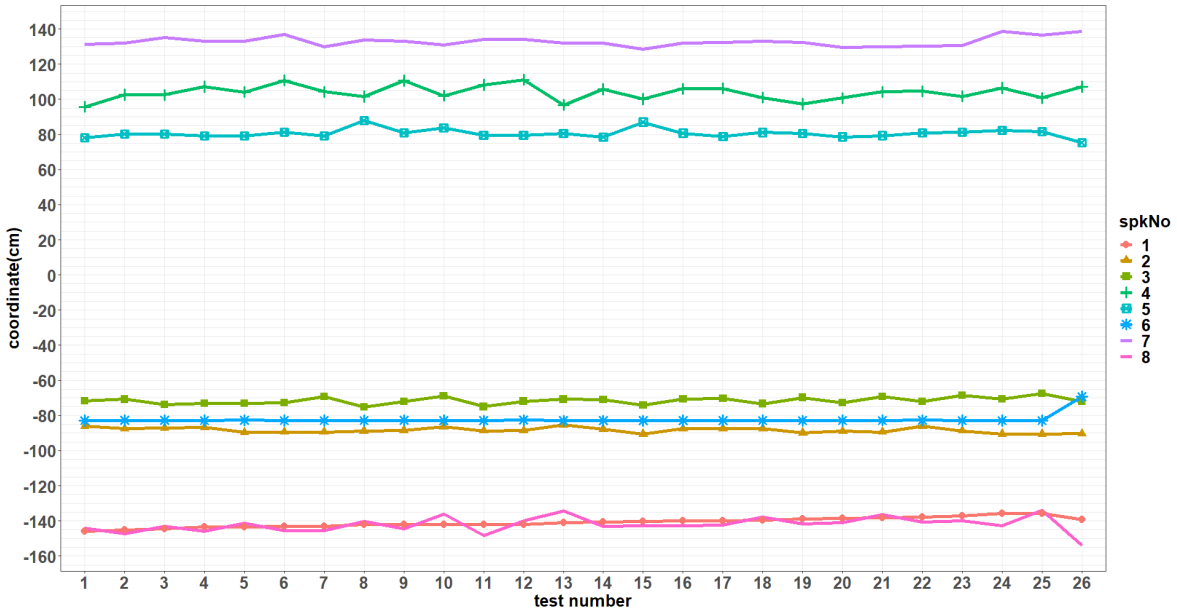


Figure 6.4: 26 results for all speakers' x-coordinates

Figure 6.4 shows a graph for the respective loudspeakers' x-coordinate line plots for the 26 results. From this figure, the plots of some speakers are flatter, while others are variable. A flat speaker plot indicates consistency in determining the coordinate for the respective speaker. This would then lead to a small standard deviation and range. Speakers 1, 2, 3, 6 and 8 have plots which are almost flat, with Speaker 6 being flatter than the rest. Figure 6.5 shows a plot of the 26 x-coordinates for Speaker 6 (as the most consistently determined speaker from the statistics). The SDIAS results (test Number 1 to 25) lie closely together, around -83cm. However, the manual result (test number 26) is far off the others. This could be due to marginal errors which occurred at the time of measuring

the distances manually, which would then propagate into the coordinates results.

On the other hand, a jagged plot points to some inconsistency or high variability in determining the respective speaker's coordinate, resulting in a large standard deviation and range. The speakers with more variability in their plots are for Speakers 4, 5 and 7, with Speaker 4 having the most variability. Figure 6.6 shows a similar plot for Speaker 4, being the worst consistently determined. Despite the relatively larger errors, the results are distributed close to the manual coordinate.

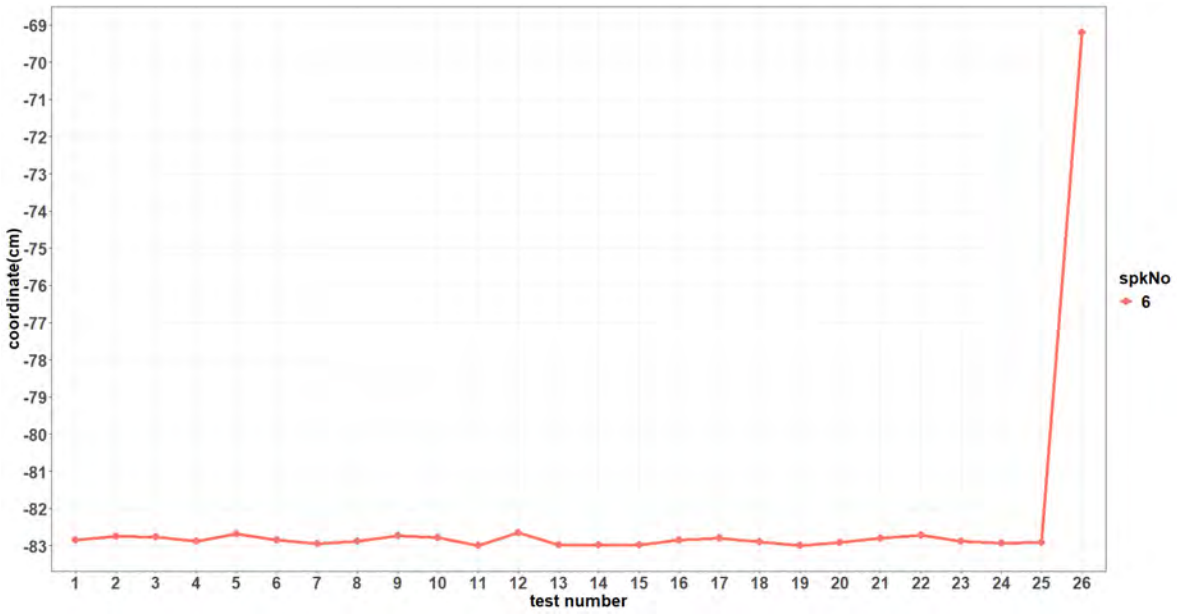


Figure 6.5: 26 results for Speaker 6 x-coordinates

For the y-coordinates, the ranges lie between 2.9cm to 12.6cm for Speaker 6 and 4 respectively. This range lies within the 19.1cm depth of test speakers. The standard deviation lies in the range 0.8cm to 3.6cm for Speakers 6 and 4 respectively. Both of the range and standard deviation limits are smaller than of the x-coordinates. This implies that the y-coordinates were more consistently determined than the x-coordinates. Similarly, the quantities indicate that Speaker 6 and 4 were respectively best and worst consistently determined by SDIAS. Speaker plots for this axis are shown in Figure 6.7.

The z-coordinate results have a range of 4.9cm for Speaker 6 to 13.3cm for Speaker 4 for a range. These limits are within the margins of 24.7cm height of the test speakers, suggesting consistency in determining the z-coordinate for the various speakers. The mean values for all the speakers further point to the consistency of the determination algorithm used by SDIAS since they lie between their respective maximum and minimum coordinates. The standard deviation varies from 1.2cm for Speaker 6 to 3.8cm for Speaker

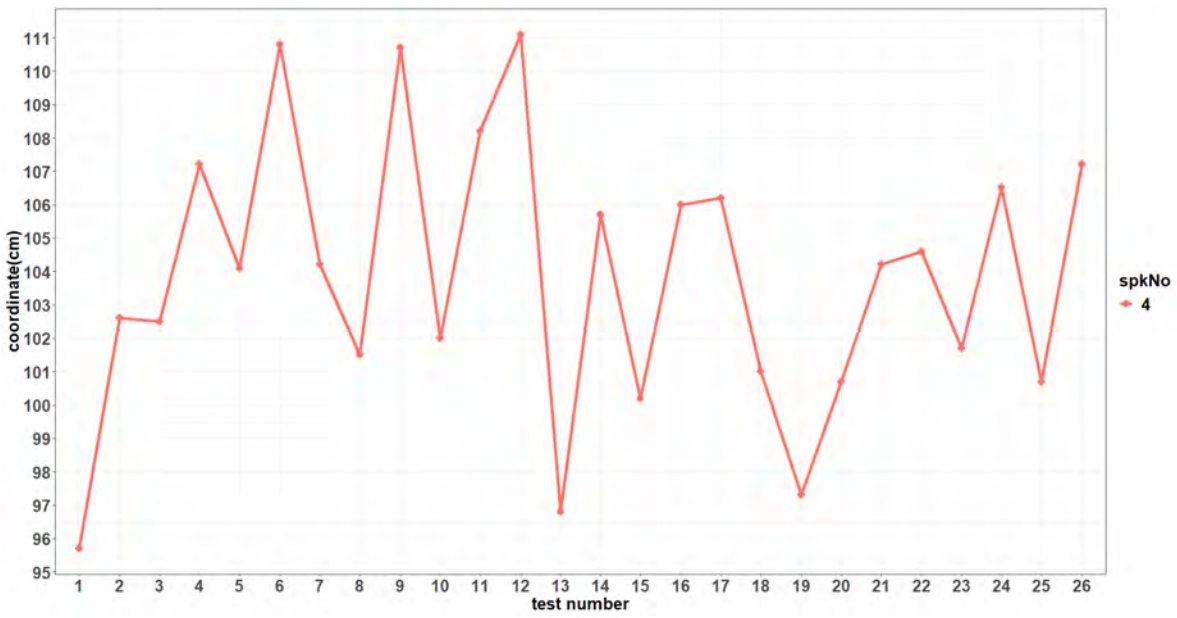


Figure 6.6: 26 results for Speaker 4 x-coordinates

4. This standard deviation range is also smaller than those of the x and y coordinates, suggesting more consistency in determining coordinates for the z-axis. The plots for the z-axis are shown in Figure 6.8.

Table 6.6 shows a side-by-side comparison of three sets of coordinates - two sets from the statistical mean and mode and one from the results of the distances measured manually. The variation between the coordinates computed from the manual distances and those obtained dynamically from SDIAS can be attributed to factors that play a role in the acquisition of the measurements. These include the calibration distance precision from the actual source of sound versus the measured point, the orthogonality of the microphone positions in terms of the axes and the equality of the distances along the axes for those positions. Inconsistencies in the placements of the microphone at the four positions, together with the calibration distance, affect these three quantities, and subsequently results in the deviations in the final coordinates. The effects of these distances are explained in Section 6.2.6.

Table 6.6: Comparison of coordinates

Coordinates by factor									
	MEAN			MODE			MANUAL		
Speaker #	X	Y	Z	X	Y	Z	X	Y	Z
1	-140.9	128.2	79.4	-141.9	130.5	81.1	-139.1	149.3	76.2
2	-88.3	-67.7	13.9	-89.4	-69.2	14.7	-90.4	-90.4	9.7
3	-71.5	125.8	13.5	-69.3	124.6	14.2	-71.9	144	12.1
4	103.7	107.9	15.9	100.7	112.8	12.8	107.2	115.1	7.2
5	80.8	-57.9	11.4	80.6	-58.7	12.9	75.4	-80.6	10.6
6	-82.9	37.7	12.2	-82.9	37.8	11.7	-69.2	33.4	11.4
7	132.6	-14.4	97.7	132.3	-15.5	97.8	138.6	-30.5	94.1
8	-141.7	-83.2	80.3	-142.8	-80.5	81.9	-154	-102.9	82.3

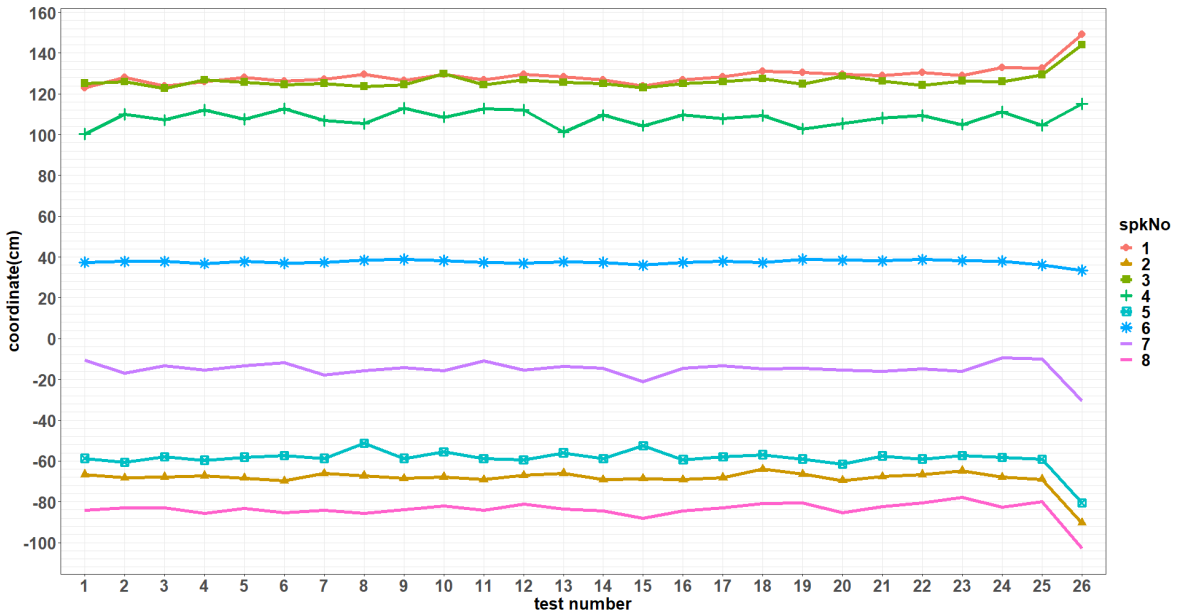


Figure 6.7: 26 results for all speakers' y-coordinates

An XML file which holds a layout for the 3D speaker coordinates is sent by the server to the client. An example of such a file is shown in Listing 6.1. The listing corresponds to test number 17 in Table C.1.

Listing 6.1: Small test room XML speaker layout for test 17

```
<?xml version="1.0" encoding="UTF-8"?>
        <config>
        <speaker number = "1" xpos = "-140.1" ypos = "127" zpos = "82.2"></speaker>
        <speaker number = "2" xpos = "-87.5" ypos = "-69.2" zpos = "11.8"></speaker>
        <speaker number = "3" xpos = "-70.7" ypos = "125.1" zpos = "16.5"></speaker>
        <speaker number = "4" xpos = "106" ypos = "109.6" zpos = "13.2"></speaker>
        <speaker number = "5" xpos = "80.7" ypos = "-59.3" zpos = "8.1"></speaker>
        <speaker number = "6" xpos = "-82.8" ypos = "37.3" zpos = "13.1"></speaker>
        <speaker number = "7" xpos = "132.1" ypos = "-14.5" zpos = "97.4"></speaker>
        <speaker number = "8" xpos = "-142.8" ypos = "-84.5" zpos = "81"></speaker>
        </config>
```

6.2.4 Large Room Test SDIAS Measurements

Tests for this configuration were run in a slightly different environment to that of small test room. The effect of sound reflection from the room surfaces and the magnitude of the orthogonal distances were investigated. The two factors could potentially have negative effect on the accuracy of the resultant 3D coordinates, as evidenced by the poor results

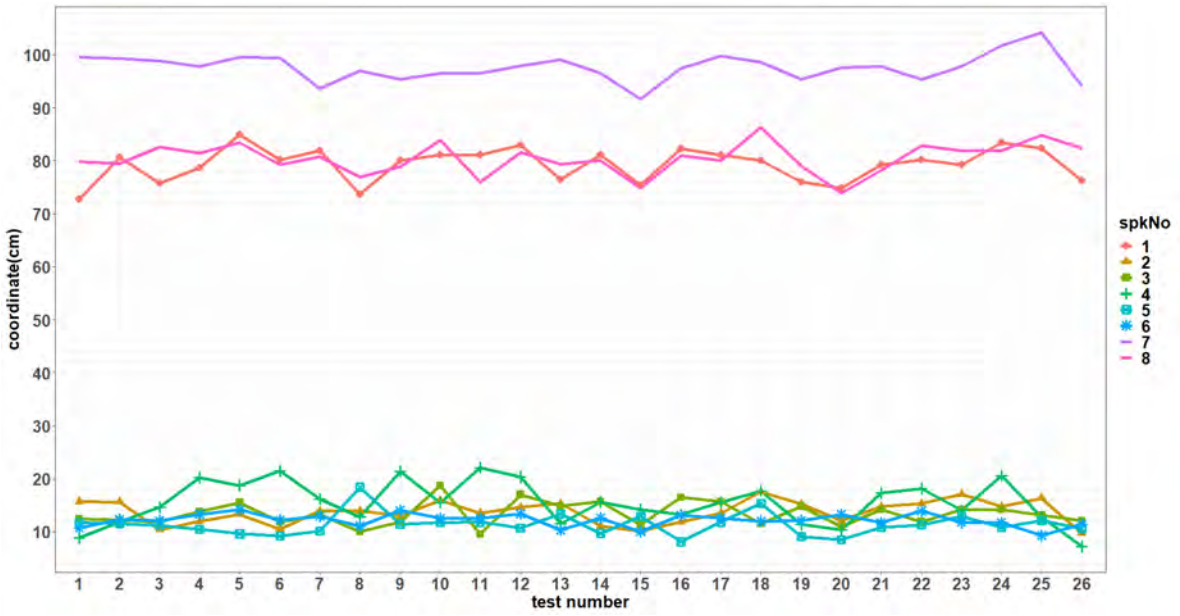


Figure 6.8: 26 results for all speakers' z-coordinates

from the initial tests in a larger room. To investigate the effect of reflection, a series of tests was run in a room with natural reflective walls, while others were run in a room with sound damping. The orthogonal distance was varied for each series. The variation distances used were 20cm, 40cm and 80cm. The tests series were run using a single speaker layout, that is, speakers were at the same positions for the reflective and damping tests, across the three orthogonal distances. This is the layout shown in Figures 6.3. This meant that the expected 3D coordinates for these series would be similar.

Coordinate translation (explained in Section 6.2.2) was used so as to reference the final coordinates with respect to one of the speakers in the layout - by default Speaker 1. The reference point, also called the origin, (0,0,0), was therefore the point on the floor below the reference speaker. The 3D coordinates of Speaker 1 would then be in the form (0,0,h), where h is the calculated z-coordinate, which is the height of the speaker from the floor.

The results of the tests are classified according to the combinations of the wall reflectivity and orthogonal distances. These combinations are labelled as follows:

- D20: for damped room and 20cm orthogonal distance,
- R20: for reflective room and 20cm orthogonal distance,
- D40: for damped room and 40cm orthogonal distance,
- R40: for reflective room and 40cm orthogonal distance,
- D80: for damped room and 80cm orthogonal distance, and
- R80: for reflective room and 80cm orthogonal distance.

The results for these combinations are given in Tables C.2 through C.7 of Appendix C. As with the previous small test room configuration, the tables show the x, y and z coordinates for Speakers 1 through 8. Also, the z-coordinate values from all the results are calculated from the x and y coordinates, without using the measured z microphone position. Some statistical analysis is given in the tables to describe the respective datasets. For each orthogonal distance, the results are plotted in a graph. The graphs are placed side-by-side for comparison between the results from the damped and reflective environments. For each point in a respective speaker's line plot, the first 10 coordinates are from SDIAS, while test number 11 and 12 are from the manual measurements. For test number 11, the z-coordinate was obtained from the microphone position placed at the z-axis, whereas for test number 12, the z-coordinate was obtained using only the x and y coordinates, as explained in Section 4.8.2.

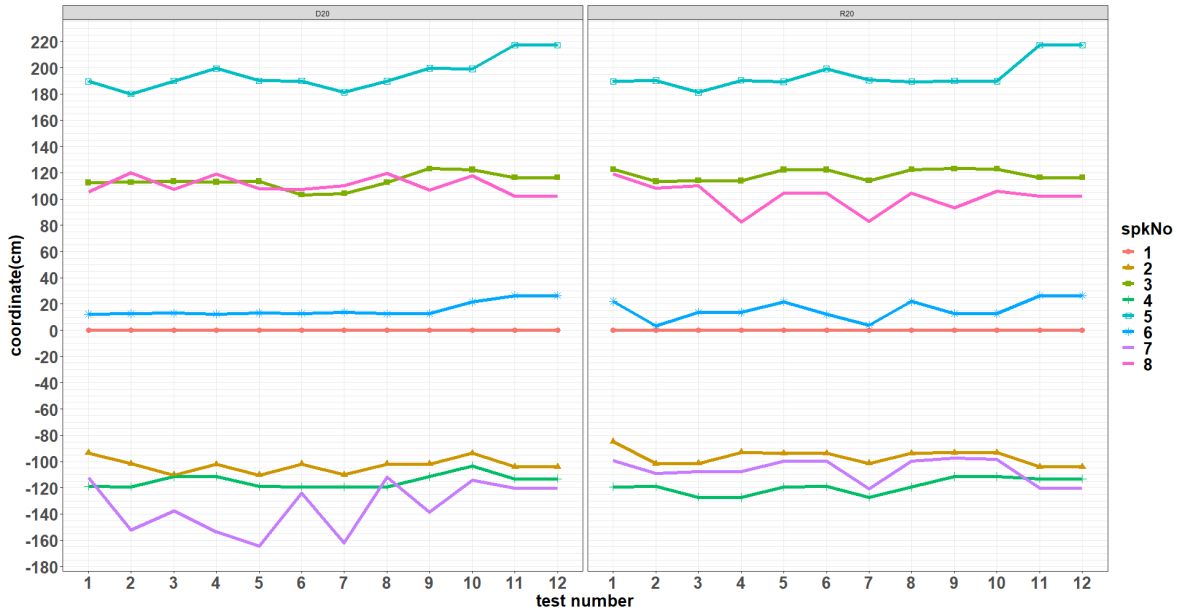


Figure 6.9: 20cm orthogonal distance speaker x-coordinates

D20 X-axis Results

From the results for the damped room with the microphone positions at 20cm orthogonal distance (D20), the range of the x-axis (excluding the values of Speaker 1 as those have been translated to always be zero) varies from 9.3cm (from Speaker 6) to 52.7cm (from Speaker 7). The range greatly exceeds the dimensions of the 15.1cm width of the speakers used for the tests, which are used as a reference for the error margins. Consequently, the

large range values imply less accuracy in determining the x coordinates. The standard deviation varies from 2.7cm to 19.5cm from the corresponding speakers. The upper value also slightly exceed the 19.1cm speaker width. From these tests, Speaker 6 was therefore most consistently determined, while Speaker 7 was the least consistently determined. In Figure 6.9 D20, Speaker 6's line plot is smoother than the rest, and Speaker 7's line plot shows more variability than the rest, as indicated by the standard deviation and range. The SDIAS results for test all speakers, except Speaker 5, are also close to the manual results, as evidenced by the proximity of their results to those for test number 11 and 12.

R20 X-axis Results

For the R20 configuration, the x-axis range extends from 9.6cm for Speaker 3, to 36.4cm from Speaker 8. The extent of range values exceeds the 15.1cm test speakers' width dimension, hence less accuracy in determining the x coordinates for these tests. The standard deviation varies from 4cm for Speaker 5 to 11.2cm for Speaker 8. This is a narrower spread than that of D20, implying improved consistency in determining the coordinates for this test configuration over the D20 results. The mean values also lie close to the manual x coordinate, as shown in Figure 6.9 R20. Speaker 3 was more consistently determined for this test, while Speaker 8 was least consistently determined.

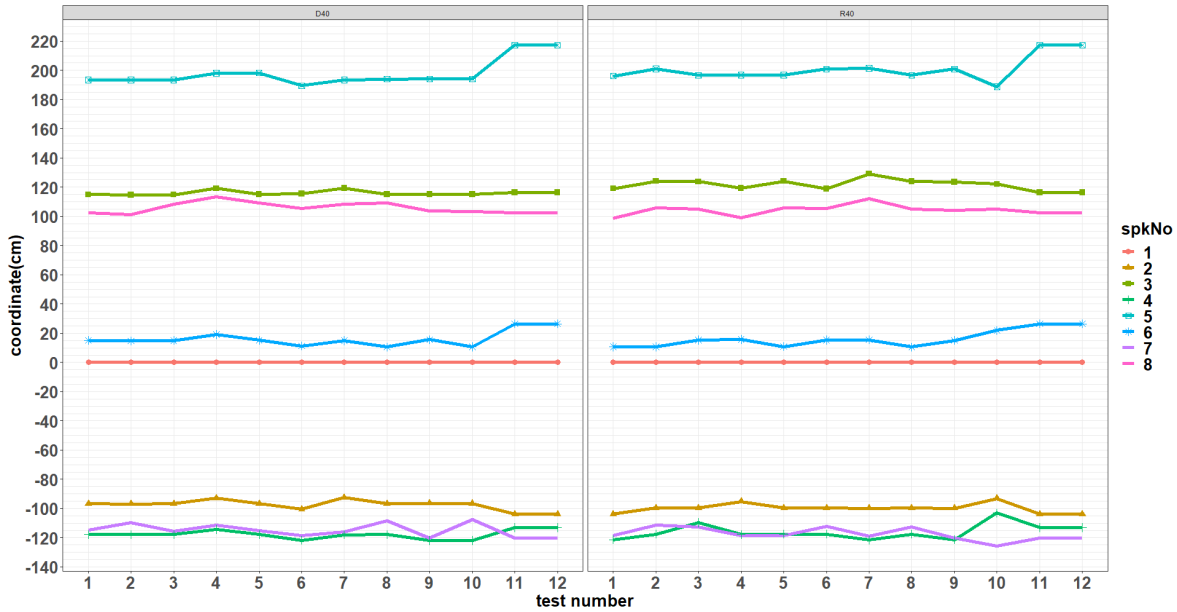


Figure 6.10: 40cm orthogonal distance speaker x-coordinates

D40 X-axis Results

The x-axis results for the D40 configuration range varies from 4.6cm for Speaker 3 to 12.7cm for Speaker 7. The range values are a significant improvement over the D20 and R20 results. They fall within the 15.1cm test speaker's width dimension, hence within acceptable error margin. As a result, these imply increased accuracy in the x coordinate determination. The standard deviation also has a narrow spread from 1.7cm for Speaker 3 to 4.1cm for Speaker 7. The relatively smaller range for the standard deviation suggests an increased consistency in the x coordinates determination for these tests. The mean values are close to the manual x coordinate. The best and worst determined speakers for this configuration are Speaker 3 and 7 respectively. This can be evidenced in Figure 6.10 D40, by the flatter plot for Speaker 3, while that of Speaker 7 is more variable than the rest of the other speaker plots.

R40 X-axis Results

The R40 x-axis range is from 10.3cm to 18.6cm for Speakers 3 and 4 respectively. This range also slightly exceeds the 15.1cm test speakers' width, but it still shows an improved accuracy in determining the x coordinates, especially over the D20 and R20 results. The standard deviation varies from 2.7cm for Speaker 2 to 5.5cm for Speaker 4. This spread is also a significant improvement over the D20 and R20 spread. It thus suggests more accuracy for the x coordinates. These statistics, however, shows less accuracy than that of the D40 results. The mean values also indicate more proximity to the manual x coordinate values across the speakers. The best consistently determined speakers are Speaker 2 and 3, while the least is Speaker 4, as depicted in Figure 6.10 R40.

D80 X-axis Results

The range for the x-axis for D80 varies from 4.1cm for Speaker 7 to 8cm for Speaker 8. The values fall within the 15.1cm test speaker's width dimension, hence renders them within acceptable limits. The range values are also a significant improvement over all the 20cm and 40cm test results. This implies increased accuracy in the x coordinate determination. The standard deviation also has a narrow extent of 1.3cm to 2.2cm. The narrow spread of the standard deviation additionally suggests a further increase in the consistency of the x coordinates determination for the 80cm orthogonal distance over the previous 20cm

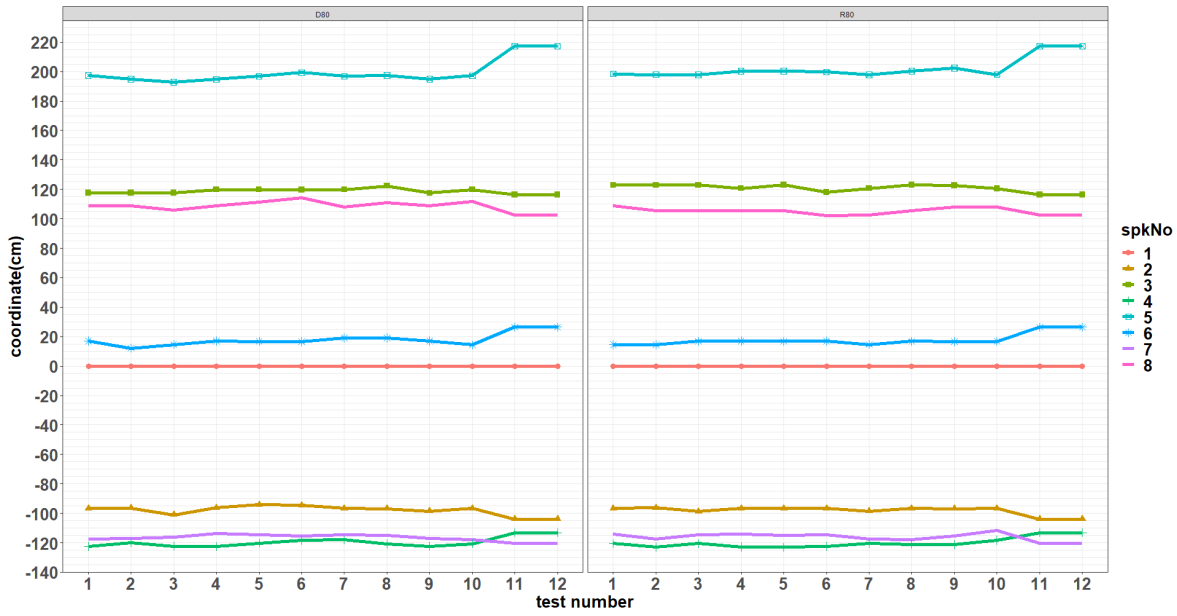


Figure 6.11: 80cm orthogonal distance speaker x-coordinates

and 40cm results. The mean values closely approximate coordinates from the manual x coordinate, adding to the observation of increased accuracy in the determination of the x coordinates. From Figure 6.11 D80, all line plots are flatter than those of the D20 and D40 results. This implies that the results improve with increase in the orthogonal distance. Speaker 7 and 8 are therefore the most and least consistently determined for this configuration, despite all speakers being more consistently computed than those for the 20cm and 40cm results.

R80 X-axis Results

The range for the x-axis for R80 varies from 2.4cm from Speaker 2 to 6.8cm from Speaker 8. The extent of the x-axis range is better than that of the damped environment at the same orthogonal distance - D80. Also, the values fall significantly below the 15.1cm test speaker's width dimension, hence acceptable. The range also implies increased accuracy in the x coordinate determination for all test in the large test room. The standard deviation extends from 0.8cm for Speaker 2 to 2.2cm Speaker 8 for all tests in the large test room. This the narrowest extent of the standard deviation in all the test for the large test room. The narrow spread of the standard deviation additionally suggests a further increase in the consistency of the x coordinates determination for the 80cm orthogonal distance over the previous 20cm and 40cm results. The mean values closely approximate coordinates

for the manual x coordinate, adding to the observation of increased accuracy for the x coordinates. From Figure 6.11 R80, the plots are the flattest of all configurations.

The results from the x-coordinates indicate a general improvement of accuracy in determining the 3D coordinates across most of the speakers. This suggests that the larger orthogonal distance has a positive effect on the accuracy of the results. An example of this improvement is Speaker 7 across the damped or reflective environments. The plots for this speaker get flatter from D20, where they are the most variable, to D40 where they are less variable, through to D80 where they are the least variable. A similar observation can be made for the reflective environments. For reflective and damped environments, it is not clear that the damped environment yields better results than the reflective one as it would be expected. For instance, Speaker 5 does not show significant differences between any of the pairs D20-R20, D40-R40 or D80-R80.

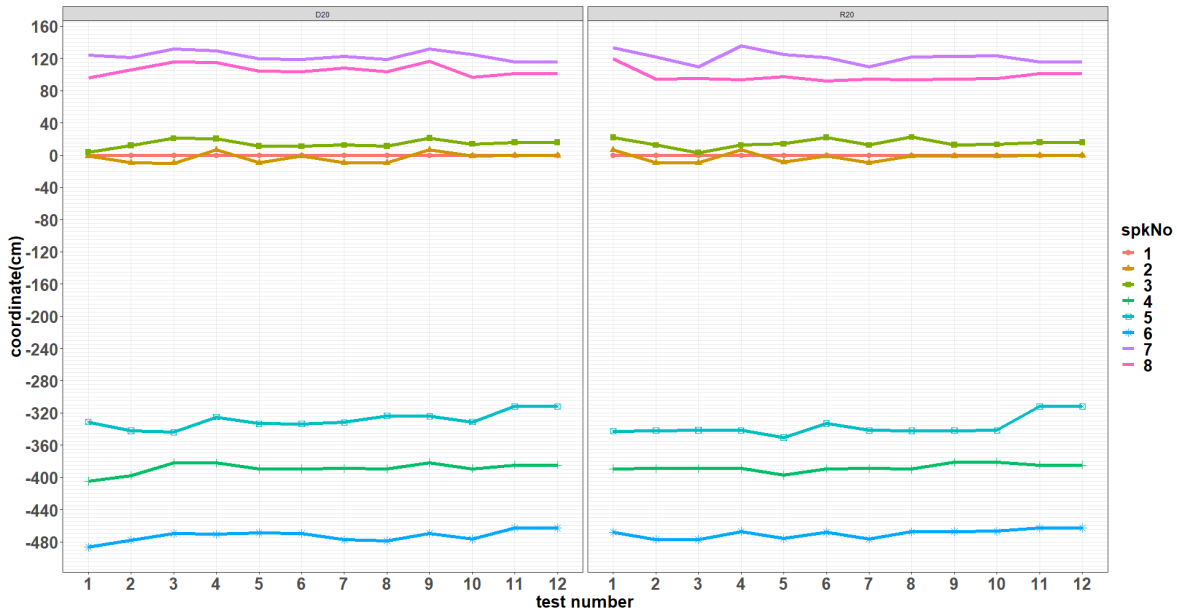


Figure 6.12: 20cm orthogonal distance speaker y-coordinates

D20 Y-axis Results

For the y-axis (also excluding the values of Speaker 1), the range is from 13.1cm, for Speaker 7, to 26.5cm for Speaker 4. This range slightly exceeds the acceptable boundaries of the test speaker 19.1cm dimension used as a reference for error margin for the y-coordinates. The upper bound of the range (26.5cm) is not too far from 19.1cm. The implication is therefore moderate accuracy in determining the y coordinates for this series.

The standard deviation varies from 4.8cm for Speaker 7 to 19.5cm for Speaker 8. This quantity slightly exceeds the speaker depth, and therefore adds to the moderate accuracy of the y values conformity. The mean also shows that the centre of the values lies close to the manually measured y coordinate for most speakers. The speakers with the most and least accuracy within test configuration are Speakers 7 and 8 respectively. Figure 6.12 D20 shows line plots for the y-coordinates for the 12 results. Speaker 7 plot seems least variable of all others, while Speaker 4 and 8 have the most variable plots, suggesting least consistency in determining their results.

R20 Y-axis Results

The y-axis range starts from 10.4cm, for Speaker 6, to 27.2cm for Speaker 8. The range exceeds 19.1cm test speaker depth. The implication is therefore less accuracy in determining the y coordinates. The standard deviation varies from 4cm for Speaker 5 to 11.2cm for Speaker 7. These values indicate a wide spread of the datasets, suggesting less accuracy in determining the y coordinates. The mean, however, lies close the manually measured y coordinate. Figure 6.12 R20 shows a graph for this configuration. The plot of Speaker 5 is flatter than the rest, while that of Speaker 7 is more jagged than the others, implying more and less consistency in determining the y-coordinates for these speakers respectively.

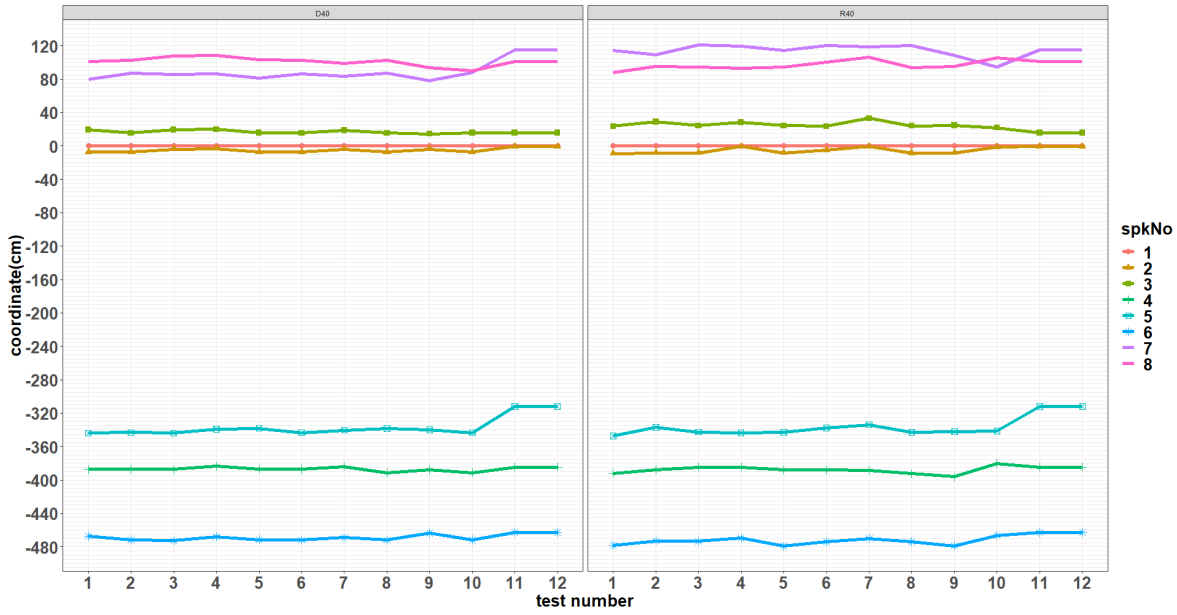


Figure 6.13: 40cm orthogonal distance speaker y-coordinates

D40 Y-axis Results

For the y-axis, the range starts from 4.1cm, for Speaker 2, to 18cm for Speaker 8. This is within the acceptable boundaries of the test speaker dimension of 19.1cm depth. The range therefore suggest improved accuracy in determining the y coordinates for this series, especially over the D20 and R20 results. The standard deviation varies from 1.7cm to 5.3cm for the corresponding speakers. The narrow range of standard deviation additionally suggests an improved accuracy for the y values. The mean shows that the values lie close the manually measured y coordinate. Figure 6.13 D40 shows an improvement in the determination of the y-coordinates for this configuration over the D20 and R20 results. This is shown by the flatter plots, in comparison the those of the D20 and R20 plots. Also, the plots for Speaker 2, 3 and 4 are flatter than the others, hence most consistently determined. Speaker 8 plot shows the most variable plot, hence is least consistently determined.

R40 Y-axis Results

The y-axis range starts from 8.4cm for Speaker 2, to 26.5cm for Speaker 7. The upper bound of the range exceeds the 19.1cm test speaker depth. This implies less accuracy in determining the y coordinates, despite being a slight improvement over the all the 20cm test results. The standard deviation varies from 3.2cm for Speaker 3 to 7.9cm for Speaker 7. This quantity, too, suggests an improved accuracy of the y values over the D20 and R20 results. The mean also shows that the centre of the values lies close to the manual y coordinate, except for the last two speakers (7 and 8). Figure 6.13 R40 shows a flatter plot for Speaker 2 and a variable plot for Speaker 7, rendering their results the best and worst consistently determined respectively.

D80 Y-axis Results

The y-axis range starts from 3.6cm for Speaker 2, to 10.8cm for Speaker 7. The values, too, lie within acceptable limits of the test speaker dimension of 19.1cm depth. The range therefore implies improved accuracy in determining the y coordinates. It is also a significant improvement over the y-coordinates test results for the large test room. The standard deviation varies from 1cm to 4.3cm for the corresponding speakers. The narrow range of standard deviation additionally suggests improved accuracy of the y values. The

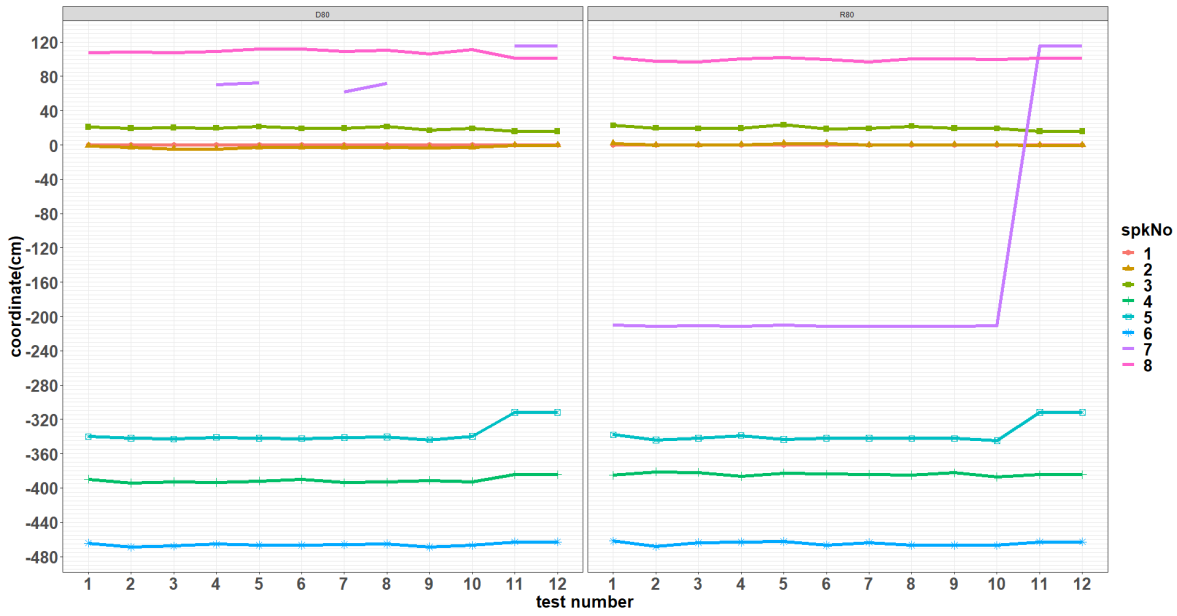


Figure 6.14: 80cm orthogonal distance speaker y-coordinates

mean lies close to the y coordinate from the manual measurements across the speakers, except for Speakers 5 and 7. Figure 6.14 D80 shows flatter plots than those of the 40cm 20cm test results, corroborating the observation of the improved accuracy from the statistics. However, some of the tests for this configuration could not yield values for Speaker 7. All speakers for this configuration, except Speaker 7, indicate increased accuracy in determined results.

R80 Y-axis Results

The y-axis range starts from 1.9cm for Speaker 7, to 7cm for Speaker 5. The values fall within the limits of the test speaker dimension of 19.1cm depth. The range therefore implies improved accuracy in determining the y coordinates for this series. It is also an improvement over the 20cm and 40cm results. The standard deviation varies from 0.6cm to 2.2cm. This is the narrowest range of standard deviation of all configurations. It therefore suggests improved accuracy of the y values. The mean lies close to the manually measured y coordinate across the speakers except for Speakers 7, which was consistently determined at the wrong value. Figure 6.14 R80 shows a graph for this configuration. The plots from this figure are the flattest of all plots from other configuration.

As with the x-coordinates results, the y-coordinates results show an improvement with an increasing orthogonal distance. This solidifies the observation that the increasing

orthogonal distance improves the accuracy of the results. Again, a similar observation to the x-coordinates can be made for the reflective environments, that no major differences are observed between the pair-wise damped and reflective environments.

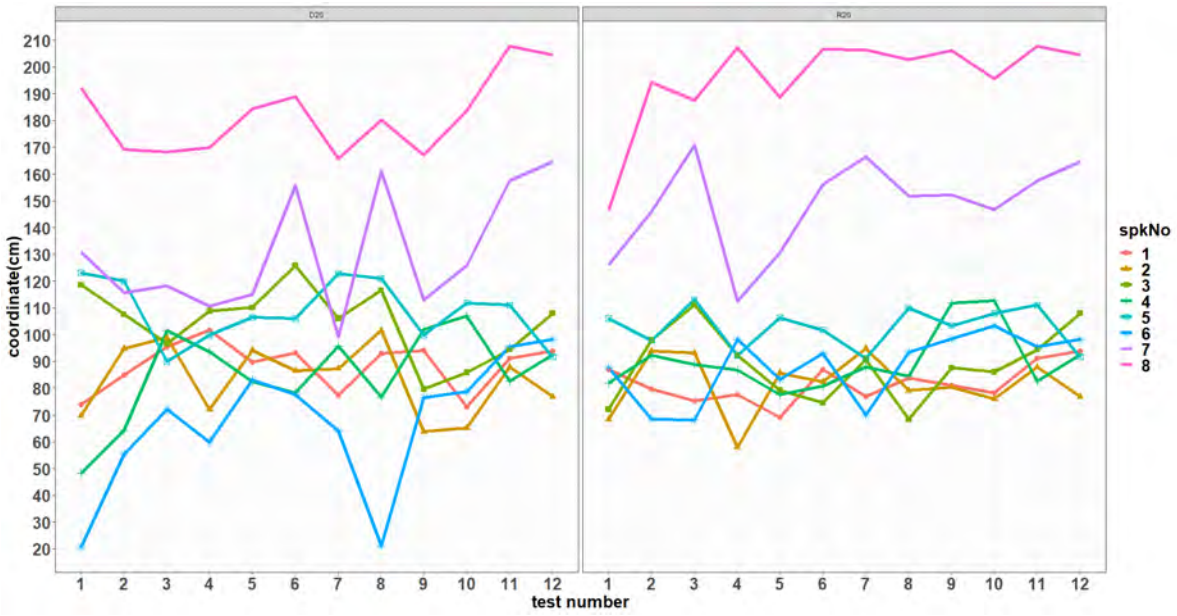


Figure 6.15: 20cm orthogonal distance speaker z-coordinates

D20 Z-axis Results

The range is from 26.5cm for Speaker 8, to 62.6cm for Speaker 6. This is wider than the other two axes. Consequently, this implies less accuracy in determining the z coordinates for this series. The standard deviation varies from 9.4cm for Speaker 1 to 21.7cm for Speaker 6, indicating less accuracy in the results. Figure 6.15 D20 shows a graph of the various plots for the respective speakers across all the tests. These graphs indicate very high variation, hence the worst consistency in determining coordinates.

R20 Z-axis Results

For this configuration, the range is from 18.1cm for Speaker 1, to 61cm from Speaker 8. This range exceeds the 24.7cm height dimension of the width of the test speakers. Consequently, this implies low accuracy in determining the z coordinate for this series, especially compared to the other axes. However, it is an improvement over D20 results. The standard deviation varies from 5.2cm for Speaker 1 to 17.5cm for Speaker 8, which

is also an improvement of over the results for D20. The plots of this configuration are shown in Figure 6.15 R20. All the plots are highly variable, yet less so than those of D20.

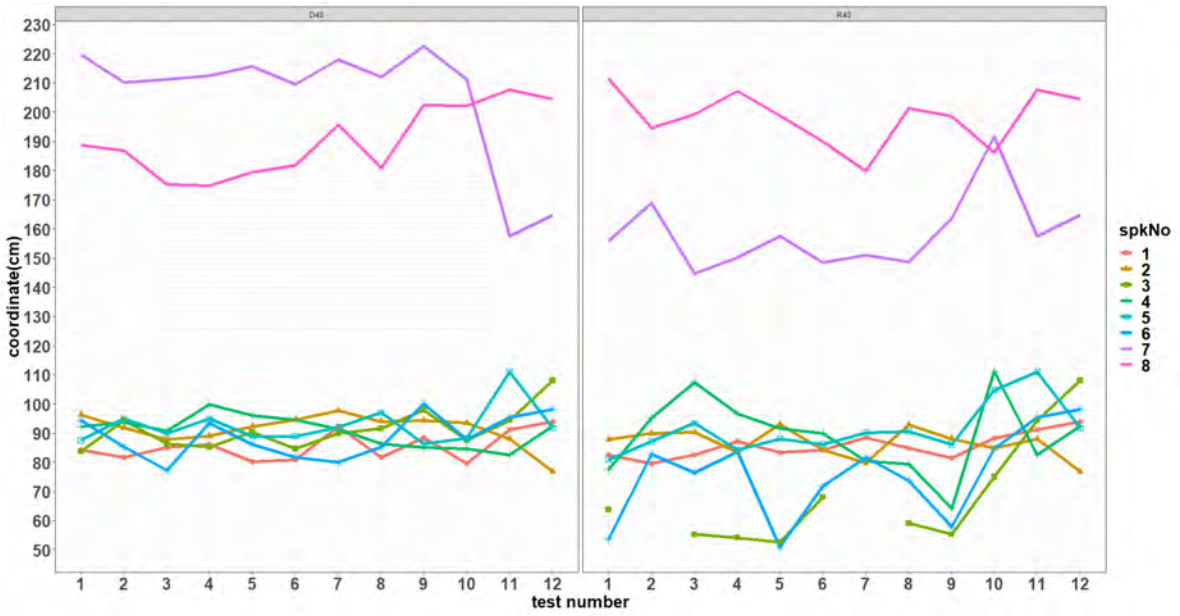


Figure 6.16: 40cm orthogonal distance speaker z-coordinates

D40 Z-axis Results

The range is from 9.9cm for Speaker 2, to 26.5cm for Speaker 8. This is a slightly wider range than the other two axes of the 40cm test results. However, the range, as with the x and y coordinate ranges, is an improvement over the D20 and R20 results. The upper bound of the range (26.5cm) slightly exceeds the 24.7cm height of the test speakers. Consequently, this implies less accuracy in determination process. The standard deviation varies from 2.9cm to 9.5cm from the same speakers as the range. The mean of the z coordinates also lies close to the expected manually measured values. The three statistics (range, standard deviation and mean) show a significant improvement over both D20 and R20 results. This is shown in Figure 6.16 D40, where the line plots for the respective speaker tests are less variable than those from the D20 and R20 plots. Speaker 2 and Speaker 8 are best and worst consistently determined for this configuration.

R40 Z-axis Results

The z-axis range is from 8.9cm for Speaker 1, to 47.1cm for Speaker 7. The range significantly exceeds the 24.7cm height dimension of the test speakers. Consequently, it implies

less accuracy in determining the z coordinates than the other axes for the same orthogonal distance. It is, however, an improvement over the D20 and R20 results. The standard deviation varies from 2.8cm for Speaker 1 to 13.6cm for Speaker 4. This is also an improvement over the D20 and R20 results, as shown in Figure 6.16 R40, by the reduction in variability of the respective speaker plots. Speaker 1 shows the least variability in its plot, making it the most accurately determined, while Speakers 4 and 7 show the most variability, making them the least accurately determined in this configuration.

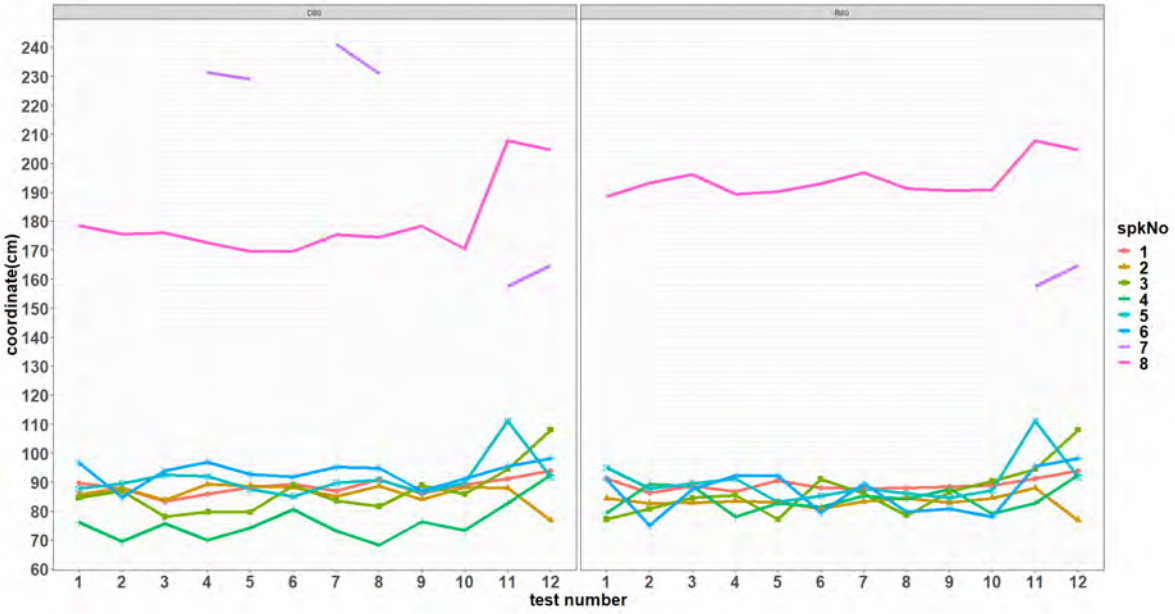


Figure 6.17: 80cm orthogonal distance speaker z-coordinates

D80 Z-axis Results

The z-axis range is from 5.4cm for Speaker 2, to 12.2cm for Speaker 4. This range is an improvement over the 20cm and 40cm results. The 12.2cm upper bound of the range is within the 24.7cm height dimension of the test speakers. Consequently, this implies improved accuracy in determining the z coordinates. The standard deviation varies from 2cm for Speaker 2 to 9.5cm for Speaker 7. The z coordinate mean lies close to the manually measured values. These statistics show a general improvement over all the 20cm and 40cm results. Figure 6.17 D80 shows the plots for this configuration. This indicates an improvement in the accuracy, as shown by the reduction of variability in the plots. Speaker 2 shows more consistency, while the least is shown by Speaker 5. Some tests, such as test number 1 to 3, for Speaker 7, could not yield results. This is due to the distance input values lying outside, due to the wrongly detected tone peaks by the

determination algorithm, the various trigonometric functions used.

R80 Z-axis Results

The z-axis range is from 3.6cm for Speaker 2, to 6.2cm for Speaker 6. This range is an improvement over the 20cm and 40cm results. The upper bound of the range (6.2cm), lies within the 24.7cm height of the test speakers, hence less improvement in determining the z coordinates. The standard deviation varies from 1cm for Speaker 2 to 9.5cm for Speaker 6. The z coordinates mean lies close to the manually measured values except for Speakers 5 and 7. Speaker 7 results are affected by the incorrect y coordinates (see Figure 6.14 R80), as such, they automatically become incorrect. The three statistics show an improvement over both the test results for the 20cm and 40cm values for both the damping and reflective results, and also over the 80cm orthogonal distance damping results from D80. Figure 6.17 R80 plots are the least variable of all plots in the large test room.

There is an observed general improvement of the results as the orthogonal distance increasing, as with the other axes, also noting that the z-coordinates for these configurations are obtained from the x and y coordinates, without using the z microphone position. This means there will be cumulative errors from those two axes, which will likely be compounded by the use of two axes values. No changes between the reflective and damped environment results for the z-axis.

The coordinate results from both tables indicate close proximity to the physical layout as all coordinates are reasonably accurate compared to the manual measurements. However, other statistics indicate some significantly wide ranges and standard deviations per axis. Both quantities imply less consistency in the determined coordinates, albeit their proximity to the results calculated using the manually measured distances (test number 11 and 12).

The results for the large test room show an increasing accuracy as the orthogonal distance increases (from 20cm, through 40cm to 80cm). This is due to the reduction in the margin of error obtained from the calculated microphone-to-speaker distances as the orthogonal distance increases. Another reason is the reduction of sensitivity of the trigonometric calculations due to the increase in the angle sizes of the involved amongst the various positions and the associated distances. On the other hand, the damping does not seem to have enhanced the accuracy over reflective surfaces as reflections were suspected to

have an impact on the accuracy of the calculated coordinates. In most cases the reflective environment results were, on the contrary, better than those from the environment with damping.

6.2.5 Client UI

The 3D coordinates discussed are displayed on the client UI. Figure 6.18 is an example of the client UI screenshot for results corresponding to D20 test number 1 in Table C.2. The client UI screenshot is annotated to highlight the following information it provided:

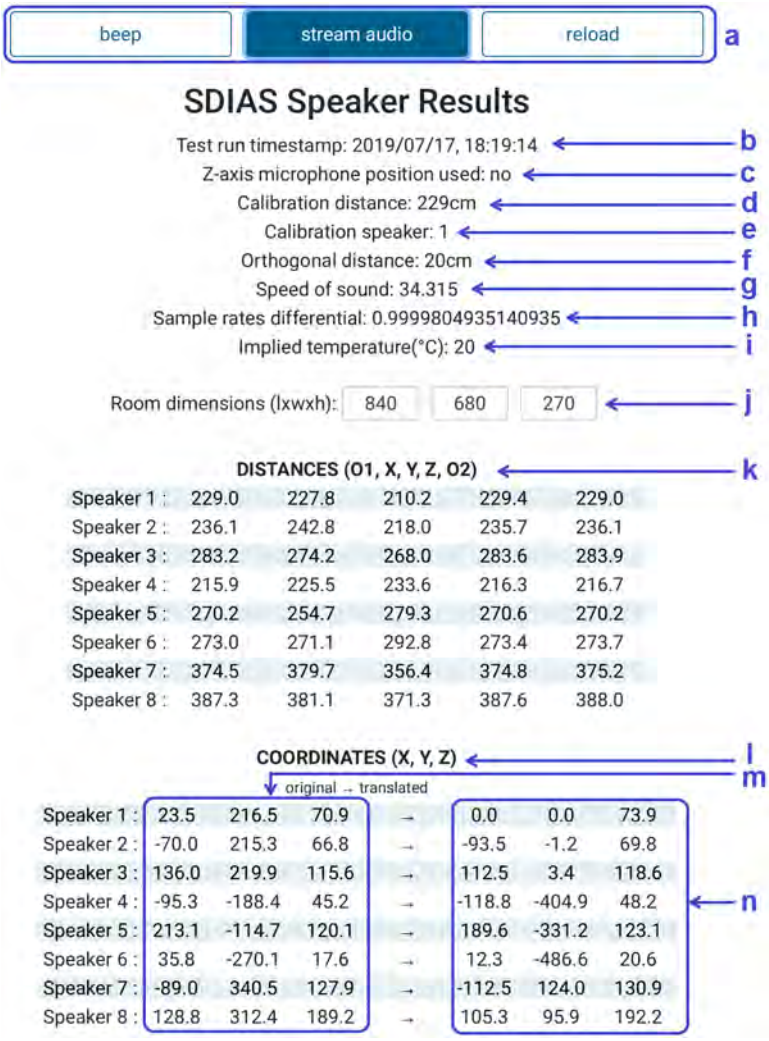


Figure 6.18: L1D20 test 1 client results

- a: three buttons for controlling events for beep, stream and reload
- b: timestamp for the results as given by the server

- c: indication of whether or not the z microphone position was used
- d: the calibration distance
- e: the calibration speaker
- f: the orthogonal distance
- g: the value of speed of sound used
- h: the sample rates differential, SRD in equation 4.4
- i: the temperature corresponding to the speed of sound used
- j: test room dimensions
- k: distances extracted from the tones audio samples, from O1 as the first origin microphone position, through X, Y, Z and back to the origin as O2
- l: the resulting 3D coordinates
- m: the original coordinates with respect to the microphone position origin O
- n: the translated coordinates relative to the point on the floor directly below the reference speaker, Speaker 1 in the UI.

6.2.6 Sensitivity of the Results to the Orthogonal and Calibration Distances

The calibration and orthogonal distances affect the outcome of the determination process. Figure 6.19 shows these distances. The calibration distance is the green line, marked k, from the origin to the speaker, while the orthogonal distances are the red lines OX = OY = OZ shown by the right-angles on the axes forming the 3D space for the microphone positions. Since these quantities are obtained from manual portion of the speaker determination process, precision in their measurement also contributes to the accuracy of the results.

The Effect of the Orthogonal Distance

The trigonometric results are sensitive to deviations in the values used for the calculations. Imprecise placement of the microphone positions during streaming can lead to significant differences in the final coordinates. An example is Speaker 3 in Table 6.6, where the x-coordinate values from the mean, mode and manual sets are respectively -71.5cm, -69.3 and -71.9cm. The largest difference is 2.6cm between the last two of the three values.

However, for the y-coordinate of Speaker 3, the values from the sets are 125.8cm, 124.6cm and 144cm. These give 19.4cm and 1.2cm as the largest and smallest differences respec-

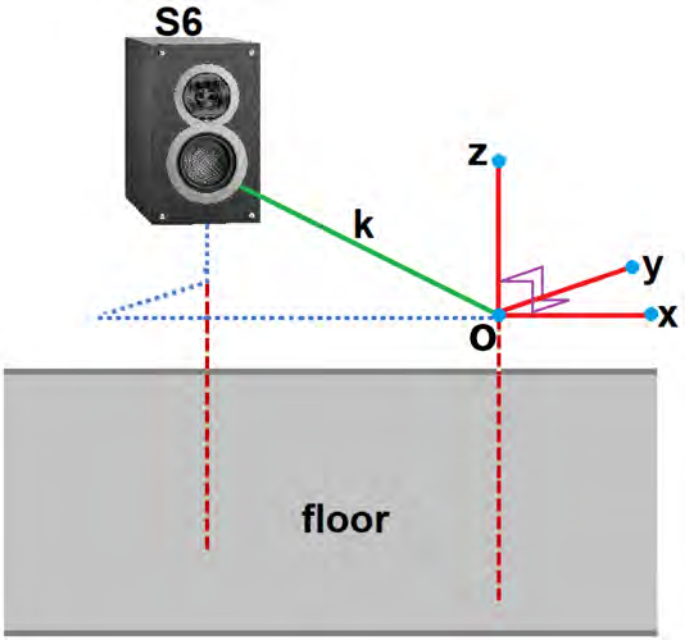


Figure 6.19: Orthogonality and the calibration distance

tively amongst the values. This large deviation slightly exceeds the y-axis's margin of error reference which is the test speaker's depth dimension of 19.1cm. The results could be attributed to the sensitivity of the trigonometric calculations to the distances of the microphone positions. This issue is illustrated in Figure 6.20.

The figure is a screenshot of a Node.js program output that tests the sensitivity of the trigonometric calculations for the speaker distances and coordinates involved. The figure shows how small variations in the distance of the microphone position along the y-axis from the origin significantly influence the corresponding y-coordinate. It shows a range of Y distance values that generate the range of y-coordinate values for Speaker 3 found in Table 6.6, namely 144cm, 125.8cm and 124.6cm.

The Y distances and their corresponding y-coordinates are highlighted in yellow, red and green in the figure. The two limits from the table for this speaker are obtained from the corresponding Y distances of 111.1cm and 116.6cm. The difference in the input Y distances are slightly less than 6cm apart while the difference in the output coordinates is 19.4cm. This shows a relatively significant sensitivity of the output coordinates to the input distances. The sensitivity is also shown by the calculations highlighted in blue and yellow. These show how a small granularity of 1mm in the measurements yields numerically different results - 111cm does not yield a valid y-coordinate (NaN in the Node.js code, which signifies that the result of the function cannot yield a number -

Not-A-Number (MDN, 2019a)) while 111.1cm yields 143.7cm.

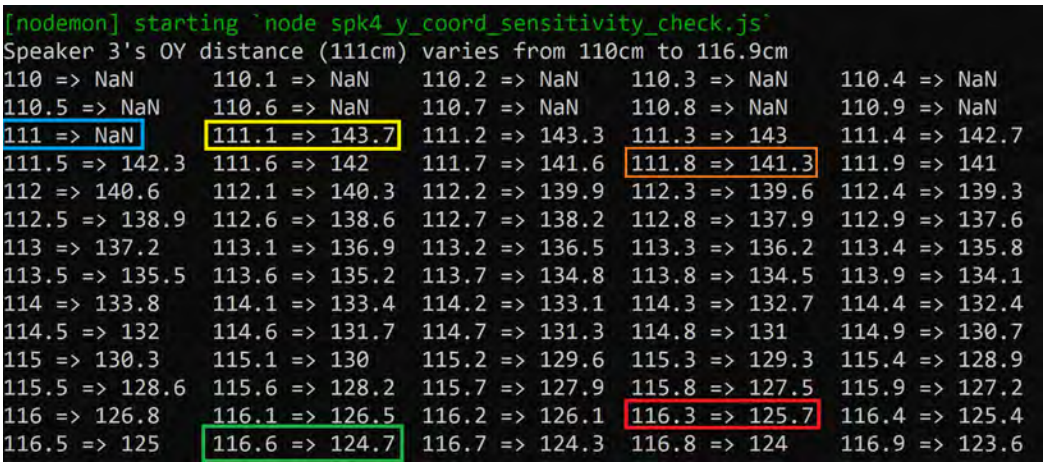


Figure 6.20: Sensitivity of coordinates computations

At the sampling rate of 48,000Hz and 343ms^{-1} for speed of sound, one sample is equivalent to 7.1mm and this is the smallest granularity of measure in this system. This is approximately seven times the difference which delineates between a value and a NaN in the illustrative program above. The domain of the arc cosine function used is from -1 to 1, from equation 4.10 in Chapter 4. The JavaScript NaN produced from the calculations in this equation is due to the input value lying outside the domain of the arc cosine function. This means a tone detected one sample off could render the arc cosine's input value outside the function domain, hence yield a NaN. This further shows the sensitivity of the system in general to the various quantities used. For the case of Speaker 3 on the y-axis, one sample difference leads to a 2.4cm change in the y-coordinate. For example, a change from 111.1cm to 111.8cm (the yellow and orange boxed pairs) in the Y distance results in a change from 143.7cm to 141.3cm in the y-coordinate.

The Effect of the Calibration Distance

Similarly to the orthogonal distance, precision on the measurement of the calibration distance has a bearing on the outcome of the results. Figure 6.21 shows how a variation in the calibration distance affects the outcome of the 3D coordinates shown in Figure 6.18. The variation is in increments of 10cm and in the range ±50cm around the 229cm of the physically measured calibration distance which is shown in the shaded row. The first column, *cDist*, is the variation of the calibration distance, where *xi*, *yi* and *zi* are the three coordinates for speaker i (i = 1,...,8). From the table, a variation of 10cm

introduces a 0.6cm (at x1) and 9.9cm (at y6) at extremes, which increases proportionally, to the 50cm variation with 3cm at x1 and 49.5cm at y6. The coordinates are sensitive to the variations in the measurement of the calibration distance, though to a relatively lesser extent than they are to the variations of the orthogonal distance.

cDist	x1	y1	z1	x2	y2	z2	x3	y3	z3	x4	y4	z4	x5	y5	z5	x6	y6	z6	x7	y7	z7	x8	y8	z8
179	20.5	169.5	53.9	-53.3	170.1	53.3	113.4	181.8	92.1	-71.5	-144.3	40.2	174.4	-92.0	98.0	31.0	-220.6	10.7	-75.9	295.3	111.0	113.3	272.5	163.2
189	21.1	178.9	57.3	-56.6	179.2	56.0	117.9	189.4	96.8	-76.2	-153.1	41.3	182.2	-96.5	102.4	32.0	-230.5	12.2	-78.5	304.3	114.4	116.4	280.5	168.4
199	21.7	188.3	60.7	-60.0	188.2	58.7	122.5	197.0	101.5	-81.0	-161.9	42.3	189.9	-101.1	106.9	32.9	-240.4	13.7	-81.2	313.4	117.8	119.5	288.5	173.6
209	22.3	197.7	64.1	-63.3	197.3	61.4	127.0	204.6	106.2	-85.8	-170.8	43.3	197.6	-105.6	111.3	33.9	-250.3	15.0	-83.8	322.4	121.1	122.6	296.4	178.8
219	22.9	207.1	67.5	-66.7	206.3	64.1	131.5	212.2	110.9	-90.5	-179.6	44.3	205.4	-110.1	115.7	34.8	-260.2	16.4	-86.4	331.4	124.5	125.7	304.4	184.0
229	23.5	216.5	70.9	-70.0	215.3	66.8	136.0	219.9	115.6	-95.3	-188.4	45.2	213.1	-114.7	120.1	35.8	-270.1	17.6	-89.0	340.5	127.9	128.8	312.4	189.2
239	24.1	225.9	74.3	-73.3	224.4	69.5	140.5	227.5	120.3	-100.1	-197.2	46.1	220.9	-119.2	124.6	36.7	-280.0	18.9	-91.7	349.5	131.2	131.9	320.3	194.4
249	24.7	235.3	77.7	-76.7	233.4	72.2	145.1	235.1	125.0	-104.8	-206.1	47.0	228.6	-123.8	129.0	37.7	-289.9	20.1	-94.3	358.6	134.6	135.0	328.3	199.6
259	25.3	244.7	81.1	-80.0	242.5	74.8	149.6	242.7	129.7	-109.6	-214.9	47.8	236.3	-128.3	133.4	38.6	-299.8	21.4	-96.9	367.6	138.0	138.1	336.3	204.8
269	25.8	254.1	84.5	-83.4	251.5	77.5	154.1	250.3	134.4	-114.4	-223.7	48.6	244.1	-132.8	137.9	39.6	-309.7	22.5	-99.5	376.7	141.4	141.2	344.3	210.0
279	26.4	263.5	87.9	-86.7	260.6	80.2	158.6	257.9	139.0	-119.2	-232.6	49.4	251.8	-137.4	142.3	40.5	-319.6	23.7	-102.2	385.7	144.7	144.3	352.2	215.2

Figure 6.21: Sensitivity of coordinates computations

Despite its sensitivity being less than that of the orthogonal distance, errors in the measurement of the calibration distance still introduce errors in the final coordinates, as such, precision is crucial during the measurement step so as to unsure minimal errors.

6.3 Chapter Summary

This chapter gave a detailed description of the various comprehensive tests performed. The accounts of the test configurations were given, together with the test procedures, inputs, expected outputs and finally the obtained results. The results from the two configurations used were presented and analysed. The details of the sensitivity of the trigonometric calculations used in the input values was observed. This resulted from the placement of the microphone at the various locations, as this affects the various resultant triangle lengths and angles used in the involved trigonometric functions. Speaker 3 in the small test room was used to illustrate the sensitivity to seemingly slight variations, which indicated how one sample difference in the trigonometric input resulted in 2.4cm change in the corresponding coordinate output in the y-axis.

A discussion of the results from a larger testing environment highlighted the problems associated with the orthogonality precision and how this affected the resulting coordinates. The problems include the precision in placing the microphone at the precise right-angles as the core the computations rely on this precision. A distinct observation was that the accuracy of the results generally improved with increasing orthogonal distance. It was,

however, not so clear that the reflection affected the results as the results from the damped environments did not generally show significant variations within these contexts, in the accuracy of the results.

An analysis of the effects of the errors introduced in the measurements of the orthogonal and calibration distances was also given. This demonstrated how the results are affected by these distances, with the coordinates being more sensitive to the orthogonal distance than the calibration distance. The speaker determination algorithm for SDIAS was tested in room environments of approximately up to 8m long, 6.8m wide and 2.7m high room dimensions. The next chapter concludes the research work for SDIAS.

Chapter 7

Conclusion

This research has investigated the use of mobile devices to determine the coordinates of loudspeaker configurations in a 3D space. The research aimed to develop a client-server system for the determination process. The system aimed to use a mobile device with an attached microphone for recording tones. A tones server uses the capabilities of the Ethernet AVB standard to deliver the tones at equally-spaced intervals. This chapter concludes the work done towards this system.

7.1 Chapter Summaries

Chapter 2: the chapter gave a background review of the technologies available within the field of immersive sound. The height dimension in audio was introduced, together with the various techniques used for creating its content and reproducing it for playback. Immergo (Foss & Rouget, 2015) was introduced as an example 3D spatial audio content creation system. Common spatialisation algorithms used by the 3D spatial audio content systems, such as Immergo, were introduced. These include DBAP, VBAP, Ambisonics and WFS. A discussion on the various speaker configurations deployed by each algorithm, together with benefits and limitations of the algorithm were highlighted. A brief introduction to the 3D audio formats was given, followed by a concluding discussion of various speaker layouts used for delivering immersive sound. Throughout the chapter, the need for the knowledge of speaker coordinates at various stages of immersive sound workflow was pointed out.

Chapter 3: an overview of the proposed system solution was described in this chapter. The chapter gave details of the architectural client-server style of the system, and how the

style was chosen to match with that of Immergo. The client recorded tones as they were transmitted by the tones server. The recorded tones were then processed and used to determine the 3D coordinates of a given speaker layout. The 3D loudspeaker coordinates were saved as an XML configuration, which could be accessed by Immergo or other 3D spatial systems to use in their spatialisation algorithms.

Chapter 4: Various technologies were used to design and implement SDIAS. These were given in detail in this chapter. The novelty of the solution to solving the clock drifts common in networked environments was described. This approach sought to achieve synchronisation of the system components in both the time and frequency domains. User interaction with the system was also explained, with the client UI screenshots used for highlighting some of the steps in the process.

Chapter 5: The design and implementation processes were discussed in this chapter. The chapter gave details of the system design model steps, which led to four use cases which defined the system goals. The class and sequence diagrams were also created for visualising the system behaviour and structure. System implementation followed the system design modelling, giving detailed discussions of the steps involved in each of the use cases by the various components of SDIAS (the Node.js server, the web client and the JUCE tones server). Completion of SDIAS development led to the testing phase.

Chapter 6: This chapter gave the details of the two testing environments used, together with the procedure followed and the obtained results for a 6.0.2 speaker configuration used by SDIAS. The results showed a varying degree of accuracy, which showed improvement as the magnitude of the orthogonal distance of the microphone positions increased.

7.2 Review of Research Goals

This research set out to achieve three objectives based on the hypothesis that mobile devices can be used to determine speaker coordinates for immersive sound configurations. This section discusses the degree to which the objectives were achieved, pointing to achievements and failures of the system.

1. Review current literature and standards pertaining to immersive audio content, loudspeaker configurations and sound spatialisation: the discussion on the various technologies used in creating and rendering immersive audio content was motivated by this objective. Spatialisation algorithms discussed in Chapter 2, together with

the various 2D and 3D audio content and speaker configurations sought to further fulfil this objective.

2. Explore various approaches that could be used to determine 3D loudspeaker coordinates: various approaches were explored towards achieving a speaker position measurement system. These include the potential sound sources for use in playing tones from the loudspeakers, as well as the technologies used. Additionally the microphones were also tested on various client devices which included computer desktops and laptops, tablets and smartphones, and including those embedded in the mobile devices. These approaches, described in Chapter 3 and 4, were geared towards building a solution that could yield the most accurate 3D speaker coordinates.
3. Design and implement a system that will enable speaker configuration via a mobile device: the system design model, implementation and tests results provide evidence of achieving this objective. These were discussed in Chapter 5 and 6, with test results indicating the feasibility of using mobile devices for determining speaker coordinates.

7.3 Achievements and Limitations

The work carried out in this research has provided some evidence that the use of mobile devices to determine 3D speaker coordinates is feasible. This is indicated by the results described in the testing phase.

The tests have yielded reasonably accurate coordinates compared to those measured manually during the testing phases. However, it was evidenced in some of the results that increasingly accurate 3D coordinates were mostly obtained with the use of external microphones attached to the mobile devices, while the embedded microphones performed poorly.

The results also improved with the increase in the magnitude of the orthogonal distance used for microphone positions. The tests performed were limited primarily to two test rooms of small to medium size (with the largest length dimension of 8.4m).

Despite the limitations, this research has provided an alternative semi-automatic method to the problem of speaker coordinates measurement. A fully automatic system was not possible to achieve with commodity hardware.

7.4 Future Work

While the work carried out in this research has demonstrated the feasibility of using readily available (and low-cost) mobile devices to measure speaker coordinates, it also leaves room for more work to be carried out due to various limitations of this research.

- With the myriad of mobile devices available, the use of only a few devices on the client was a limitation and this could be improved by increasing the range of mobile device types, to determine how various devices perform with external or embedded microphones.
- Besides the omnidirectional microphones used in SDIAS, it would be useful to test other types, such cardioid microphones, especially with the mechanism deployed by the system, to determine their potential for speaker locations measurements.
- Other types of sound source techniques such as MLS could be explored as substitutes for the sine-based sound sources used in this research, accounting for their various requirements such as longer execution times, as opposed to the short times - ≈10.667ms long sine wave - used in SDIAS, used by MLS (Policardi, 2011).
- It would desirable to perform tests in a large environment to see how the SDIAS algorithm for determining the coordinates performs under real-world conditions, as tests were limited to small to medium test environments.
- Various interaction styles available in the mobile devices were not explored. This includes the use of various sensors such as gyroscopes. Future work could explore this avenue, to determine the possibilities they could bring into the speaker positions measurement system implemented in the research.

Bibliography

Adenot, P., & Toy, R. (2018). Web audio api. *s Candidate Recommendation [online specification], Version*, *1*.

Ahveninen, J., Kopčo, N., & Jääskeläinen, I. P. (2014). Psychophysics and neuronal bases of sound localization in humans. *Hearing research*, *307*, 86–97.

Andersen, R. L., Crockett, B. G., Davidson, G. A., Davis, M. F., Fielder, L. D., Turner, S. C., ... Williams, P. A. (2004). Introduction to dolby digital plus, an enhancement to the dolby digital coding system. In *Audio engineering society convention 117.*

Auro Technologies. (2015). *Auro-3d home theater setup installation guidelines.* Retrieved from `https://www.auro-3d.com/wp-content/uploads/documents/Auro-3D-Home-Theater-Setup-Guidelines_lores.pdf` (Accessed on 22 Oct 2019)

Auro Technologies. (2019). *Auro-3d/auro technologies: Three-dimensional sound.* Retrieved from `https://www.auro-3d.com/professional/superior-solution/` (Accessed on 1 Oct 2019)

Baalman, M. A. (2010). Spatial composition techniques and sound spatialisation technologies. *Organised Sound*, *15*(3), 209–218.

Barco, A. T. (2015). *Auromax whitepaper.* Retrieved from `https://www.auro-3d.com/wp-content/uploads/documents/AuroMax_White_Paper_24112015.pdf` (Accessed on 24 Jan 2020)

Berkhout, A. J., de Vries, D., & Vogel, P. (1993). Acoustic control by wave field synthesis. *The Journal of the Acoustical Society of America*, *93*(5), 2764–2778.

Blauert, J. (1983). *Spatial hearing: the psychophysics of human sound localization.* MIT press.

Breebaart, J., Herre, J., Jin, C., Kjörling, K., Koppens, J., Plogsties, J., & Villemoes, L. (2006). Multi-channel goes mobile: Mpeg surround binaural rendering. In *Audio engineering society conference: 29th international conference: Audio for mobile and handheld devices.*

Burnett, D., Bergkvist, A., Jennings, C., Aboba, B., Bruaroey, J., & Bostrom, H. (2018). Webrtc 1.0: Real-time communication between browsers. *s Candidate Recommen-*

dation [online specification], Version, 1.

Burnett, D., Bergkvist, A., Jennings, C., Aboba, B., Bruaroey, J., & Bostrom, H. (2019). Media capture and streams. *s Candidate Recommendation [online specification], Version, 1.*

Cheng, C. I., & Wakefield, G. H. (1999). Introduction to head-related transfer functions (hrtfs): Representations of hrtfs in time, frequency, and space. In *Audio engineering society convention 107.*

Daniel, J., Moreau, S., & Nicol, R. (2003). Further investigations of high-order ambisonics and wavefield synthesis for holophonic sound imaging. In *Audio engineering society convention 114.*

Dehaan, D. (2018). *Sound localization & spatialization techniques for portable digital environments.* Retrieved from `https://www.danielrdehaan.com/sound-spatialization` (Accessed on 31 Jan 2020)

De Vito, L., Rapuano, S., & Tomaciello, L. (2008). One-way delay measurement: State of the art. *Instrumentation and Measurement, IEEE Transactions on*, *57*(12), 2742–2750.

Digital Theater Systems. (2016). *Home theater sound gets real.* Retrieved from `https://dts.com/dtsx` (Accessed on 1 Oct 2019)

Digital Theater Systems. (2018). *Home solutions.* Retrieved from `https://dts.com/professional/home-solutions` (Accessed on 5 Oct 2019)

Digital Theater Systems. (2019). *Dts:x pro puts you there.* Retrieved from `https://dts.com/dtsxpro` (Accessed on 29 Jan 2020)

Dolby Laboratories. (2003a). *7.1 surround sound setup.* Retrieved from `https://www.dolby.com/us/en/guide/speaker-setup-guides/7.1-virtual-speakers-setup-guide.html` (Accessed on 1 Oct 2019)

Dolby Laboratories. (2003b). *9.1 surround sound setup.* Retrieved from `https://www.dolby.com/us/en/guide/surround-sound-speaker-setup/9-1-setup.html` (Accessed on 1 Oct 2019)

Dolby Laboratories. (2003c). *Dolby 5.1-channel music production guidelines.* Retrieved from `https://www.dolby.com/` (Accessed on 1 Oct 2019)

Dolby Laboratories. (2012). *Dolby atmos.* Retrieved from `https://www.dolby.com/us/en/brands/dolby-atmos.html` (Accessed on 1 Oct 2019)

Dolby Laboratories. (2016). *Dolby atmos® enabled speaker technology.* Retrieved from `https://www.dolby.com/us/en/technologies/dolby-atmos/dolby-atmos-enabled-speaker-technology.pdf` (Accessed on 28 Jan 2020)

Dolby Laboratories. (2018a). *Dolby atmos cinema sound.* Retrieved from `https://www.dolby.com/us/en/technologies/cinema/dolby-atmos.html` (Accessed on

29 Jan 2020)

Dolby Laboratories. (2018b). *Dolby atmos® home theater installation guidelines.* Retrieved from `https://www.dolby.com/us/en/technologies/dolby-atmos/dolby-atmos-home-theater-installation-guidelines.pdf` (Accessed on 25 Jan 2020)

Echo. (2019a). *Nic-1.* Retrieved from `https://echoaudio.com/products/streamware-nic-1` (Accessed on 24 January 2019)

Echo. (2019b). *Streamware workbench.* Retrieved from `https://echoaudio.com/products/streamware-workbench#tech-specs-tab` (Accessed on 31 January 2019)

Farina, Angelo. (2017). *Conversion from ambisonics to tbe format.* Retrieved from `http://pcfarina.eng.unipr.it/TBE-conversion.htm` (Accessed on 29 Jan 2020)

Foss, R., & Rouget, A. (2015). Immersive content creation using mobile devices and ethernet avb. In *Audio engineering society convention 139.*

Gasull Ruiz, A., Sladeczek, C., & Sporer, T. (2015). A description of an object-based audio workflow for media productions. In *Audio engineering society conference: 57th international conference: The future of audio entertainment technology–cinema, television and the internet.*

Genelec. (2019). *1029a studio monitor.* Retrieved from `https://www.genelec.com/support-technology/previous-models/1029a-studio-monitor` (Accessed on 22 Feb 2019)

Hak, C. C. J. M., van Haaren, M. A. J., Wenmaekers, R. H. C., & van Luxemburg, L. C. (2009). Measuring room acoustic parameters using a head and torso simulator instead of an omnidirectional microphone. In *Inter-noise and noise-con congress and conference proceedings* (Vol. 2009, pp. 2411–2419).

Herre, J., Hilpert, J., Kuntz, A., & Plogsties, J. (2015). Mpeg-h audio — the new standard for universal spatial/3d audio coding. *Journal of the Audio Engineering Society*, *62*(12), 821–830.

Hodges, P. (2011). *Ambisnic info — channel formats.* Retrieved from `https://ambisonic.info/ambisonics/channels.html` (Accessed on 31 Jan 2020)

Hollerweger, F. (2013). An introduction to higher order ambisonic. *Florian Hollerweger's Website.*

Holman, T. (2012). *Surround sound: up and running.* Routledge.

IEEE 1722. (2016). IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks. *Document Standard.*

IEEE 802.1AS. (2011). IEEE Standard for Local and Metropolitan Area Networks -

Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks. *Document Standard*.

IEEE 802.1BA. (2011). IEEE Standard for Local and metropolitan area networks - Audio Video Bridging (AVB) Systems. *Document Standard*.

IEEE 802.1Qat. (2010). IEEE Standard for Local and Metropolitan area networks - Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol. *Document Standard*.

IEEE P1722.1. (2013). IEEE Standard for Device Discovery, Connection Management and Control Protocol for IEEE 1722 Based Devices. *Document Standard*.

immersiveDSP. (2019). *Object-based audio made easy.* Retrieved from `https://www.immersivedsp.com/` (Accessed on 25 Sep 2019)

ITU-R. (1994). *Bs.775 : Multichannel stereophonic sound system with and without accompanying picture.* Retrieved from `https://www.itu.int/rec/R-REC-BS.775-1-199407-S/en` (Accessed on 2 Oct 2019)

ITU-R. (2013). *Bs.2076: Audio definition model.* Retrieved from `https://www.itu.int/dms_pub/itu-r/opb/rep/R-REP-BS.2266-2013-PDF-E.pdf` (Accessed on 28 Jan 2020)

ITU-R. (2019). *Bs.2127: Audio definition model renderer for advanced sound systems.* Retrieved from `https://www.itu.int/dms_pub/itu-r/rec/bs/R-REC-BS.2127-0-201906-!!!PDF-E.pdf` (Accessed on 28 Jan 2020)

JUCE. (2019). *Discover.* Retrieved from `https://www.juce.com` (Accessed on 17 December 2019)

Kaiser, F. (2011). A hybrid approach for three-dimensional sound spa-tialization. *Séminaire IEM algorithmes en acoustique et informatique musicale, Graz*.

Kostadinov, D., Reiss, J. D., & Mladenov, V. (2010). Evaluation of distance based amplitude panning for spatial audio. In *Icassp* (pp. 285–288).

Kurose, J. F., & Ross, K. W. (2013). *Computer networking: A top-down approach: International edition.* Pearson Higher Ed.

Lombardo, V., Arghinenti, A., Nunnari, F., Valle, A., Vogel, H. H., Fitch, J. P., ... Weinzierl, S. (2005). The virtual electronic poem (vep) project. In *Icmc.*

Lossius, T., Baltazar, P., & de la Hogue, T. (2009). Dbap–distance-based amplitude panning. In *Icmc.*

Ludé, P. (2014). *Your host.* Citeseer.

Malham, D. G., & Myatt, A. (1995). 3-d sound spatialization using ambisonic techniques. *Computer music journal*, *19*(4), 58–70.

Marston, D. (2013). *The audio definition model.* Retrieved from `https://www.w3.org/2013/10/tv-workshop/papers/webtv4_submission_4.pdf` (Accessed on 28 Jan

2020)

MathWorks. (2020). *Square wave from sine waves.* Retrieved from `https://www.mathworks.com/help/matlab/math/square-wave-from-sine-waves.html` (Accessed on 13 Jan 2020)

MDN. (2019a). *NAN.* Retrieved from `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN` (Accessed on 15 Feb 2019)

MDN. (2019b). *Window.* Retrieved from `https://developer.mozilla.org/en-US/docs/Web/API/Window` (Accessed on 18 Dec 2019)

MDN. (2019c). *Window: load event.* Retrieved from `https://developer.mozilla.org/en-US/docs/Web/API/Window/load_event` (Accessed on 18 Dec 2019)

MDN. (2020a). *Iife.* Retrieved from `https://developer.mozilla.org/en-US/docs/Glossary/IIFE` (Accessed on 10 Feb 2020)

MDN. (2020b). *MediaDevices.getUserMedia().* Retrieved from `https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia` (Accessed on 7 Feb 2020)

MDN. (2020c). *Promises.* Retrieved from `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise` (Accessed on 5 Feb 2020)

Microsoft. (2019). *Windows 10 release information.* Retrieved from `https://docs.microsoft.com/en-us/windows/release-information/` (Accessed on 17 Dec 2019)

miniDSP. (2019). *UMIK-1.* Retrieved from `https://www.minidsp.com/products/acoustic-measurement/umik-1` (Accessed on 13 Jan 2019)

miniDSP. (2020). *SPK-4P / PoE+ IP Loudspeaker / AVB Powered.* Retrieved from `https://www.immersivedsp.com/product/spk-4p-poe-ip-loudspeaker-avb-powered/` (Accessed on 26 Jan 2020)

Minnaar, P., Olesen, S. K., Christensen, F., & Møller, H. (2001). The importance of head movements for binaural room synthesis..

MKLabs. (2019). *Staruml 3.* Retrieved from `http://staruml.io/` (Accessed on 18 December 2019)

Møller, H. (1992). Fundamentals of binaural technology. *Applied acoustics*, *36*(3-4), 171–218.

Möller, S., & Raake, A. (2014). *Quality of experience: advanced concepts, applications and methods.* Springer.

MOTU. (2019). *Introducing the ultralite avb.* Retrieved from `http://motu.com/products/avb/ultralite-avb` (Accessed on 5 March 2019)

Nachbar, C., Zotter, F., Deleflie, E., & Sontacchi, A. (2011). Ambix-a suggested am-

bisonics format. In *Ambisonics symposium, lexington* (p. 11).

Pohlmann, K. C. (2010). *Principles of digital audio.* McGraw Hill.

Policardi, F. (2011). MLS and sine-sweep measurements. *Università di Bologna, Italia ELEKTROTEHNIŠKI VESTNIK*, *78*(3), 91–95.

Pulkki, V. (1997). Virtual sound source positioning using vector base amplitude panning. *Journal of the Audio Engineering Society*, *45*(6), 456–466.

Pulkki, V. (1999). Uniform spreading of amplitude panned virtual sources. In *Proceedings of the 1999 ieee workshop on applications of signal processing to audio and acoustics. waspaa'99 (cat. no. 99th8452)* (pp. 187–190).

Purple Panda. (2019). *Purple panda lavalier microphone kit.* Retrieved from `https://www.purplepandastore.com/products/purple-panda-lavalier-microphone-kit` (Accessed on 23 December 2019)

Roberts, B. (2019). *Inside the uk's first dolby cinema: 400 speakers, dolby atmos and a compton organ.* Retrieved from `https://www.whathifi.com/features/inside-the-uks-first-dolby-cinema-400-speakers-dolby-atmos-and-a-compton-organ` (Accessed on 25 Jan 2020)

Rumsey, F., Griesinger, D., Holman, T., Sawaguchi, M., Steinke, G., Theile, G., & Wakatuki, T. (2001). Multichannel surround sound systems and operations. *The Audio Engineering Society, Tech. Rep. AESTD1001. 0.01-05*.

Ryan, P. J. (1986). *Euclidean and non-euclidean geometry: an analytic approach.* Cambridge university press.

Samsung Electronics. (2019). *Samsung Galaxy Tab S2.* Retrieved from `https://https://www.samsung.com/global/galaxy/galaxy-tab-s2/` (Accessed on 16 Sep 2019)

Sasidharan, V. (2018). *Dts:x vs dolby atmos: Similarities and differences.* Retrieved from `https://www.gizbot.com/accessories/features/dts-x-vs-dolby-atmos-similarities-differences/articlecontent-pf88293-050238.html` (Accessed on 2 Feb 2020)

Sharp, H., Rogers, Y., & Preece, J. (2015). *Interaction design: beyond human-computer interaction.* Wiley.

Shinn-Cunningham, B. (2000). Learning reverberation: Considerations for spatial auditory displays..

SMPTE. (2018a). Immersive audio metadata. *Document Standard*.

SMPTE. (2018b). SMPTE Standard - Immersive Audio Metadata. *Document Standard*.

Stan, G.-B., Embrechts, J.-J., & Archambeau, D. (2002). Comparison of different impulse response measurement techniques. *Journal of the Audio Engineering Society*, *50*(4), 249–262.

Steinberg. (2019). *Our technologies.* Retrieved from `http://www.steinberg.net/en/`

company/technologies.html (Accessed on 22 Decemeber 2019)

Tarzan, A., Alunno, M., & Bientinesi, P. (2017). Assessment of sound spatialisation algorithms for sonic rendering with headsets. *arXiv preprint arXiv:1711.09234*.

The Node.js Foundation. (2019a). *Express.* Retrieved from https://expressjs.com/ (Accessed on 9 Dec 2019)

The Node.js Foundation. (2019b). *Node.js.* Retrieved from https://nodejs.org/en/ (Accessed on 7 Dec 2019)

Vaseghi, S. V. (2008). *Advanced digital signal processing and noise reduction.* John Wiley & Sons.

Watkinson, J. (2001). *The art of digital audio* (3rd ed.). Focal Press.

Weber, T. J. (2019). *The wave file format.* Retrieved from http://www.lightlink.com/tjweber/StripWav/WAVE.html (Accessed on 31 January 2019)

Wolter, K., Reinecke, P., & Mittermaier, A. (2011). Model-based evaluation and improvement of ptp syntonisation accuracy in packet-switched backhaul networks for mobile applications. In *Computer performance engineering* (pp. 219–234). Springer.

Appendix A

Listings

A.1 JavaScript Code for Verification of Coordinates Computation

A JavaScript program that verifies the coordinates computation functionality from each of the octants of the 3D space.

```
/*
For each octant, the program uses four locations to derive distances from an audio source.
The distances are known inputs, then the corresponding x, y and z
coordinates are computed using trigonometry.

NB: distances are given in centimetres
*/

// create a 3D point
function getPt(oct) {
    var orthogonalDistance = 1000,  // the orthgonal distance for the 3 axes - 10 metres, can be any length!

        // a random 3D point
        pt = [Math.floor(Math.random() * orthogonalDistance),
            Math.floor(Math.random() * orthogonalDistance),
            Math.floor(Math.random() * orthogonalDistance)];

    for (var i = 0; i < pt.length; i++) { // go over octant 'signs'
        if (oct[i] == '1')      // change '1' to '0' to change order of octants
            pt[i] = -pt[i];     // set coordinate to negative depending on octant
        }
    return pt;
```

```
}

// add a method 'hypot' to Math class
Math.hypot = function(a,b,c) {
    return Math.sqrt(a*a + b*b + c*c);
}

// format a float, (short-form)
function f(f) {
    return Math.floor(Math.round(f));
}

// round a number f, to n decimal places (see http://www.jacklmoore.com/notes/rounding-in-javascript/)
function round(f, n = 0) {
    return parseInt(Number( Math.round( f + 'e' + n ) + 'e-' + n ));
}

/** MAIN **/
var hd = '\nSPEAKER DISTANCES VERIFICATION (in cm)\n\
------------------------------------------------------------------------\n\
dO\t dX\t dY\tdZ\t  Derived Point\t\t  Old point\n\
------------------------------------------------------------------------';

    a = 100, // 100 centimetres, distance for each axis from origin
    octs = 8, // 8 octants in a 3d space
    iter = 5, // iterations for octants
    parityFails = 0; // track the number of new and old coordinates disparity

console.log(hd);

for (var i = 0; i < iter*octs; i++) {
    var oct = ('00' + (i.toString(2))).slice(-3),   // create a 3-bit octant (0 - 7)
        pt = getPt(oct),    // create a point within an octant

        // calculate distances from four locations
        dO = Math.hypot(pt[0],pt[1],pt[2]),
        dX = Math.hypot(pt[0]-a,pt[1],pt[2]),
        dY = Math.hypot(pt[0],pt[1]-a,pt[2]),
        dZ = Math.hypot(pt[0],pt[1],pt[2]-a),

        // calculating angles between dO and each new location - using cosine law
        x_theta = Math.acos((a*a + dO*dO - dX*dX)/(2*a*dO)),
        y_theta = Math.acos((a*a + dO*dO - dY*dY)/(2*a*dO)),
        z_theta = Math.acos((a*a + dO*dO - dZ*dZ)/(2*a*dO)),

        // compute new values, along each axis
        x1 = dO*Math.cos(x_theta),
        y1 = dO*Math.cos(y_theta),
        z1 = dO*Math.cos(z_theta),
        spacer = '\t\t';

    // compare the new point (x1, y1, z1) with the old point pt
    if ( pt[0] != round(x1)) {
        parityFails += 1;
        console.log('PARITY FAIL: %s != %s', pt[0], x1);
```

```
78      }
79      if ( pt[1] != round(y1)) {
80          parityFails += 1;
81          console.log('PARITY FAIL: %s != %s', pt[1], y1);
82      }
83      if ( pt[2] != round(z1)) {
84          parityFails += 1;
85          console.log('PARITY FAIL: %s != %s', pt[2], z1);
86      }
87      // formatting...
88      if((i+1) % octs == 0) {
89          if( (pt[0].toString().length == 4) & (pt[1].toString().length == 4) & (pt[2].↩
                toString().length == 4))
90              spacer = '\t';
91      }
92      // display derived/computed & original points
93      console.log(f(d0) + '\t' + f(dX) + '\t' + f(dY) + '\t' +
94       f(dZ) + '\t(' + f(x1) + ',' + f(y1) + ',' + f(z1) + ')' +
95         spacer + '(' + pt + ') ');
96      if((i+1) % octs == 0) {
97          console.log();
98      }
99  }
100 // display the number of failed old-new coordinate pairs if there are any
101 if (parityFails) {
102     console.log('%s parity fails', parityFails);
103 }
```

A.2 JavaScript Code for 3D Coordinates Computation from the Manual Measured Distances

This JavaScript program uses computes the coordinates from the distances from the manually measured distances in the two testing environments. The algorithm used in this program is the same algorithm used for the distances obtained dynamically when the short impules are used.

```
1  /*
2  Author  :   Motebang Lebusa
3  ID      :   614L4400
4  Date:   :   8 Feb 2019
5  Feature :   Coordinates Computation from Physical Distances
6  Project :   Speaker Position Determination
7  Group   :   Audio Networks Research
8  Comment :   Use the physically measured distances to compute coordinates
9  */
10
```

```
11 // envinronment 1 setup - Sys Dev Lecture Room
12 var config1 = { measuredDistances: [[195 ,225.5 ,183 ,192],
13     [184.5 ,217 ,191 ,181.5],
14     [284 ,307 ,260 ,282],
15     [194 ,170 ,217 ,191],
16     [285 ,269.5 ,257.5 ,282],
17     [290 ,259 ,286.5 ,288],
18     [391.5 ,417.5 ,377 ,379],
19     [362.5 ,391.5 ,363.5 ,348]],
20     orthogonalDistance:33,
21     roomDims:{length:820, height:270, width:680},
22     roomDesc:"Large Room - Sys Dev Room",
23     coords:[]},
24
25 // envinronment 2 setup - Office 120
26     config2 = { measuredDistances: [[202 ,226 ,179 ,192],
27         [112 ,140 ,140 ,114],
28         [144 ,163 ,111 ,145],
29         [153 ,132 ,130 ,155],
30         [96 ,73 ,125,98],
31         [83 ,112 ,76 ,85],
32         [168 ,142 ,177 ,152],
33         [187 ,215 ,207 ,175]],
34         orthogonalDistance:33,
35         roomDims:{length:500, height:270, width:400},
36         roomDesc:"Small Room - Office 120",
37         coords:[]},
38     environments = [config2, config1];
39
40 // compute each setup's coordinates
41 environments.forEach(config => {
42     config.coords = config.measuredDistances.map(a => {
43         return computeCoordinates(a, config.orthogonalDistance);
44     });
45 });
46
47 // display environments info: distances & coordinates
48 environments.forEach((config, i) => {
49
50     console.log("ENVIRONMENT %s: %s\nDimensions: %scm x %scm x %scm", i+1, config.↩
        roomDesc, config.roomDims.width,
51         config.roomDims.height, config.roomDims.length )
52
53     console.log("\nSpeaker Distances".toUpperCase());
54
55     config.measuredDistances.forEach((b,j) => {console.log("Speaker %s: (%s)", j+1,b)})↩
        ;
56
57     console.log("\nSpeaker Coordinates".toUpperCase());
58
59     config.coords.forEach((b,j) => {console.log("Speaker %s: (%s)", j+1,b.map(c=>round(↩
        c,1)))});
60
61     console.log("\n--------\n");
62 });
63
```

```
64
65 // compute coordinates -- extracted from the server
66      function computeCoordinates( locationDistancesArray, orthogonalDistance ) { // ↩
            array of 4 distances from locations o, & 3 axes
67
68         let _do = locationDistancesArray[0],
69             _dx = locationDistancesArray[1],
70             _dy = locationDistancesArray[2],
71             _dz = locationDistancesArray[3],
72
73             // calculating the coordinates
74             coordX = _do * Math.cos(Math.acos((orthogonalDistance * orthogonalDistance ↩
                   + _do * _do - _dx * _dx) / (2 * orthogonalDistance * _do))),
75             coordY = _do * Math.cos(Math.acos((orthogonalDistance * orthogonalDistance ↩
                   + _do * _do - _dy * _dy) / (2 * orthogonalDistance * _do))),
76             coordZ = _do * Math.cos(Math.acos((orthogonalDistance * orthogonalDistance ↩
                   + _do * _do - _dz * _dz) / (2 * orthogonalDistance * _do)));
77
78         return [ coordX, coordY, coordZ ];
79     }
80
81 // round a number f, to n decimal places (see http://www.jacklmoore.com/notes/rounding-↩
       in-javascript/)
82 function round(f, n = 0) {
83     return Number( Math.round( f + 'e' + n ) + 'e-' + n );
84 }
```

Appendix B

UI and Diagrams

This appendix chapter contains various screenshots of the UI and diagrams for various components of SDIAS.

B.1 User Interface Screenshots

The measurement process goes through various steps during the configuration process as shown in the following UI screenshots.

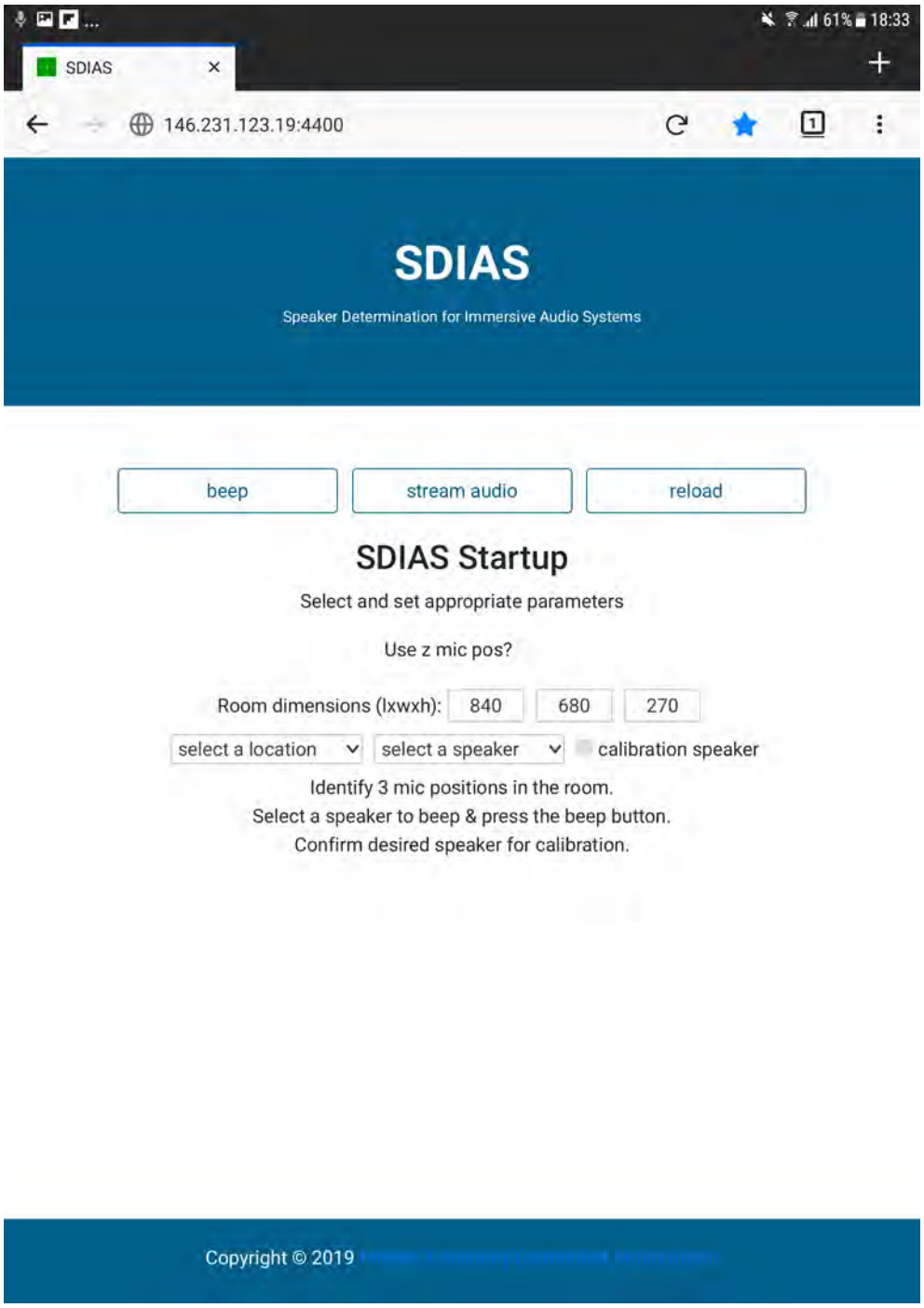


Figure B.1: Client Calibration

Figure B.2: Client Calibration - Calibration Speaker Selection

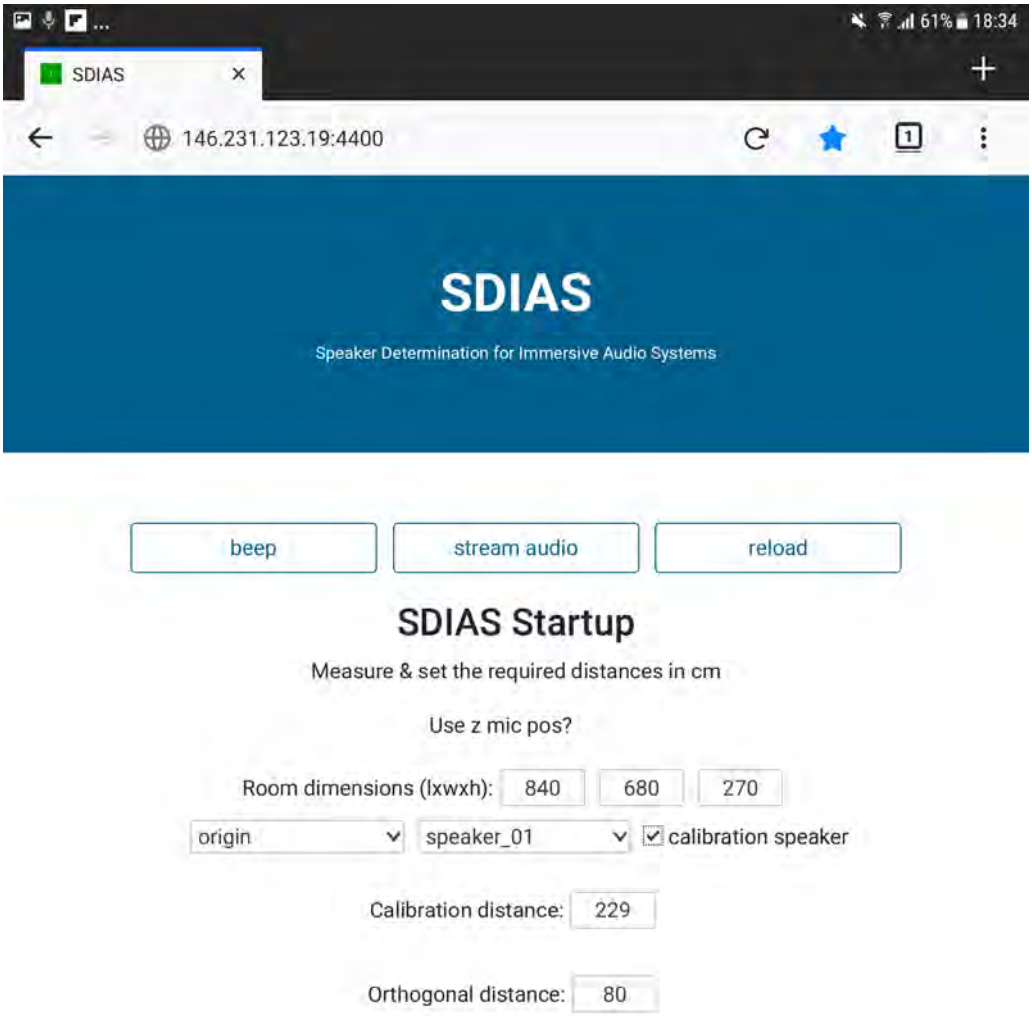


Figure B.3: Client Calibration - Calibration & Orthogonal Distances

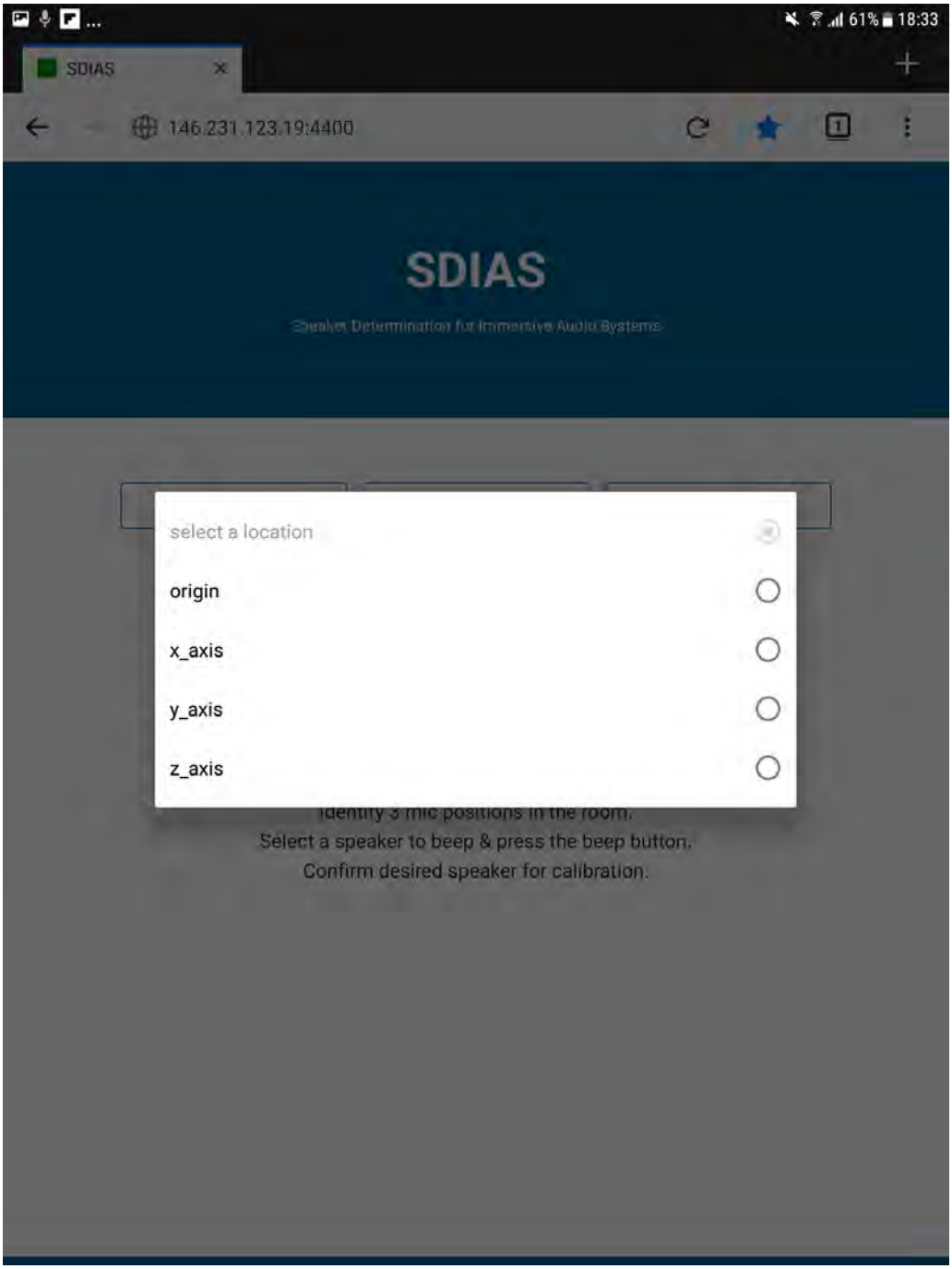


Figure B.4: Client Streaming - Location Selection

Figure B.5 is an example of a visualisation drawn from the audio data saved to the disk as plain text during each SDIAS run. This visualisation is for the same SDIAS run which is shown in Figure 3.15, which the client draws on an HTML 5 canvas element. The visualisation is confined by the width of the display device, whose pixels are significantly smaller than the available audio samples which need to be drawn.

The function which draws this visualisation, *visualiseAudioData()* shown in line 225 of Listing 5.8, therefore draws a compressed version of the recorded (and saved) data to fit the display device. For instance, with 184320 (5 locations x 8 speakers x 4608 samples for each tone) samples from the recorded audio data and a mobile device with a display width of 1536 pixels, the compression ratio used by the function becomes 120 (184320/1536), which means the function picks every 120th sample to plot. This limitation hides the visualisation features described in Section 3.4.4 and 5.3.13.

Figure B.5 shows the colour-coded impulses for each of the eight speakers at each of the five locations for the entire SDIAS run data, that is, without compression implemented by *visualiseAudioData()*. The x-axis intervals are marked at the location duration, that is, location O1 starts at sample position 0 and ends at position 36863, the next location, X, starts at position 36864 and ends at position 73727 and all the way through to O2 which starts at position 147456 and ends at position 184319. Within each location, for instance, O1, each speaker's tone is shown with its respective colour and demarcated by the minor, unmarked intervals, where each interval is 4608 samples long. The y-axis shows the dynamic range of the 32-bit floats of the audio sample data, whose nominal range is the [1,-1].

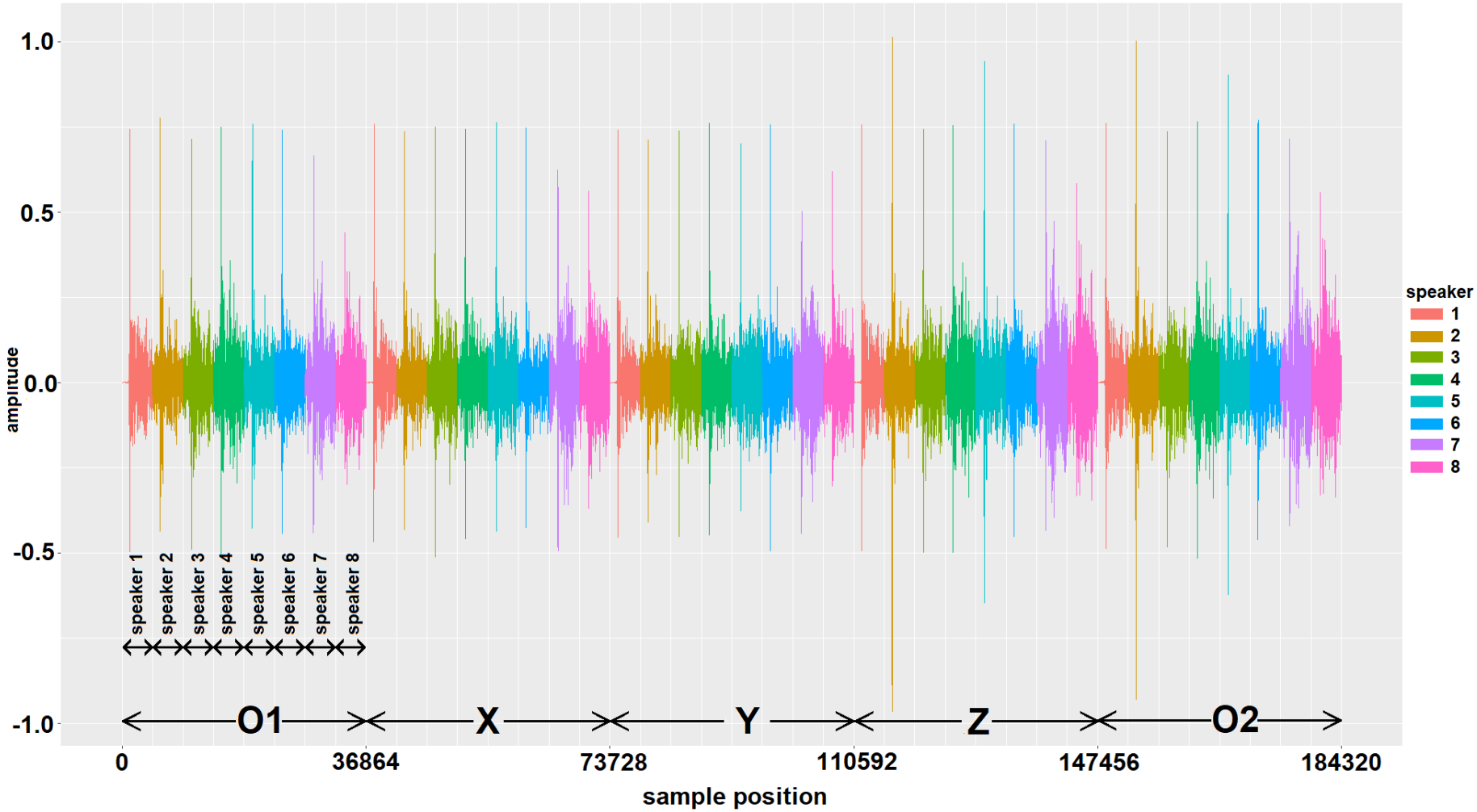


Figure B.5: Five locations, each with eight impulses

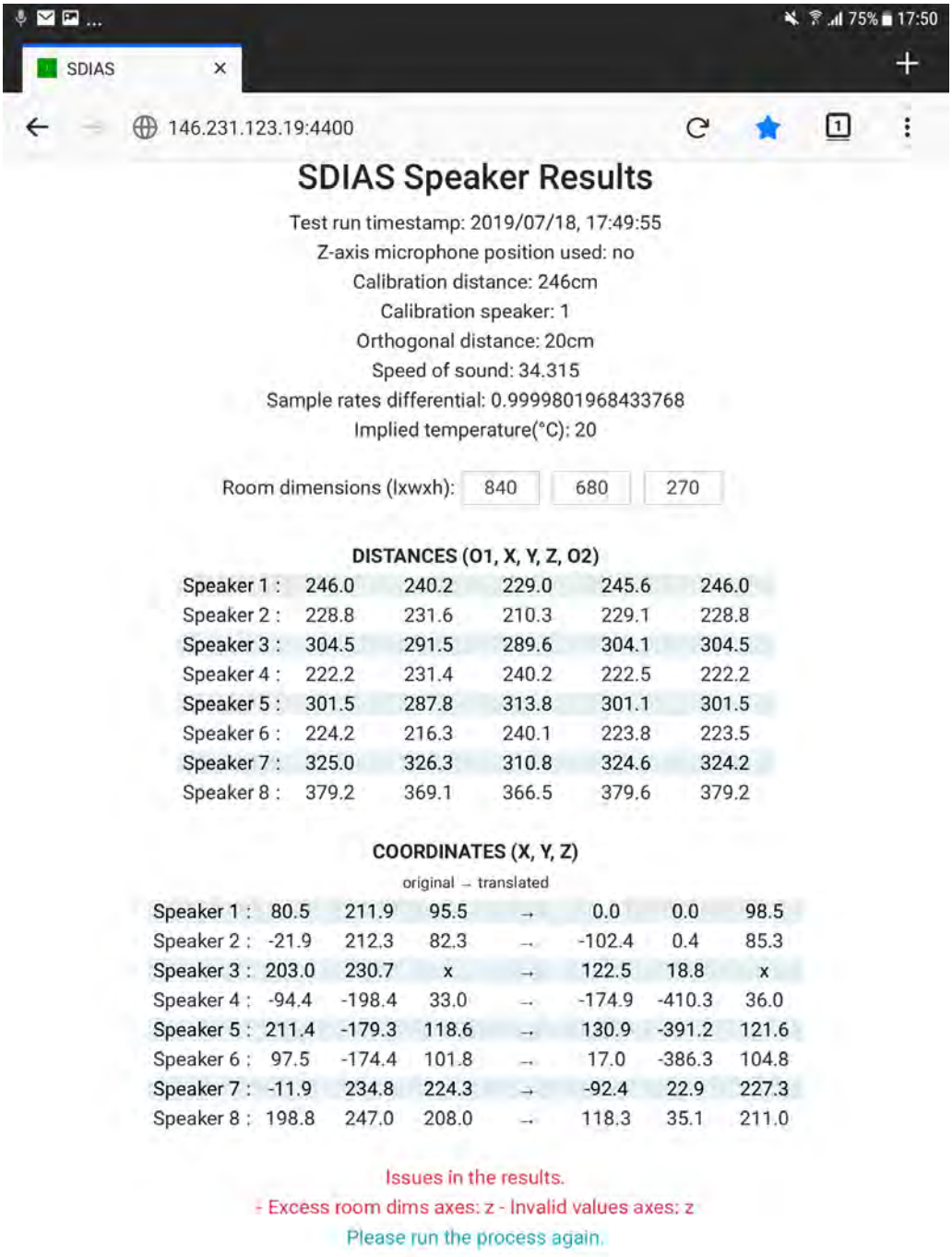


Figure B.6: Client Unsatisfactory Results

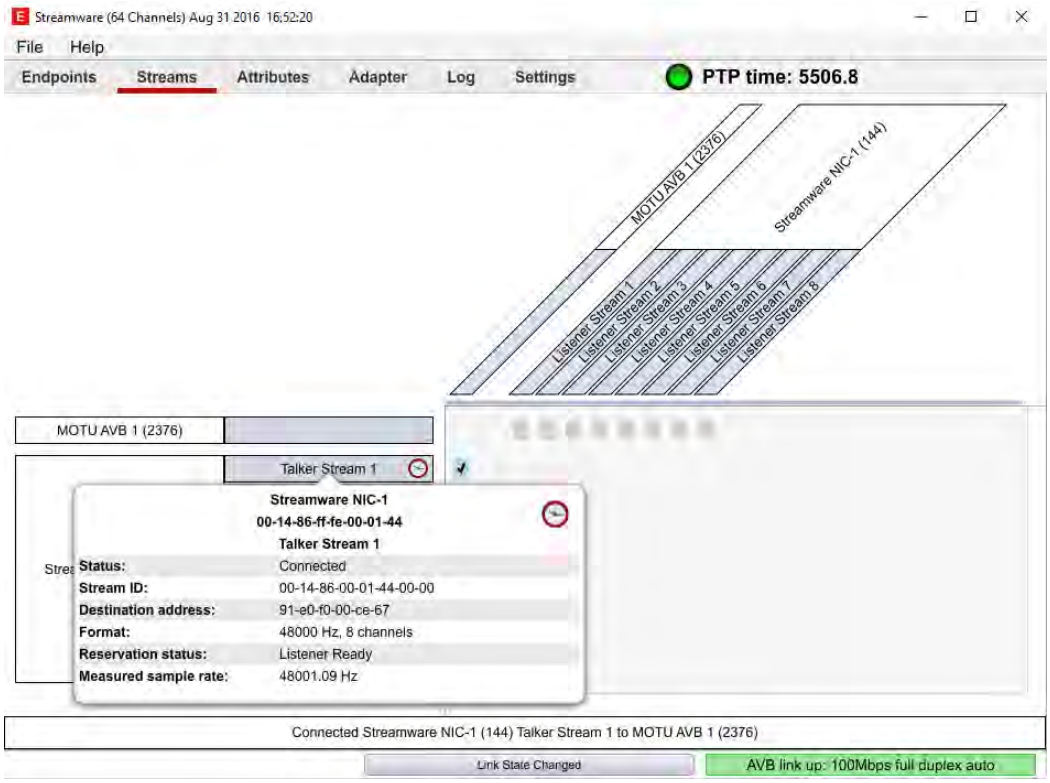


Figure B.7: Clock drift within NIC-1

Appendix C

3D Coordinate Tables

The results of the various tests from the two test rooms are presented in this appendix.

Table C.1: Small test room coordinates

Test No.	Speaker 1			Speaker 2			Speaker 3			Speaker 4			Speaker 5			Speaker 6			Speaker 7			Speaker 8		
	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
1	-145.9	123	72.7	-86.1	-66.6	15.7	-71.8	125.2	12.4	95.7	100.4	8.8	78.2	-58.7	11.7	-82.8	37.4	10.7	131.2	-10.6	99.5	-144.2	-84.2	79.8
2	-145.1	128.2	80.6	-87.6	-68.3	15.6	-70.8	125.9	12.2	102.6	110	11.8	80.1	-60.6	11.5	-82.7	37.8	12.3	132	-17	99.3	-147.4	-83.1	79.5
3	-144.6	123.9	75.8	-87.1	-67.8	10.4	-73.9	122.8	11.8	102.5	107.2	14.6	80.1	-58	11.2	-82.8	37.8	12.1	135	-13.3	98.8	-143	-83.1	82.6
4	-143.5	125.9	78.7	-86.7	-67.2	12	-73.1	126.8	13.8	107.2	112.1	20.2	79.1	-59.6	10.5	-82.9	36.8	13.3	133	-15.5	97.8	-146	-85.7	81.4
5	-143.3	128.1	84.9	-89.6	-68.4	13.3	-73	125.8	15.5	104.1	107.6	18.7	79.2	-58.1	9.6	-82.7	37.8	14.2	133.1	-13.4	99.5	-141.3	-83.2	83.4
6	-143.1	126.2	80.2	-89.4	-69.8	10.5	-72.8	124.6	11.9	110.8	112.8	21.5	81.3	-57.3	9.2	-82.9	37	12.2	136.8	-11.7	99.4	-145.6	-85.3	79.2
7	-143	127.1	81.9	-89.4	-66.2	13.9	-69.3	125.2	13.2	104.2	106.9	16.2	79.2	-58.7	10.1	-83	37.4	12.9	129.8	-17.9	93.6	-145.5	-84.3	80.7
8	-142.2	129.6	73.7	-88.8	-67.2	13.9	-75.2	123.7	10	101.5	105.4	12.8	87.9	-51.2	18.4	-82.9	38.4	11.1	133.9	-15.6	97	-140.2	-85.8	76.9
9	-142	126.7	80	-88.5	-68.6	13	-72	124.6	11.9	110.7	113	21.4	81	-58.7	11.4	-82.7	38.9	14	133.1	-14.2	95.4	-144.5	-84	78.9
10	-142	129.5	81.1	-86.3	-67.8	15.9	-69	129.8	18.7	102	108.6	15.5	83.8	-55.5	11.7	-82.8	38.3	12.5	131	-15.7	96.5	-136.1	-82	83.9
11	-141.9	126.9	81.1	-88.7	-69.2	13.5	-75	124.5	9.5	108.2	112.8	22.1	79.6	-58.9	11.8	-83	37.3	12.5	134	-10.9	96.5	-148.4	-84.2	76
12	-141.9	129.7	82.9	-88.6	-67.1	14.6	-71.9	126.9	17	111.1	112.1	20.3	79.6	-59.5	10.7	-82.6	36.9	13.4	134.1	-15.5	97.9	-139.9	-81.3	81.6
13	-140.9	128.4	76.4	-85.3	-66	15.3	-70.8	125.7	14.9	96.8	101.4	11.5	80.6	-56	13.4	-83	37.7	10.3	132	-13.5	99	-134.5	-83.7	79.4
14	-140.5	127	81.1	-87.8	-69.2	11.1	-70.9	125.1	15.7	105.7	109.6	15.6	78.5	-58.8	9.7	-83	37.3	12.6	131.9	-14.5	96.5	-143.1	-84.4	80
15	-140.2	123.8	75.4	-90.6	-68.7	10.1	-74.2	123.1	11.3	100.2	104.4	14.2	87	-52.5	12.9	-83	36	10	128.5	-21.1	91.6	-142.8	-88	74.8
16	-140.1	127	82.2	-87.5	-69.2	11.8	-70.7	125.1	16.5	106	109.6	13.2	80.7	-59.3	8.1	-82.9	37.3	13.1	132.1	-14.5	97.4	-142.8	-84.5	81
17	-139.8	128.3	81.1	-87.4	-68.2	13.5	-70.4	126	15.7	106.2	107.8	15.6	78.7	-57.9	11.8	-82.8	37.9	12.6	132.3	-13.2	99.7	-142.5	-82.9	80
18	-139.5	131	80	-87.4	-64.1	17.5	-73.3	127.4	11.5	101	109.4	17.7	81.1	-56.9	15.3	-82.9	37.2	12	132.9	-14.7	98.6	-137.7	-80.8	86.3
19	-139	130.5	76	-89.8	-66.5	15.2	-69.8	124.9	14.8	97.3	102.9	11.4	80.6	-59	9.1	-83	38.8	12.1	132.3	-14.6	95.3	-141.7	-80.5	79
20	-138.7	129.6	74.8	-89	-69.6	12.1	-72.7	128.6	10.9	100.7	105.5	10.4	78.6	-61.6	8.5	-82.9	38.4	13.3	129.5	-15.5	97.5	-140.9	-85.4	73.9
21	-138.3	128.9	79.2	-89.4	-67.7	14.7	-69.3	126.3	14.2	104.2	108.2	17.3	79.3	-57.5	10.8	-82.8	38.1	11.7	129.8	-16.1	97.8	-136.5	-82.3	78.2
22	-137.8	130.5	80.2	-86.1	-66.6	15.3	-72	124.3	11.9	104.6	109.4	18.1	81	-59	11.3	-82.7	38.7	13.9	130.1	-14.7	95.4	-140.5	-80.5	82.8
23	-137.2	129	79.2	-88.7	-64.8	17.1	-68.5	126.4	14.2	101.7	105	14	81.2	-57.4	12.9	-82.9	38.2	11.7	130.5	-16	97.8	-139.9	-77.9	81.9
24	-135.8	132.8	83.4	-90.5	-68	14.7	-70.6	126.1	14.2	106.5	111.2	20.6	82.3	-58.2	10.8	-82.9	37.9	11.7	138.8	-9.4	101.7	-142.7	-82.7	81.9
25	-135.6	132.6	82.3	-90.6	-69	16.3	-67.4	129.4	13.1	100.7	104.6	12.8	81.5	-59.1	12.1	-82.9	36.1	9.3	136.7	-10.1	104.2	-134.1	-79.8	84.8
26(*)	-139.1	149.3	76.2	-90.4	-90.4	9.7	-71.9	144	12.1	107.2	115.1	7.2	75.4	-80.6	10.6	-69.2	33.4	11.4	138.6	-30.5	94.1	-154	-102.9	82.3

Table C.2: D20 coordinates

Test No.	Speaker 1			Speaker 2			Speaker 3			Speaker 4			Speaker 5			Speaker 6			Speaker 7			Speaker 8		
	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
1	0	0	73.9	-93.5	-1.2	69.8	112.5	3.4	118.6	-118.8	-404.9	48.2	189.6	-331.2	123.1	12.3	-486.6	20.6	-112.5	124	130.9	105.3	95.9	192.2
2	0	0	85	-101.8	-9.7	94.7	112.8	11.8	107.6	-119.2	-397.6	64.3	180.1	-342.2	120	12.6	-477.7	55.3	-152.4	121.2	115.8	119.9	106	169.1
3	0	0	95.5	-110.3	-9.9	98.9	113.5	20.7	96.6	-111.4	-382	101.5	190	-344	90	13	-469.7	72.1	-137.8	131.4	118.3	107.4	115.8	168.2
4	0	0	101.7	-102.1	6.3	72.1	112.9	20.1	108.8	-111.3	-382.2	93.7	199.4	-325.1	99.8	12.4	-470.6	59.9	-153.6	129.7	110.7	119.2	114.6	169.9
5	0	0	89.8	-110.3	-9.8	94.2	113.3	11.2	110.1	-119.1	-389.5	82.4	190.1	-333.3	106.4	13.1	-468.8	83.2	-164.5	119.6	115.1	107.8	104.3	184.3
6	0	0	93.1	-102	-1.4	86.5	103.3	10.9	125.8	-119.2	-389.6	78.3	190	-333.6	106.1	12.9	-469.3	77.7	-124.2	118.8	155.6	107.2	103.4	189
7	0	0	77.3	-110.1	-9.6	87.3	104.2	12.4	106	-119.2	-389	95.7	181.4	-331.8	122.7	13.8	-477.5	64.1	-162.2	122.7	99.2	110.1	107.7	165.7
8	0	0	93	-101.9	-9.8	101.6	112.7	11	116.5	-119.2	-389.6	76.7	189.7	-323.6	121	12.5	-478.9	21.2	-111.8	118.9	161.2	119.7	103.6	180.1
9	0	0	94.2	-102	6.4	63.8	123.1	21	79.6	-111.4	-381.9	102	199.6	-324.1	99.9	12.8	-469.4	76.5	-138.6	131.9	112.9	106.7	116.4	167.1
10	0	0	73	-93.6	-1.1	65.1	122.2	13.7	85.8	-103.3	-389.3	107	199.1	-331.3	111.9	21.6	-476.5	78.7	-114.3	125.1	125.7	117.4	96.3	183.6
11**	0	0	93.8	-104	-0.5	76.8	116.4	15.7	108	-113.1	-384.6	92.4	217.3	-311.9	91.7	26.4	-462.9	98.2	-120.4	115.3	164.7	102.4	101.2	204.5
Coordinates statistics																								
min	0	0	73	-110.3	-9.9	63.8	103.3	3.4	79.6	-119.2	-404.9	48.2	180.1	-344	90	12.3	-486.6	20.6	-164.5	118.8	99.2	105.3	95.9	165.7
max	0	0	101.7	-93.5	6.4	101.6	123.1	21	125.8	-103.3	-381.9	107	199.6	-323.6	123.1	21.6	-468.8	83.2	-111.8	131.9	161.2	119.9	116.4	192.2
range	0	0	28.7	16.8	16.3	37.8	19.8	17.6	46.2	15.9	23	58.8	19.5	20.4	33.1	9.3	17.8	62.6	52.7	13.1	62	14.6	20.5	26.5
mean	0	0	87.7	-102.8	-4	83.4	113.1	13.6	105.5	-115.2	-389.6	85	190.9	-332	110.1	13.7	-474.5	60.9	-137.2	124.3	124.5	112.1	106.4	176.9
std dev	0	0	9.4	5.8	6.3	13.7	6	5.3	13.7	5.3	6.9	17.7	6.5	6.6	10.9	2.7	5.6	21.7	19.5	4.8	18.8	5.8	10	9.5

Table C.3: R20 coordinates

Test No.	Speaker 1			Speaker 2			Speaker 3			Speaker 4			Speaker 5			Speaker 6			Speaker 7			Speaker 8		
	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
1	0	0	86.9	-84.9	6.5	68.3	122.6	21.8	72.1	-119.2	-389.3	81.9	189.6	-342.7	106	22	-468.1	87.8	-99.2	133.5	126.1	118.9	119.5	146.4
2	0	0	79.7	-101.6	-9.6	93.8	113.5	12.3	98	-119.1	-389.1	92.2	190.2	-342	97.9	3.5	-477	68.5	-109.3	121.7	146.1	108.4	94	194.2
3	0	0	75.2	-101.4	-9.6	93.1	114	3	111.2	-127.2	-389	88.9	181.4	-341.6	113.1	13.8	-477.3	68.2	-107.6	109.7	170.9	110.1	94.9	187.6
4	0	0	77.7	-93.1	6.6	57.9	113.9	12.4	92.3	-127.3	-389	86.8	190.4	-341.8	92.3	13.7	-467.1	98.2	-107.9	135.3	112.5	82.5	93.7	207.4
5	0	0	69	-93.6	-8.9	85.7	122.4	13.9	79.3	-119.2	-397	77.6	189.4	-350.7	106.3	21.8	-476.1	83.3	-99.9	125.1	130.6	104.4	97.1	188.7
6	0	0	87.1	-93.6	-1.3	82.3	122.4	21.8	74.5	-119.1	-389.3	80.7	199.2	-333	101.8	12.1	-468.1	92.9	-99.9	120.8	156.2	104.4	92.3	206.5
7	0	0	77	-101.4	-9.6	94.9	114.1	12.4	90.7	-127.2	-389	87.9	190.5	-341.8	91.2	4	-476.8	70	-120.9	109.3	166.5	82.9	93.9	206.4
8	0	0	83.8	-93.6	-1.2	79.1	122.5	22.2	68.3	-119.5	-389.8	84.4	189.5	-342.3	109.9	21.9	-467.6	93.5	-99.7	121.7	151.8	104.6	93.3	202.8
9	0	0	81.1	-93.4	-1.2	80.4	123.1	12.8	87.6	-111.4	-381.3	111.7	189.9	-342	103.4	12.8	-467.3	98.5	-97.6	122.2	152.1	93.1	93.9	206.1
10	0	0	78.2	-93.4	-1.2	75.9	122.9	13.1	86.1	-111.4	-381.2	112.7	189.8	-341.7	108	12.6	-466.9	103.2	-98.3	122.9	146.6	106	94.7	195.6
Coordinates statistics																								
min	0	0	69	-101.6	-9.6	57.9	113.5	3	68.3	-127.3	-397	77.6	181.4	-350.7	91.2	3.5	-477.3	68.2	-120.9	109.3	112.5	82.5	92.3	146.4
max	0	0	87.1	-84.9	6.6	94.9	123.1	22.2	111.2	-111.4	-381.2	112.7	199.2	-333	113.1	22	-466.9	103.2	-97.6	135.3	170.9	118.9	119.5	207.4
range	0	0	18.1	16.7	16.2	37	9.6	19.2	42.9	15.9	15.8	35.1	17.8	17.7	21.9	18.5	10.4	35	23.3	26	58.4	36.4	27.2	61
mean	0	0	79.6	-95	-3	81.1	119.1	14.6	86	-120.1	-388.4	90.5	190	-342	103	13.8	-471.2	86.4	-104	122.2	145.9	101.5	96.7	194.2
std dev	0	0	5.2	4.9	6	11.2	4.3	5.6	12.4	5.5	4.3	11.6	4	4	6.9	6.4	4.6	12.6	7	8	17.2	11.2	7.7	17.5

Table C.4: D40 coordinates

Test No.	Speaker 1			Speaker 2			Speaker 3			Speaker 4			Speaker 5			Speaker 6			Speaker 7			Speaker 8		
	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
1	0	0	81.3	-96.7	-7.4	93.3	114.9	19.4	80.7	-117.9	-387.3	89.3	193.4	-343.8	84.6	15.1	-467.2	91.3	-114.9	79.8	216.6	102.5	101.2	185.6
2	0	0	78.6	-96.9	-7.3	88.8	114.7	15.5	91.3	-117.9	-386.9	90.6	193.4	-343.1	91.9	14.8	-471.8	82.3	-109.6	87.3	207.1	101.4	102.3	183.7
3	0	0	82.1	-96.8	-3.9	84.7	114.8	19.4	83.3	-117.9	-387.3	87.8	193.4	-343.8	86.8	15	-472.7	74.2	-115.6	85.6	208.1	108.5	107.8	172.1
4	0	0	83.2	-93	-3.3	86	119.2	19.9	82.3	-114.3	-383.6	96.8	198	-339.2	91.9	19.1	-468	90.4	-111.2	86.1	209.4	113.3	108.3	171.8
5	0	0	77.2	-96.7	-7.2	89.3	115	15.6	87.3	-117.8	-386.7	93.1	198.2	-338.2	85.9	15.2	-471.5	83.2	-115	81.1	212.6	109.1	103.2	176.4
6	0	0	77.8	-100.6	-7.3	91.6	115.6	15.4	81.6	-121.8	-387	91.5	189.5	-343.3	85.9	11.3	-472	78.7	-118.7	86.4	206.5	105.4	102.5	178.6
7	0	0	89.3	-92.5	-4	94.6	119.3	18.5	86.8	-118	-383.8	88.4	193.4	-340.3	89.1	14.9	-469.1	77	-115.9	83.4	214.9	108.2	99.2	192.5
8	0	0	78.6	-96.7	-7.3	91	115	15.4	88.6	-117.9	-391.3	83.3	194	-338.5	94	10.5	-471.9	82.3	-108.3	87.1	209	109.3	102.6	177.9
9	0	0	85.4	-96.5	-4	91.3	115.2	14.5	94.9	-121.9	-387.8	82	194.1	-340	83.3	15.6	-463.6	97.1	-120.3	78.4	219.5	103.9	94.1	199.3
10	0	0	76.6	-96.6	-7.3	90.5	115.1	15.6	84.5	-122	-391.1	81.6	194.1	-343.3	85.3	10.7	-471.6	84.7	-107.6	87.5	208.2	103.2	90.3	199.2
**	0	0	93.8	-104	-0.5	76.8	116.4	15.7	108	-113.1	-384.6	92.4	217.3	-311.9	91.7	26.4	-462.9	98.2	-120.4	115.3	164.7	102.4	101.2	204.5
Coordinates statistics																								
min	0	0	76.6	-100.6	-7.4	84.7	114.7	14.5	80.7	-122	-391.3	81.6	189.5	-343.8	83.3	10.5	-472.7	74.2	-120.3	78.4	206.5	101.4	90.3	171.8
max	0	0	89.3	-92.5	-3.3	94.6	119.3	19.9	94.9	-114.3	-383.6	96.8	198.2	-338.2	94	19.1	-463.6	97.1	-107.6	87.5	219.5	113.3	108.3	199.3
range	0	0	12.7	8.1	4.1	9.9	4.6	5.4	14.2	7.7	7.7	15.2	8.7	5.6	10.7	8.6	9.1	22.9	12.7	9.1	13	11.9	18	27.5
mean	0	0	81	-96.3	-5.9	90.1	115.9	16.9	86.1	-118.7	-387.3	88.4	194.2	-341.4	87.9	14.2	-469.9	84.1	-113.7	84.3	211.2	106.5	101.2	183.7
std dev	0	0	3.8	2.1	1.7	2.9	1.7	2	4.3	2.3	2.4	4.7	2.3	2.2	3.4	2.2	2.8	6.7	4.1	3.2	4.2	3.6	5.3	9.8

Table C.5: R40 coordinates

Test No.	Speaker 1			Speaker 2			Speaker 3			Speaker 4			Speaker 5			Speaker 6			Speaker 7			Speaker 8		
	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
1	0	0	79.5	-103.8	-9.1	84.8	119	23.6	60.8	-121.3	-391.8	74.4	196.1	-347.4	77.9	10.7	-478.4	50.7	-118.5	114	152.7	98.8	87.7	208.4
2	0	0	76.6	-99.6	-8.4	86.8	124.1	29		-117.5	-387.6	92.1	200.9	-336.8	84.2	10.7	-472.9	79.7	-111.6	109.4	165.9	105.9	95.5	191.5
3	0	0	79.6	-99.7	-8.4	87.3	123.8	24.3	52.3	-109.7	-384.5	104.4	196.6	-342.5	90.4	15.3	-473.4	73.5	-112.8	120.8	141.6	104.8	94.8	196.1
4	0	0	84.2	-95.3	-0.7	80.4	119.4	28.2	51.2	-117.8	-384.5	93.7	196.7	-343.4	81.2	15.7	-469.7	80.6	-118.4	119.4	147.1	99.3	93.1	204.1
5	0	0	80.3	-99.6	-8.4	90	124.1	24.2	49.7	-117.5	-387.9	88.5	196.7	-342.7	85.1	10.8	-479.2	47.9	-118.6	114.5	154.4	105.9	94.4	195.7
6	0	0	81.2	-99.7	-4.9	81.2	118.8	23.5	65.1	-117.5	-388	87	200.8	-337.6	83.2	15.4	-473.9	68.7	-112.2	120.3	145.4	105.3	100.5	186.9
7	0	0	85.5	-99.9	-0.8	76.7	129.1	33.1		-121.7	-388.6	77.4	201.5	-333.7	87.1	15.5	-470	78.7	-119	119	148	112.3	106	176.8
8	0	0	81.8	-99.7	-8.5	89.8	123.9	24	56.1	-117.5	-392.5	76.3	196.6	-342.9	87.5	10.5	-473.9	70.6	-112.5	120.2	145.6	105.1	94.1	198.2
9	0	0	78.5	-99.9	-8.4	85.1	123.6	24.5	52.3	-121.4	-396	61.3	200.8	-342.2	83.1	15.1	-478.7	54.8	-120.1	108.5	160.5	104.2	95.3	195.6
10	0	0	85.2	-93.1	-1.2	81.8	122.1	21.8	71.9	-103.1	-380.3	108.2	188.8	-341.3	101.7	21.9	-466.3	81.6	-125.8	94.3	188.7	104.9	105.8	183.1
Coordinates statistics																								
min	0	0	76.6	-103.8	-9.1	76.7	118.8	21.8	49.7	-121.7	-396	61.3	188.8	-347.4	77.9	10.5	-479.2	47.9	-125.8	94.3	141.6	98.8	87.7	176.8
max	0	0	85.5	-93.1	-0.7	90	129.1	33.1	71.9	-103.1	-380.3	108.2	201.5	-333.7	101.7	21.9	-466.3	81.6	-111.6	120.8	188.7	112.3	106	208.4
range	0	0	8.9	10.7	8.4	13.3	10.3	11.3	22.2	18.6	15.7	46.9	12.7	13.7	23.8	11.4	12.9	33.7	14.2	26.5	47.1	13.5	18.3	31.6
mean	0	0	81.2	-99	-5.9	84.4	122.8	25.6	57.3	-116.5	-388.2	86.3	197.6	-341	86.1	14.2	-473.6	68.7	-117	114	155	104.7	96.7	193.6
std dev	0	0	2.8	2.7	3.4	4.1	3	3.2	7.4	5.5	4.3	13.6	3.6	3.7	6.1	3.4	4	12.3	4.3	7.9	13.3	3.5	5.4	9

Table C.6: D80 coordinates

Test No.	Speaker 1			Speaker 2			Speaker 3			Speaker 4			Speaker 5			Speaker 6			Speaker 7			Speaker 8		
	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
1	0	0	86.6	-96.6	-1.2	82.7	117.7	21	81.5	-122.5	-389.7	73	197.3	-340.1	84.7	16.9	-464.2	93.7	-117.3			108.7	107.7	175.5
2	0	0	84.8	-96.5	-2.7	84.9	117.7	19.3	83.8	-120.1	-394.3	66.5	194.8	-341.8	86.4	12.1	-469	81.7	-117.1			108.9	108.4	172.6
3	0	0	80.3	-101	-4.7	80.8	117.7	20	75	-122.4	-392.9	72.7	192.8	-343	89.4	14.6	-467.6	91	-116.4			106.1	107.3	173
4	0	0	82.8	-96.3	-4.8	86.2	119.9	19.6	76.7	-122.6	-393.7	67	194.7	-341	88.9	16.9	-465.6	93.9	-113.6	70.5	228.3	108.7	109.2	169.6
5	0	0	85.2	-94.3	-2.7	85.7	119.8	21.2	76.6	-120.3	-391.8	71.2	196.8	-341.9	84.5	16.7	-466.6	89.6	-114.4	72.5	225.9	111.5	111.8	166.6
6	0	0	86.2	-94.6	-2.7	85.1	119.7	19.2	85.8	-118.5	-390	77.5	199.3	-342.4	82.1	16.5	-466.8	88.8	-115.4			114.1	111.6	166.6
7	0	0	83.8	-96.8	-2.7	81.9	119.8	19.5	80.5	-118	-393.9	70.2	196.8	-341.3	86.7	19.1	-465.9	92.2	-114.6	61.7	238	108	109	172.3
8	0	0	87.9	-96.9	-2.8	85.6	122.3	21.3	78.6	-120.8	-393.2	65.2	197.4	-340.4	87.7	19	-465.4	91.7	-115	71.5	227.9	111.1	110.8	171.5
9	0	0	83.1	-98.6	-3.2	81	117.4	17	85.8	-122.4	-391.2	73.4	194.8	-343.9	83.5	17	-468.8	83.9	-116.9			109	106.3	175.2
10	0	0	86	-96.7	-2.7	85.2	119.8	19.1	82.8	-120.7	-392.7	70.3	197.2	-339.8	86.6	14.3	-467	88.1	-117.7			111.8	111.5	167.4
**	0	0	93.8	-104	-0.5	76.8	116.4	15.7	108	-113.1	-384.6	92.4	217.3	-311.9	91.7	26.4	-462.9	98.2	-120.4	115.3	164.7	102.4	101.2	204.5
Coordinates statistics																								
min	0	0	80.3	-101	-4.8	80.8	117.4	17	75	-122.6	-394.3	65.2	192.8	-343.9	82.1	12.1	-469	81.7	-117.7	61.7	225.9	106.1	106.3	166.6
max	0	0	87.9	-94.3	-1.2	86.2	122.3	21.3	85.8	-118	-389.7	77.5	199.3	-339.8	89.4	19.1	-464.2	93.9	-113.6	72.5	238	114.1	111.8	175.5
range	0	0	7.6	6.7	3.6	5.4	4.9	4.3	10.8	4.6	4.6	12.3	6.5	4.1	7.3	7	4.8	12.2	4.1	10.8	12.1	8	5.5	8.9
mean	0	0	84.7	-96.8	-3	83.9	119.2	19.7	80.7	-120.8	-392.3	70.7	196.2	-341.6	86.1	16.3	-466.7	89.5	-115.8	69.1	230	109.8	109.4	171
std dev	0	0	2.1	1.8	1	2	1.5	1.2	3.7	1.6	1.5	3.5	1.8	1.2	2.2	2	1.4	3.8	1.3	4.3	4.7	2.2	1.9	3.2

Table C.7: R80 coordinates

Test No.	Speaker 1			Speaker 2			Speaker 3			Speaker 4			Speaker 5			Speaker 6			Speaker 7			Speaker 8		
	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z	x	y	z
1	0	0	83.7	-96.6	-2.7	83.4	117.7	19.4	82.5	-122.9	-389.5	78	197.3	-338.9	88.1	16.9	-466	92.1	-117.2	-211.7		105.6	100.7	189.3
2	0	0	85.9	-96.8	-2.7	84.4	117.5	19.1	87.3	-118.4	-390.1	79.1	194.9	-344.9	84.1	16.7	-470	78.1	-114.7	-210.9		108.2	99.9	190.8
3	0	0	88.1	-96.6	-1.3	84.3	119.9	22.7	74.2	-120.6	-388.3	79.2	195.4	-337.9	91.9	14.4	-465	91	-117.3	-209.9		108.8	101.7	188.4
4	0	0	85	-96.6	-1.3	80.8	115.1	18.7	87.9	-122.5	-386.7	81.3	196.7	-341.9	82.3	16.9	-469.8	79.4	-117.4	-211.2		102	99.6	193
5	0	0	87.5	-96.7	-1.3	83.1	120.1	23.4	74.1	-123	-385.6	82.7	197.3	-343.3	80.1	16.8	-465.4	92.1	-117.9	-210.2		105.3	102	190.1
6	0	0	85	-96.7	-2.7	84.4	120.1	21.8	75.5	-121.1	-387.9	84.2	197.2	-342.3	83	16.8	-469.7	79.8	-121.1	-211.2		105.3	100.2	191.3
7	0	0	84.8	-98.6	-3.3	83.3	117.7	19.2	83.1	-120.5	-387.3	85.2	194.7	-341.9	84.8	14.6	-466.6	89.4	-120.3	-211.1		102.6	96.7	196.7
8	0	0	85.4	-96.9	-2.7	82.8	119.7	19.3	83.7	-121.3	-385.3	87.6	199.3	-342.2	81.6	16.5	-469.6	80.8	-118.6	-211.2		107.8	100.3	190.5
9	0	0	85.5	-98.7	-3.3	82.9	119.9	19.2	81.5	-120.6	-384.9	88.6	194.8	-342	86.4	16.8	-466.8	87.6	-117.4	-210.9		105.3	96.5	196
10	0	0	83.3	-96.3	-3.2	82.7	119.9	19.6	77.7	-122.9	-384.2	89.3	194.9	-344	84.7	14.5	-471.3	75	-120.6	-211.8		105.4	97.5	193.1
Coordinates statistics																								
min	0	0	83.3	-98.7	-3.3	80.8	115.1	18.7	74.1	-123	-390.1	78	194.7	-344.9	80.1	14.4	-471.3	75	-121.1	-211.8		102	96.5	188.4
max	0	0	88.1	-96.3	-1.3	84.4	120.1	23.4	87.9	-118.4	-384.2	89.3	199.3	-337.9	91.9	16.9	-465	92.1	-114.7	-209.9		108.8	102	196.7
range	0	0	4.8	2.4	2	3.6	5	4.7	13.8	4.6	5.8	11.3	4.6	7	11.8	2.5	6.3	17.1	6.4	1.9		6.8	5.5	8.3
mean	0	0	85.4	-97.1	-2.5	83.2	118.8	20.2	80.8	-121.4	-387	83.5	196.3	-341.9	84.7	16.1	-468	84.5	-118.3	-211		105.6	99.5	191.9
std dev	0	0	1.4	0.8	0.8	1	1.6	1.6	4.9	1.4	1.9	3.9	1.5	2	3.3	1	2.2	6.2	1.8	0.6		2.1	1.9	2.6

Appendix D

Miscellaneous Information

D.1 The Node.js Package File

Node.js uses a JSON file called *package.json*. This file is used to simplify management and sharing of packages. The file stores configuration information related to the application, such as application name, version, author, and dependency packages (their names and versions) of packages included in the application. The *package.json* file is created by running the *'npm init'* command on the command prompt. The command then initiates the process, which then requires the developer to specify the basic required fields, as key-value pairs as the JSON format. Additional fields such as scripts, dependency and development dependency packages can then be added after the file creation. If dependency packages are specified in the file, they need to be downloaded (or installed) prior to running the server. Running the command *'npm install'* performs the installation. Listing D.1 shows the contents of the *package.json* file for the SDIAS server.

Listing D.1: The *package.json* file contents

```
1  {
2    "name": "sdias",
3    "version": "2.0.0",
4    "description": "this app determines speaker configuration for a 3d audio system",
5    "keywords": [
6      "audio, speaker configuration, speaker positions, immersive audio, web recording, ↩
           web audio api"
7    ],
8    "private": true,
9    "scripts": {
10     "test": "set NODE_ENV=test&& nodemon -harmony sdias_server.js",
```

```
11      "dev": "set NODE_ENV=development&& nodemon -harmony sdias_server.js",
12      "start": "set NODE_ENV=production&& nodemon -harmony sdias_server.js"
13    },
14    "license": "MIT",
15    "author": "Motebang Lebusa",
16    "dependencies": {
17      "debug": "^4.1.1",
18      "express": "^4.17.1",
19      "screenshot-desktop": "^1.6.0",
20      "socket.io": "^2.2.0"
21    }
22  }
```

D.2 Running a Node.js Server in Various Modes

There are several ways of running Node.js applications. These include running through the standard Windows or Node.js command prompt, or any command line interface. From the command line program, several options can be used to launch a Node.js application. These are the Node.js's *node*, *npm*, or any other third-party tools which run Node.js applications such as *nodemon*[1] during development. A Node.js server application can be executed in various modes. Some of these modes include the test mode, production mode and development.

To run the application in *production* or *test* mode, an environment variable, *NODE_EVN*, must be set to *production* or *test* before the command above. However, and for convenience, the scripts can be defined in the *package.json* file to simplify the chaining of commands and switches required to run applications in different environments. As such, Node.js applications can be run easily in pre-defined modes as defined in the scripts definition (Lines 9-13 in Listing D.1). For instance, the SDIAS server can be launched in *development* mode by simply running the *dev* script defined in Line 11 in Listing D.1 with the *npm*'s *run* command as follows:

```
> npm run dev
```

Running in development mode means Node.js will not use packages meant for development. The scripts specify a utility called *nodemon* as a choice for running the Node.js server. The *nodemon* utility watches for file changes and restarts the servers whenever

[1] https://nodemon.io/

changes are detected during the lifetime of the server application. Furthermore, the scripts pass the *harmony* option which allows use of new experimental ES6 features as they are used in the server source code. The *development* mode is mostly used for implementation. In this mode, extensive debugging tools is used. For instance, verbose logging is used to inspect various application states and variable values throughout the application. Alternatively, the *start* script allows the SDIAS server to run in *production* mode but excluding behaviour from non-*production* code, such as extensive logging.

D.3 The JUCE Audio Application Project Creation

An audio engineer must create an audio application project to implement the desired audio functionality. The JUCE framework provides a management tool for cross-platform audio projects. This is called the *Projucer*. It manages enables the engineer to make code modifications, manage module and exports to various target platforms (JUCE, 2019). The creation of an audio application project using *Projucer* creates the necessary files. Some of these include the *Main.cpp*, *MainComponent.cpp* and *MainComponent.h* files. The *Main.cpp* C++ file contains the source which launches an audio application. Nothing needs to be changed within this file. The changes which implement an audio application functionality are done in the *MainComponent.cpp* and *MainComponent.h* files. The C++ header and implementation files implement standard C++ software engineering patterns - they separate the program elements declarations from their use.

SDIAS used the *MainComponent.cpp* and *MainComponent.h* files to implement its functionality. The application variables declared in the header file include:

- *myTone*: this is a multichannel buffer is used for describing the sine wave audio data used for streaming. This is declared as a pointer variable of type *AudioSampleBuffer*, and
- *currentAudioSetup*: this is a set of properties for the current audio setup,

The *MainComponent.cpp* file contains the implementation of the tones server. Some key methods are described in the thesis text. The file contains other methods, such the destructor - *~MainComponent()*, *releaseResources()*, *paint()* and *resized()*, which implement the MainComponent class.