# An investigation into the use of intuitive control interfaces and distributed processing for enhanced three dimensional sound localization

Submitted in fulfilment of the requirements of the degree

MASTER OF SCIENCE

Rhodes University - Grahamstown

M. L. Hedges

2015

# Abstract

This thesis investigates the feasibility of using gestures as a means of control for localizing three dimesional (3D) sound sources in a distributed immersive audio system.

A prototype system was implemented and tested which uses state of the art technology to achieve the stated goals. A Windows Kinect is used for gesture recognition which translates human gestures into control messages by the prototype system, which in turn performs actions based on the recognized gestures. The term *distributed* in the context of this system refers to the audio processing capacity. The prototype system partitions and allocates the processing load between a number of endpoints. The reallocated processing load consists of the mixing of audio samples according to a specification. The endpoints used in this research are XMOS AVB endpoints. The firmware on these endpoints were modified to include the audio mixing capability which was controlled by a state of the art audio distribution networking standard, Ethernet AVB. The hardware used for the implementation of the prototype system is relatively cost efficient in comparison to professional audio hardware, and is also commercially available for end users.

the successful implementation and results from user testing of the prototype system demonstrates how it is a feasible option for recording the localization of a sound source. The ability to partition the processing provides a modular approach to building immersive sound systems. This removes the constraint of a centralized mixing console with a predetermined speaker configuration.

# Acknowledgments

Firstly I would like to acknowledge my family for all the love and support they have given me throughout the course of my studies. I am confident in saying that I would never have been where I am today if it wasn't for the motivation that they have given me throughout the years of my life.

I would like to thank my fellow postgraduates in the Audio Engineering research division at Rhodes University for making this journey such an enjoyable one. The support and help you have given me throughout this project has been remarkable.

I would like to thank the National Research Foundation (NRF) for funding my masters studies, and the Telkom Centre of Excellence for Distributed Multimedia for the funding they provide to Rhodes University's Computer Science department.

I would like to thank Mr Tom Ammermann of New Audio Technology who was never hesitant to provide help towards this project.

Lastly I wish to thank Professor Richard Foss for supervising both my Honours and Masters research. The support and dedication that Richard has given me throughout this research is immeasurable. The knowledge that Richard has imparted to me within and beyond the realm of computer science will be carried with me forever. I firmly believe that Richard's wisdom and values make him a role model to all of the computer science students at Rhodes University. Thank you Richard.

# Table of Contents

# List of Figures

# Chapter 1

# Introduction

Audio distribution systems are used in a wide variety of venues for both commercial and personal use. Audio distribution in venues such as cinemas, home theaters, and live music venues is more commonly referred to as *surround sound.* Surround sound aims to provide sound all around the user at ear level, enhancing their overall listening experience. This is achieved by localizing and moving sounds within a multitrack audio production to different areas of a listening environment. Standard surround sound has led to the concept of *immersive* surround sound. Immersive sound contains speakers at height levels that are higher and lower than the height of the user's ear. This enables a system to envelop the user in a listening environment with a much improved listening experience.

Conventional surround sound content production systems use a preset speaker configuration such as the Dolby 7.1 surround [1] or the Auro3D 9.1 immersive high and low [4], and have a central mixing matrix. This will mix the audio and distribute analog audio signals to speakers over fixed physical cabling. The mixing matrix receives the input audio from a live source such as a microphone preamplifier or a storage medium such as a DVD.

The use of network technology as a means of audio transport is becoming increasingly popular due to the advantages it provides over conventional systems. These advantages include easy configuration, convenient cabling, and digital patch bays where the audio signal is not degraded [7]. The use of digital networks as opposed to physical cabling has allowed intelligent routing of audio streams as well as providing remote control of the audio devices. Digital audio must be converted into an analog signal before it can be played. By using a digital approach to distributed audio, each audio device endpoint must convert the digital signals to an analog signal that is ready for output.

Sound panning in the context of immersive sound is the process of adjusting the level of sound at multiple endpoints such that the sound appears to be emanating from a particular location, or appears to be moving. Sound panning is done in the film and music industry, where a sound is moved between a number of speakers. The locations for a moving sound source can be encoded in a variety of ways, and then decoded by an audio decoder before it is distributed. The encoding method that is used must be compatible with the decoder to achieve sound localization. There are a number of different methods that have been used to localize sound sources in professional systems. Examples of these are Higher Order Ambisonics [8], Binaural Delay [9], Vector Based Amplitude Panning [10], and Distance Based Amplitude Panning [11] [12]. These four methods are frequently used to localize sound in professional systems.
The panning of an audio source is often done as a post-recording effect. This typically requires an audio engineer to pan the audio using instruments such as a *Panning Potentiometer* (pan pot) and encode the result so that it may be reproduced at a later stage. The digital approach to panning uses software as a post-recording tool to localize the audio into areas of the listening space. This provides a much simpler way than using conventional panning instruments such as the pan pot. Panning a sound source in three dimensions (3D) is slightly more complicated, as it requires movement in two different planes. This is required because of the height factor that is introduced with 3D immersive audio. With the increase in the use of immersive audio, there are commercially available systems that have been released which provide 3D localization. An example of one of these systems is the Spatial Audio Producer [13].

## 1.1 Research Question

This research investigates the requirements and implementation of a gesture controlled immersive sound system to overcome the limitations of currently available systems. The system uses *intelligent endpoints* in a distributed audio configuration. Intelligent endpoints aim to extend the functionality of each endpoint by enabling them to perform additional audio processing and Digital Signal Processing (DSP). By doing this, a significant portion of the processing would be reallocated from the workstation to the endpoints. This removes the limitation of a preset speaker configuration and provides the capability of adding endpoints incrementally. The configuration of this system will be simplified even further by utilizing the Power over Ethernet (PoE) capability [14]. This would allow complete configuration and control via Ethernet cabling.

This research also investigates the use of human gestures and movements as a method of localizing a sound source. This will be done by utilizing the features of a Windows Kinect "hands free" controller [15]. This aims to enable dynamic sound source localization in a 3D environment by representing a user's hand as a sound source. The use of dynamic panning for 3D movements eliminates the need of having to locate a sound in two different planes as 3D localization can be performed with a single movement of a user's hand. To achieve this, the location of the user's hand and thus the location of a sound source has to be translated into a series of mix levels applied at each endpoint.

## 1.2 Research Objectives

The research described in this thesis had the following goals:

- To investigate various speaker configurations that are used for current surround sound and immersive sound systems, and from this to determine a suitable configuration for use in testing the prototype system.

- To determine the effectiveness of utilizing a distributed processing system for the implementation of immersive sound control. An implementation goal was to use low-cost hardware that is easily available and doesn't require any specialized configuration, together with open standard for control.

- To incorporate and evaluate a form of device-free control to be included as part of the implementation. The intention was to control the system via intuitive user interfaces using human gestures. The localization of the sound sources was to be performed via the detection of the user's 3D hand position.

- To acquire both qualitative and quantitative feedback from user testing. This required a sample of users to test the system implemented for this research and also to test a commercially available immersive sound system.

The system implemented for this research will be referred to as the *KinectSound system* throughout the thesis.

## 1.3 Thesis Layout

The layout of the chapters contained in this thesis are described below.

- **Chapter 2** - Investigates various surround and immersive sound speaker configurations and determines a configuration that is optimal to implement and test the KinectSound system. This chapter also investigates methods of encoding audio, and four different techniques used to localize audio in a 3D space.

- **Chapter 3** - Defines usability metrics and the determining factors of a *usable* system. Included in this chapter is a review of existing sound panning and localization software, as well as software that does not use regular workstation control devices.

- **Chapter 4** - Gives an overview of technologies used to distribute audio across a network and introduces Ethernet AVB and the control protocol associated with it. It also describes why Ethernet AVB was the protocol of choice for the KinectSound system's configuration and introduces the components that make up the KinectSound system.

- **Chapter 5** - Provides details of the design and implementation of the key areas of the KinectSound system, and how they fit together to fulfill the requirements.

- **Chapter 6** - Describes the steps taken to perform user testing and provides consolidated quantitative and qualitative results. The results obtained from the tests are also discussed in this chapter.

- **Chapter 7** - Presents a closing discussion of this research and future work.

The research and system implementation are also described more concisely in two papers presented at the 2013 South African Telecommunications, Networking, and Applications Conference (SATNAC) [16] and 2014 Audio Engineering Society (AES) convention [17].

# Chapter 2

# Surround Sound and Immersive Sound - State of the Art

## 2.1 Speaker Placement

Speaker placement is a key element of surround sound and immersive sound systems as a large part of the sound experience depends on where the speakers are located. The goal of surround sound and immersive sound systems is to provide an enhanced listening experience for the user by using speakers arranged around the user. Each speaker is the source of one or more audio channels. In cases where there is more than one channel of audio, they will have to be mixed together in order to produce a final output. The simplest surround sound system is known as "2.1 Surround Sound", which is simply two stereo speakers and a sub woofer. The purpose of the sub woofer is to manage the bass by playing the low frequency component of each channel [18]. The ".1" is derived from the sub woofer's strictly limited frequency bandwidth [3]. This section discusses the most common speaker setups used in surround sound and immersive sound environments. These range from the 2.1 speaker configuration to the Auro3D 9.1 speaker configuration. It also discusses the 7 speaker configuration used by the KinectSound system.

### 2.1.1 2.1 Surround

The simplest and earliest form of surround sound is the Stereophonic 2.1 speaker setup[1]. Stereophonic sound is sound that is divided into two separate channels which have been pre-recorded. The sound is then mixed to an extent and reproduced through 2 speakers and a sub woofer. The earliest recorded instance of "Stereo"

[1] Although one must be careful when designating this configuration as "Surround Sound"

being produced was by French engineer Clement Ader. He achieved this by using a series of telephone transmitters on the stage of an opera theatre, and sent these signals along short telephone lines. People at the Paris Electrical Exhibition could listen to the stream of the live stereo audio by holding a telephone receiver to each ear [19].

Stereo is by far the most common form of surround sound. Almost all computers, radios, and other home sound systems use a 2 speaker stereo setup. Stereo evolved from Monophonic sound, which is a single channel of audio that can be sent to one or more speakers. This results in all the speakers playing the same audio and is the most basic format of sound output [20]. Monophonic sound is easier as well as cheaper to record, and is used in telecommunications, intercoms, and public address systems where the surround experience is insignificant. Stereo is used in media players, cinema, and television. The speaker placement for 2.1 systems is described as the standard "Equilateral Triangle" setup [18]. The triangle's points are comprised of the left speaker, right speaker, and user. The triangle cannot always be strictly equilateral, due to space limitations as well as objects obstructing the speaker placement. For a home entertainment system, the speakers should be between 22° - 30° on each side of user, where 0° is the line from the user to the television screen [1]. As the viewing distance between the user and the television screen increases, so will the distance between the speakers in order to keep the triangle consistent.

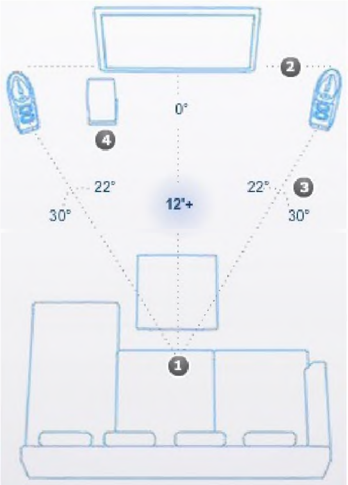


Figure 2.1 : The “Equilateral Triangle” setup used for 2.1 speaker placement [1]

Figure 2.1 shows the setup of the speakers with the user at a 12 foot viewing distance (approximately 3.5 meters). The viewing distance is not fixed as this may vary depending on the size of the display between the speakers. The viewing distance and the screen size are directly proportional to each other [1]. The components in the figure are numbered as follows:

1. The User

2. The Speakers

3. The Speaker Angles

4. The Sub-woofer

### 2.1.2 4.1 Surround

There are two speaker configurations for a 4.1 surround system [3], these are known as the *Quadraphonic* and *Cinema* configurations. Figure 2.2 shows the Quadraphonic configuration (A), and the Theater configuration (B).

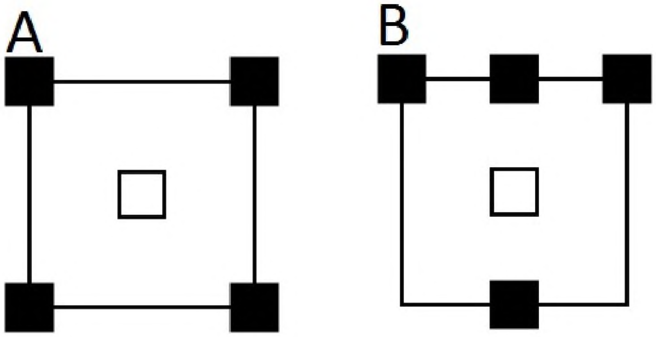


Figure 2.2 : The two different 4.1 speaker configurations used.

The quadraphonic configuration is the 4.1 configuration used in home theater systems. This setup is often referred to as the *phantom configuration* as it uses a stereo mix between the front speakers to generate a center channel. The purpose of the surround speakers is to provide ambience channels from the back, creating an "immersive" environment. These channels are rendered using a matrix format, either with four mono channels, or two stereo channels feeding the left/right speakers.

The theater configuration uses three front channels, and one surround channel. The surround channel feeds one (or more) speakers located behind or to the sides of the audience in cases where more than 1 surround speaker is used. The use of a single channel feeding the surround speakers limits the 360° localization capability.

The inability to localize sound in the theater setup, as well as the lack of a center speaker in the quadraphonic configuration, led to the development of 5.1 surround. This configuration was able to produce the desired localization effects not possible with 4.1.

### 2.1.3 5.1 Surround

The first instance of complete "surround sound" dates back to 1941 with Disney's release of the movie Fantasia. Disney achieved this by recording a different reel of sound for a specific scene and splicing it in for that scene when it was needed in the movie [21].

Surround sound, as we know it today, became digital in 1992 and was introduced by

Dolby Laboratories as 'AC-3' (Audio Encoding 3). This encodes the sound into a film's optical tracks, but in digital form as opposed to analog [21]. In 1993 Digital Theater Systems (DTS) also introduced their first surround sound technology. DTS recorded their sound on a CD and synchronized that with a DTS time code printed between the frames and sprocket holes [22]. DTS's first commercial film with surround sound was Steven Spielberg's award winning "Jurassic Park". Dolby and DTS are the two major competitors in today's home theater systems and are often referred to as the "Coke and Pepsi of modern day sound".

Five point one (5.1) is the most commonly used speaker setup in today's home theater systems [23] [18]. 5.1 speaker setups are composed of 6 audio channels being fed into 6 independent speakers.

A 5.1 surround system is composed of 3 speakers at the front, 2 at the rear, and a sub-woofer situated below the front right speaker or directly behind the user [18]. 5.1 configurations have been specified by the Audio Engineering Society (AES), as well as by Dolby and Digital Theater Systems (DTS).

### The AESTD1001 configuration

The Audio Engineering Society have provided a standardized speaker configuration, which is intended to provide the listener with an optimal surround sound experience [18] [2].

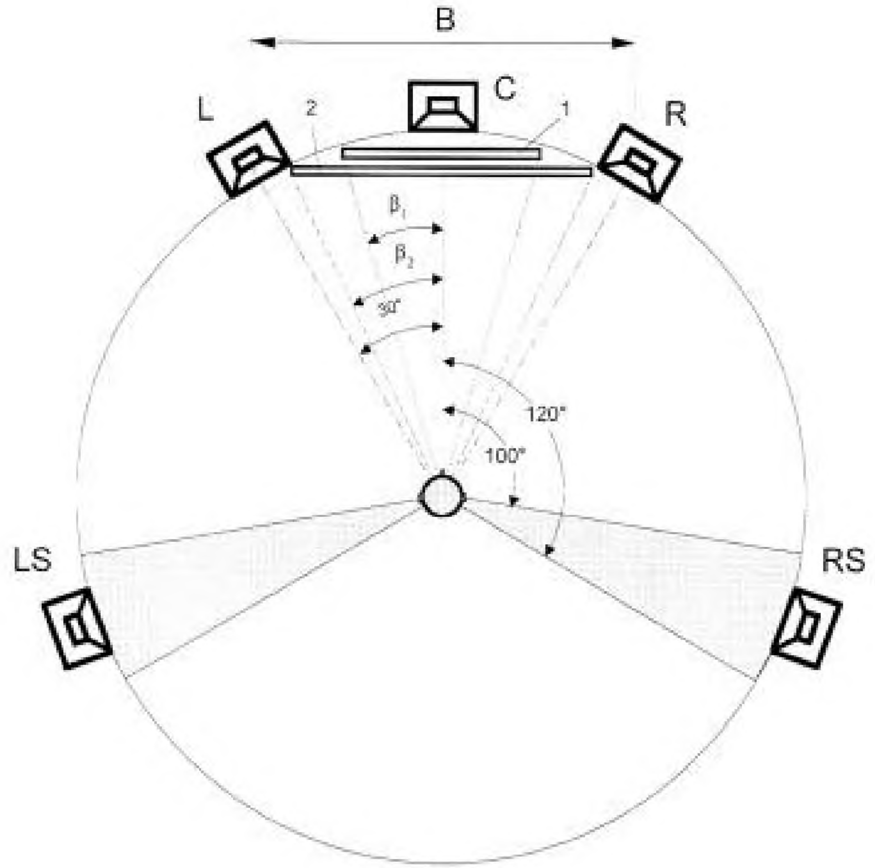


Figure 2.3 : The 5.1 Speaker configuration defined by the AES [2]

Figure 2.3 shows the optimal speaker placement for a 5.1 surround sound system. The left and right speakers (labelled 'L' and 'R') are in the same position as they would be in a 2.1 speaker setup. This is 30 ° on each side of the listener where 0 ° is directly in front of the listener in the center of the viewing screen. In addition to 2.1, there is a center speaker which is located at 0 ° and is either above or below the screen (labelled 'C'), as well as two more speakers. The height that these speakers should ideally be placed is 1.2 meters above the ground [2] with the center speaker height being dependent on its size, so that it does not obscure the screen.

The two additional speakers are labelled 'LS', and 'RS' which stands for Left Surround and Right Surround. These audio channels are known as "Ambience Channels" [2]. These speakers are located between 100 ° and 120 ° to each side of the listener [18].

Their height can be anywhere above 1.2 meters with their tilt at a maximum of 15° downwards.
The speaker location angles can be described mathematically as follows, where $\theta$ is 0° or the center of the screen as viewed by the user. The letters used in the equations denote the angles of the speaker locations rather than the channels:

- $C = \theta$
- $L|R = \theta \pm 30^\circ$
- $LS|RS = \theta \pm (110^\circ \pm 10^\circ)$

### The Dolby / DTS configuration

The Dolby or DTS configuration[2] has a similar configuration to that of the AES configuration, but discards the accuracy of the speaker position. This is due to inevitable space limitations and shapes of the venues. The AES configuration requires the speakers to be placed in a circular fashion around a single listener at certain angles. The Dolby/DTS configuration requires the user to be in the center of the five speakers which are placed as either a square or rectangle around the user [1].This allows the configuration to be used in venues such as cinema theaters and homes where there are space limitations.

[2] This configuration is used by more than just these two audio/visual brands

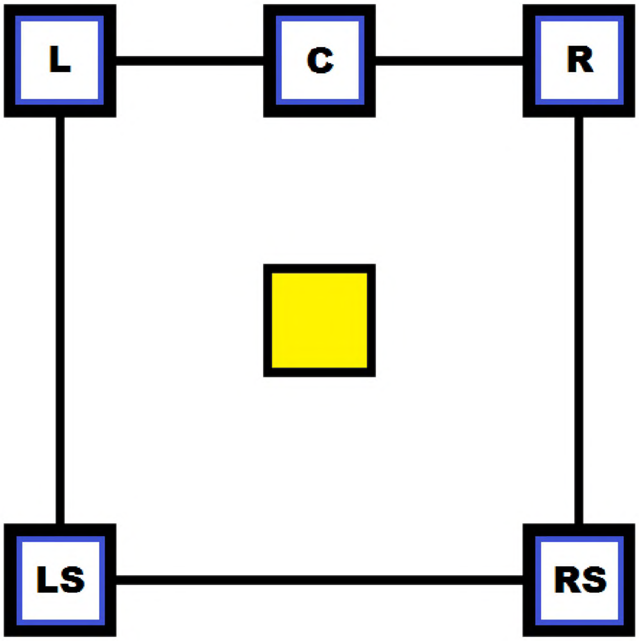


Figure 2.4 : The 5.1 Speaker configuration defined by Dolby/DTS

Shown in Figure 2.4 is a diagram of the Dolby/DTS layout of a 5.1 surround system. The center square shows where the user (or audience) is located.

### 2.1.4 7.1 Surround

The seven point one (7.1) surround sound system is a surround sound speaker setup that is also commonly used in home theater systems today. There are two main configurations that are used for 7.1 surround. These are known as the Dolby Home Theater configuration [1] and the Sony Dynamic Digital Sound (SDDS) configuration.

#### The Dolby Home Theater configuration

This 7.1 surround system uses all the components of a 5.1 channel system, but has two extra audio channels in order to provide better rear localization [24]. The 7.1 configuration has two speakers adjacent to the user, and two behind the user [24].

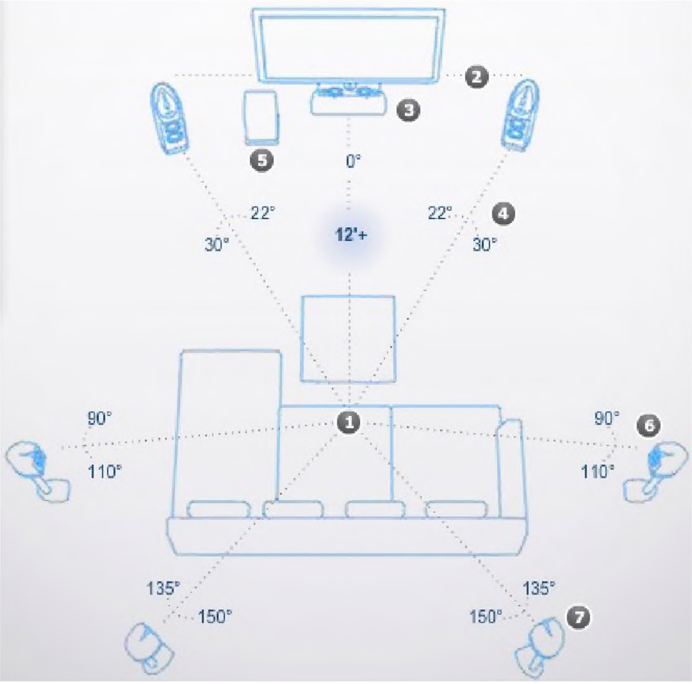


Figure 2.5 : The 7.1 Speaker configuration defined by Dolby [1]

Figure 2.5 shows the 7.1 surround configuration defined by Dolby. The left and right speakers (labelled 2) are placed exactly how they would be in the 2.1 stereo placement. The center speaker (labelled 3) is located in the same place as in the 5.1 configuration.

The components in the figure are numbered as follows:

1. The User
2. The left and right speakers (L/R)
3. The center speaker (C)
4. The left and right speaker angles
5. The Sub-woofer
6. The left and right surround speakers (LS/RS)
7. The left and right back speakers (LB/RB)

The speaker configuration can be described in a similar mathematical fashion as the 5.1 configuration, where $\theta$ is 0°, and is the center of the screen as viewed by the user.

- $C = \theta$
- $L|R = \theta \pm (26° \pm 4°)$
- $LS|RS = \theta \pm (100° \pm 10°)$
- $LB|RB = \theta \pm (142.5° \pm 7.5°)$

### The Sony Dynamic Digital Sound (SDDS) configuration

The Sony Dynamic Digital Sound (SDDS) configuration was first used in the movie *Last Action Hero* in 1993. It is a configuration that supports up to 8 independent channels, however only a small fraction of films using this method of encoding have used all eight channels. This configuration is not intended for private consumers, but rather for large venues such as cinemas and auditoriums. There are two main reasons as to why this configuration is not used privately:

1. The SDDS encoding is not available in any form of home entertainment
2. The equipment required for this setup is very expensive

The cinemas where this would be used are ones where the screen is large enough to accommodate two further speakers, for example IMAX cinemas. This speaker layout is similar to the DTS 5.1 configuration, with an additional two speakers in between the center, and left/right speakers. These speakers are known as the center left (CL) and center right (CR) speakers.

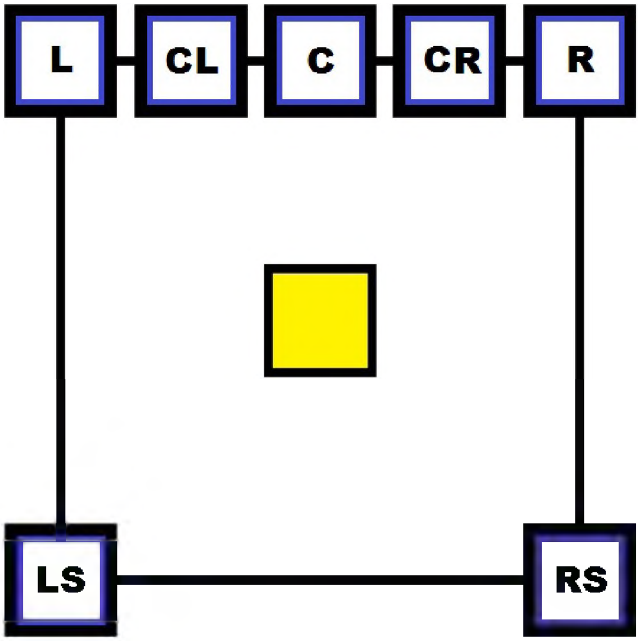


Figure 2.6 : The 7.1 SDDS configuration [3]

### 2.1.5 Auro3D 9.1

Auro3D is an audio format which has enabled immersive sound by adding on to current surround configurations [4]. The Auro3D configurations have added channels for height, as well as over head, to the current surround channels contained in Dolby 5.1 configurations. Due to the fact that the Auro3D configurations add speakers on to the default 5.1 layout, the configurations are often referred to as *5.1 + $x$* where $x$ is the number of speakers that have been added on to the 5.1 configuration.

The 9.1 configuration (alternatively known as *5.1 + 4*) uses the 5.1 configuration with 4 added speakers for the height channels. Shown below is the speaker layout of the Auro3D 9.1 surround [4].

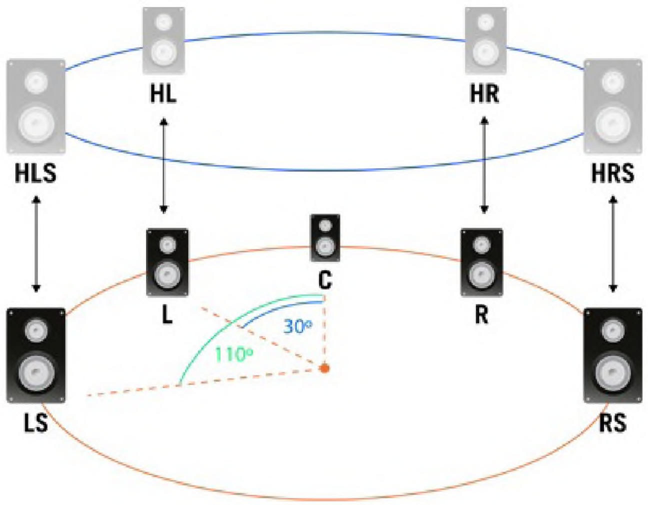


Figure 2.7 : The Auro3D 9.1 speaker layout [4]

Shown in Figure 2.7 is the 5.1 Dolby layout as well as the added Auro3D height layer. The speakers in the 5.1 configuration are shown around the lower circle, while the Auro3D height layer is shown around the upper circle.
The speaker positions can be described by a rotation in degrees relative to $\theta$, where $\theta = 0°$, and an elevation in degrees relative to $\phi$, where $\phi = 0°$ [3] [4].

| | Speaker rotation | Speaker Elevation |
|---|---|---|
| C | $\theta$ | $\phi$ |
| L | $\theta$ - 30° | $\phi$ |
| R | $\theta$ + 30° | $\phi$ |
| LS | $\theta$ - 110° | $\phi$ |
| RS | $\theta$ + 110° | $\phi$ |
| HL | $\theta$ - 30° | $\phi$ + 30° |
| HR | $\theta$ + 30° | $\phi$ + 30° |
| HLS | $\theta$ - 110° | $\phi$ + 30° |
| HRS | $\theta$ + 110° | $\phi$ + 30° |

### 2.1.6 The KinectSound system Speaker Configuration

The KinectSound system configuration uses a variation of the Auro3D 9.1 system [4], which is the 7.1 with height configuration [25]. This configuration excludes the *RB* and *LB* speakers from the 7.1 Dolby configuration shown in Figure 2.5, and replaces them with two height speakers located in the front height layer($\theta \pm$ 30°, $\phi$ + 30°). The 7.1 with height configuration was chosen for the following reasons:

1. Audio routing - The AVB bridge (Chapter 4) provided 8 populated RJ45 ports. Due to the fact that one port was reserved for the workstation[3], it only allowed 7 endpoints to be connected. The *Presonus Firepod* which was used to route audio conventionally, also provided 8 outputs, which allowed 7 speakers and a sub-woofer to be connected. The Presonus Firepod was the audio interface used with commercial sound localization systems for comparative purposes.

2. Endpoint availability - Each speaker used within the system contained an Attero Tech AVB endpoint. During the construction of this system only 7 Attero Tech endpoints were available for use. This limited the number of surround speakers to 7. The subwoofer was not included in the configuration as the system was testing panning and localization rather than the overall user experience.

Figure 2.8 shows the layout of the KinectSound system speaker configuration. The Dolby 5.1 speakers are shown on the grey plane, with the LH and RH speakers in the front. The ideal listening position of a user is also shown.

[3] This was for the audio multicast and control packets for the endpoints

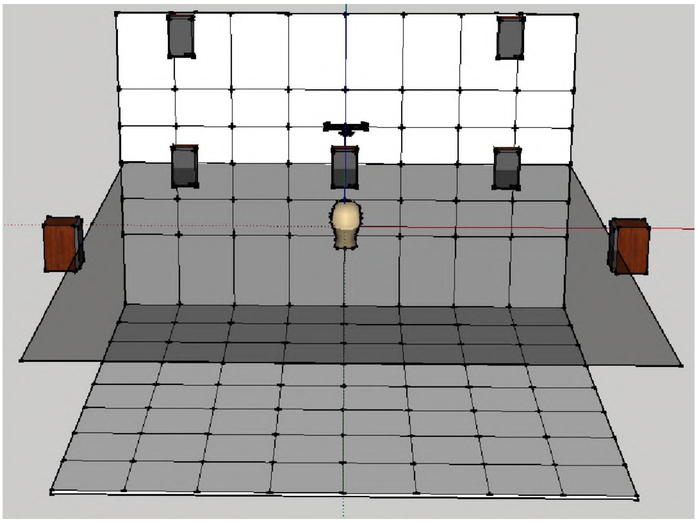


Figure 2.8 : The 7 speaker KinectSound system configuration modelled in Sketchup

The height surround speakers were left out in preference for height front speakers due to the fact that a humans echo location accuracy decreases when a sound source is moved beyond the $\theta\pm$ 90° rotation [26]. Having speakers in this region would not be as effective as having them in front of the user.

## 2.2 3D Audio Encoding

3D Audio, or spatial audio, refers to the technology used to recreate a scene that is able to "immerse" a person in a sound space [28] [29]. Enveloped immersive audio refers to the ability to virtualize a sound source at a position in space with accurate audio localization [29]. Several 3D audio encoding standards are beginning to use a similar paradigm to Object Oriented Programming (OOP). The OOP languages encapsulate attributes and operations that relate to each other, in single objects [28]. The object approach is used for audio encoding as it enables transport of an audio scene as discrete entities as opposed to fixed channels. These entities are often referred to as audio metadata. Audio renderers are able to re-create the audio scene for various speaker configurations from the attributes of the audio objects. Since the audio scene is recreated at the decoding stage, this allows for greatly improved scalability when

compared to the transmission of fixed signals. This also allows various audio effects to be applied to the audio signal in realtime [28].
Extensible Markup Language (XML) schemas are commonly used to describe audio scenes as they provide a simple, yet robust method of representing audio metadata. XML is able to describe most parameterized data via the use of tags. An XML tag may contain either a set of attributes, or further XML tags. This enables an object oriented approach since tags containing sets of attributes or further tags may be treated as objects. Immersive audio systems are now evolving to have flexible audio formats, rather than a fixed channel approach [30]. XML plays an important role in this evolution because it can be easily parsed. This allows a sound localization system to re-create an audio scene, as long as it is able to parse the XML file correctly, and a prescribed standard is being followed.

### 2.2.1 Dolby Atmos

Dolby *Atmos* is an immersive sound technology developed by Dolby Laboratories and released in 2012 [31]. Atmos can pan up to 128 audio tracks which are dynamically rendered depending on the speaker configuration's requirements. There are three components which make up an Atmos mix, these are:

1. *Bed audio* - consists of channel based premixes or stems. This audio is often used for ambient audio effects.

2. *Object audio* - is any mono or stereo audio that requires dedicated panning. The panning of the object audio is controlled by the Atmos metadata.

3. *Atmos metadata* - contains the data which describes the panning automation for each audio object.

The 128 tracks in an Atmos encoded audio piece consist of up to 9.1 bed audio tracks and 118 independent audio objects. The audio beds are routed directly to the speakers whereas the objects have their positions described by the metadata and are rendered in realtime. The Atmos' metadata format is not open and an Atmos mix requires a Dolby Rendering and Mastering Unit (RMU) to be rendered.

### 2.2.2 Auro3D Octopus

The *Auro3D Octopus* codec provides an immersive playback format with 3D spatial audio capabilities. The Auro3D codec is used by Barco [32], a Belgian visual hardware manufacturer for cinema configurations. This codec is backward compatible and is able to downmix audio streams for standard surround sound systems. Upmixing a stream requires the Auro-Matic plugin in order to decode the signal [33]. This codec uses a channel-based approach as a means of audio transport.

The Auro3D Octopus codec uses 24 bits for the audio delivery format, mainly for compatibility purposes with existing high-resolution audio standards. An audio signal using 18 bits provides a range of up to 105dB, which is sufficient for the reproduction of most professional audio. The remaining bits are utilized by Auro3D Octopus to carry audio metadata. Some of the important attributes contained in Auro3d Octopus' metadata are [33]:

- An identification and channel count of an audio stream.
- Simple post-production effects such as range reduction.
- Instructions for the upmixing or downmixing of the audio stream.

As the Auro3D Octopus codec is proprietary, the format of this codec is not available.

### 2.2.3 MPEG-H 3D Audio

MPEG-H is a group of audio standards in development by the Moving Picture Experts Group (MPEG), a group working on the development of standards for digital audio [34]. The goal of MPEG-H audio is to provide an audio codec that is able to:

- Render 3D audio for a variable number of speakers in a surround/immersive speaker configuration.
- Complement Ultra High Definition (UHD) displays which currently provide up to an 8K (7680 x 4320 pixels) resolution

MPEG-H implements a format converter which allows audio to be converted for playback to a variety of speaker configurations [35]. It uses a combination of channel based audio, object based audio, and Higher Order Ambisonics (HOA) to reproduce an audio scene. There are 5 operations in the rendering of MPEG-H audio, which are [35]:

1. Audio signals are decoded in a Unified Speech and Audio Coding (USAC) stage known as USAC-3D. USAC is the audio encoding scheme used by MPEG which maps a single input channel to multiple output channels. USAC-3D enhances USAC by allowing it to decode audio into a 3D context.

2. Channel based signals are mapped to the target speaker configuration by a *format converter*. The format converter renders the sound to a speaker configuration using the raw audio that has been decoded in step 1.

3. Object based signals are rendered to the target speaker configuration by an object renderer which uses MPEG metadata. Contained in the metadata are attributes such as azimuth and elevation angles, radius, gain, element spread, and an optional dynamic priority[4].

4. Signals encoded by the original Spatial Audio Object Coding (SAOC) are rendered to the target configuration using their associated metadata. This step serves as an alternative measure for signals that use the SAOC codec as opposed to the USAC codec.

5. Higher Order Ambisonics (HOA) signals are rendered to the target configuration using their associated HOA metadata.

[4] A full description of the MPEG metadata can be found in [36]. Each object uses these attributes to describe their position in a 3D space. The object renderer uses Vector Based Amplitude Panning (VBAP) to localize the objects' sound source.

### 2.2.4 Digital Theater Systems' Multi Dimensional Audio

Digital Theater Systems (DTS) is an American digital audio solutions provider [37]. Their solutions are used in a variety of audio configurations ranging from commercial to home systems. The DTS-HD Master is the most frequently used audio codec for Blu-ray encoding [38]. DTS has announced the release of DTS:X, an immersive audio technology based on object audio. This technology was announced for release in March 2015 [38]. The object audio in this codec uses X, Y, and Z co-ordinates which are attached to objects in order to localize them in a listening environment. The speaker placement can be flexible to accommodate configurations that aren't similar to professional configurations. This will be complemented by backwards compatibility to allow current surround sound and stereo configurations to use DTS:X [38].

### 2.2.5 SMPTE 25CSS

The Society of Motion Picture and Television Engineers (SMPTE) announced the TC-25CSS project in March 2013 [39]. The aim of this project is to standardize areas within the digital cinema (D-cinema) architecture, and in turn provide interoperability between independently developed audio systems [40]. There is no current standard for audio systems that utilize Object Based Audio Essence (OBAE) for the distribution of audio. The TC-25CSS project has 5 five main goals within its scope, these are [40]:

1. Create a standard that specifies an OBAE file format which contains all the metadata about an audio piece. This audio piece is able to play back on different configurations and have the same localization intent as the original encoding.

2. Create a specification which specifies how the OBAE bitstream is transported throughout the configuration. This specification needs to take into account any security protocols used by the D-cinema specification.

3. Create a standard renderer that is able to render the received bitstream to a variety of different speaker configurations. This requirement should fulfil a

variation of speakers along with the different sizes of the venues that will be used.

4. Ensure synchronization from a central server which is accurate to each distributed audio sample. The synchronization messages sent from the server will be interoperable with the current D-cinema architecture.

5. Work with the TC-21DC standard, a standard for high frame rates in D-cinema applications. This should allow the mapping of OBAE into the Digital Cinema Package (DCP). The DCP is a collection of digital audio and image files used to create cinema content.

6. Supply a document to consumers with instructions on how to calibrate the system to achieve the desired interoperability.

7. Create any other standards that are necessary in order for TC-25CSS to reach its specified goal

### 2.2.6 The EBU MPEG Surround Encoding

The European Broadcasting Union (EBU) provides a specification for the metadata description of audio objects. This does not include an actual format for carrying audio data, but rather the metadata which is carried with the audio in order to reproduce an audio scene [30]. The entire model is split into two different parts, the *content* part, and the *format* part. The essential elements of the EBU standard that provides the metadata description for an audio piece use the following structure[5]:

- The audio objects that are to have their metadata specified contain a start time and a duration.

- Each object is associated with a track which is an object that is associated with a Broadcast Wave Format (BWF) file.

[5] Only the elements that are relevant to the metadata description are shown in this diagram. The full description of the EBU specification is shown in Appendix 8.1

- Each audio track is contained within an audio stream and consists of audio channels and audio packs. An Audio Pack is responsible for grouping channels together. An Audio Channel consists of audio blocks
- The audio blocks specify co-ordinates for object localization, as well as a start time and duration for each set of co-ordinates.

The layout of the structure is shown in Figure 2.9

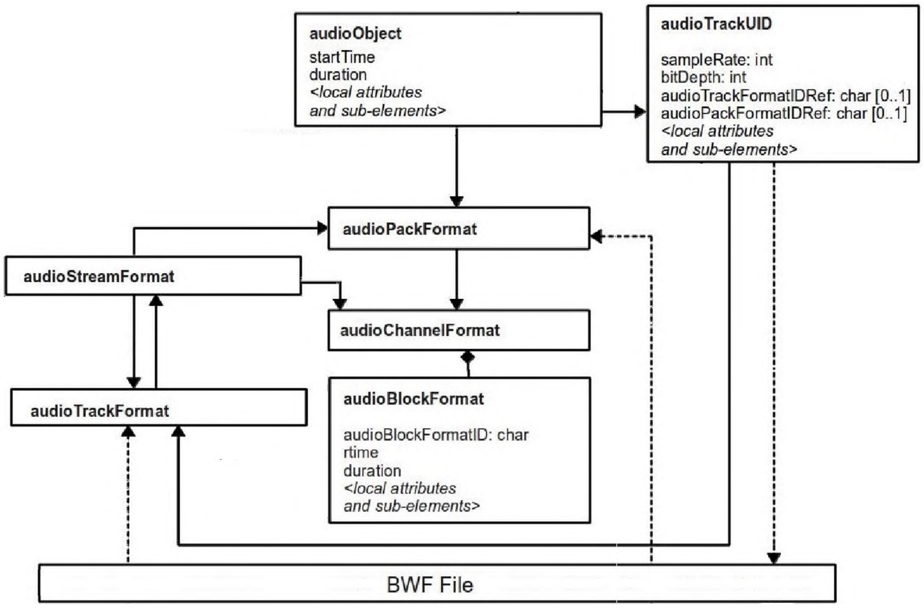


Figure 2.9 : The metadata structure of the EBU specification

### 2.2.7 The KinectSound system Encoding

The KinectSound system uses an encoding scheme similar to that of the EBU standard. The KinectSound system's encoding utilizes the DSP capability at each end-point to mix the audio according to the localization's metadata provided with each audio object. A layout of the decoding process is shown in Figure 2.10

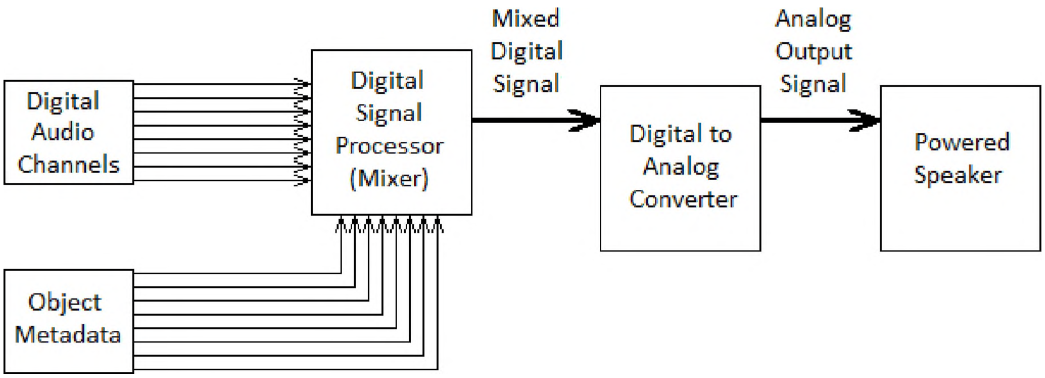


Figure 2.10 : The decoding stage in the KinectSound system

The metadata for the KinectSound system is encoded using XML. This method of encoding uses three different tags, which all appear in their own separate layers. These provide a simple approach for the main system to read the data about tracks and use it for calculating the mix levels required for each track. Due to the simple appearance of this metadata, it is also easily readable by an end user.

### The ***AudioPiece*** tag

The main tag within which all the data is contained, is called the *AudioPiece* tag. The KinectSound system only allows one *AudioPiece* to be stored at a time. Shown in Listing 2.1 are the opening and closing *AudioPiece* tags as they would appear in the XML data file.

```
<AudioPiece>
    <!—Additional  track  metadata—>
</AudioPiece>
```

Listing 2.1: The *AudioPiece* tags.

The purpose of the *AudioPiece* is distinguish between the beginning and end of the audio piece.

### The ***Track*** tag

Within the *AudioPiece* tag there are several *Track* tags which make up the *AudioPiece* block. Each track tag contains metadata about a single track. Shown in Listing 2.2

is how the several *Track* tags appear within the *AudioPiece* block.

```
<AudioPiece>
    <Track><!—TRACK1—>
        <!—Data defining track 1>
    </Track>
    <Track><!—TRACK2—>
        <!—Data defining track 2>
    </Track>
    <!—Additional tracks>
    <Track><!—TRACKN—>
        <!—Data defining track n>
    </Track>
</AudioPiece>
```

Listing 2.2: *Track* tags within the *AudioPiece* tag.

The *AudioPiece* tag may contain as many tracks as the system requires. The current KinectSound system is set to deal with a maximum of 8 tracks, so the XML file will only contain 8 *Track* tags within the *AudioPiece*. All the data pertaining to a single track will be found within its respective tag.

### The *TimedCoOrd* tag

The *TimedCoOrd* tags provide localization information. Each *TimedCoOrd* tag represents the location of the sound source at $\frac{1}{30}$ second intervals, the rate at which frames are received from the Kinect. The X, Y, and Z co-ordinates represent the distance in millimeters from the bottom front left corner of the room, referred to as the *KinectSound Origin* which is described in Chapter 5. The X, Y, and Z co-ordinates have maximum values, which are set by the KinectSound system for the current speaker configuration. Each *TimedCoOrd* tag contains a frame count, as well as an X, Y, and Z co-ordinate. Shown in Listing 2.3 are the first 20 *TimedCoOrd* tags for Track 1 of an audio piece, as they would appear in the XML file.

```
<AudioPiece>
   <Track><!—TRACK1—>
      <TimedCoOrd frameCount=‘‘0” X=‘‘1377.895” Y=‘‘1176.144” Z=‘‘1542.129”/>
      <TimedCoOrd frameCount=‘‘1” X=‘‘1428.809” Y=‘‘1175.578” Z=‘‘1539.043”/>
      <TimedCoOrd frameCount=‘‘2” X=‘‘1472.935” Y=‘‘1172.690” Z=‘‘1554.777”/>
      <TimedCoOrd frameCount=‘‘3” X=‘‘1505.147” Y=‘‘1174.622” Z=‘‘1558.291”/>
      <TimedCoOrd frameCount=‘‘4” X=‘‘1551.846” Y=‘‘1171.641” Z=‘‘1581.572”/>
      <TimedCoOrd frameCount=‘‘5” X=‘‘1585.886” Y=‘‘1171.407” Z=‘‘1599.222”/>
      <TimedCoOrd frameCount=‘‘6” X=‘‘1637.469” Y=‘‘1165.953” Z=‘‘1625.263”/>
      <TimedCoOrd frameCount=‘‘7” X=‘‘1677.192” Y=‘‘1163.507” Z=‘‘1647.021”/>
      <TimedCoOrd frameCount=‘‘8” X=‘‘1731.982” Y=‘‘1153.885” Z=‘‘1684.686”/>
      <TimedCoOrd frameCount=‘‘9” X=‘‘1767.906” Y=‘‘1151.372” Z=‘‘1708.127”/>
      <TimedCoOrd frameCount=‘‘10” X=‘‘1800.588” Y=‘‘1146.840” Z=‘‘1738.155”/>
      <TimedCoOrd frameCount=‘‘11” X=‘‘1856.947” Y=‘‘1144.492” Z=‘‘1761.973”/>
      <TimedCoOrd frameCount=‘‘12” X=‘‘1913.656” Y=‘‘1141.081” Z=‘‘1785.314”/>
      <TimedCoOrd frameCount=‘‘13” X=‘‘1968.975” Y=‘‘1139.749” Z=‘‘1797.092”/>
      <TimedCoOrd frameCount=‘‘14” X=‘‘2018.503” Y=‘‘1138.705” Z=‘‘1816.894”/>
      <TimedCoOrd frameCount=‘‘15” X=‘‘2063.627” Y=‘‘1141.803” Z=‘‘1805.307”/>
      <TimedCoOrd frameCount=‘‘16” X=‘‘2134.559” Y=‘‘1121.020” Z=‘‘1855.604”/>
      <TimedCoOrd frameCount=‘‘17” X=‘‘2165.708” Y=‘‘1123.483” Z=‘‘1851.895”/>
      <TimedCoOrd frameCount=‘‘18” X=‘‘2207.926” Y=‘‘1121.637” Z=‘‘1858.915”/>
      <TimedCoOrd frameCount=‘‘19” X=‘‘2288.948” Y=‘‘1116.037” Z=‘‘1863.848”/>
      <!—Additional TimedCoOrd tags>
   </Track>
   ...
   <Track><!—TRACKN—>
      <!—TimedCoOrd tags for Track n>
   </Track>
</AudioPiece>
```

Listing 2.3: The *TimedCoOrd* tags contain data about a track.

The interval between each *TimedCoOrd* tag is a constant $\frac{1}{30}$ of a second. This is because the Kinect is able to stream co-ordinates and the camera image to the main system at a rate of 30 frames per second [15]. Section 2.3.4 discusses the feasibility of using the Kinect's frame rate as an interval between co-ordinates for sound localization.

### Adoption of audio encoding standards

Storage formats such as DVDs have adopted surround sound encoding which allows an end user to play the surround sound through a home theater system. The Dolby AC-3 method of encoding is considered the surround sound standard and is used in both DVD and Blu-ray discs. DTS has been adopted by DVDs after its initial use by the *Jurassic Park* movie, however there are very few that use it. DTS uses a higher bitrate therefore providing better quality audio, but the difference is often considered neglible [27].

Dolby and DTS both provide further encoding for 7.1 surround on Blu-ray discs. Dolby Digital Plus and DTS-HD are considered ”lossy” due to the audio compression used. Dolby TrueHD and DTS-Master HD are uncompressed and lossless, making them identical to the studio master. Due to the fact that most Blu-ray discs have optional extras on them, storage space is often limited so these further encoding standards are often not included on the discs [27].

## 2.3 Sound localization and panning

Sound localization is a process naturally carried out by humans on a daily basis. It is the process of determining where a specific sound, or combination of sounds is coming from. A sound source is localized by its distance and direction. The direction can be described by its *Azimuth* (horizontal rotation), and its *Elevation* (vertical rotation). Two of the properties of a sound source which enable a human to identify its location are the Interaural Amplitude Difference (IAD), and the Interaural Time Difference (ITD) [41]. The IAD refers to the difference in amplitude of the sound perceived by the two ears. This allows a human to determine the distance from the sound source. The ITD refers to the difference in time taken for a sound to reach each ear. This allows a human to determine the direction in which the sound is coming from. Most of the speaker-based sound localization techniques today utilize speaker amplitude difference to localize a sound [41].

The following sections describe four different sound localization techniques, outlining the differences and discussing which was best suited for the KinectSound system.

### 2.3.1 Ambisonics and Higher Order Ambisonics

Ambisonics is an audio format based on Spherical Harmonics [8] [41] that is able to reproduce a sound field from a given direction rather than a sound source from a specific location [41]. Ambisonics works from the assumption that sound waves are planar and that the listener is located in the center of a spherical co-ordinate system [8]. A speaker configuration described as *Spherical* is a 3D speaker configuration where the distance between the user and any speaker remains constant. This method divides the sound field's direction into two components: the *Azimuth*, and the *Elevation* angles. The concept of using a general field or direction rather than a specific location renders the encoding and decoding of a sound (or series of sounds) using ambisonics, independent of the speaker configuration. Although the loudspeaker configuration is independent of the encoded audio, the best possible localization is achieved by having orthogonal speaker positions [10]. The accuracy of the ambisonic field is dependent on the extent of the *ambisonic order* that the encoder has used. Increasing the number of speakers in an ambisonic configuration does not increase the accuracy of the localization to a significant extent [10].

Ambisonics begins at the *Zeroth* order, which represents a mono signal (W) in a single channel. The $W$ channel is given by $S * \frac{1}{\sqrt{2}}$, where S is the signal sound source and is omni-directional. First order ambisonics uses three additional channels to encode sound fields on the X, Y, and Z, axes. Figure 2.11 shows the sound fields created by first order ambisonics.

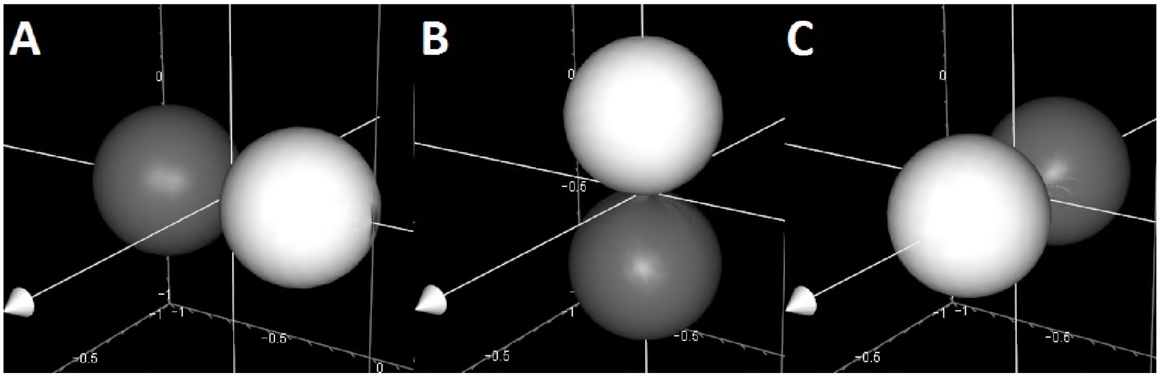


Figure 2.11 : The three additional channels used for first order ambisonics

Figure 2.11 shows the sound fields generated by the 3 channels that have been added to create first order ambisonics. As mentioned above, these are generated by the formulae for spherical harmonics, which use an input signal to generate an amplitude for a particular field. The positive and negative results for each ambisonic field are shown in white and black respectively. The formulae which correspond to each set of fields are given below [42]. The azimuth angle is given by $\theta$, the elevation angle is given by $\phi$, and the source signal is given by $S$:

A. $X = S * cos\phi.cos\theta$

B. $Y = S * sin\theta.cos\phi$

C. $Z = S * sin\phi$

Simple ambisonic panners encode a source by either recording with a soundfield microphone or by supplying the ambisonic formulae with a mono source in order to calculate the sound fields. The main ambisonic encoding format is called *B-format*, which allows multichannel audio [43]. The 4 channels resulting from the encoding equations are decoded by an ambisonics decoder. For 2D ambisonics, the decoding equations for a 4 speaker layout are as follows:

1. Front Left = $\sqrt{8}$ * (2W + X + Y)

2. Front Right = $\sqrt{8}$ * (2W + X - Y)

3. Back Left = $\sqrt{8}$ * (2W - X + Y)

4. Back Right = $\sqrt{8}$ * (2W - X - Y)

The formulas shown above can be extended to include a height factor Z for decoding in a 3D environment. In the decoding stage the decoder supplies a portion of the spatial encoded audio to each speaker. This is done proportionally for the speaker layout [44]. As the ambisonic order increases, the number of ambisonic fields increases. This results in the size of each ambisonic field decreasing to achieve a more accurate result. The number of fields in a given ambisonics order can be given by $f = (r+1)^2$, where $f$ is the number of fields and $r$ is the ambisonics order, which begins at the *zeroth order*. Figure 2.12 provides a visualization of the single vertical ambisonics field up the the 4th order.

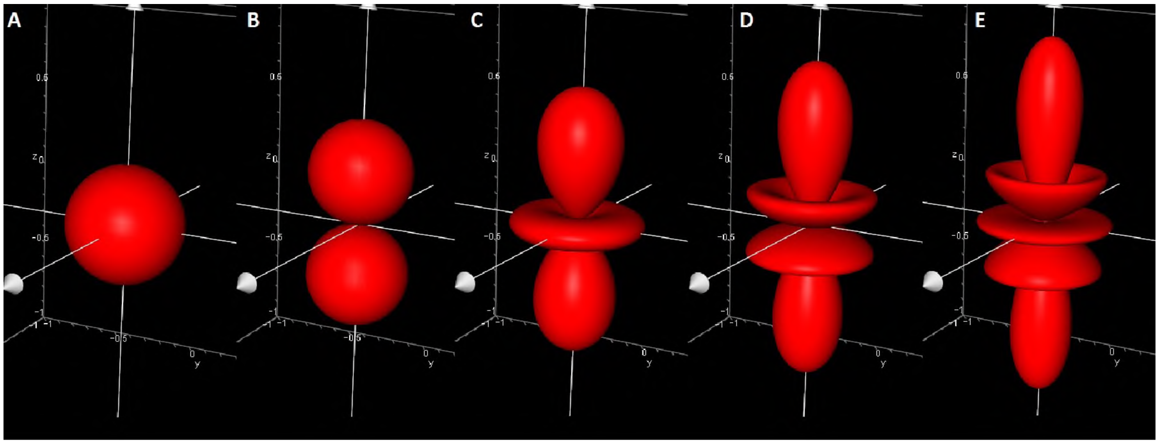


Figure 2.12 : The vertical ambisonics field shown from zeroth (A) to fourth (E) order.

Ambisonics was considered as the localization method in the KinectSound system due to it being independent of speaker configurations. Due to the accuracy requirement, Higher (third) Order Ambisonics (HOA) would be used. The azimuth and elevation angles can be determined by using the 3D co-ordinates provided by the Kinect in conjunction with the tan/arctan rule. This is given by ($tan\theta = \frac{o}{a}$ ; $tan^{-1}(\frac{o}{a}) = \theta$) where $o$ is the distance of the side opposite $\theta$ and $a$ is the distance of the side adjacent to $\theta$. This approach was discarded due to the computational complexity of the sound field calculations. The encoding stage requires 2 sets of *tan* calculations for each of the 16 channels in HOA to calculate the azimuth and elevation angles. Following this the B-format channels would be created using the azimuth and elevation angles [43]. These channels would need to be decoded to provide a signal to each speaker.

### 2.3.2 Binaural localization

Binaural localization refers to the ability of humans to localize a single sound source using sound stimuli at their two ears (binaural). Binaural localization was introduced in this section for the sake of completeness. The intention of the KinectSound system was to use an immersive configuration, however using a stereo configuration or headphones could be an option in future.

Binaural localization utilizes two components, the Interaural Time Delay (ITD), and the Interaural Amplitude Difference (IAD)[6] [9]. The human brain is able to detect subtle differences in the sound received by each ear, which enables it to locate the position of a sound source. ITD occurs when the sound from a source reaches the ear that is located physically closer to the source before the ear that is further from the source. The IAD occurs due to the fact that the listeners head will block a small portion of the sound energy from the ear located further away from the sound source [9]. Both ITD and IAD are greater when the sound source is located at ±90° from the user's facing direction, where 0° is directly in front of the user. These will also decrease as the source approaches 0° or 180°. When the source is positioned at 0° or 180° there will be no difference in the amplitude of the sound source or the time that it takes to reach the listener's two ears.

The ITD can be calculated under the assumption that the head is perfectly round [45]. Figure 2.13 shows a sound source labelled $S$ approaching a listener from an angle of 45°

[6] Also referred to as Interaural Intensity Difference (IID)

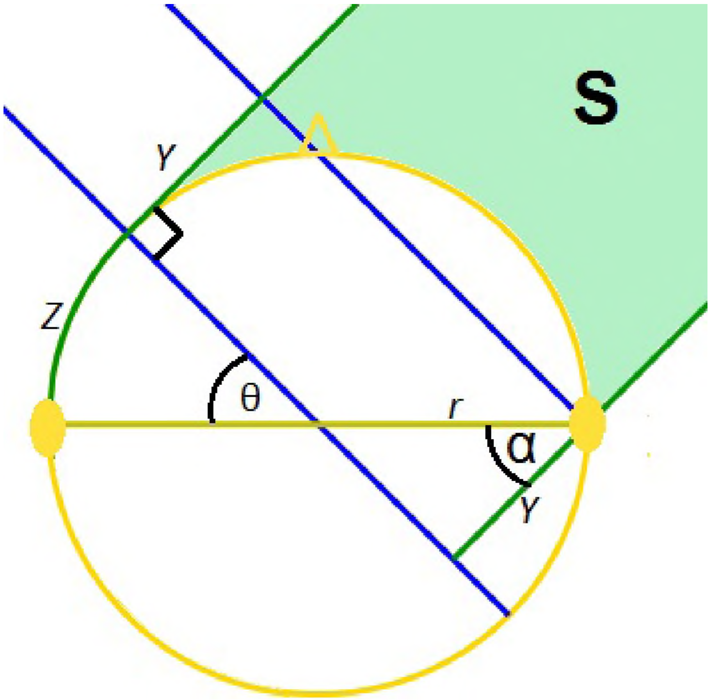


Figure 2.13 : Delay shown between a listeners two ears

The sound source is shown as $S$ in Figure 2.13 and will approach the human head at a certain angle, given by $\theta$. The ITD can be calculated by using the following formula, where $\Delta D$ is the extra distance the sound has to travel to reach the user's right ear and $S$ is the speed of sound:

$$\text{ITD} = \frac{\Delta D}{S} \qquad \text{- Eq. 2.1}$$

The extra distance the sound has to travel is broken down into two components shown in Figure 2.13 as $Y$, and $Z$. The value of Z is given by $Z = (\frac{\theta}{360})2\pi r$ as it is a portion of the circumference of the head. There are two line segments labelled $Y$ in Figure 2.13, which are equal lengths as both sets of intersecting lines are parallel. The $Y$ at the lower right of the figure can be calculated using the cosine law, due to the fact that the angle $\alpha = 90^{\circ} - \theta$, and $r$ are known. The value of $Y$ is given by $Y = r.cos(90\text{-}\theta)$, this can further be simplified as $cos(90\text{-}\theta) = sin(\theta)$, so $Y = r.sin(\theta)$.

Assuming the average human head has a diameter of approximately 20cm and the speed of sound at sea level travels at 340.29 meters per second, the delay between ears can be calculated using the following formula:

$$\frac{(\frac{\theta}{360})2\pi(0.1)+(0.1)sin(\theta)}{340.29} \qquad \text{- Eq. 2.2}$$

Localizing a sound using ITD and IAD can result in what is known as a *cone of confusion* which occurs when the ITD and IAD are identical. Cones of confusion are a result of the sound being deflected off the pinnae into the ear canal which causes a slight change in the ITD and IAD.

The use of binaural localization is most commonly used for stereophonic headphones, or two loudspeakers. It has been proven that for loudspeakers, it is better to pan using 3 speakers as 2 speakers produce sound images that are often more widely spaced than the actual sound source location [45]. The use of binaural delay for headphones works effectively and allows a simple localization technique which does not require any complex encoding or decoding.

### 2.3.3 Vector Based Amplitude Panning

Vector Based Amplitude Panning (VBAP) uses speaker locations as well as the listener location in order to produce a sound from a specific source within the listening environment [10]. For two speakers, 2D VBAP gives the same results as the tangent law [41]. The tangent law predicts the location of a sound source (given by $A$) in a 2D environment with the following equation [45]:

$$\mathrm{A} = arctan(tan(45)*(g_1\text{-}g_2)\ /\ (g_1 + g_2)) \qquad \text{- Eq. 2.3}$$

This equation implies that the speakers are located at a direction of -45° and 45° in relation to the user. The two variables labelled as $g_1$ and $g_2$ are the gain factors of the sound to be located for the two different speakers. These control the amplitude of the sound. In order to maintain a constant sound power, the sum of the speaker gain factors needs to have a constant value. This is shown by:

$$g_1^2 + g_2^2 = C \qquad \text{- Eq. 2.4}$$

Where $g_1$ and $g_2$ are the gain factors, and $C$ is the sound power that is required to remain constant.

Figure 2.14 shows a virtual source placed in the listening environment with an azimuth angle of $\alpha$ from the listener's 0° position. The speakers in a VBAP configuration are required to be equidistant from the location of the listener [10].

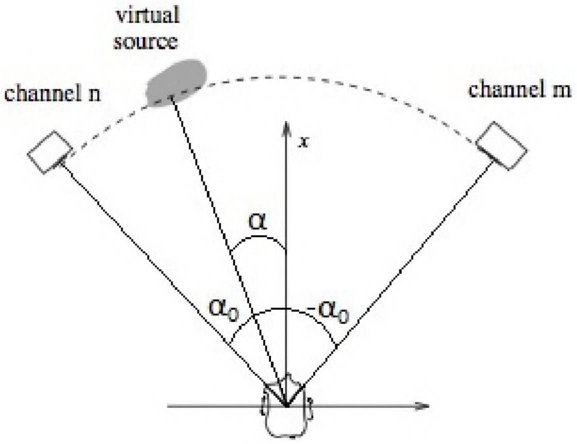


Figure 2.14 : A virtual source shown by an angle $\alpha$, in a 2D layout

Figure 2.14 uses $\pm\alpha_0$ to describe the speaker angles. The layout in Figure 2.14 shows the listener at an orthogonal base where $\pm\alpha_0 = \pm 45°$. The vector basis which describes the sound source location for 2D panning is formulated from the speaker configuration shown in Figure 2.14. Figure 2.15 shows the vectors used for 2D vector formulation.

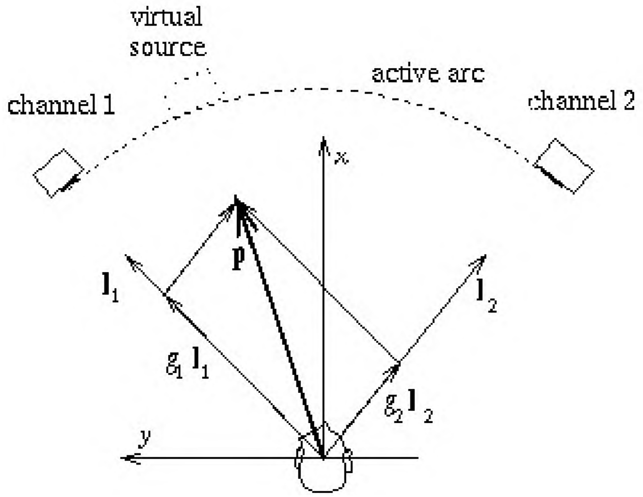


Figure 2.15 : The vector formulation for 2D VBAP

Figure 2.15 introduces a new vector, $\mathbf{p} = [p_1 \;\; p_2]^T$, which points toward the virtual sound source location[7]. This is the resultant vector of a linear combination of the loudspeaker vectors $l_1 = [l_{11} \;\; l_{12}]^T$, and $l_2 = [l_{21} \;\; l_{22}]^T$ and can be formulated as follows:

$$\boldsymbol{p} = g_1\mathbf{l}_1 + g_2\mathbf{l}_2. \qquad \text{- Eq. 2.5}$$

This equation can be written in matrix form $\boldsymbol{p}^T = \boldsymbol{g}\boldsymbol{L}_{12}$ where $\boldsymbol{g} = [g_1 \;\; g_2]$ and $\boldsymbol{L}_{12} = [\boldsymbol{l}_1 \;\; \boldsymbol{l}_2]^T$, and can be used to calculate the gain factors for both speakers. This can be solved if $\mathbf{L}_{12}^{-1}$ exists [10] [41]:

$$\boldsymbol{g} = \boldsymbol{p}^T\boldsymbol{L}_{12}^{-1} = [p_1 \;\; p_2] \begin{bmatrix} l_{11} & l_{12} \\ l_{21} & l_{22} \end{bmatrix}^{-1} \qquad \text{- Eq. 2.6}$$

The inverse matrix of $L_{12}^{-1}$ exists when $\alpha \neq 0°$ or $90°$ [10]. When $\pm\alpha_0 \neq \pm 45°$, the gain factors have to be normalized in order to maintain a constant sound energy. The normalized gain factors are given by the following formula [10]:

[7] The $T$ superscript denotes a matrix transposition.

$$\mathbf{g}^{scaled} = \frac{\sqrt{C}\,\mathbf{g}}{\sqrt{g_1^2+g_2^2}} \qquad \text{- Eq. 2.7}$$

Two dimensional VBAP with two speakers may be extended to localize sources between more than two speakers by reformulating the vector base to the pair of speakers closest to the localization point. The configuration would consist of a series of speaker pairs, with each speaker belonging to two pairs. Figure 2.16 shows the vector bases for a 2D VBAP configuration using 4 speakers.

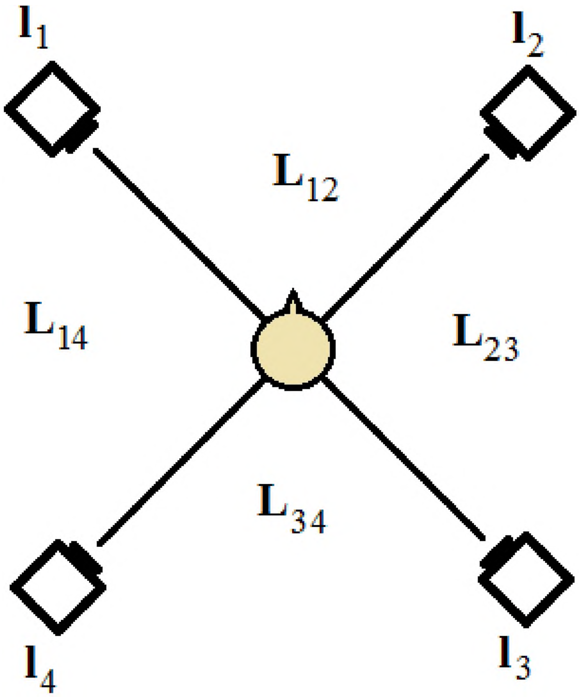


Figure 2.16 : 2D VBAP for more than two speakers

Figure 2.16 shows the configuration divided into 4 arcs. Each arc represents an area for localizing the sound source. Two of the speakers in an *active* arc are used to localize the sound source. The active arc refers to the vector base inbetween the two speakers where the sound source is to be located [10]. The fundamentals of this approach are the same is stereo VBAP, but with a changing vector base.

VBAP is able to pan in a 3D space by introducing a height component. The speakers for a 3D configuration can be in an arbitrary position but still require that they are equidistant from the listener [46]. VBAP allows a virtual sound source to be localized within a triangle formed by a triplet of speakers. The triangle is required to be observable from the listeners position [10]. Figure 2.17 [10] [46] shows three speakers, and the triangle that is formed amongst them.

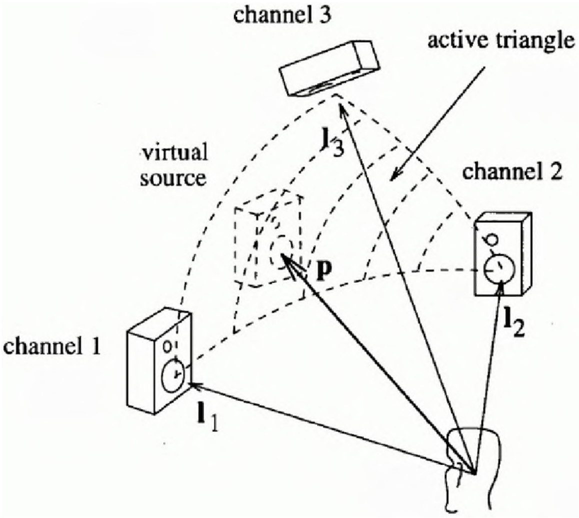


Figure 2.17 : The vector formulation for 3D VBAP

The same laws apply for 3D VBAP as 2D VBAP, this includes the gain normalization as well as the vector calculation. The vector $p$ in a 3D area can be calculated by $\mathbf{p} = g_1\boldsymbol{l}_1 + g_2\boldsymbol{l}_2 + g_3\boldsymbol{l}_3$ [10]. The gain coefficients are also required to maintain the sound amplitude, and are normalized by the following [10] [41]:

$$\mathbf{g}^{scaled} = \frac{\sqrt{C}\,\mathbf{g}}{\sqrt{g_1^2+g_2^2+g_3^2}} \qquad \text{- Eq. 2.8}$$

### 2.3.4 Distance Base Amplitude Panning

Distance Based Amplitude Panning (DBAP) is a matrix-based spatialization technique [11] which uses the actual positions of speakers in a speaker configuration to pan a sound. DBAP overcomes limitations associated with ambisonics and VBAP, which assume that the listener is located at a fixed position, and the speakers are in a ring or spherical formation around the listener [11] [12]. The distance between a virtual source in a 3D space with co-ordinates $(x_s, y_s, z_s)$ and any speaker in a 3D space with co-ordinates $(x_i, y_i, z_i)$ can be calculated using the Pythagoras Theorem in a 3D space [11]. This is illustrated in Figure 2.18

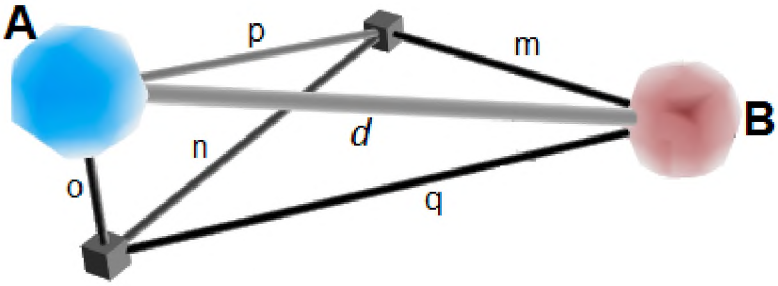


Figure 2.18 : Pythagoras' Theorem being used in 3D

Points $A$ and $B$ in Figure 2.18 show an example of a speaker's position (A), and the position of a sound source to be located (B). The side labelled $d$ is the direct distance between the speaker and the sound source. Given below are the distances related to the calculation of the required distance, $d$:

$$m = x_s - x_i$$
$$n = y_s - y_i$$
$$o = z_s - z_i$$
$$p = \sqrt{(y_s - y_i)^2 + (z_s - z_i)^2}$$
$$q = \sqrt{(x_s - x_i)^2 + (y_s - y_i)^2}$$
$$d = \sqrt{(x_s - x_i)^2 + (y_s - y_i)^2 + (z_s - z_i)^2}.$$

The gain for each speaker is required to be normalized in order to maintain a constant power for the system, similar to the process performed within VBAP. This formula is shown below [12], where $g$ is the gain factor for each speaker indicating the change in sound amplitude of the speaker:

$$\sum_{n=1}^{N} g_n^2 = 1 \qquad \text{- Eq. 2.9}$$

This equation reflects the fact that power is proportional to the square of the amplitude. The gain factor for each speaker can be calculated since the amplitude, and hence sound pressure at a sound source is inversely proportional to the distance of the sound source from the speaker [3]. This can be given by $g_n = c/d_n$ where $g$ is the gain factor of speaker $n$, $d$ is the distance of the sound source from speaker $n$ in meters, and $c$ is a constant dependent on the exact nature of the inverse distance law for sound propagation. These two formulae may be combined to produce the following speaker gain:

$$g_n = \frac{1}{d_n\sqrt{\sum_{i=1}^{N}\frac{1}{d_i^2}}} \qquad \text{- Eq. 2.10}$$

### DBAP in the KinectSound system

A modified form of DBAP was used in the KinectSound system. DBAP was chosen because it did not impose any constraints on the position of the speakers or of the user. The size of the room was also taken into account when the KinectSound system's DBAP calculations were being formulated. The room was small, approximately 4 meters in length and width with a 2.5 meter high ceiling. Multitrack audio samples were generated every $\frac{1}{48000}$ seconds ($20.8\mu s$) which required the audio mixing for a set of samples to be done in less than $20.8\mu s$. When an endpoint receives a multitrack set of samples, it multiplies each sample by an amplitude factor $a_i$, associated with track $i$. This amplitude factor varies between 0 and 1. When the amplitude is adjusted, the pressure of the sound waves travelling through the air is similarly adjusted.

As indicated previously, the inverse distance law for a single speaker's gain can be expressed as:

$$g_n = \frac{c}{d_n} \qquad \text{- Eq. 2.11}$$

where $g_n$ is a gain factor, and $c$ is a constant that is context dependent.

In the following equations and graphs, we use the term amplitude factor $a_n$, rather than gain factor $g_n$, since gain is often associated with dB units. So the equation is:

$$a_n = \frac{c}{d_n} \qquad \text{- Eq. 2.12}$$

The problem with this is that the amplitude factor tends to infinity as the distance tends to zero. An offset of 1 meter can be applied which will result in a gain factor that varies from 1 to approximately 0.2 if a constant value of $c = 1$ is used. This is given by the formula:

$$a_n = \frac{1}{1+d_n} \qquad \text{- Eq. 2.13}$$

A graphical representation of this is shown in Figure 2.19

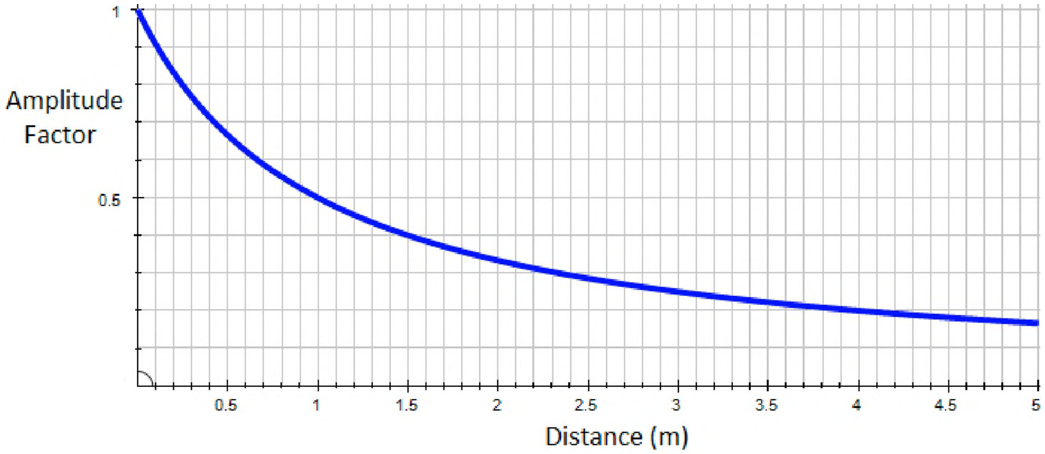


Figure 2.19 : Relationship between speaker amplitude (Y) and distance (X)

By graphing the dB value rather than the amplitude factor, it can be seen that there is a 10dB reduction and a perceptual halving of the loudness when the sound is located in the middle of the room. This is shown in Figure 2.20.

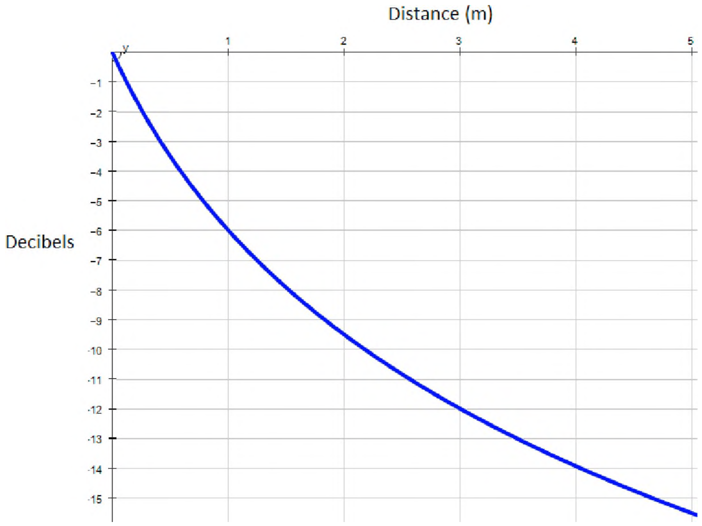


Figure 2.20 : Relationship between decibel value (Y) and distance (X) for the inverse distance calculation

Listening tests conducted in a room of a similar small size with a system that used the inverse distance law [47] showed that there was a large spatial width. This resulted in the sound being difficult to locate when it was moved around. The KinectSound system proposed and used an inverse square calculation. This was done by using the following equation for sound amplitude, where $a_n$ is the amplitude factor applied to each sample for a speaker $n$, and $d_n$ is the distance of the speaker from the sound source:

$$a_n = \frac{c}{1+d_n^2} \qquad \text{- Eq. 2.14}$$

Line $A$ in Figure 2.21 shows that by using this formula with $c = 1$, there is a faster drop in the amplitude of the sound source with distance, appropriate for a small room. The line marked $B$ in Figure 2.21 shows the amplitude drop when the inverse distance law is used.

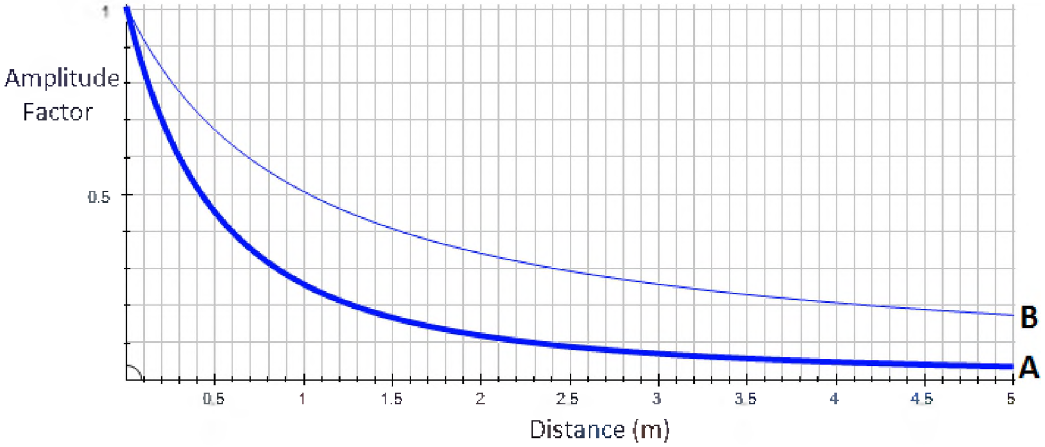


Figure 2.21 : Relationship between speaker amplitude (Y), and distance (X) for inverse squared (A), and inverse (B) functions

Looking at the decibel values for the inverse distance squared calculation, there is a 20 decibel reduction in the center of the room. The decibel values for the inverse distance squared calculation are shown as line $A$ in Figure 2.22. The decibel values for the inverse distance law are shown as line $B$ for comparative purposes.

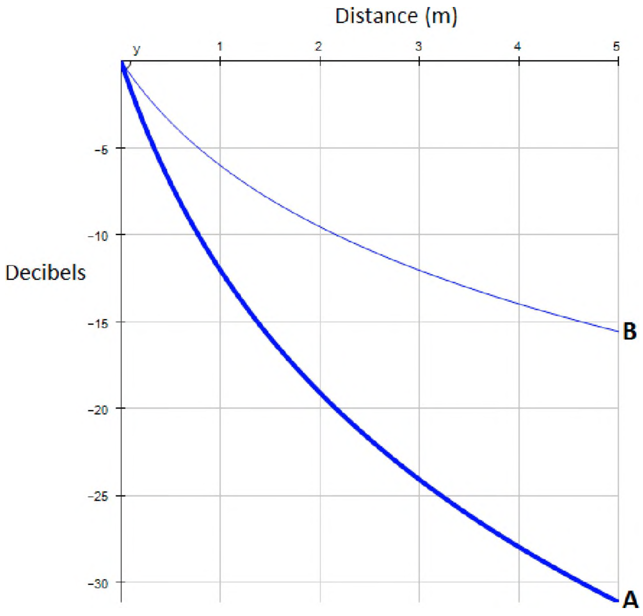


Figure 2.22 : Relationship between decibel value (Y) and distance (X) for the inverse distance squared calculation

This approach was confirmed with listening tests presented in Chapter 6. These listening tests consisted of users testing the KinectSound system against a commercial system that used VBAP for localizing sound sources.

One of the assumptions for DBAP is that the energy of a virtual sound source should be constant as it moves [11] [12]. This assumption does not hold when the speakers are not evenly spaced. The energy will tend to be higher when the sound source is located close to a cluster of speakers, and lower when the sound source is located further away from a cluster of speakers. This assumption implies the following, since the energy of a sound wave is proportional to the square of its pressure:

$$\sum_{i=1}^{N} a_i^2 = 1 \qquad \text{- Eq. 2.15}$$

This equation can be combined with the inverse square distance calculation used in the KinectSound system to get:

$$\sum_{i=1}^{N} \frac{c^2}{(1+d_i)^4} = 1 \qquad \text{- Eq. 2.16}$$

From this it can be inferred that:

$$c = \frac{1}{\sqrt{\sum_{i=1}^{N} \frac{1}{(1+d_i)^4}}} \qquad \text{- Eq. 2.17}$$

From this, the final amplitude factor for a speaker $n$ can be calculated using:

$$a_n = \frac{1}{(1+d_n)^2 * \sqrt{\sum_{i=1}^{N} \frac{1}{(1+d_i)^4}}} \qquad \text{- Eq. 2.18}$$

The factor added to this calculation from the original DBAP calculation serves to lower the speaker amplitude when the virtual sound source is close to a cluster of speakers and will increase the amplitude when the source is distant from any clusters, thereby ensuring constant energy.

### The time feasibility of DBAP in the KinectSound system

A performance test was done to calculate the feasibility of using these calculations in the KinectSound System. For this performance test, every calculation consisted of the distance and mix level calculations for 8 soundtracks per endpoint. Each time the calculation was done, the sound source distances for each track were assigned random values in the range that was feasible for the 3D sound source positions.

The processing times for the distance calculations as well as the amplitude factor calculations are shown Table 2.1. The amplitude factors are used to determine the mix levels at the speaker endpoints. Table 2.2 shows the processing times for a set of 8 samples, keeping the sound energy of the room constant as shown in the calculation above.

| $n$ | **1** | **2** | **3** | **4** | **5** | $\bar{x}$ **per sample** |
|---|---|---|---|---|---|---|
| 10000 | $75ms$ | $74ms$ | $75ms$ | $77ms$ | $78ms$ | $7.6\mu s$ |
| 50000 | $365ms$ | $368ms$ | $370ms$ | $368ms$ | $371ms$ | $7.4\mu s$ |
| 100000 | $748ms$ | $774ms$ | $746ms$ | $751ms$ | $746ms$ | $7.5\mu s$ |
| 500000 | $3770ms$ | $3749ms$ | $3765ms$ | $3787ms$ | $3777ms$ | $7.5\mu s$ |

Table 2.1 : Times taken to process a set of 8 samples for an endpoint

| $n$ | **1** | **2** | **3** | **4** | **5** | $\bar{x}$ **per sample** |
|---|---|---|---|---|---|---|
| 10000 | $470ms$ | $477ms$ | $472ms$ | $471ms$ | $463ms$ | $47.1\mu s$ |
| 50000 | $2314ms$ | $2285ms$ | $2335ms$ | $2270ms$ | $2387ms$ | $48.3\mu s$ |
| 100000 | $4971ms$ | $4678ms$ | $4623ms$ | $4593ms$ | $4616ms$ | $47.0\mu s$ |
| 500000 | $22960ms$ | $23769ms$ | $22956ms$ | $22990ms$ | $23365ms$ | $46.4\mu s$ |

Table 2.2 : Times taken to process a set of 8 samples for an endpoint, keeping the energy constant.

The tables above show the times taken to perform the distance and mix level calculations for the inverse squared distance law, and the inverse squared distance law with a constant overall energy. Each calculation calculates the distance between the endpoint and a sound source in a virtual space, then uses the distances to determine mix levels.

The column labelled $n$ shows the number of sample sets that have had their distances and mix levels calculated. Each sample set contained 8 randomized 3D co-ordinates. Each calculation performed a distance and mix level calculation for the co-ordinates of a track and another fixed co-ordinate, which would represent a speaker in the context of the KinectSound system.

The columns labelled 1 - 5 show the time taken to perform the calculations on 5 different occasions. The column furthest to the right shows the average time taken per set of samples to be processed. This number can be multiplied by the number of endpoints to calculate the amount of time it will take to process a single set of samples for a given number of endpoints in a system. The KinectSound system performs these calculations for 7 endpoints every time a frame is received by the system.

The processing time of the mix levels would only start to affect the system once the total time taken for the processing exceeds the time interval between when frames are presented to the system. The processing time exceeding the frame interval may be caused by increasing the number of endpoints, which would increase the overall processing time.

The KinectSound system receives new frames at a rate of 30 frames per second. It uses this new frame indicator as a trigger to dispatch a set of mix levels to the endpoints in the system. The performance test was done to determine whether the system was able to complete the mix level calculations within the interval between receiving two consecutive frames from the Kinect.

With the KinectSound system having 7 endpoints and a Kinect that is able to transmit at 30 frames per second, the time surplus that is available to be used for further processing may be calculated by using the following equation:

$$t = f - 48.3\text{x}10^{-6} * S$$

$$t = 33.33\text{x}10^{-3} - 48.3\text{x}10^{-6} * 7$$

$$t = 32.99\text{x}10^{-3}\text{s} = 32.99ms \qquad \text{- Eq. 2.19}$$

where $f$ is the frame interval in seconds, $S$ is the number of endpoints used in the configuration, and $t$ is the time surplus. The longest time taken to process a set of samples is taken from Table 2.2 and used in the above formula, which is a constant $48.3\mu s$

In order to determine that a co-ordinate sampling rate of 30 times a second provided an accurate sound panning motion, a test was conducted to determine the average distance a sound would move when the user is panning a sound very quickly. The sound source was panned steadily across the space of 3 meters over the time of approximately 800ms (24 Kinect frames). The average distance that the sound source moved during each frame interval was 13cm. A gap this small between each interval would appear to be a smooth movement of sound when it is detected by the human ear. Figure 2.23 shows the movement of the sound location as it would appear during this test.

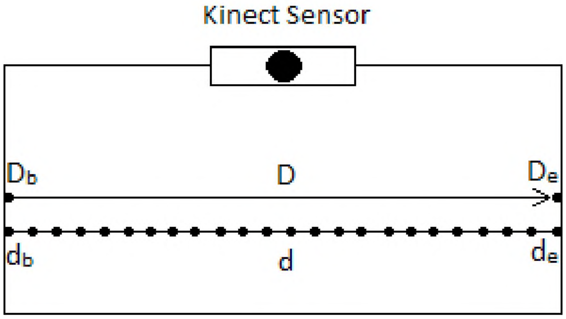


Figure 2.23 : The co-ordinate movement over a space of 24 Kinect frames

Figure 2.23 shows two lines labelled $D$ and $d$. $D$ is the total movement of the sound source between two points of the movement area, labelled $D_b$ and $D_e$. The line labelled $d$ shows the location of the sound source as it is localized after each frame during the movement from $d_b$ to $d_e$. Fast panning of sounds with the KinectSound system is enabled by the scaling of the co-ordinates. The KinectSound system scales the co-ordinates received from the Kinect by a factor of 7. This was initially done in order for the user to reach the front corners of the movement space. This would be impossible without scaling as the user's hand would not be within the Kinects vision. The scaling of the user's hand co-ordinates is discussed further in Chapter 5.

## 2.4 Chapter Summary

This chapter has discussed 3 core topics relevant to surround sound and immersive sound, namely speaker configurations, digital audio encoding, and surround sound localization techniques.

The first section covered various speaker configurations ranging from small home stereo systems, up to the Auro3D 9.1 configuration. The configuration for the KinectSound system was shown, and was described as a *7.1 with height* configuration, excluding the sub-woofer. Two main reasons were provided as to why this configuration was chosen, they were:

- 7 free ports on the AVB bridge

- Limited availability of Attero Tech AVB boards

Section 2 of this chapter discussed digital audio encoding techniques used in the industry, these were:

- Dolby Atmos
- Auro3D Octopus
- MPEG-H 3D Audio
- DTS Multi Dimensional Audio
- SMPTE 25CSS (A standardization of audio encoding)
- EBU MPEG Surround Encoding

The encoding method used for the KinectSound system was described. This was shown as an XML representation of 3D co-ordinates in which the endpoints would decode and localize in real time.

Section 3 of this chapter discussed four different methods used to pan audio in a 3D space. These methods were Higher Order Ambisonics (HOA), Binaural Localization, Vector Base Amplitude Panning (VBAP), and Distance Base Amplitude Panning (DBAP). The principles behind each method were discussed, along with diagrams and examples. The section on DBAP gives a further description of the relationship between sound pressure, sound power, sound energy, and amplitude. This section also described why DBAP was chosen as the preferred panning method for the KinectSound system. The reasons for this are:

- The listener does not have to be located in a specific listening position.
- This method uses any number of speakers, which are not required to be in any specific configuration.
- It does not require complex decoding and can be performed in real time

The section following DBAP gives a time feasibility analysis of the KinectSound system. It also describes how the control packets are dispatched whenever a frame is sent from the Kinect detecting the frames to the main system and how the frame rate may affect the processing if it gets too high. This was shown as an unlikely event to occur as only a small fraction of the total time is being used. If this did happen, alternative measures are able to be put in place in the form of timed sampling. This would change the procedures to happen on a timed interval as opposed to whenever a frame comes in from the Kinect.

# Chapter 3

# Human Computer Interaction (HCI) for Digital Audio

In recent years graphical user interfaces have been used to good effect by the software for audio systems. This has enabled enhanced control over audio systems. However the audio industry has not fully embraced current interactive interfaces [48]. There have been systems developed that allow remote control using devices such as tablets with a touch screen interface. These often, however, do not display more than the physical appearance of a device and contain controls such as knobs and faders [48].

This chapter describes usability metrics and compares various interfaces currently used for the production of audio including sound placement. Usability metrics measure various parameters of a system in order to determine a generalized measure of *Usability*, a term that has been loosely defined in the past [49]. The following sections describe user interfaces that enable the panning of audio sources, including control that is neither a mouse or keyboard based. A need for 3D control is presented and discussed by example of a system that was developed as part of this thesis, the KinectSound system.

The KinectSound system displays how a gesture controlled system used to pan audio allows the user to have a more interactive approach to sound control. It also satisfies the requirements of 3D control. There is a discussion of the various factors that had to be taken into consideration throughout the design and implementation stages of this system.

## 3.1 Usability Metrics

The *usability* of a system has been defined in various different ways in the past. The definitions of usability fall loosely into three main categories, these are [50]:

- **Semantics** - Semantics refer to how *User-friendly* a system is. This definition of usability identifies a system as highly "usable" if users are able to complete the task relatively easily, in a reasonable amount of time. This definition of usability may also refer to the amount of training a user needs with the system in order to use it.

- **Features** - This refers to the presence or absence of features contained in the interface that aid the user in completing a task. This requires interfaces to have enough features in order to complete a task, but not so many as to render the system complicated to use

- **Operations** - This measures the system in terms of how satisfied the user is with the end result of the operations the system has performed.

Evaluations of systems require a more detailed definition of the term *usability* in order to give an accurate determination of the user's experience of the system. Such a definition needs to encompass various aspects of the user's experience in order to provide an indication of how satisfied users may be with the system. The usability metrics of a system can be categorized into what is known as the *Five E's of System Usability* [51]. The *Five E's* are defined by the following terms [49] [51]:

- **Efficient** - Efficiency refers to the time it takes a user to complete a task, and the accuracy of the completeness. This can often vary depending on the user's past experience with similar software, or if they have been trained to use the software.

- **Effective** - Effectiveness refers to the user being able to accomplish the goals they wish to achieve. This is the kingpin of system usability because if the main goal of the system cannot be achieved, then nothing else really matters.

- **Engaging** - *Engaging* refers to what most people know as *user friendly.* The broad areas of engagement are the visual presentation of data and functions, types of images used, and any colour variation on the interface. A system's engagement may include many other factors which are application specific.

- **Error tolerance** - The error tolerance of a system refers to the prevention and recovery of errors that may be caused by the user. It may also refer to the ability to recover from any errors caused by the system if there are any.

- **Easy to learn** - A system that is classed as *easy to learn* assumes that little or no training is required in order to use the system effectively.

## 3.2 Device Controlled HCI for Real-time Audio Panning

This section describes the approaches that two current device controlled systems have taken towards 3D sound localization. The term *device controlled systems* refers to systems that use a standard keyboard and mouse as the user interaction. The usability of each of these systems is discussed in context of the Five E's described in the previous section. The examples given in this section were the only 3D localization systems that enabled localization of sound in real time.
Current real-time audio panning applications typically provide the following views of a venue within which a sound source will be located via user controlled audio panning. These are either:

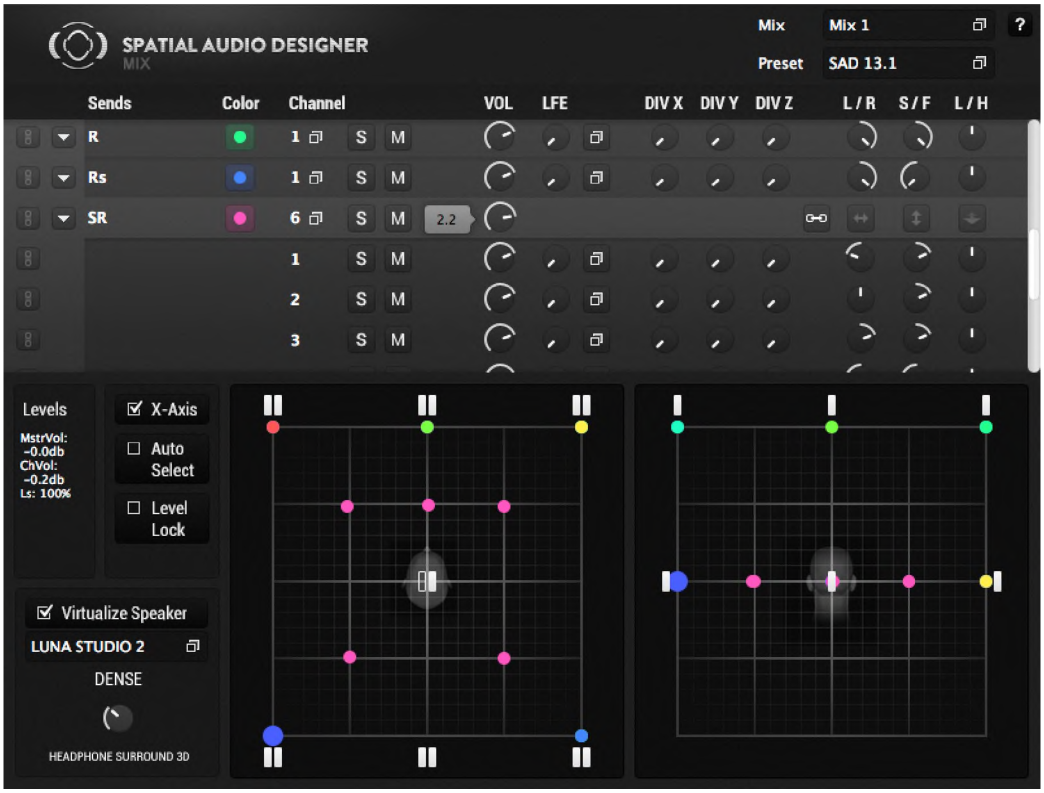
1. Two dimensional (2D) views of the venue, viewed from different aspects which allow the user to view the speakers and sound source with reference to any two of the X, Y, and Z axes.

2. A 3D view of the venue which enables the user to view the speakers and sound source in a 3D environment.

Examples of each of these views are shown later in the chapter.

### 3.2.1 New Audio Technology's Spatial Audio Designer

Spatial Audio Designer (S.A.D) is a Virtual Studio Technology (VST) [52] plugin for a Digital Audio Workstation (DAW). It provides mixing and monitoring of sound formats from mono up to 13.1 surround [13]. The plugin provides different speaker layouts that are pre-defined, an example of this being the *Auro3D 9.1* layout [4]. Figure 3.1 shows the mixing interface for the S.A.D using an 11.1 preset as the speaker layout.



Mix: Mix 1

Preset: SAD 13.1

| Sends | Color | Channel | | | VOL |
|---|---|---|---|---|---|
| R | ● | 1 | S | M | |
| Rs | ● | 1 | S | M | |
| SR | ● | 6 | S | M | 2.2 |
| | | 1 | S | M | |
| | | 2 | S | M | |
| | | 3 | S | M | |

Levels<br>
MstrVol: -0.0db<br>
ChVol: -0.2db<br>
Ls: 100%

☑ X-Axis

☐ Auto Select

☐ Level Lock

☑ Virtualize Speaker

LUNA STUDIO 2

DENSE

Figure 3.1 : The *mix* interface for the Spatial Audio Designer

In the lower region of Figure 3.1, the user's head is shown in relation to the various speakers. The user is then able to pan any of the sound sources listed in the upper region by clicking on the coloured dot that represents a particular sound source and moving it around in the 3D space. The left hand grid allows the user to pan the front/back and left/right movements of the audio, whereas the right hand grid allows the user to pan the left/right and high/low movements of the audio. Any specific movements that the user wishes to record may be done so and exported using the *Track Automation* feature, a feature which is provided by the supporting DAW.

### 3.2.2 Iosono Anymix Pro

Anymix Pro is a VST plugin created by *IOSONO*, a provider of end user electronics for audio [53]. The goal of Anymix Pro is to mix audio at preset inputs to preset outputs. These inputs and outputs range from mono to 8.1 surround. It can upmix or downmix from any input to any output. Figure 3.2 shows the interface for Anymix Pro.

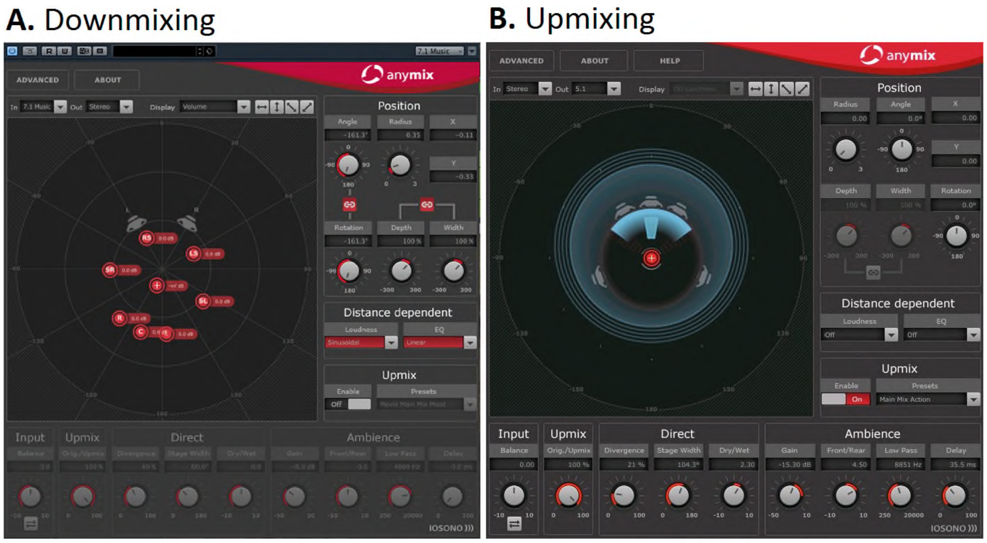


Figure 3.2 : The user interface for Anymix Pro

Figure 3.2 shows a downmix from a 7.1 input to a stereo output (A) and an upmix from a mono input to a 5.1 output. A major advantage of Anymix Pro is that it is able to localize sound sources outside of the speaker configuration. This allows it to localize *sound approaches* and *pass bys* where the sound source moves from outside the speaker configuration, through the configuration and out of it again [53]. The *Distance dependent* parameters on the right of each interface in Figure 3.2 allow the user to automate the loudness and equalization of the sound movement, enabling a more realistic sound when the audio is approaching or moving away from the listener. A major disadvantage of this panning software is that it is limited to a maximum of 8 speakers, which does not include height. Immersive surround sound configurations require height panning.

## 3.3 HCI for Audio Production Using 3D Hardware

This section describes and discusses two audio panning systems that do not use the keyboard and mouse for panning, but rather use specialized 3D hardware to pan audio. These two systems were the only two systems at the time of writing that allowed localization without the use of a keyboard and mouse.

### 3.3.1 The Haptic Feedback Prototype

A prototype for a sound localization system which utilizes haptic feedback was introduced in 2013 by Melchior *et al.* [5]. The hardware used for localizing audio in this system is a Novint Falcon, a haptic controller. This device contains a small controller which the user may grip and move around in 3 dimensions. The controller is connected to three arms which allows the free 3D movement of the controller. It also contains four small buttons on the top which allow additional input from the user. Figure 3.3 shows the structure of a Novint Falcon with the controller labelled 'A' and the two visible arms labelled $B_1$ and $B_2$.

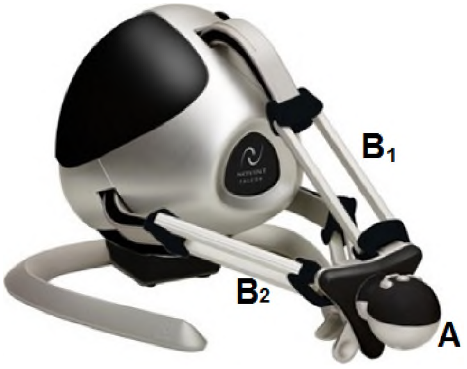


Figure 3.3 : The Novint Falcon haptic device

The device provides haptic feedback in the form of a force that is applied when the user is approaching the boundaries of the panning area. The boundaries of the panning area are predefined and are visible in the scene rendering section of the custom developed software. When a user panning the source approaches a boundary, the device will impose a force in the opposite direction.
The Novint Falcon is connected to a workstation via a USB port. This workstation contains a DAW and uses custom built software to provide visualization and audio control. The visualization is a 3D representation of the localization area and contains coloured objects to show the positions of the sound sources. Figure 3.4 shows the 3D display as it appears on the haptic feedback system.

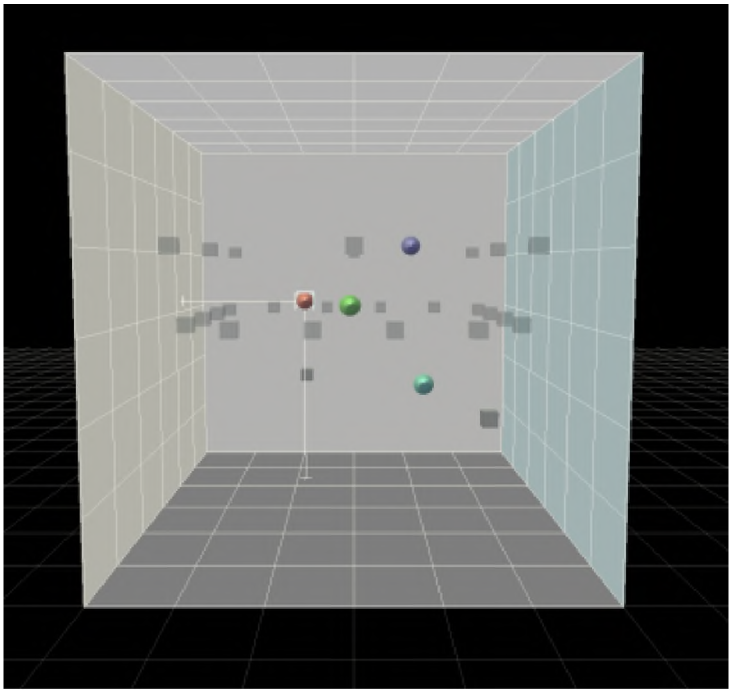


Figure 3.4 : The haptic feedback 3D display [5]

A second workstation is used which renders the spatial audio using commercial software. This workstation connects to the main workstation via a Multichannel Audio Digital Interface (MADI) for the audio data from the DAW, and an Ethernet connection for the control (localization) data. Figure 3.5 shows the configuration of the haptic feedback system.

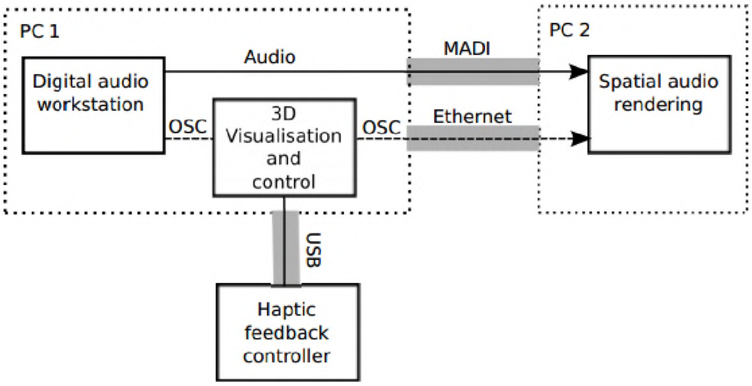

PC 1
Digital audio workstation
Audio
OSC
3D Visualisation and control
OSC
MADI
Ethernet
PC 2
Spatial audio rendering
USB
Haptic feedback controller

Figure 3.5 : The configuration of the workstations in the haptic feedback system [5]

The system was tested and evaluated by a group of sound production professionals. It was tested against another sound localization system which used the mouse to change the sound source location from a bird's eye view with an additional slider to change the height of the sound source location. The users were required to perform a series of movements of a sound source using both systems and give feedback on their experience with the system. The results from this test indicated that the haptic feedback system was more practical for movements involving all 3 dimensions. The users gave a better evaluation for the mouse controlled system when the movement was only through 2 dimensions as the mouse provided better control for smaller and more accurate movements.

### 3.3.2 The JL Cooper NUAGE Surround Panner

The JL Cooper NUAGE surround panner is a surround sound controller developed by JL Cooper electronics. It contains a joystick, a series of buttons, rotary encoders, and an LED display as the control mechanisms that the user may interact with [54]. The surround panner is used in conjunction with the system "NUAGE" which it connects to via an Ethernet interface. NUAGE is a system that incorporates both Yamaha hardware and a Steinberg DAW to create a real time or post production tool [55].

The DAW used for NUAGE is Steinberg's Nuendo [56]. Nuendo allows NUAGE to apply a variety of sound effects to an audio piece and provides support for Anymix Pro, the plugin by IOSONO described in Section 3.2.2. Figure 3.6 shows the surround panner.

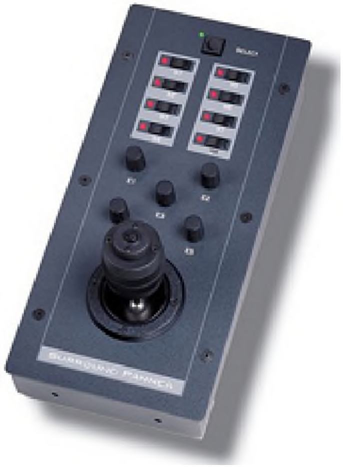


Figure 3.6 : The JL Cooper surround panner

The joystick on the surround panner is able to rotate around 3 axes, providing the capability to pan in a 3D environment. Although the joystick can perform 3D movements, NUAGE does not yet provide a tool to perform 3D panning using the JL Cooper surround panner.
As stated in the requirements of the KinectSound system, the intention was to create a system with complete device free control. The user should be able to control the system without having to interact with a keyboard, mouse, or any other device.

### 3.3.3 A Usability Review of 3D Localization Control Approaches

This section revisits the Sections 3.2 and 3.3 in this Chapter and reviews them in terms of the Five E's as described in Section 3.1.

### Standard Mouse and Keyboard Systems

The localization systems that have been discussed in Section 3.2 do not allow a user to move a sound source through all three dimensions in a single movement. They require a separate action to move the source through the height dimension. This makes the system less *efficient* than a system which enables immediate 3D input. However the mouse does provide accurate input and in this regard could be seen to be *effective*. This requires experience with VST plugins and for a user unfamiliar with VST this would require a learning curve. The time it takes for a user (efficiency) of each system is directly related to the user's previous experience with audio recording software and this may vary from user to user. The only real errors that would be encountered in these systems would be incorrect sound localization which may be attributed to the user, and this is easily rectifiable.

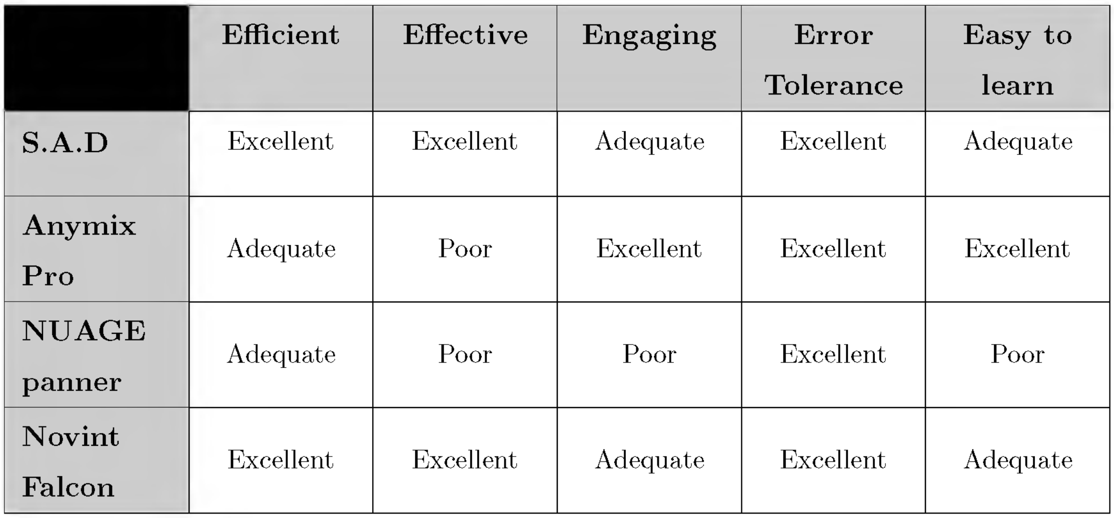
### 3D Hardware Control

The 3D hardware systems described in Section 3.3 provide a more *efficient* approach to sound panning than the systems described in Section 3.2 because they allow sound sources to be localized with 3D hardware rather than the standard keyboard and mouse combination. The ability to pan audio sources in three dimensions enables faster and more intuitive (efficient) entry than both the S.A.D which requires two different movements for 3D panning, and Anymix Pro which does not provide 3D panning.

The JL Cooper NUAGE panner and the Novint Falcon require the user to translate their perception of a 3D sound location into control movements. The control movements can then be made by interacting with the 3D control hardware. Although this 2 step stage of localizing a sound source provides a solution, complete device free control is more desirable. Device free control would allow the user to localize the sound source at the desired location by direct movements instead of a movement from a perceived location.

The Kinect was able to provide the device free control that was required by using the

3D co-ordinates of the user's hand as the location of the sound source. This allowed the user to control the location of the sound source via direct hand movements. It did not require perception of the sound source to be translated into a movement which they had to control via the 3D hardware.
Table 3.1 gives an indication of the degree to which each system has satisfied each of the Five E's of system usability for a 3D sound localization system.



| | **Efficient** | **Effective** | **Engaging** | **Error Tolerance** | **Easy to learn** |
|---|---|---|---|---|---|
| **S.A.D** | Excellent | Excellent | Adequate | Excellent | Adequate |
| **Anymix Pro** | Adequate | Poor | Excellent | Excellent | Excellent |
| **NUAGE panner** | Adequate | Poor | Poor | Excellent | Poor |
| **Novint Falcon** | Excellent | Excellent | Adequate | Excellent | Adequate |

Table 3.1 : A table describing the Five E's for the localization systems in this chapter.

## 3.4 Hardware That Enables Device Free Control

This section presents hardware that enables device free control and discusses each one's potential to be incorporated into a spatial audio system.

### 3.4.1 The Leap Motion Controller

The Leap Motion controller is a device that uses hand recognition in order to provide control of a workstation via a USB port. The dimensions of the Leap Motion are *76mm* x *13mm* x *13mm* [57] which allows it to be located in front of a seated user. The Leap Motion controller provides accurate hand and finger tracking and can track the joints of each hand independently. Figure 3.7 shows a 3D rendering of a user's hands as they are detected by the Leap Motion controller in real time. Figure 3.8 shows a 3D rendering of the joints located on each of a user's hands as they are detected by the Leap Motion [57].

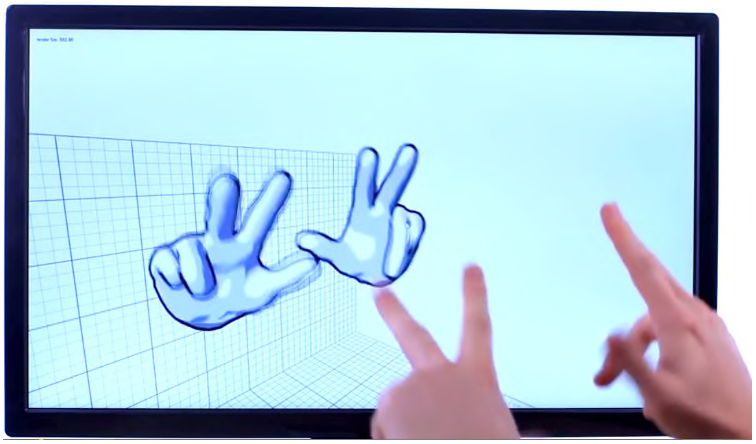


Figure 3.7 : A user's hands being detected by the Leap Motion and rendered in 3D

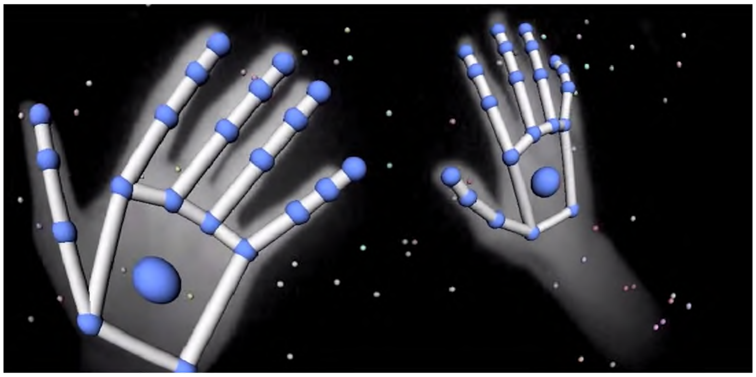


Figure 3.8 : The joints on a user's hand as detected by the Leap Motion

Along with the accurate hand tracking, the Leap Motion provides an Application Programming Interface which renders it suitable to be used for device free control for audio panning and localization. The Leap Motion has been used for audio production and panning in systems such as the Motion Mix [58] system.

### 3.4.2 Standard Web Camera

Standard USB webcams may be used to provide control over a workstation via the use of hand tracking. Many of the systems that use hand tracking do so by integrating a library known as Open source Computer Vision, or more commonly referred to as *OpenCV*. OpenCV is an open source computer vision library which is supported across various programming languages and is compatible with most operating systems [59]. It provides classic and state of the art tracking and learning algorithms which may be included and utilized in programming modules. Two of the most common webcam based hand tracking algorithms included in OpenCV are:

#### 1. CAMShift

The Continuously Adaptive Mean Shift (CAMShift) algorithm tracks hands based on a colour that has been selected. It performs this by adapting colour sequences on an image to the selected colour. To show the colour field it is tracking, it shows an ellipse which encapsulates the colour being detected by the CAMShift algorithm. The accuracy of CAMShift tracking is dependent on how unique the selected colour to be tracked is. If this colour is similar to other objects in the room, the algorithm may not function properly and track incorrect objects. Systems that use CAMShift often choose to track luminous objects that are easily distinguished from any other objects in the room. Figure 3.9 shows a user's hand being tracked with CAMShift along with the colour histogram of the user's hand.

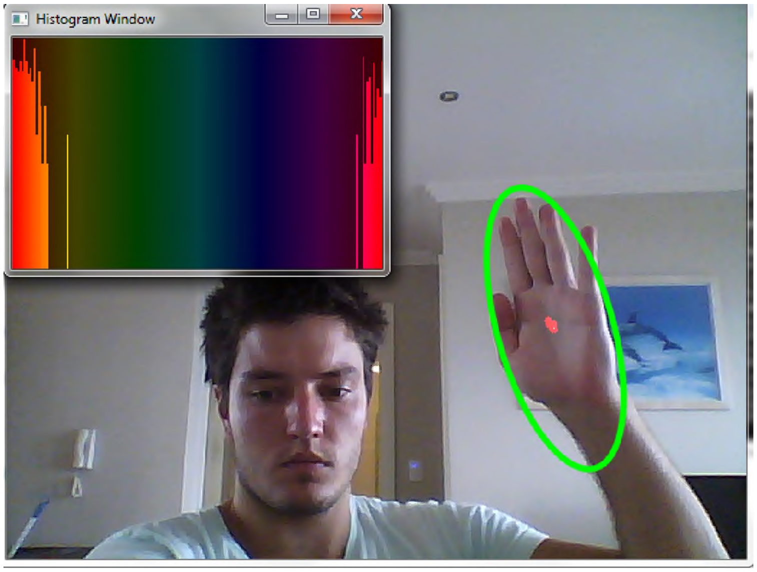


Figure 3.9 : A user's hand as detected by the CAMShift algorithm

### 2. Background subtraction

Background subtraction uses a static background as a template and subtracts it from newly detected frames derived from a webcam to determine where changes in the frame have occurred. These changes are then passed through a filter so that only significant sections will be shown. The changes that will appear in the subtracted frames will be the user's hands that are not contained in the static background frame. A *running average* frame is often used for background subtraction rather than a static background. This eliminates stationary objects that have entered the frame. The running average implementation is better suited for hand tracking and gesture recognition as it would only detect changes in the hands gesture or position when it changes [60].

OpenCV provides a contour extraction mechanism which is used to find the edges of a user's hand and identify the concave and convex regions which are the user's fingers. Figure 3.10 shows the hand recognition using background subtraction.

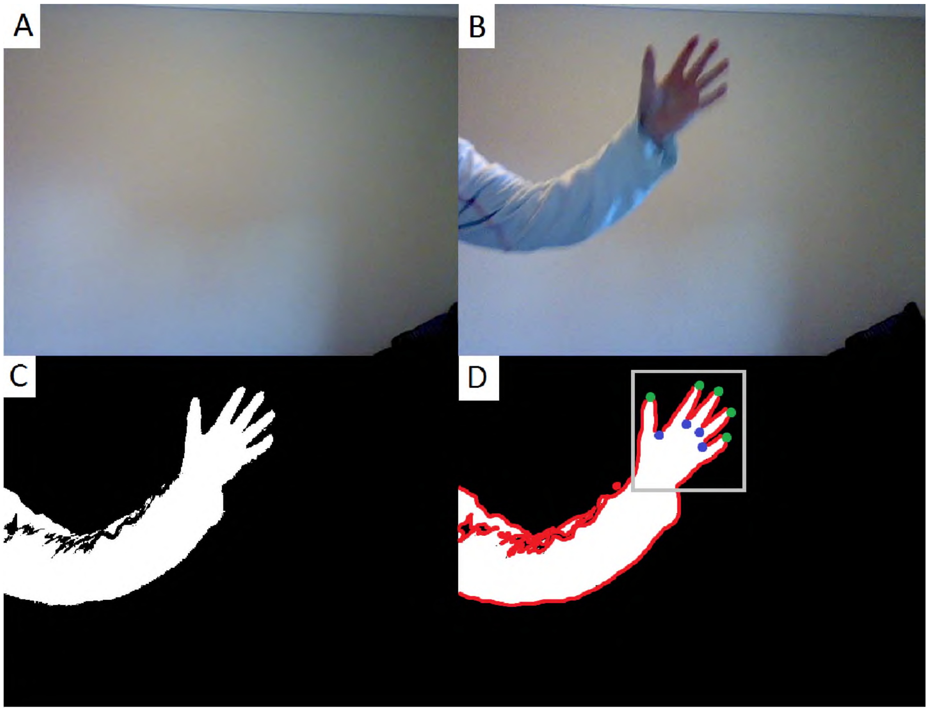


Figure 3.10 : Detection of a user's hand via background subtraction

Figure 3.10 shows 4 different images that are used in the background subtraction hand recognition process, these are:

A. The static background used as the template for background subtraction.

B. The frame containing the user's hand as it is seen by the webcam.

C. The difference between Frame A and Frame B converted into a black and white image.

D. Frame C once the OpenCV contour and hand recognition have been applied. This frame shows the concave and convex points on the users hand which are used to detect the user's fingers.

### 3.4.3 The Windows Kinect

The Windows Kinect is a device developed by Microsoft that was initially used for motion sensing with the XBox 360. This provided a user with an interactive way of gaming by using their motions to control certain features of a game. The Kinect contains a standard RGB camera, an infrared emitter, an infrared receiver, and a microphone array [61]. It is able to detect human skeleton structures via a series of algorithms, after it has used the *light structuring* technique [62]. The Kinect performs light structuring by using the infrared emitter to project a known pattern of infrared light onto a scene. The infrared sensor is able to render 3D objects by detecting any deformations of the emitted infrared light. The skeleton structure detected by the Kinect is composed of 20 joints as shown in Figure 3.11.

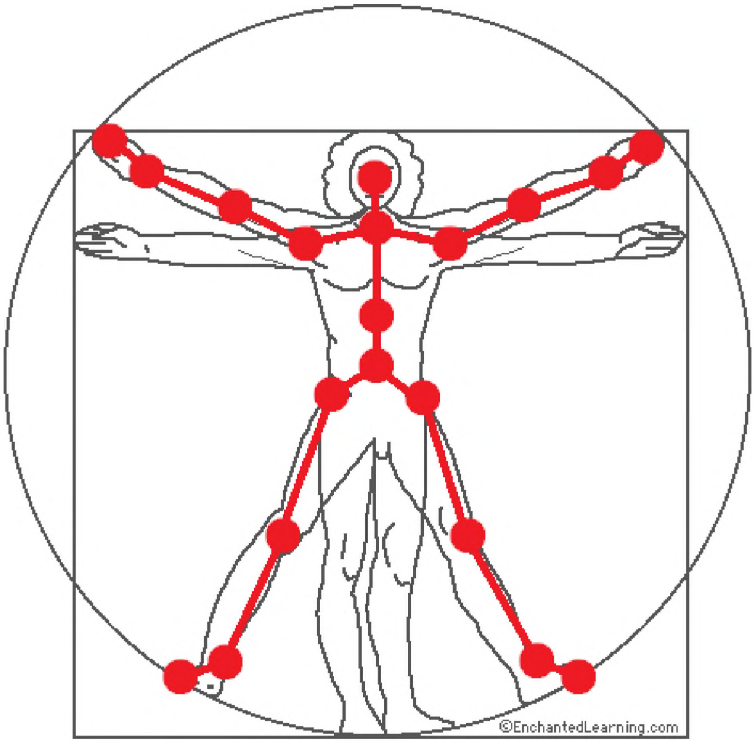


Figure 3.11 : The Kinect skeleton

The Kinect may be connected to a workstation via a USB port and accessed through the SDK released by Microsoft [63]. This allows software modules to access and use the features of the Windows Kinect. All of the algorithms required for skeleton tracking are encapsulated within the SDK and accessed directly. This eliminates the need for a user to implement the algorithms as they would have to for a webcam implementation. The Kinect's movement tracking is able to track any user uniquely. This is independent of their size (as human bodies differ) and does not require any further configuration [64]. These features make the Kinect an attractive choice of hardware for device-free workstation control.

### 3.4.4 An Analysis of the Usability of Device Free Control Interfaces

The Leap Motion provides a form of device free localization that fitted all the requirements of the implementation for this research. The Leap Motion was considered as the choice of hardware for the project presented in this thesis, however it was only made available in July 2013 to customers that had pre-ordered it. This was more than 6 months into the start of this research, in which a significant amount of implementation work had already been completed.

Tracking a user's hand movements and gestures with a webcam would be possible, but the requirement of 3D control makes this approach less attractive. The concept of depth perception would require 2 webcams and a significant amount of extra processing to be done by the system. By using a Kinect, the skeleton tracking performed by the Kinect could be utilized and the 3D co-ordinates of the user's skeleton retrieved directly from the Kinect SDK. The Kinect's skeleton tracking capabilities are also a lot more accurate and reliable than a user-implemented hand tracking algorithm done using a webcam [64]. This minimized the risk of error when controlling the workstation.

## 3.5 Sound Localization with the KinectSound system

The Kinect provided a solution to the limitations described in the previous sections. This section describes how the Kinect was used to provide a usable system that is able to localize audio in three dimensions.

### 3.5.1 The KinectSound System's User Interface

*Device free control* of the application had to be implemented because the user is not able to use the mouse or keyboard due to the position they are in when they are using the KinectSound system. Device free control was achieved by superimposing controls and any relevant information onto the displayed camera image acquired from the Kinect and displaying this in a different dialog box as shown in Figure 3.15. The KinectSound system's consists of three different *User Interfaces*. Each user interface consists of a set of controls. The controls on each interface provided either a similar function or provide navigation to another interface. A user is able to select a control option (provided it is selectable) by moving their left hand into the control area, then moving their left hand forward as if they were pressing a button in the virtual space. The controls' functions were grouped by colour to indicate their functionality. The colour of each control refers to the colour of it's border. These colours were chosen based on their visibility and how likely they were to contrast against background colours. The colour groups are as follows:

- **Dark green** buttons indicate navigation buttons. By selecting one of these buttons, the user will be directed to a new interface.

- **Yellow** buttons indicate commands sent to the system. These buttons perform a specific command, which is described within the button.

- **Light blue / light green** buttons are track buttons. Tracks are all light blue by default and change to light green when they have been selected.

- **Purple** buttons are buttons that have changed the state of the system. The only buttons that may change to purple are the *Record* and *Playback* buttons on the recording interface. These change the state of the system to recording and playing respectively.

- **Dark grey** buttons are buttons that have been disabled. The user is not able to select a button that has been disabled.

The three interfaces are described further in the following sections. A black background is used for each interface. When the system is in use, the background will appear as a frame from the Kinect's camera.

### The Home Interface

The Home Interface is the interface that is displayed upon system startup. This interface contains 3 control options which do the following:

- Proceed to the track selection interface.
- Clear any recorded track data.
- Shutdown the system.

By only having three control options on this interface, the user would not be overwhelmed by their first interaction with the system. This design decision was made to improve the engaging factor of the system, and to let the user gain a level of confidence when they use the system for the first time. The layout of the Home Interface is shown in Figure 3.12.

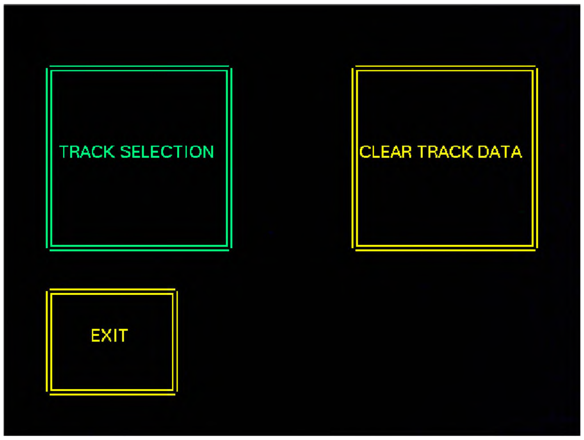


Figure 3.12 : The KinectSound system's Home Interface

### The Track Selection Interface

The track selection interface consists of 8 track controls, one for each track that can be recorded or muted. A user is able to choose which tracks are unmuted/muted and select a single track for recording. The track that is being recorded will be unmuted by default. Each track button contains a text display that informs the user of its muting state, and a red circle to indicate if it has been selected for recording. This interface also contains controls that do the following:

- Set recording state.
- Toggle the muting state.
- Return to the Home Interface.
- Proceed to the Recording Interface.

The controls for selecting a track for recording or toggling the muting state will perform their respective commands on a track that has been selected. A user may select a track in the same manner as they would select any other interface control. As mentioned in this section's introduction, a selected track will have its border shown in light green. When a command is applied to a track, its state will change to *unselected* after the command has been applied.

The navigation controls allow the user to return to the home interface, or proceed to the Recording Interface. The layout of the Track Selection Interface is shown in Figure 3.13.

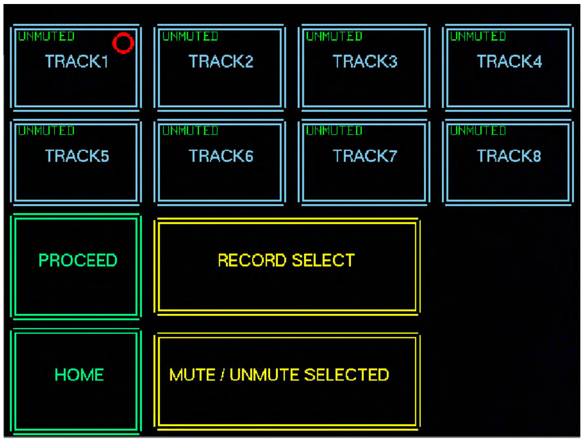


Figure 3.13 : The KinectSound system's Track Selection Interface

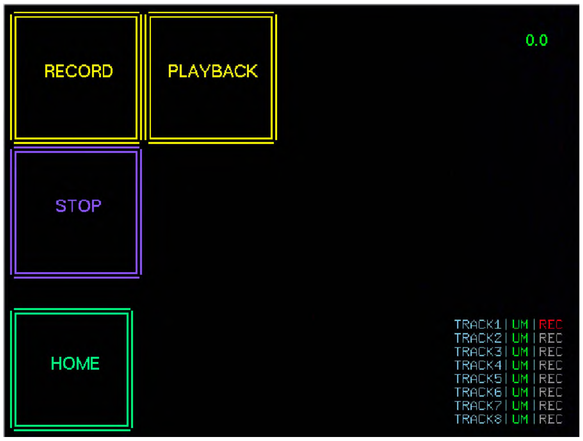
### The Recording Interface

The Recording Interface is the interface that allows the user to record sound localization data for tracks as well as play back any tracks using the recorded data. This interface contains 4 controls which do the following:

- Begin recording localization co-ordinates for the track that has been selected for recording.
- Play the audio piece back using newly recorded data as well as stored data for other tracks.
- Stop the system from recording/playing.
- Return to the Home Interface.

This interface also contains a display which shows the 8 tracks and their muting/recording state and a time which shows how long the track has been recording or playing for. When the user begins recording or playback, all of the controls on the user interface are disabled except the control used to stop the recording/playback. This prevents the user from switching states during recording or playback and also prevents the user from navigating to the Track Selection interface and changing the states of tracks during recording or playback.

The layout of the Recording Interface is shown in Figure 3.14.



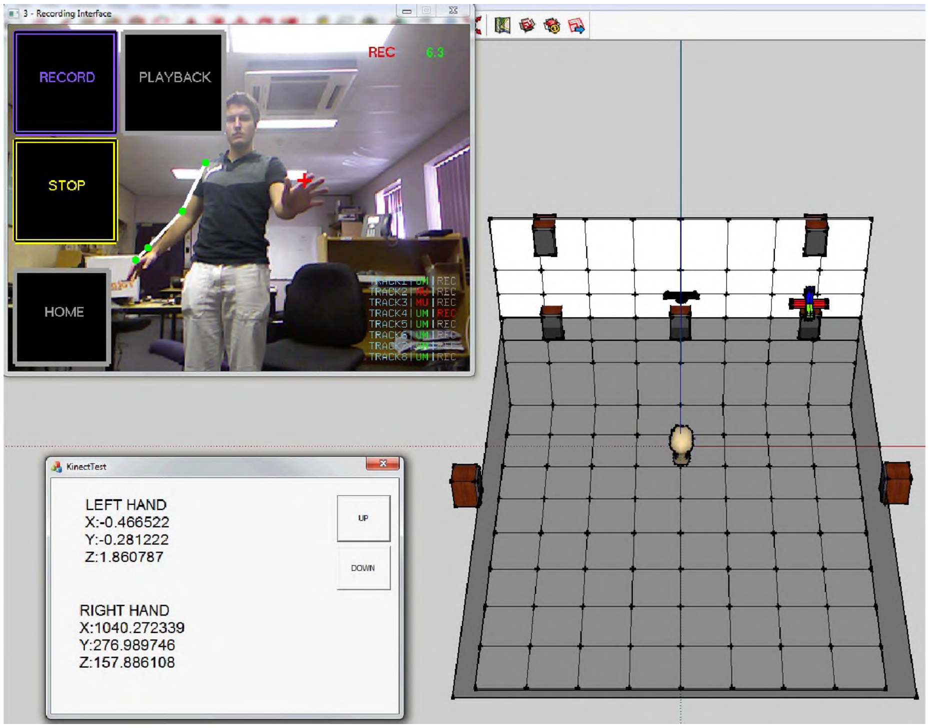
```
TRACK1 | UM | REC
TRACK2 | UM | REC
TRACK3 | UM | REC
TRACK4 | UM | REC
TRACK5 | UM | REC
TRACK6 | UM | REC
TRACK7 | UM | REC
TRACK8 | UM | REC
```

Figure 3.14 : The KinectSound system's Track Recording Interface

Figure 3.14 shows the Recording Interface as it would appear to the user with all of the tracks unmuted and track 1 selected for recording. The timestamp is shown at the top right of the interface. A time of *0.0* shows that the system is neither recording or playing the audio piece.

### 3.5.2 Panning Audio with the KinectSound System

The Kinect provides a dynamic way of panning a sound source around an area with simple hand movements. The KinectSound system is able to pan a sound source in 3D as it treats the users right hand as the sound source and can perform panning that involves all 3 axes in a single movement. It provides feedback about the location of the users hand via a dialog showing the 3D co-ordinates of the hand, as well as Sketchup, a 3D modelling tool. Figure 3.15 shows a user panning using the KinectSound.



```
TRACK1|UM|REC
TRACK2|MU|REC
TRACK3|MU|REC
TRACK4|UM|REC
TRACK5|UM|REC
TRACK6|UM|REC
TRACK7|UM|REC
TRACK8|UM|REC
```

LEFT HAND
X:-0.466522
Y:-0.281222
Z:1.860787

RIGHT HAND
X:1040.272339
Y:276.989746
Z:157.886108

Figure 3.15 : A user panning in 3D with KinectSound

The crosshair in Figure 3.15 shows the location of the user's right hand, which the system treats as the sound source. The dialog box on the left hand side of the figure shows the co-ordinates of the right hand as X, Y, and Z. These co-ordinates represent the lateral displacement, height displacement, and depth of the users right hand in millimeters.

### 3.5.3 A Usability Review of the KinectSound system

In order for the KinectSound system to achieve a high degree of usability, there were several design considerations that took place before it was implemented. The KinectSound system was intended to provide a usable system in which users could intuitively record and playback immersive audio.

The system was made efficient and easy to learn by using large buttons with concise descriptions of their functions. The same button layout is used for commands, selecting tracks, and navigation. Each interface displays relevant details such as track muting states on the track selection interface and the recording time on the recording interface. These details are also made clearly visible to a user. The buttons are grouped into three different interfaces. This in turn allows each interface to be associated with a specific functionality to improve the engaging aspect of the system by not overwhelming the user with controls. The tracks are able to be recorded and played back on the same interface, which allows the user to overwrite any pre-recorded data that they wish to change.

The visual aid of Sketchup complements the dialog display of the 3D co-ordinates. This allows the user to get a 3D representation of the sound source's localization point so as to minimize the chance of localization errors. The error tolerance is aided by the scaling of the user's right hand's co-ordinates. This allows the user to localize the sound source in a more constrained area and pay more attention to the 3D display indicating where the source is located.

The KinectSound system was tested by a sample of users. Each user was required to complete a set of tasks using the KinectSound system and fill in a questionnaire which related to the system's usability. The results and discussion of these tests can be found in Chapter 6.

## 3.6 Chapter Summary

This chapter has described the *Five E's* of usability metrics. These are efficiency, effectiveness, engagement, error tolerance, and ease of learning. If all of these are satisfied (or mostly satisfied), the system will have high *usability.* The rest of the components that make up this chapter are described and summarized in terms of the Five E's in order to gain a measure of their usability

Various audio localization systems were introduced. The interfaces for Spatial Audio Designer (S.A.D) and Iosono Anymix Pro used a 2D view of the listening environment. The 2D interface allowed a user to view 2 axes of the listening environment and localize the sound within these 2 axes. The Haptic Feedback Prototype and The JL Cooper NUAGE surround panner were described. This section also described how these systems used 3D hardware to localize audio. The term *3D hardware* refers to hardware that is able to localize an audio source through the X, Y, and Z axes in a single movement or motion. The Haptic Feedback Prototype used the Novint Falcon for localization while the JL Cooper NUAGE surround panner used a 3D joystick. Due to the requirement of device-free control, several current approaches to device-free control were analyzed and discussed. These approaches were discussed in terms of their capability to provide a solution to a device-free system that enabled 3D control. The device-free approaches that were discussed were the Leap Motion Controller, a standard webcam, and the Windows Kinect. The Kinect was chosen due to its ability to perceive depth as well as provide an accurate skeleton tracking algorithm. The Leap Motion wasn't publicly available when this research began which is why it wasn't used. The KinectSound system was then introduced. This section explains the procedures involved in using the system to record the tracks in an audio piece. The KinectSound system contains three different *User Interfaces* that the user may navigate between to achieve what is desired. A 3D view of the listening environment is also provided by Sketchup. This allows the user to gain a better understanding of where the sound is being located in the 3D environment.

# Chapter 4

# A Distributed Approach to Surround Sound Processing

This chapter introduces current audio networking protocols and explains the Ethernet AVB transport and control mechanisms used by the KinectSound system. It also introduces the distributed processing approach that the KinectSound system Implements.

One of the main goals of the KinectSound system was to distribute a significant portion of the audio processing from the workstation, to “intelligent endpoints”. Figure 4.1 shows the allocation of audio processing in a non-distributed processing system (1), and a distributed processing system (2).

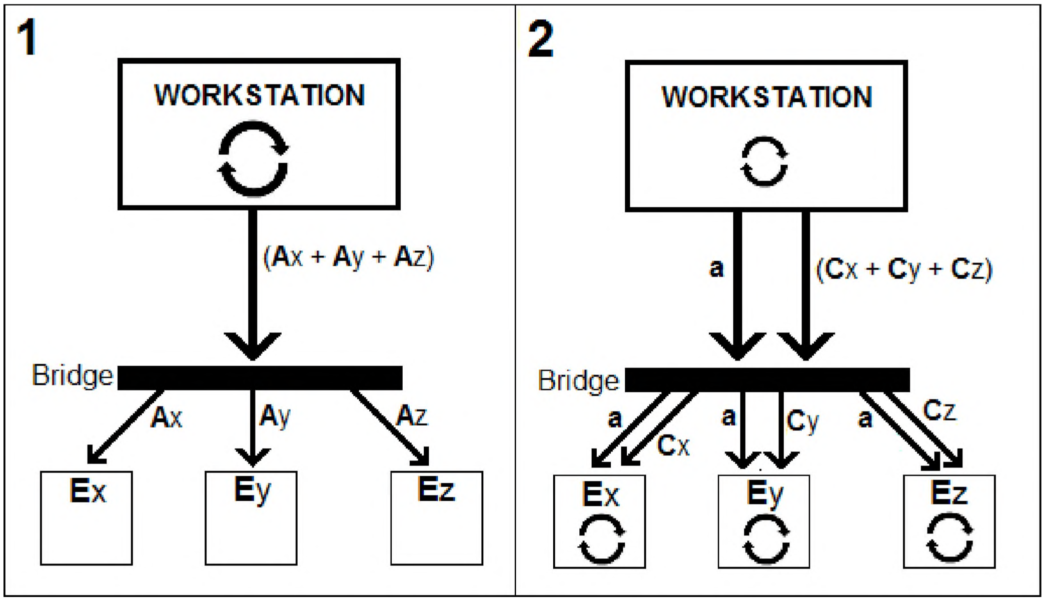


Figure 4.1 : The processing distribution between centralized and decentralized systems.

The figure represents audio processing using circular arrows, and data flow with uppercase and lowercase letters. The uppercase $A$ shown in system 1 represents unicast audio data that is endpoint specific. This data is sent to the bridge and routed to each endpoint. The distributed configuration of system 2 shows multicast audio data as a lowercase $a$, and endpoint specific control packets as an uppercase $C$. Audio mix ratios are extracted from each control packet and then applied to the audio samples. These samples are then mixed and sent to their respective speakers for presentation.

## 4.1 Audio/Video Networking Technologies

Audio/Video (AV) networks are networks of devices that allow real-time streaming of AV data between devices that are connected to the network. Although there are various protocols that enable AV networking, each of them has a similar set of requirements in order to successfully transport AV data. These are:

1. **Synchronization** - When a single AV stream is transported from a source such as a workstation to more than one destination, then the endpoints, which might be powered speakers are required to present the samples for each channel simultaneously. If delay is required, then this should be precisely controllable. All devices on the network must therefore have the same clock. Apart from the issue of simultaneous presentation, if source and destination don't operate from the same clock, audio buffer underflow or overflow may occur.

2. **Bandwidth reservation** - The network should be able to guarantee the resources that are required to transport the AV data from its source to its destination.

3. **Low latency** - AV networks transmit the data in real-time which requires them to have low latency. If there is latency in the network, it must be minimized and deterministic. The most common maximum allowable latency is $5ms$ [65]

There are several existing protocols which satisfy the requirements mentioned above and allow endpoints to connect and communicate with each other. Examples of these systems are described below:

### IEEE 1394 (Firewire)

The IEEE 1394-1995 task group specified an isochronous, as well an asynchronous method of transporting data through buses [66], more commonly known as *Firewire*. The main advantage of Firewire is the speed at which it is able to transfer data. This ranges from 50MB/s (Firewire 400) up to 400MB/s (Firewire S3200). Due to the speed requirement of AV traffic, the speed of Firewire makes this technology suitable for AV transport. The devices on a Firewire network are organized into a logical tree structure according to node ID, with a *root* node at the base of the tree.

Firewire devices are required to obtain an isochronous channel number and bus bandwidth from the *Isochronous Resource Manager* (IRM) before they are able to transmit [67]. When a node has acquired the resources to transmit, it will subtract its bandwidth usage from the IRM's *BANDWIDTH_AVAILABLE* register. The IRM is located in the *Bus Management Layer* of the Firewire communications model. This layer is responsible for bus configuration and management [7]. If more than one node requires to transmit data, the root node decides the order in which nodes are able to send their data via a process known as *network arbitration*. Arbitration happens in $125\mu s$ cycles, which play an important role in the synchronization of devices. The start of each cycle is determined by the device which is the root node. At the beginning of each cycle, the root node broadcasts a *cycle start* packet to each node. Upon a node receiving a cycle start packet, it will [68]:

1. Synchronize its time base to the time in the cycle start packet.
2. Send an arbitration request if it has data to transmit

The root node will grant a single node at a time the permission to transmit its isochronous data over the bus. When a node has finished sending its data, the remaining nodes will re-transmit their arbitration requests after the bus has been idle for $40ns$. When all of the isochronous transfers are complete, the remainder of the $125\mu s$ cycle will be used for asynchronous transfers. Isochronous transfers are able to occupy up to 80% of the bus bandwidth [69].

The major problem with Firewire is the inability to transport data over long distances. The maximum distance a Firewire cable may run is 4.5 meters [7]. Another disadvantage of Firewire is that it does not use existing infrastructures. Many computers do not come with a Firewire port. In this case a Firewire card has to be purchased and installed. The network infrastructure for Firewire networks can be expensive to install due to the cable length limitations. If the distance between two devices is greater than 4.5 meters, a repeater box will have to be used. IEEE 1394c specifies extensions to which allow maximum cable distances of up to 100m over CAT5 cable [70]. Products that utilize the IEEE 1394c are not available, which limits Firewire configurations to 4.5m cable lengths. This in turn limits the capabilities of a surround sound system as distances between speakers are often greater than 4.5m.

### EtherSound

EtherSound is an Ethernet based layer 2 audio transport protocol. EtherSound is able to transmit audio at a rate of 44.1KHz, 48KHz, and multipliers of those with low latency [71].

EtherSound achieves synchronization through a device known as the *primary master*. This is the first device on the network, to which the rest of the devices are referred to as *slaves*. The primary master is responsible for generating the wordclock for the network and propagating it to all the slave devices on the network.

EtherSound devices may be configured in the following topologies:

1. **Daisy-chain** - Devices are connected in a series, and audio packets must propagate through each device to reach a destination. Packets in this configuration can be sent bi-directionally.

2. **Ring** - Similar to the daisy-chain configuration but the last device is connected to the first device. This provides redundant cabling in the case of cable failure.

3. **Star (tree)** - Many devices are connected to a single Ethernet switch. This provides intelligent routing, however the devices are only able to travel uni-

directionally.

Due to the topologies used by EtherSound, the network latency is stable and deterministic, as each device adds 1.5$\mu$s to the overall latency [72]. The latency is also independent of the number of channels in an audio stream [73]. Figure 4.2 shows a configuration of EtherSound devices [74].

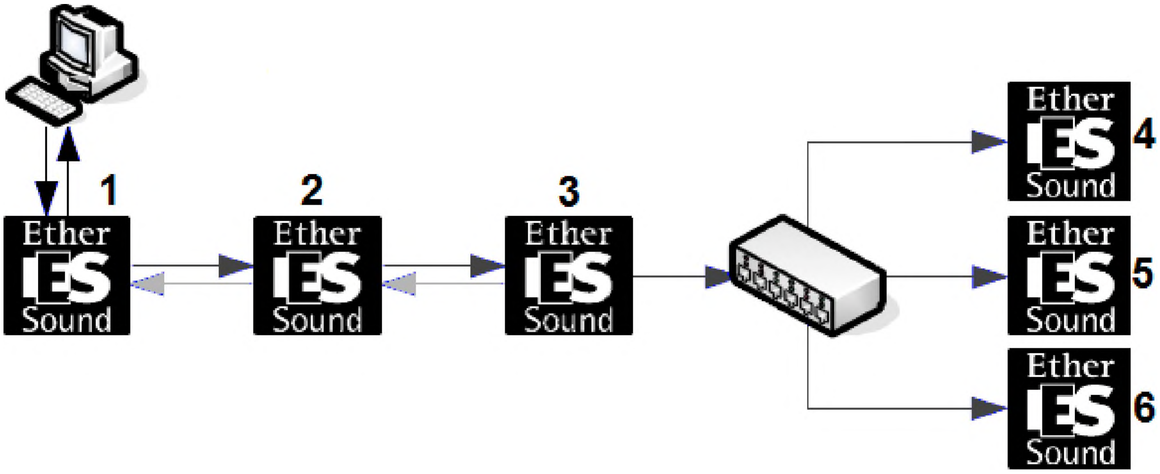


Figure 4.2 : An example of an EtherSound device configuration

The configuration shown in Figure 4.2 shows a hybrid of a daisy-chain topology (devices 1 - 3) and a star topology (devices 4 - 6). If Device 3 was connected to Device 1, the daisy-chain section of this configuration would be transformed into a ring topology. Device 1 would be the primary master on this network as it is the first node on the network. Connections between EtherSound devices can be classed as one of the following:

- **Downlink** - audio data travelling away from the primary master
- **Uplink** - audio data travelling towards the primary master

Both of these connections are shown in Figure 4.2 between Device 1 and Device 3. The rest of the devices have a uni-directional downlink due to the star topology.
A disadvantage of EtherSound is that it requires a dedicated infrastructure, which

removes the opportunity for convergence using existing infrastructures. The protocol used for EtherSound is also proprietary requiring specialized software for remote control and monitoring [75].

### CobraNet

CobraNet is regarded as the first successful audio over Ethernet installation. It provides successful routing in large venues such as airports, conference centers, and stadiums. CobraNet devices provide two network interfaces, the primary interface is operational and the secondary interface assumes functionality when it detects a fault or malfunction in the initial connection [76].

CobraNet networks contain a single device known as a *conductor*. A device will establish itself as the conductor if it is the first powered device to establish itself on the network, with the cabilities of being a conductor. The rest of the devices on the network are referred to as *performers*. One of these performers will become a conductor if the current conductor fails or is removed from the network [77]. The conductor is responsible for synchronization on CobraNet networks. The sampling rates supported by CobraNet networks range from 750Hz to 48KHz. The clock that generates the sampling rate will either be an external clock or a clock that is present in the conductor. The timing information is sent in a beat packet, where each packet denotes the beginning of an isochronous cycle [78].

CobraNet performs bandwidth allocation by broadcasting *reservation packets*. Reservation packets are transmitted as the devices require bandwidth or at a typical rate of 1 per second. These packets are also responsible for remote device monitoring and connection management [77]. CobraNet devices use the *Simple Network Management Protocol* (SNMP) for the transmission of reservation packets [79]. Each CobraNet device contains an SNMP *agent*, which is SNMP software running on a device that has the capability to be managed remotely. CobraNet networks are recommended to have a dedicated infrastructure due to the unpredictability of burst Ethernet traffic on existing networks [76].

The CobraNet protocol is proprietary and although CobraNet has had major success in the industry [78], the use of CobraNet devices continues to diminish with the development of open standard protocols such as Ethernet AVB and AES67.

### Dante

Dante was started by an Australian based company, Audinate, in 2006. It is a layer 3 protocol which transmits audio data using UDP packets. Dante is intended for use on Gigabit Ethernet networks, which provide the required throughput for the audio traffic [80]. As an IP based protocol, Dante provides the advantage of transporting audio via routers. Although Dante claims Ethernet AVB layer 2 interoperability, it has not provided any evidence of achieving this.

Dante devices provide network synchronization via the use of the IEEE 1588 Precision Time Protocol (PTP) [81]. This protocol uses the *Best Master Clock Algorithm* (BMCA) to choose a network clock. This protocol is described later on in this chapter as it is also used for Ethernet AVB synchronization. Dante networks also provide the capability of having an external clock which may be used for network synchronization. The number of channels that can be transmitted through a Dante network depend on the bandwidth that is available. This ranges from 96 channels on a 100Mbps to 1024 channels on a Gigabit network [82]. Dante network traffic is given a higher priority than standard network traffic.

Dante network switches have to be configured and managed for the use of Dante devices. The Dante protocol requires specialized software from Audinate for controlling their devices as their protocol is proprietary.

### Ethernet AVB

The Institute of Electrical and Electronics Engineers (IEEE) Audio Video Bridging task group of the 802.1 standards committee defined a set of standards known as Audio Video Bridging (AVB). AVB networks enhance the current IEEE 802 architecture in a number of ways which allow the transport of AV data over existing Ethernet networks. The AVB standards are able to guarantee quality of service for time sensitive AV data [83] which is prioritized and not affected by external data. AVB *bridges* are required to route the data between a source and a destination. These perform the same functions as network switches in Ethernet networks, however, they provide the mechanisms which allow the routing of time sensitive AV data.

Ethernet AVB can be described as a combination of protocols that co-operate in order to fulfill the requirements for a fully functional audio distribution network. These protocols are:

- **802.1Qat** - The Stream Reservation Protocol (SRP) provides admission control and is used by AVB for bandwidth reservation.
- **802.1Qav** - Traffic shaping algorithms that provide queuing and forwarding enhancements for AVB network traffic.
- **802.1As** - A tightly-constrained profile of the IEEE 1588 protocol, which provides synchronization for IEEE 1722 devices.

A major benefit of AVB is that there is an open standard for the protocol, which allows it to be easily incorporated into audio systems. AVB traffic uses standard Ethernet cabling, which provides opportunity for convergence with existing infrastructure.

Ethernet AVB provides a solution to a custom built distributed surround sound system and is able to satisfy the requirements for AV transport. Due to the open protocol, endpoints are able to be remotely controlled by a custom built system, as long as the control messages follow the predefined AVB standard.

### AES67

AES67 is a layer 3 protocol specified by the Audio Engineering Society which allows low latency transport of professional audio [84]. This refers to audio that is sampled at 44.1KHz and higher. AES 67 is designed to allow interoperability between layer 3 transport protocols.

Devices implementing AES67 synchronize their clocks to a single device on the network using the IEEE 1588-2008 precision time protocol [84]. The Quality of Service

(QoS) for AES67 networks is achieved through the use of the Differentiated Services (DiffServ) architecture described in RFC 2474 [85]. This architecture provides mechanisms for classifying and managing networking traffic of different priorities by the use of a Differentiated Services Code Point (DSCP) field in the packet header [85]. Network packets with varying DSCP values will be treated differently at the switches and endpoints according to their DSCP values.

The standard for AES67 is open and can be downloaded from the AES website [86]. An open standard provides the opportunity of integrating this protocol into an end user developed sound system. This protocol was not chosen for the KinectSound system as it was only standardized in September 2013, 8 months after the start of the KinectSound system's development.

## 4.2 Ethernet AVB in Depth

Devices on an AVB network can be classified into two main categories which are:

- **Endpoints** - Endpoints are devices on the network that are responsible for transmitting and receiving audio data to and from the network respectively.
- **Bridges** - Bridges are devices on the network that provide routing of audio packets between endpoints or other bridges. They contain mechanisms for queuing and forwarding of AV data.

Devices that transmit audio data onto a network are known as *Talkers*, while devices receiving audio data are known as *Listeners*. Quality of service in AVB networks is enabled by three different IEEE 802.1 sub protocols. Each protocol works towards satisfying the synchronization, bandwidth reservation, and latency requirements that were mentioned earlier in the chapter. Each of these protocols is described in further detail below.

### 4.2.1 IEEE 802.1 Qat

The 802.1 Qat protocol defines the mechanisms which reserve network bandwidth to guarantee Quality of Service (QoS). The Stream Reservation Protocol (SRP) utilizes three signaling protocols to guarantee network resource from a single talker to one or more listeners (end-to-end). These protocols are [87]:

1. **Multiple VLAN Registration Protocol (MVRP)** - One of the mechanisms used to guarantee QoS is the transmission of AV data over Virtual LANs (VLANs). MVRP provides a dynamic way of assigning unique identifiers to VLANs that are assigned to potential AV streams and registering bridge ports that are being used for these streams.

2. **Multiple MAC Registration Protocol (MMRP)** - MMRP provides the ability to group together a number of devices. This provides the ability to confine specific multicast messages to groups of devices, thus reducing unnecessary traffic on the network.

3. **Multiple Stream Registration Protocol (MSRP)** - This provides an indication of whether there is sufficient bandwidth to transmit a stream from a talker to a listener. This involves determining:
   - Which listeners will be receiving the stream.
   - Which routes across the network are capable of supporting the stream's requirements between the talker and prospective listeners.

Figure 4.3 shows the events that occur when a talker advertises on a small AVB network. The network consists of a talker (T), and three listeners (L1, L2, and L3) networked across two AVB bridges (B1 and B2).

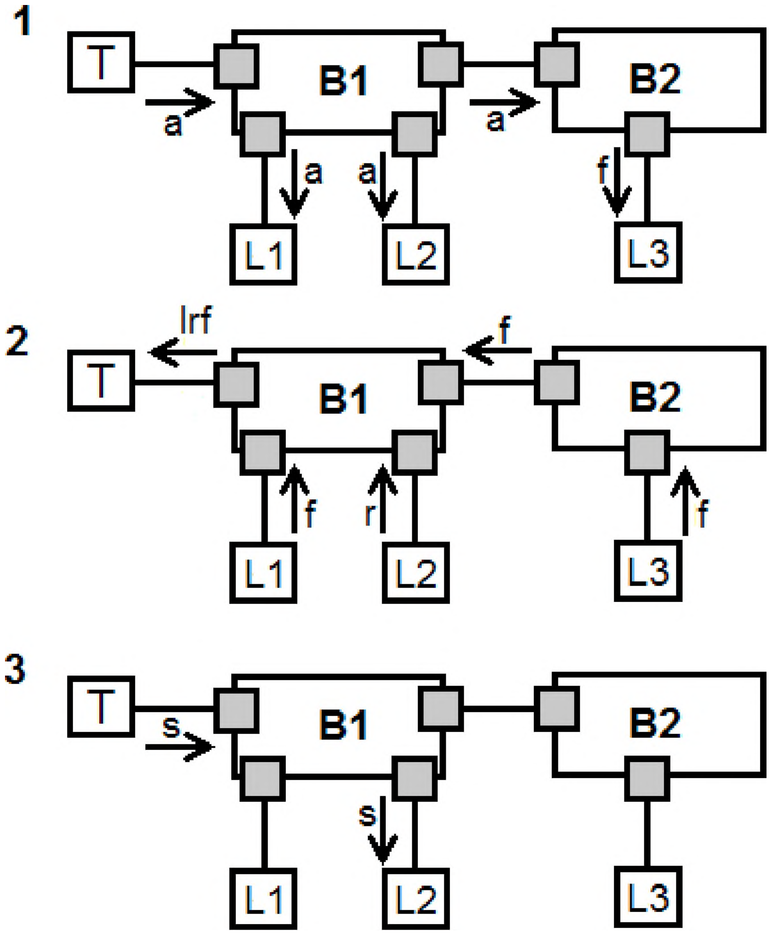


Figure 4.3 : MSRP talker advertisement with responding listeners

The following events occur in Figure 4.3:

1. Talker advertising

   - The talker broadcasts an advertise message (a) to all the listeners on the network.

   - Listeners $L1$ and $L2$ receive the message. Bridge $B2$ is incapable of transmitting a stream, so it forwards a *Talker Failed* message (f) to the listener.

2. Listeners responding

   - Listeners $L1$ and $L3$ respond with *Listener Asking Failed* messages (f). These indicate that they are interested in receiving the stream but there are insufficient

resources available.
- Listener $L2$ responds with a *Listener Ready* message (r). This indicates that the listener is interested in receiving the stream and that there are sufficient resources to transmit a stream between the talker and listener.
- The talker receives a *Listener Ready Failed* (lrf) message from the bridge. This message indicates that there are at least 2 listeners requesting the stream, and at least one capable of receiving the stream [87].

3. Stream reservation
   - The talker begins transmitting a stream onto the network.
   - Bridges that are aware of listeners requesting the stream will allocate resources accordingly and forward the AV data to the respective listeners.

AVB talkers that wish to transmit data across the network must make an MSRP declaration. This declaration contains the bandwidth requirements in order to successfully transmit a stream of AV data between two points in the network, until the stream is terminated. The declaration broadcasts to all the listeners on the network, which will respond by giving their ability (or inability) to receive a stream. The inability to receive a stream may be determined by limitations such as the listener already receiving it's maximum number of streams, or insufficient network resources. The talker will then begin transmission if the required resources are available.
802.1 Qat satisfies the AV requirement of *bandwidth reservation*, as described earlier in the chapter.

### 4.2.2 IEEE 802.1 Qav

The 802.1 Qav protocol defines the queuing and forwarding enhancements for AVB endpoints and bridges. The endpoints and bridges are required to support two different types of traffic classes, of which one must be MSRP compatible [80]. The MSRP compatible traffic class is required in order to treat the time-sensitive AVTP traffic as *High Priority* traffic. The other class is used to transmit any *Low Priority* network traffic that is not time-sensitive or loss-sensitive.

The high priority queues in the endpoints and bridges are drained using a *Credit Based Shaper Algorithm* [88]. Queues that support this algorithm have the following attributes:

- **Credit** - The *transmission credit* that is available to each port. If the port's *transmitting* attribute is false, this value will constantly increase if there are packets in the queue and reset to zero if the port is idle with a positive credit value. The rate at which the *credit* slope increases is determined by the *idle slope* attribute. If the port's *transmitting* attribute is true, the *credit* will constantly decrease. The rate at which *credit* decreases is determined by the *send slope* attribute.
- **Transmitting** - Set to true when the current port is transmitting a packet, set to false when the port is idle.
- **Transmission rate** - The rate at which the port is able to transmit data. This is measured in bits per second.
- **Idle Slope** - The rate at which a port gains *credit* when it is not transmitting and has packets in the queue.
- **Send Slope** - The rate at which a ports *credit* decreases when it is transmitting a packet.
- **Transmit Allowed** - Determines whether the port is allowed to transmit a packet or not. This is true when the *credit* is greater than or equal to 0.

Shown in Figure 4.4 is an example of the credit based shaper algorithm being used in a port with conflicting traffic. Conflicting traffic may be the result of events such as:

1. Other ports with a higher priority currently transmitting traffic.
2. Ports of the same priority with a higher credit value currently transmitting traffic.

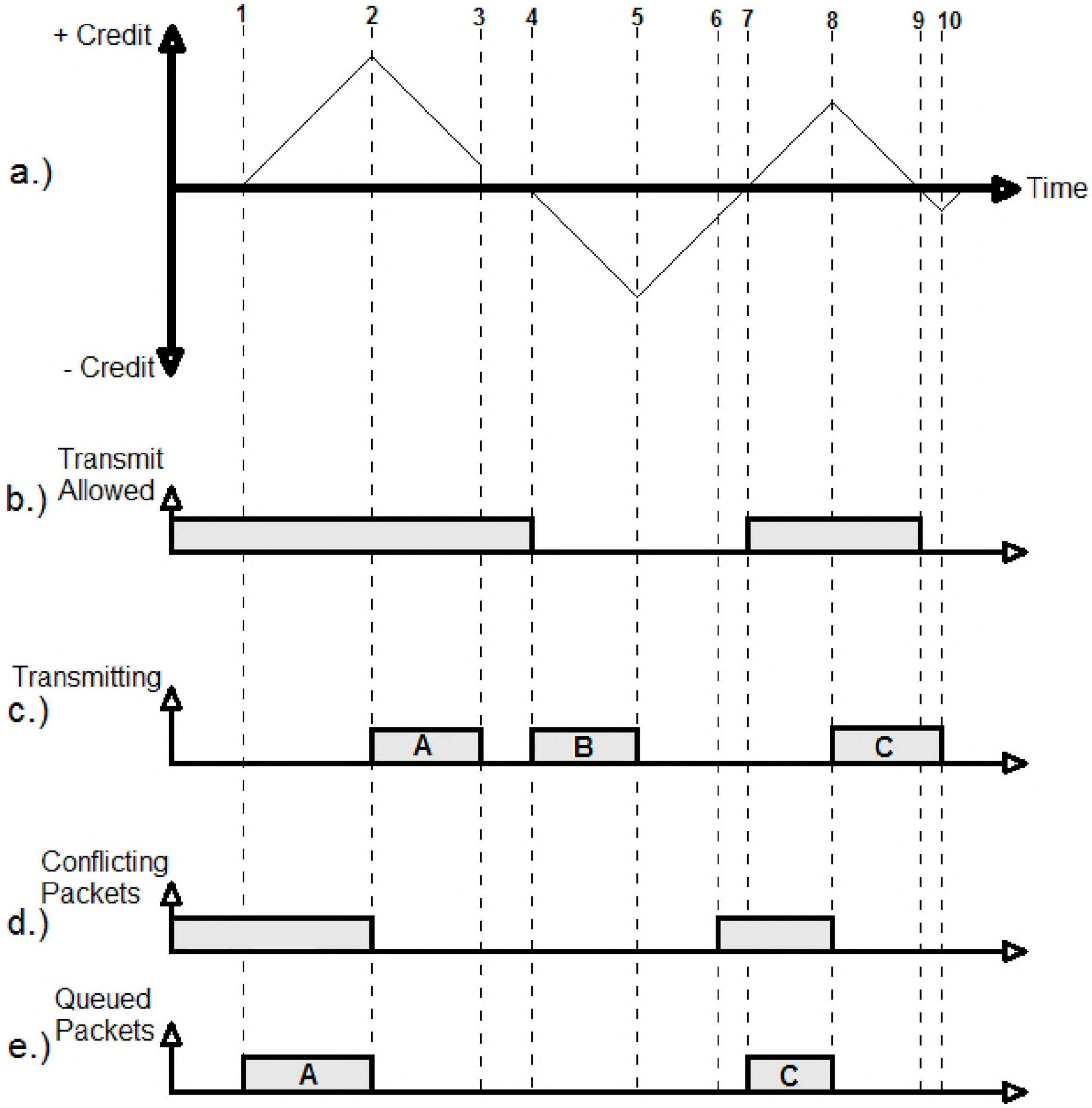


Figure 4.4 : Graphical layout of the attributes of the Credit Based Shaper Algorithm

Figure 4.4 shows 10 different numbered events that occur throughout the given time frame. Each of these events is described below:

1. Packet A is ready for transmission (e), but is queued because of conflicting traffic (d) elsewhere in the device/switch. Due to a positive credit (credit = 0), and the *Transmit Allowed* attribute being true, the port begins to gain *credit*

at a rate given by the *Idle Slope* (a).

2. There is no more conflicting traffic and the port has a positive credit, therefore Packet A begins transmitting (c). The credit slowly decreases at a rate given by the *Send Slope* (a).

3. Packet A is finished transmitting and the credit is still positive. As mentioned above, a port cannot be idle and have credit greater than 0, so the credit is reset to 0.

4. Packet B is ready for transmission and credit is equal to 0 so Packet B begins to transmit. This action decreases the credit to a negative value which in turn sets the *Transmit Allowed* attribute to false.

5. Packet B has finished transmitting and the credit is negative. This port will be unable to transmit any further packets until the credit has reached a positive value again, so credit will begin to constantly increase until it reaches 0.

6. There is conflicting traffic elsewhere on the device/switch (d).

7. Packet C is ready for transmission, the port is allowed to transmit but it cannot due to the conflicting traffic, so it is queued (e). Credit begins to increase again.

8. There is no conflicting traffic, the port is ready to transmit and has a positive credit, Packet C begins transmission (c).

9. Credit falls below zero, so the *Transmit Allowed* attribute will be set to false. This will continue until Packet C is finished transmitting.

10. Packet C is finished transmitting, credit starts to increase again until it reaches 0 (a).

802.1 Qav ensures that there is minimal traffic latency, and also guarantees the bandwidth on an existing LAN infrastructure in order to support the transport of AVTP traffic. This satisfies the AV requirement of *minimal latency* on a network.

### 4.2.3 IEEE 802.1 AS

The 802.1 AS protocol is responsible for providing a time domain in order to achieve synchronization and syntonization within the network. Synchronization refers to all the devices on the network being time synchronized with each other, in order to present audio samples at the same time. This is done by the use of the generalized Precision Time Protocol (gPTP) defined in the IEEE 1588 specification [89]. This protocol selects a device known as the *Grandmaster* from the devices on the network, which contains the base time information for the network. The network is able to select a grandmaster by using the Best Master Clock Algorithm (BMCA) [90], which selects a device with the “best” clock. The grandmaster is chosen on initialization. The BMCA works as follows:

1. All devices that are capable of being a grandmaster broadcast *announce* messages containing parameters of their own clock.

2. When a device receives an announce packet, it compares its own clock parameters to the clock parameters in the announce message.

3. If the clock parameters in the announce message indicate a better quality clock, it will move into a *slave* state, with the *master* node being the device that transmitted the message.

4. All of the devices on the network will maintain a record of the most recent announce message from the current grandmaster.

5. The BMCA is able to select a new grandmaster if the devices haven’t received an announce message from the current grandmaster for a period of time.

In order to satisfy the requirements for synchronization, any device on the network is required to know the grandmaster’s time in relation to its time, and its time in relation to the grandmaster. This time needs to be accurate due to the time-sensitivity of the traffic. This is easily achievable as the delays within local area networks (LAN) are often fixed [90].

Syntonization refers to the encoding and decoding of digital data at the same given rates [90]. Audio data is encoded at a specific rate by the talker and then transmitted onto the network. The listeners will receive the packets from the stream and decode the data into a presentable signal. If the encoding rate and decoding rate differ, the final sound will have glitches due to buffer underflow or overflow. The grandmaster plays an important role in the syntonisation of a network as each device can compare their clock against the grandmasters in order to derive a sample frequency.

## 4.3 IEEE 1722.1 - AVDECC

Ethernet AVB provides a control protocol which is known as the IEEE 1722.1 protocol, or more commonly referred to as *AVDECC* [91]. The name *AVDECC* stands for Audio\Video Discovery, Enumeration, Connection management, and Control - which are the responsibilities of this protocol for devices on AVB networks. Devices that make use of the AVDECC protocol are known as AVDECC devices and are thus capable of transmitting and receiving AVDECC messages as well as the AVTP traffic within the network. Each AVDECC device is assigned a Globally Unique Identifier (GUID). A GUID is derived using the MAC address of the device.

Apart from talkers and listeners (described in Section 4.2), AVDECC provides another type of device which is a *controller*. Given below is a description of each device in the context of AVDECC.

### 4.3.1 AVDECC Talkers

An AVDECC talker is an AVDECC entity that is capable of transmitting one or more streams onto the AVB network. A stream is set up via an AVDECC Connection Management Protocol (ACMP) message, which is sent from an AVDECC controller [91].

### 4.3.2 AVDECC Listeners

An AVDECC listener is an entity which is the recipient of an AVTP audio stream. The listener must be capable of receiving the audio data, decoding it, and presenting it as an audible signal. The listener must be able to respond to connect and disconnect messages from the AVDECC controller [91].

### 4.3.3 AVDECC Controllers

An AVDECC controller is an entity which initiates message and audio data exchange between other devices on the network. Any AVDECC entity is able to be a controller, as long as it supports the receiving and transmission of connection management and control messages. A network is able to have more than one controller, however, the *LOCK_ENTITY* command must be used in order to perform atomic operations. When an AVDECC device receives this command, it will only accept control commands from the controller that locked it [91]. The entity must be unlocked by the controller in order to be controlled by other controllers. If this is not done the lock will expire after 1 minute.
An AVDECC controller may be contained within a talker or listener, as long as it fulfills the above-mentioned requirements.

The AVDECC protocol is broken up into three separate protocols. These are known as:

- AVDECC Discovery Protocol (ADP)
- AVDECC Connection Management Protocol (ACMP)
- AVDECC Enumeration and Control Protocol (AECP)

AVDECC packets use a similar header to the AVBTP control packets however, some of the fields are redefined depending on the AVDECC sub-protocol being used. The following sections discuss each AVDECC device type in further detail, as well as the

sub-protocols that make up AVDECC [91].
The functionality of AVDECC is dependent on 3 sub protocols, these are:

1. **ADP** - The AVDECC discovery protocol which performs AVDECC discovery. This is the process by which AVDECC controllers identify all the AVDECC devices connected to the AVB network. This protocol is able to identify devices as they are added and removed from the network

2. **ACMP** - The AVDECC Connection Management protocol is responsible for creating and terminating connections between two AVDECC devices. These connections refer to audio streams between an AVDECC talker and an AVDECC listener.

3. **AECP** - The AVDECC Enumeration and Control Protocol deals with the capabilities, formats, and controls of AVDECC devices [91]. Enumeration is aimed at discovering these attributes for each device, while control deals with the manipulation of these attributes.

## 4.4 The KinectSound system configuration

The network configuration of the KinectSound system uses Ethernet AVB for transporting audio traffic. Although both AES67 and Ethernet AVB satisfy all of the requirements, AES67 was not published at the time when this project was started. The endpoints in the KinectSound configuration use AVDECC for discovery, connection management, and control. Figure 4.5a shows the Ethernet configuration of the KinectSound system. Figure 4.5b shows the system architecture of the KinectSound System. The following sections describe each component in Figure 4.5 in further detail.

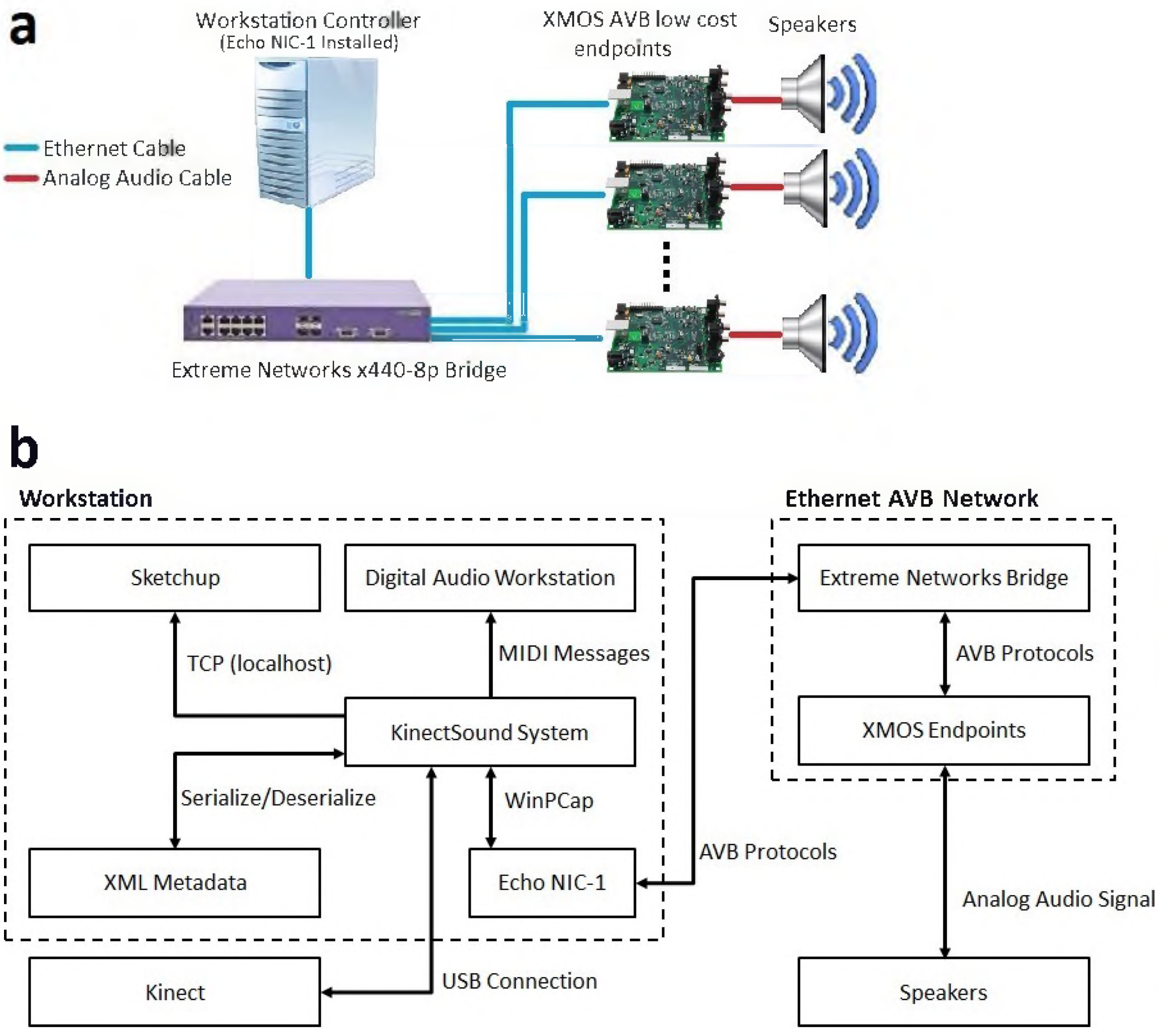


Figure 4.5 : The Ethernet Layout and System Architecture of the KinectSound system

### 4.4.1 Streamware Echo NIC-1 Network Adaptor

The Streamware NIC-1 Network Adaptor (Echo Card) serves as both a network adaptor, as well as a sound card for AVB [6]. Shown in figure 4.6 is an Echo Card.

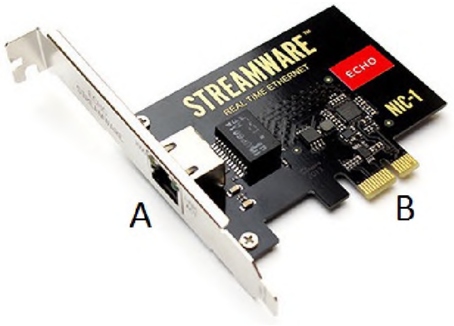


Figure 4.6 : An Echo NIC-1 Network Adaptor [6].

An Echo Card was fitted into the workstation controller as a means of providing an AVB/AVDECC link between the Ethernet devices and the workstation. The Echo Card will automatically begin to discover IEEE 1722.1 devices once it's drivers have been installed and initialized. The network adaptor requires an empty PCI slot in the workstation that it may be plugged into [6]. The PCI plug on the Echo card is shown as 'B' in Figure 4.6. Part 'A' shown in Figure 4.6 is the Ethernet interface on the Echo Card, which allows it to provide a connection between the workstation and the AVB network.

The Echo Card contains an ASIO 2.3 sound card driver which supports four streams of up to 16 channels on each stream [6]. ASIO is an API from Steinberg [92] which provides a link between software applications and sound cards.

### 4.4.2 Extreme Networks Ethernet Bridge

The Ethernet bridge used in the KinectSound system configuration is an Extreme Networks Summit x440-8p. This is a 12 port Ethernet bridge, however ports 9-12 are unpopulated and were therefore not usable with the current configuration. Ports 1-8 contain RJ45 Ethernet jacks, which allow the Echo Card to connect and control a maximum of 7 other devices[1]. This bridge is also able to provide Power over Ethernet (PoE) to devices that are PoE enabled. The bridge's AVB capability is disabled by default and required activation before it provided AVB support. This can be done by

[1] This figure can be enhanced by using further bridges

connecting to the bridge using the Serial-to-Ethernet cable connection provided with the switch, and using terminal emulator software. A user is then able to log in and access the Xtreme Operating System (XOS) where configuration can be done, and changes can be made.

### 4.4.3 XMOS AVB Low-cost Audio Endpoints

The XMOS Low-cost Audio Endpoints shown in Figure 4.5 provide an interface between the digital audio being transported over an AVB network, and an analog signal input/output. These evaluation devices were jointly developed by XMOS and Atterotech [93]. A photo of the AVB Low-cost Audio Endpoint is shown in Figure 4.7.

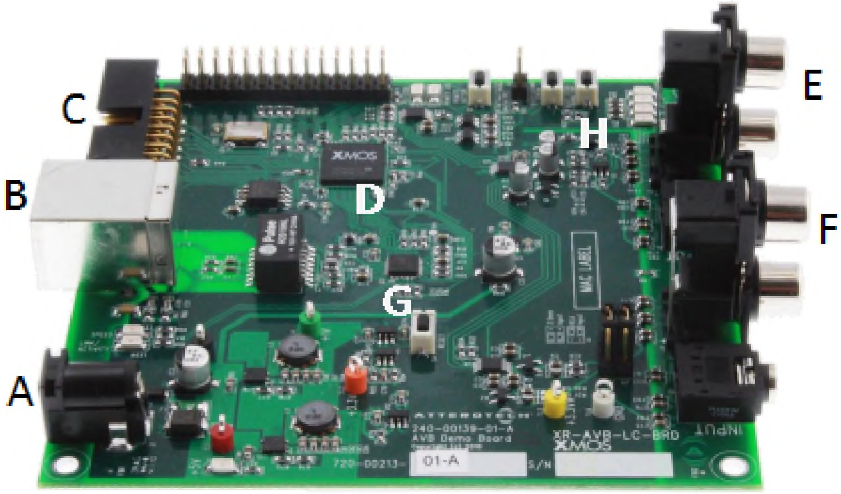


Figure 4.7 : The layout of an XMOS low-cost endpoint.

The different components of an XMOS AVB Low-cost endpoint labeled in Figure 4.7 are as follows:

A. **Power supply** - Socket for a 5V DC power adaptor which is used to power the XMOS endpoint. These devices require external power and do not support power over Ethernet.

B. **RJ45 Ethernet Port** - This port provides an interface between the XMOS endpoint and an AVB network.

C. **xTAG-2 Programming Slot** - Slot into which an xTAG-2 programmer card may fit. The programmer card provides a USB interface, allowing a user to run, flash, and time XC code on the endpoint.

D. **xCORE multicore microprocessor** - The XMOS microprocessor used on this endpoint is an *XCORE XS1-L16-128 Multicore Microprocessor* [93]. This microprocessor is capable of running at 800 - 1000 MIPS. It has 16 concurrent processing cores, each supporting high performance DSP [93]. The processing cores are distributed across a number of processing *tiles* within the microprocessor. Each tile on the XS1 microprocessor provides at least 1 logical core, a scheduler, and 64KB of RAM [80].

   Inter-process communication is enabled through the use of *channels*. Channels provide a lossless point-to-point communication mechanism between concurrent processes. A process is required to provide a *channel end* for each channel, which allows it to send and receive data through a specified channel.

E. **Output audio jacks** - The output jacks may be used to connect a speaker to an endpoint when the endpoint is defined as a listener. These jacks contain a 3.5mm stereo jack, as well as a set of RCA stereo connector jacks.

F. **Input audio jacks** - The input jacks contain the same set of jacks as the output jacks, however these are used for input devices such as microphones, media players, and any other devices capable of transmitting audio. These are used when the endpoint is defined as a talker.

G. **Reset Button** - This button is used to power cycle the board, allowing the user to reset various settings and return the microprocessor back to its original state.

H. **GPIO Buttons and LEDs** - The general purpose input/output on this endpoint is composed of 4 LED lights, and 3 buttons.

### 4.4.4 The Flow of Data

With the system doing the audio mixing at the endpoints by utilizing the xCORE microprocessor, it has successfully reallocated a significant amount of processing, typically done at the workstation, to each endpoint. The workstation is also required to send endpoint-specific control packets from the KinectSound system to each endpoint. These packets contain the mix levels for each channel, at each endpoint. The KinectSound system uses the packet capture API (PCap) to send layer 2 control packets [94]. The XMOS endpoints mix the audio according to the mix levels specified by the control packets.

Figure 4.8 shows the data flow of the audio data and control data, from the workstation to the endpoints.

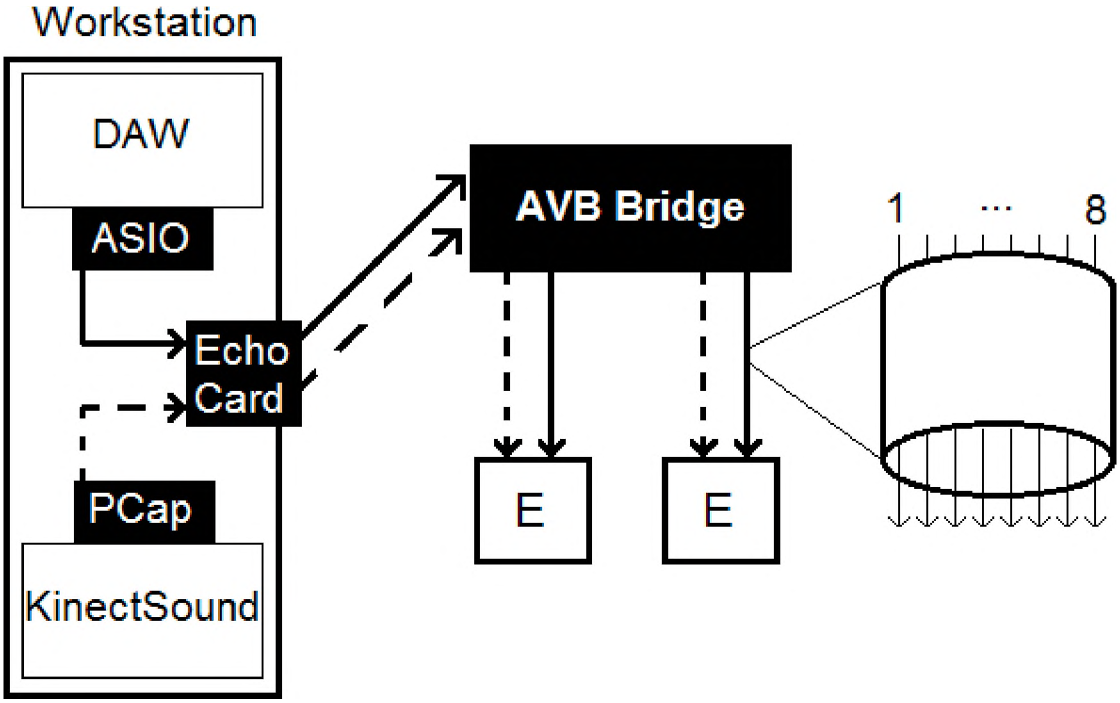


Figure 4.8 : The flow of data in the KinectSound configuration.

The audio data is shown as a solid line, which is a single stream that is multicast to each endpoint. A single audio stream consists of 8 channels, as shown on the right hand side of Figure 4.8. The dashed lines show the control data, which is sent from the KinectSound system. Ethernet AVB transports audio by using a protocol known as the AVB Transport Protocol (AVBTP) defined in the IEEE 1722 standard [83]. This standard defines the packet layout for both AVBTP data packets, and AVBTP control packets. The audio data is contained in a Common Isochronous Packet (CIP) [95], which is a format originally developed for Firewire streams. This is then encapsulated in an AVBTP data unit packet and has an Ethernet header [83]. The layout for an audio data packet is shown in Figure 4.9.

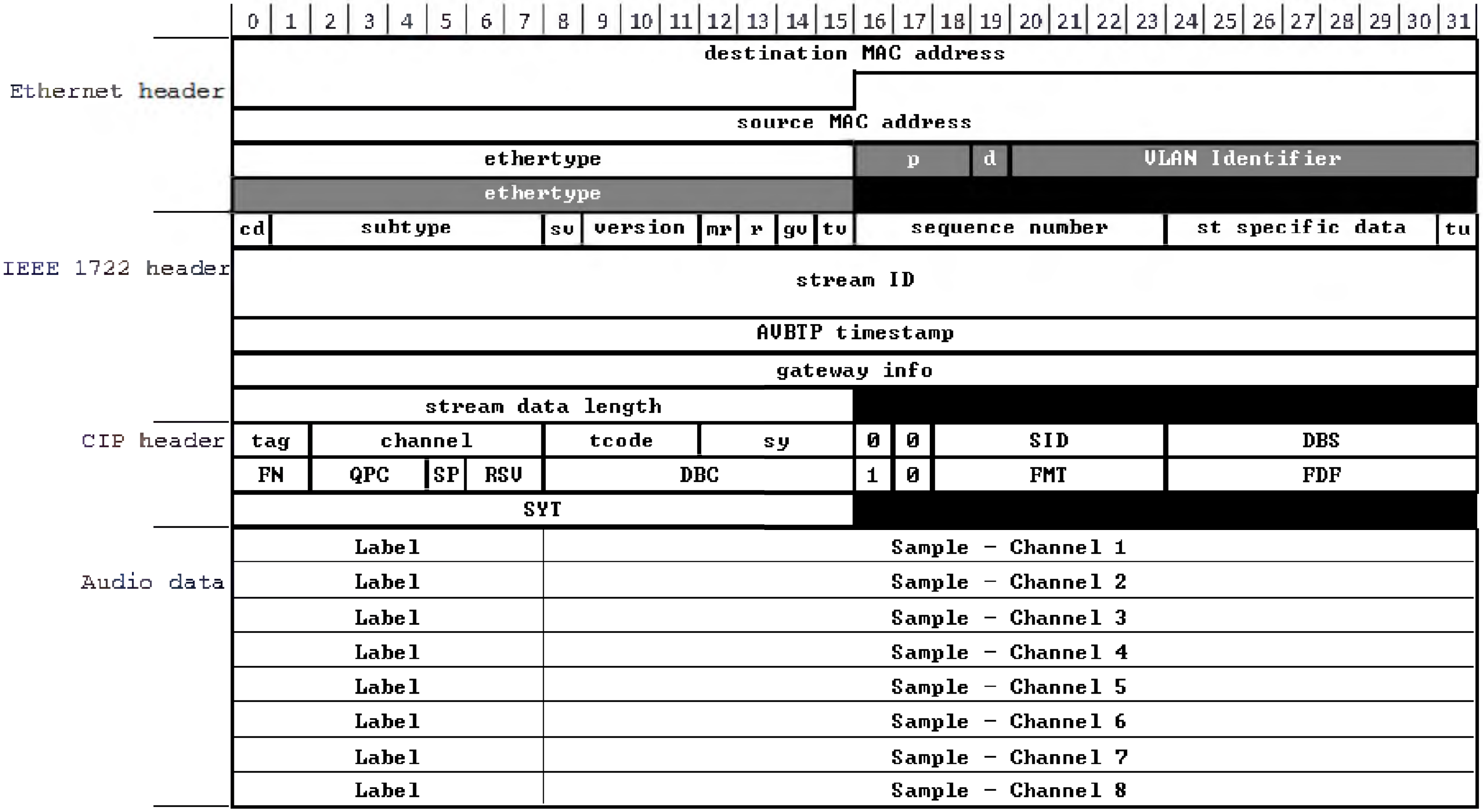


Figure 4.9 : The layout of an Audio Data packet.

Although only 1 set of samples is shown in Figure 4.9 for the 8 channels, each packet will contain a number of sets of samples, the number depending on the sample rate - 6 samples for 48KHz. For a stream of audio data that is sampled at 48KHz, the packet throughput would be $\frac{48000}{6} = 8000$ packets per second.
Header fields in Figure 4.9 that are important in the context of the KinectSound system are listed below:

- **Destination MAC address** - MAC address of the device which will receive the packet. In this case, it will be a multicast address.

- **Source MAC address** - MAC address of the device that sent the packet. This will be the Echo network adaptor as it is the only talker on this network

- **Ethertype** - Indicates the type of protocol that in encapsulated within the Ethernet packet.

The two fields that are highlighted in grey represent optional fields. These two fields will occur if the first *Ethertype* field has a value of *0*x*8100*. This value specifies that the packet is being sent on a Virtual LAN. The relevant fields within the IEEE 1722 header are described below:

- **p** - The *Priority* of the packet. This is in the range of 0 - 7 and can be used to prioritize different classes of network traffic.

- **VLAN Identifier** - Identifies which VLAN the packet belongs to. Values *0*x*0000* and *0*x*FFFF* are reserved, any other values within this range may be used. All AVB packets are VLAN tagged, as this allows them to be treated as priority traffic.

- **sv** - The *Stream ID Validation* field contains a 0 or 1 depending on whether the *Stream ID* contains a valid Stream ID (1), or if it does not contain a valid ID (0).

- **tv** - *Timestamp Validation* describes whether the *AVBTP Timestamp* field contains a valid timestamp(1), or if it should be ignored(0).

- **stream ID** - The *Stream Identification* field contains a 64 bit number associated with the AVBTP packet which uniquely identifies a single stream. End-stations which receive a packet that has a *Stream Validation* field set to 0 will simply ignore this entire field.

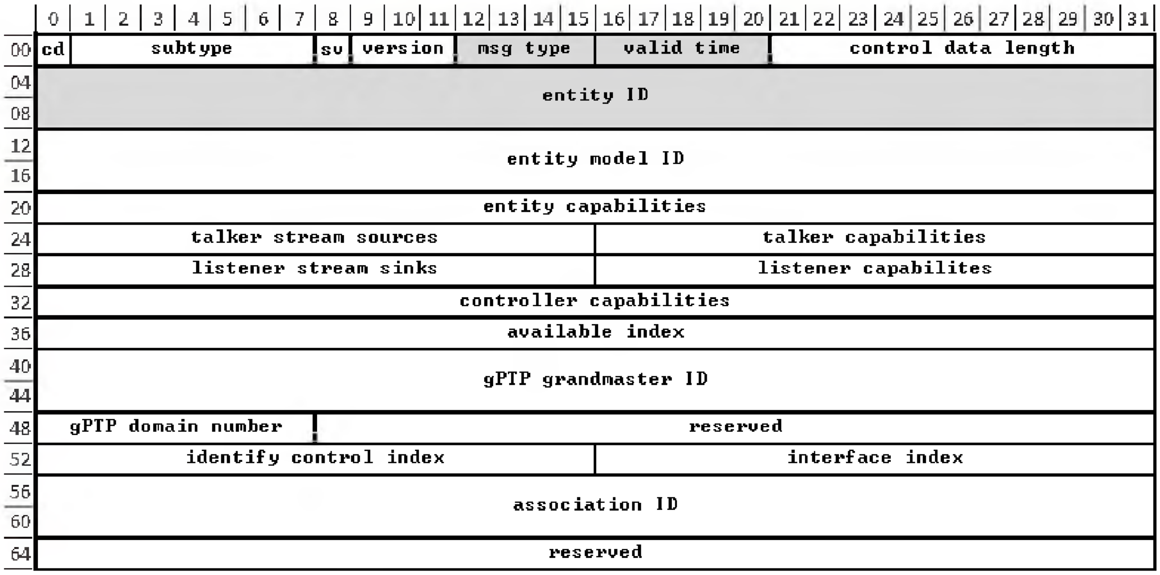
- **AVBTP timestamp** - Contains the presentation time of the samples within the AVBTP packet if the *Timestamp Validation* field is set to true. This is a value given in nanoseconds, and has a maximum value of $(2^{32} - 1)$ns.

### 4.4.5 AVDECC in the KinectSound system

As the KinectSound uses the Ethernet AVB implementation, it requires the use of AVDECC to control the endpoints in the system. AVDECC messages use the standard AVBTP control message header , but redefine certain fields depending on which AVDECC messages are sent. The redefined header is shown as the first 12 bytes of Figure 4.10, Figure 4.11, and Figure 4.12. For each figure, the redefined fields are shown in grey. Following each figure is a description of the fields that are relevant to the KinectSound system. The way in which the KinectSound system uses each AVDECC sub protocol is described in the following subsections.

#### Discovery

AVDECC discovery alerts the controllers when a device is added or removed from the network. This is done via *ADP messages.* ADP messages are sent from each device at regular intervals, and “advertise” that they are on the network. This enables the controller that is doing the discovery to know which devices are currently connected to the network. The Echo network adaptor performs AVDECC discovery and the devices can be seen in the Echo Streamware software display. This shows all the devices on the network and adds and removes devices as they are added or removed from the network. The layout of an ADP packet is shown in Figure 4.10



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | 0 | 1–7 | 8 | 9–11 | 12–15 | 16–20 | 21–31 |
| 00 | cd | subtype | sv | version | msg type | valid time | control data length |
| 04<br>08 | entity ID | | | | | | |
| 12<br>16 | entity model ID | | | | | | |
| 20 | entity capabilities | | | | | | |
| 24 | talker stream sources | | | | talker capabilities | | |
| 28 | listener stream sinks | | | | listener capabilites | | |
| 32 | controller capabilities | | | | | | |
| 36 | available index | | | | | | |
| 40<br>44 | gPTP grandmaster ID | | | | | | |
| 48 | gPTP domain number | | reserved | | | | |
| 52 | identify control index | | | | interface index | | |
| 56<br>60 | association ID | | | | | | |
| 64 | reserved | | | | | | |

Figure 4.10 : The layout of an AVDECC Discovery Protocol Packet

Important fields in the ADP packets that relate to the KinectSound system are:

- **valid time** - The valid time of an ADP message indicates how long the controller will recognize it as being connected to the network. If this time for a specific device expires before another ADP message is received, the controller will assume this device has been disconnected from the network. An AVDECC device broadcasts discovery messages at roughly $\frac{1}{4}$ of the valid time of the messages. There is a trade off in this situation between the timely detection of a device being removed from the network, and the amount of traffic generated as a result of the ADP messages.

- **entity ID** - The Globally Unique ID identifies the device that has sent the ADP message.

- **talker capabilities** - Describes the capabilities of the device as a talker. The talker capabilities of the endpoints in the KinectSound system were disabled to ensure that there is only one talker, which is the Echo network interface.

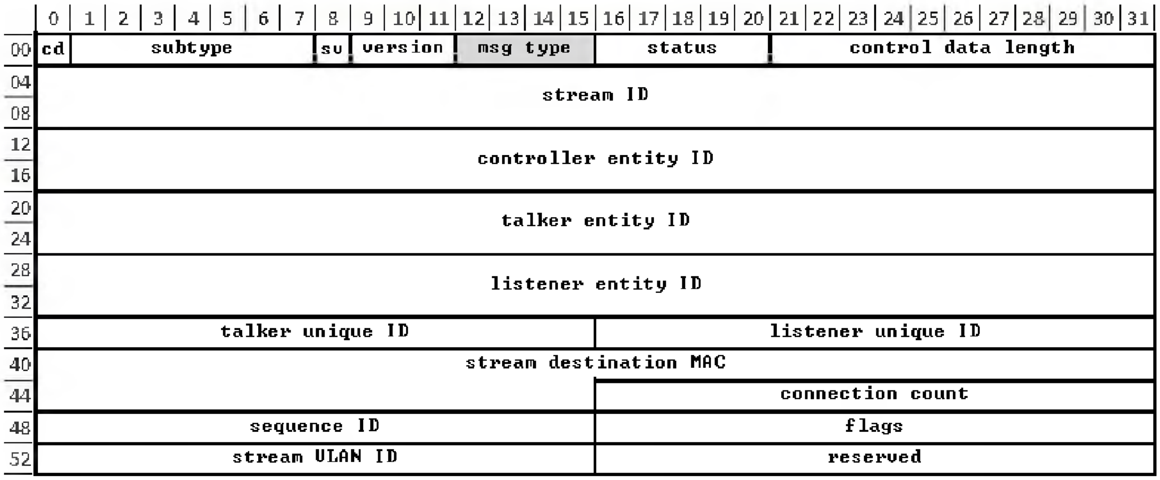
- **listener capabilities** - Describes the capabilites of the device as a listener. The endpoints in the KinectSound system all have the same listener capabilities

(*0*x*4001*). This value describes two separate capabilities, these are:

- *0*x*0001*, supports the implementation of an AVDECC listener.
- *0*x*4000*, has the capability to receive an audio stream.

### Connection Management

The connections for the KinectSound system were managed using the Streamware software provided by Echo corporation [6]. This allows a user to create audio streams between the Echo network interface and the endpoints. The workstation sends audio data via ASIO drivers to the Echo network interface, which then sends it to the endpoints specified by the connections that have been created. When a connection is created, the controller broadcasts ACMP messages that specify which device is the talker and which is the listener. The layout of an ACMP packet is shown in Figure 4.11



<table>
<tr><td></td><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr>
<tr><td>00</td><td>cd</td><td colspan="7">subtype</td><td>su</td><td colspan="3">version</td><td colspan="4">msg type</td><td colspan="5">status</td><td colspan="11">control data length</td></tr>
<tr><td>04</td><td colspan="32" rowspan="2">stream ID</td></tr>
<tr><td>08</td></tr>
<tr><td>12</td><td colspan="32" rowspan="2">controller entity ID</td></tr>
<tr><td>16</td></tr>
<tr><td>20</td><td colspan="32" rowspan="2">talker entity ID</td></tr>
<tr><td>24</td></tr>
<tr><td>28</td><td colspan="32" rowspan="2">listener entity ID</td></tr>
<tr><td>32</td></tr>
<tr><td>36</td><td colspan="16">talker unique ID</td><td colspan="16">listener unique ID</td></tr>
<tr><td>40</td><td colspan="32">stream destination MAC</td></tr>
<tr><td>44</td><td colspan="16"></td><td colspan="16">connection count</td></tr>
<tr><td>48</td><td colspan="16">sequence ID</td><td colspan="16">flags</td></tr>
<tr><td>52</td><td colspan="16">stream ULAN ID</td><td colspan="16">reserved</td></tr>
</table>

Figure 4.11 : The layout of an AVDECC Connection Management Protocol Packet

Important fields in the ACMP packets that relate to the KinectSound system are:

- **controller entity ID** - The GUID of the AVDECC controller, this is the device that is initiating the connection. This will always be the GUID of the Echo network interface in the KinectSound system as it is the only controller in the network.

- **talker entity ID** - The GUID of the AVDECC talker. This is the device transmitting the audio data over the network, which will also always be the Echo network interface.

- **listener entity ID** - The GUID of the AVDECC listener. This will be the GUID of the device that is the recipient of the audio data from the stream, an endpoint on the KinectSound system's network.

- **stream VLAN ID** - Allows the controller to specify a VLAN for the stream. As already mentioned, AVB transmits over a VLAN as it supports prioritization of the audio data.

### Control

The KinectSound system endpoints were controlled by creating AECP packets in the workstation and sending them onto the network via the Pcap API [94]. The control packets were endpoint specific and contained the mix values for the 8 different channels. These values are stored at each endpoint and applied to each audio packet that is received and are updated when a new control packet is received. The audio packets are received at a much faster rate than the control packets. The layout of an AECP packet is shown in Figure 4.12.

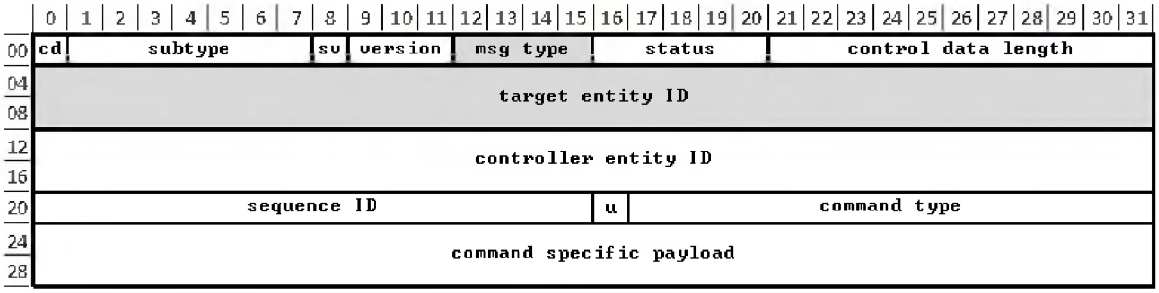


Figure 4.12 : The layout of an AVDECC Enumeration and Control Protocol Packet

Important fields in the AECP packets that relate to the KinectSound system are:

- **msg type** - The message type indicates what the purpose of the message is. Each mix level message in the KinectSound system is an *AEM_COMMAND*, which has a value of *0*x*00*.

- **target entity ID** - The GUID of the device that will receive the AECP message. This will be the GUID of one of the endpoints.

- **controller entity ID** - The GUID of the controller that is the source of the message. This is always the Echo network interface for the KinectSound system.

- **command type** - Specifies the command type of the AECP packet. The mix level packets in the KinectSound system will always be *SET_MIXER* commands, which have a value of *0*x*001C*.

- **command specific payload** - Carries the payload of the control packet. The payload contained in the KinectSound system's control packets consists of 8 bytes, each with a value between *0*x*00* and *0*x*64* (0 - 100). These values correspond to the mix levels of the 8 tracks.

## 4.5 Chapter Summary

This chapter has outlined the functionality of existing audio networking systems and introduced the concept of using a distributed processing system for surround sound processing. Current audio networking technologies that were described are IEEE 1394 (Firewire), EtherSound, Cobranet, Dante, Ethernet AVB, and AES67. Ethernet AVB, the chosen protocol was described as a combination of protocols that work together to achieve a common goal. These protocols are:

- **IEEE 802.1 Qat** - Alternatively known as the Multiple Stream Reservation Protocol, reserves network bandwidth to guarantee quality of service (QoS).

- **IEEE 802.1 Qav** - Provides queuing and forwarding enhancements for AVB endpoints and bridges. This protocol uses a *credit based shaper algorithm* in order to drain the queues at the endpoints.

- **IEEE 802.1 AS** - Uses the generalized precision time protocol which provides a time domain for the network. This enables synchronization and syntonization.

The AVDECC (Audio/Video Discovery, Enumeration, Connection management, and Control) protocol was described as a means of controlling AVB devices on a network. An AVDECC device can be classified as a *talker*, *listener*, and/or *controller*. AVDECC controllers initiate the data exchange between other devices on the network, and can perform actions such as creating an audio stream between a talker and a listener. AVDECC is comprised of 3 sub-protocols, these are:

- **ADP** - The AVDECC Discovery Protocol is responsible for identifying any AVDECC enabled devices on a network.
- **ACMP** - The AVDECC Connection Management Protocol is responsible for creating and terminating audio streams between AVDECC devices.
- **AECP** - The AVDECC Enumeration and Control Protocol is responsible for both identifying and controlling capabilities of AVDECC devices.

The network configuration of the KinectSound system was introduced. This showed how the KinectSound system utilizes Ethernet AVB and AVDECC to distribute the surround sound processing amongst the endpoints. The distributed configuration provides the following advantages:

1. The processing is distributed amongst the endpoints, which reduces the processing requirement of the workstation controller.
2. The network configuration has the capability to supply Power over Ethernet (PoE) which would result in easy configuration of the system.
3. The configuration can be built incrementally by adding new endpoints. Since there is no limitation on the number or position of the endpoints, the user is able to configure them to suit their requirements.

# Chapter 5

# Design and Implementation of the KinectSound system

This chapter describes and builds on the various aspects of the KinectSound system which have been mentioned in the previous chapters. These are the audio encoding and mixing techniques from Chapter 2, the device-free HCI from Chapter 3, and the Ethernet AVB components from Chapter 4.

The KinectSound system is a system that was developed to enable dynamic 3D panning of a virtual sound source. The system is divided up into two separate sections, the *Main System*, and *Sketchup*. The Sketchup component of the KinectSound system provides a realtime display of the user's right hand location and utilizes the Sketchup software [96]. The main system is responsible for the rest of the system's functionality, excluding audio mixing. Throughout the course of development and implementation there were several design considerations which had to fit the requirements of the system. In order to gain a better understanding of the system's functionality, this chapter also describes the design in further detail along with programming code and diagrammatic examples.

## 5.1 System Design

The KinectSound system was designed using the Unified Modelling Language (UML) [97]. The design of the system bridges the gap between the system requirements and the low level interactions between classes within the system. The system was designed using Rational Rose software [98]. This is a UML modelling tool developed by Rational Software.

The following subsections describe the steps taken in the process of designing the

KinectSound system.

### 5.1.1 Requirements Specification

The requirements specification is a high level overview of the system's functions. It describes the tasks that the system is able to perform as required by the user. There are 7 requirements which form the specification, which are:

#### Device-free Interaction

The user must be able to select menu options and fulfil all of the requirements of the system without having to use any devices. Any interaction has to be done purely by gestures and hand movements. This requirement provides a more interactive approach to the panning of audio, and also increases the user-friendliness of the system. This is done by displaying large controls that are easy to read, and provide clear instructions to anybody using the system. A user may select options that are displayed on the screen by moving their left hand into a control and then moving it forward, as if they were pressing a button. The system was required to provide accurate tracking of the user's hands in order to meet this requirement.

#### Selecting a Track

The KinectSound system presents the user with 8 tracks, which make up a single audio piece. The user must be able to select a track on which they are able to perform an action, such as change its muting state or recording state. A selected track must be made clearly visible to the user once it has been selected. This will ensure that the user does not make a mistake such as muting or recording the wrong tracks. A maximum of one track can be selected at any given time.

#### Muting/Unmuting Tracks

The user must be able to mute and unmute specific tracks in the KinectSound system. Upon system startup, all of the tracks will be unmuted by default. Each track must display its muting state so the user knows which tracks are muted or unmuted. When the user selects a *Mute/Unmute* button, the KinectSound system will check which track is selected, and change its muting state as well as deselect it. If no tracks are selected nothing will happen.

### Recording a Track

In a similar way to the muting command, a user must be able to choose a *Record Select* option, which will arm a selected track for recording. If a track is selected for recording, it will always be unmuted. If a user selects a track and selects the mute option and the track has already been selected for recording, nothing will happen. When a track has been selected for recording, a user may begin recording the sound locations for that track. This is done by sampling the position of the user's right hand at regular intervals. These positions will represent the 3D co-ordinates of a sound source, which is the track they have selected for recording. A 3D display of the user's current right hand location is also provided. This allows the user to view the location of their hand in 3 dimensions, relative to each speaker. The sound will be mixed by an endpoint at each speaker in real time, which will allow the user to hear the current position of the track being recorded. All of the tracks that are not muted will also play whilst the system is in a recording state. When a user selects the *Record* option, the system will go into a recording state after a 5 second countdown.

The data for these tracks is stored externally and is restored upon the user starting recording or playback. If a track has not been recorded, it will use its default co-ordinates.

### Playback

When the user selects the *Play* option, the KinectSound system will give a 5 second countdown and start playing the audio piece using the stored co-ordinates for each track. This procedure is similar to what happens during the recording state, with the exception that no tracks are being recorded.

### Stop Recording/Playback

When the system is in a recording or playing state, the user must be able to change the system back into an idle state when they wish to. This can be done by selecting a *Stop* option. The user is required to stop recording or playback in order to switch between the recording and playback state. This can be done in order to allow the user to playback any newly recorded tracks. When the stop option is selected, the audio data relating to each track is saved.

### Reset Tracks

The *Reset* option allows the user to reset all the tracks to the default location. The default location is determined by the average X, Y, and Z co-ordinate values for the 3D positions of each speaker. When this option is selected, all of the current data about recorded tracks is erased and replaced with the default locations. This data is then saved to an external file.

### 5.1.2 Use Case Diagrams

The UML use case diagrams provide a layout of the system's functions as well as any external actions that may influence the behaviour of the system. The KinectSound system has 2 use case diagrams. One is for the main system, and the other is for the Sketchup component.

Figure 5.1 shows the Use Case Diagram for the main system.

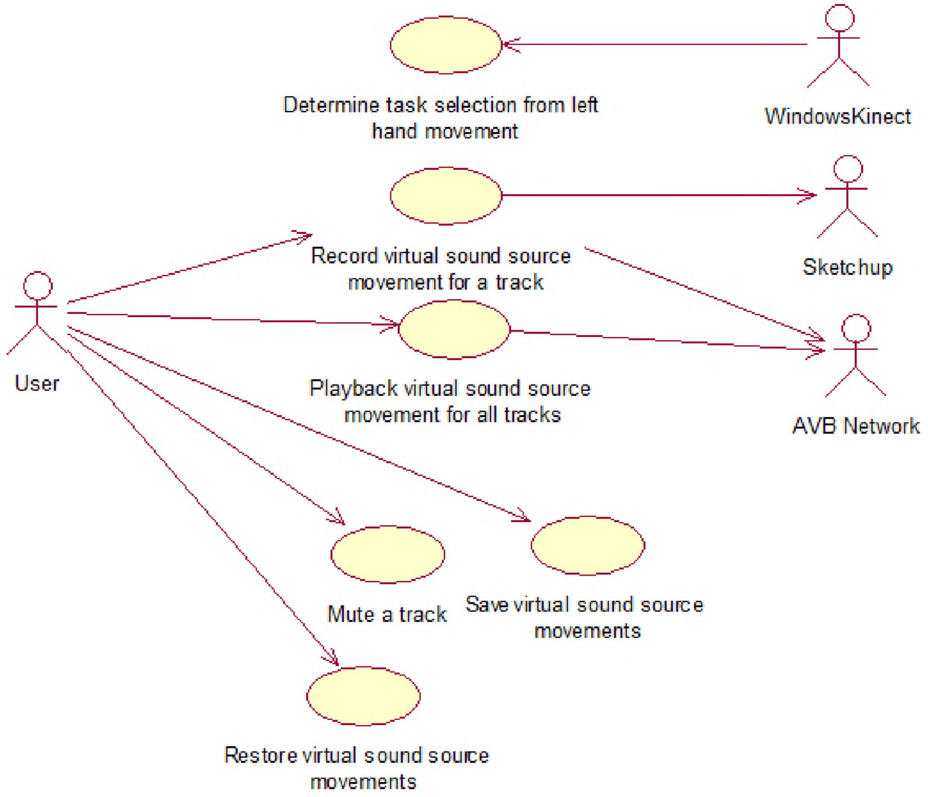

Determine task selection from left hand movement
WindowsKinect
Record virtual sound source movement for a track
Sketchup
User
Playback virtual sound source movement for all tracks
AVB Network
Mute a track
Save virtual sound source movements
Restore virtual sound source movements

Figure 5.1 : The Use Case Diagram for the main system

Every function of the system is the result of user interaction which invokes these functions. The user's hands are recognized by the Windows Kinect and have their gestures interpreted by the system, which will in turn invoke the different functions. Upon the system entering the recording or playback state, the system will send control packets to the AVB network. Each control packet contains 8 mix levels, which correspond to the 8 tracks shown in the KinectSound system. When an endpoint receives an audio packet, it will perform the audio mixing of channel samples and send the mixed samples to the speaker that it is connected to. The rest of the system's functions have been explained in the requirement specification.
Figure 5.2 shows the Use Case Diagram for the Sketchup component of the KinectSound system.

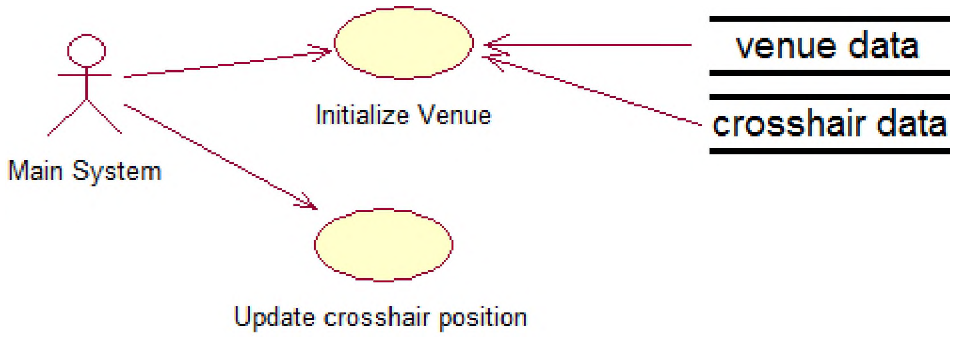


Figure 5.2 : The Use Case Diagram for the Sketchup system

On startup of the main system, Sketchup is opened and initializes the venue with the crosshair in the center of the room. This is done by inserting the models from two external Sketchup models into the current model. One of these models is the crosshair, and the other is the venue. The Sketchup camera view is also changed so the user is viewing the room from a high-angle. The Ruby code for the KinectSound system's Sketchup plugin is shown in Appendix 8.2. To ensure the accuracy of the system, the venue that Sketchup is using is built to scale with the size of the room and speakers the user is using. The venue initialization also places a crosshair at the origin, which is also saved as an external Sketchup file. The position of the crosshair is constantly updated by the main system to correspond to the user's right hand movements. The main system and Sketchup system communicate through a socket connection.

### 5.1.3 Class Diagram

Figure 5.3 shows the class diagram for the KinectSound system.

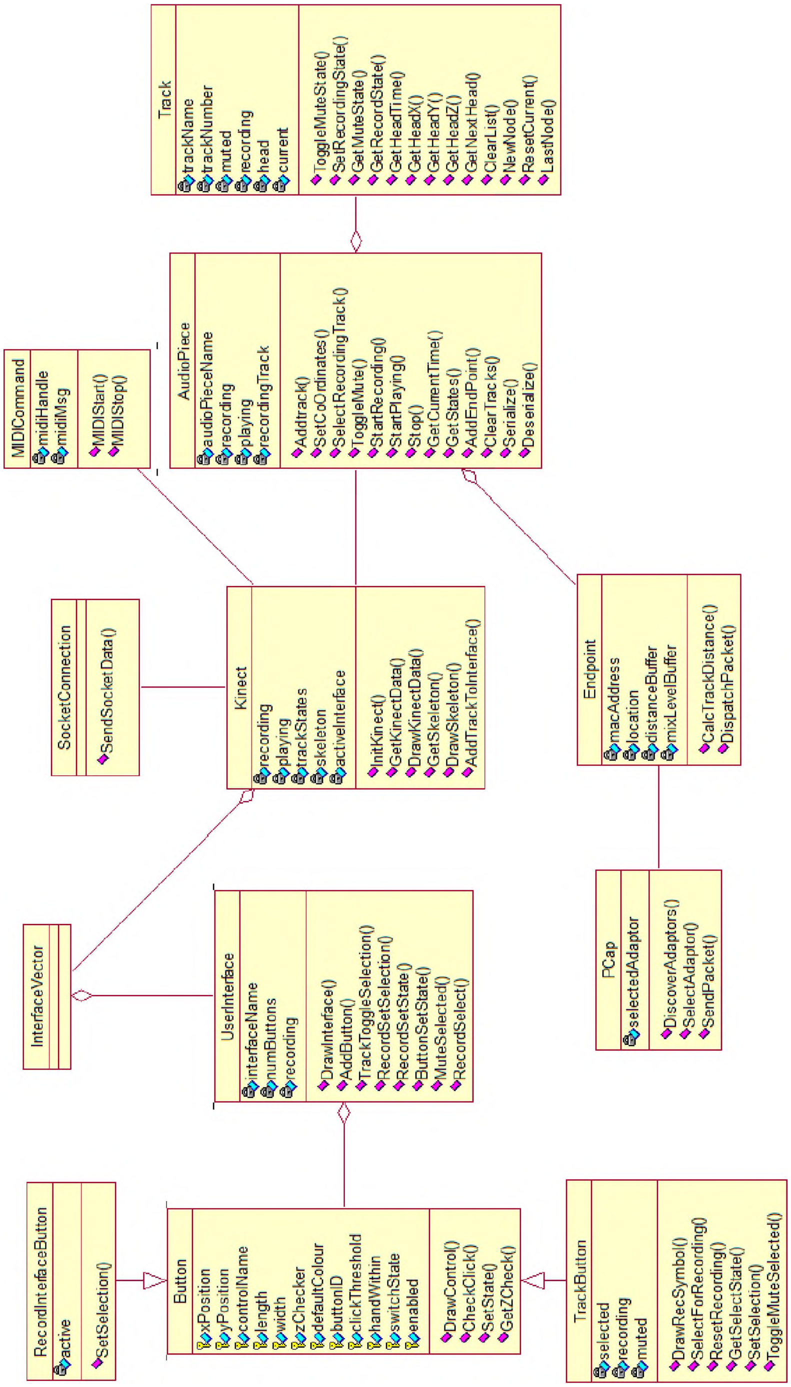


Figure 5.3 : The Class Diagram for the KinectSound system.

Each class in the diagram contains the following:

- **Class Name** - A unique identifier given to each class. This name should be relevant and be able to briefly describe the functionality of the class.
- **Class Attributes** - Each class may contain one or more attributes which aid the functionality of the class. These may be defined to be either accessible or inaccessible from other classes. Each attribute has a data type associated with it. This is not shown in the class diagram.
- **Class Operations** - Each class operation is a sequence of events that occur to achieve a specific outcome. These operations are also able to return a value depending on how they are defined. Class operations are usually directly accessible from other classes, but they can also be made inaccessible.

The attributes and components are explained further at the relevant points in this chapter and how they relate to the programming constructs used to achieve the goals of the KinectSound system.

### 5.1.4 Sequence Diagrams

Sequence diagrams show the interaction between objects as a sequence of events that occur when the system is fulfilling one of the requirements. These diagrams contain a number of messages between objects in order to achieve the desired outcome.
Figure 5.4 shows the sequence diagram for the system startup. The following subsections use sequence diagrams where they are necessary to describe the system implementation. The state transition diagram and remaining sequence diagrams can be found in Appendix 8.3.

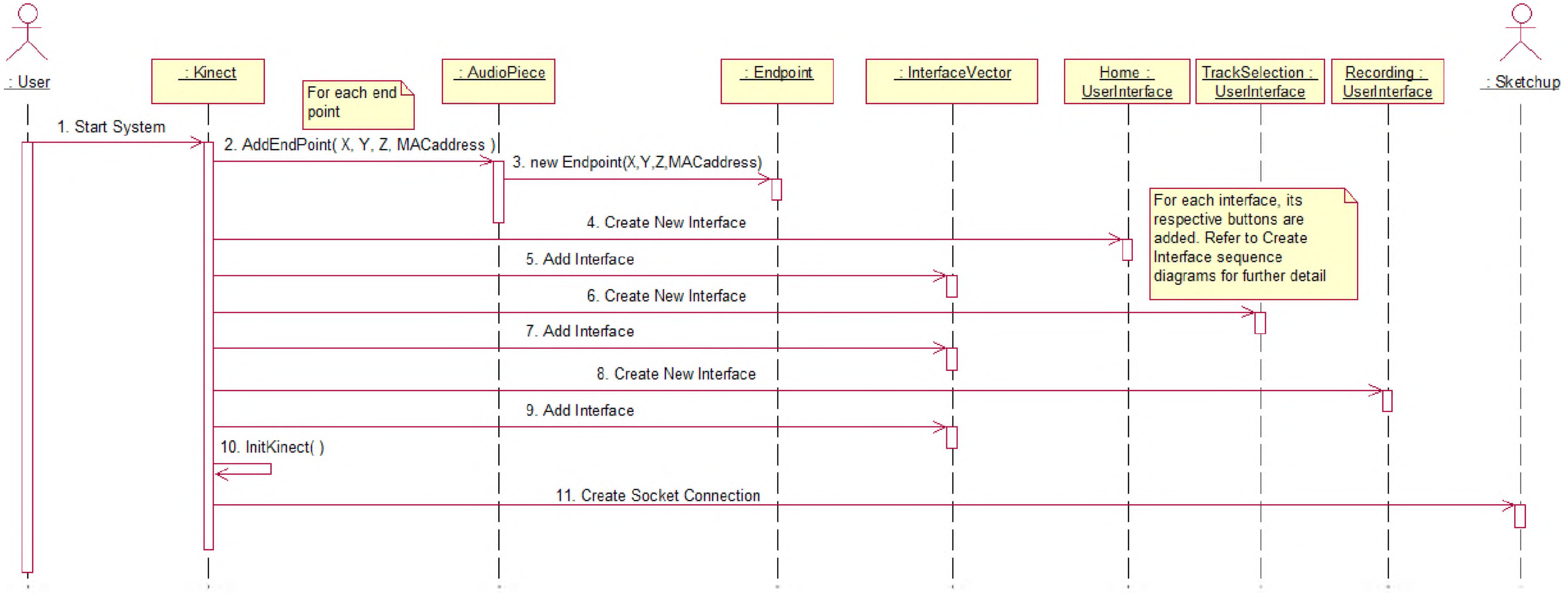

: User
: Kinect
For each end point
: AudioPiece
: Endpoint
: InterfaceVector
Home : UserInterface
TrackSelection : UserInterface
Recording : UserInterface
: Sketchup
1. Start System
2. AddEndPoint( X, Y, Z, MACaddress )
3. new Endpoint(X,Y,Z,MACaddress)
For each interface, its respective buttons are added. Refer to Create Interface sequence diagrams for further detail
4. Create New Interface
5. Add Interface
6. Create New Interface
7. Add Interface
8. Create New Interface
9. Add Interface
10. InitKinect( )
11. Create Socket Connection

Figure 5.4 : The Sequence diagram for the startup of the KinectSound system.

The functions in each sequence of events are as follows:

- The user starts the KinectSound system (1).
- The Kinect object creates instances of each audio endpoint in the AudioPiece object (2, 3). The endpoints are specified by the user in the programming code before runtime.
- The Kinect object creates an instance of each *User Interface* and adds it to the interface vector, where it is able to access them when they are needed (4 - 9).
- The Kinect object initializes the Kinect's camera so the KinectSound system is able to use it (10).
- A socket connection is created between the Kinect object and Sketchup to enable inter-process communication (11).

## 5.2 Device Free HCI

As stated in Chapter 3, one of the fundamental goals of the KinectSound system was to have device free control. This required the user to have complete control of the system without having to use a mouse or keyboard. This was achieved by the use of a *Windows Kinect*, which provided the stated requirements.

### 5.2.1 The Windows Kinect

The Windows Kinect's SDK API is used by the KinectSound system to gain access to the gesture recognition capability of the Kinect. The Kinect's camera image is accessed and displayed using Open Graphics Libraries (OpenGL) via the Simple DirectMedia Layer (SDL) or the OpenGL Utility Toolkit (GLUT) [99]. The KinectSound system used the SDL for displaying the camera image and user's skeleton to a windows dialog box. GLUT was initially used for the interfaces, but this was changed to SDL as it provided better control over the OpenGL functions that were required to implement the interfaces.

### Initializing the Kinect's camera

The Kinect's components need to be initialized before it may be used within an application. The KinectSound system uses the camera and skeleton tracking features, so there is no need to initialize the microphone array. Listing 5.1 shows the pseudocode for initializing the Kinect as it is done in the KinectSound system, adapted from [99].

```
Get Kinect Sensor
IF (KinectSensor detected)
    Initialize Kinect Sensor for use
    Initialize textures
    Initialize OpenGL
    Set camera viewport transformation
WHILE (forever)
    Try and get next frame from Kinect Sensor
    IF (frame exists)
        Draw frame
```

Listing 5.1: Pseudocode for initializing the Kinect.

The first two lines of the pseudocode check if there is a Kinect connected to the computer. This step requires the Kinect SDK to be installed. If it establishes that there is a Kinect connected, it initializes the components needed to detect and display frames from the camera. These components are [99]:

1. Sensor - The sensor includes the Kinect's camera as well as the infrared transmitter and receiver.

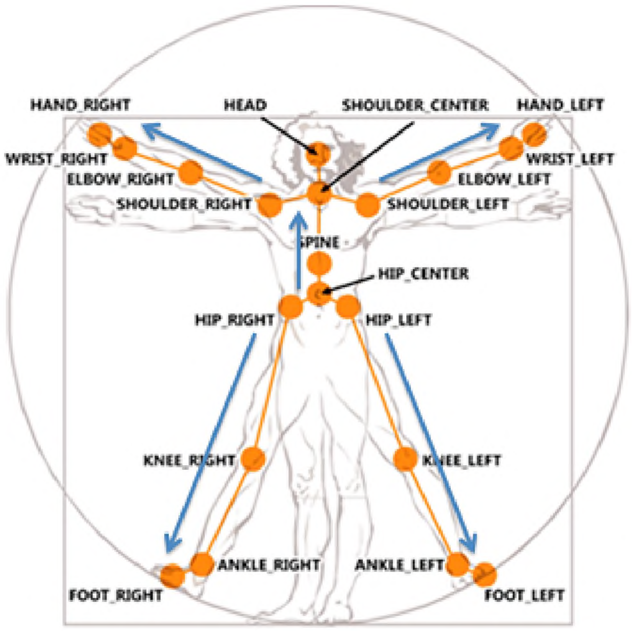
2. Textures - The textures contain the image frames that are received from the Kinect's camera. The KinectSound system uses a *640* x *480* resolution for displaying the image.

3. OpenGL - OpenGL is used to draw the frames from the Kinect's camera onto a window.

4. View-port - The OpenGL view-port specifies the region of a window in which a specific image will be displayed. In this case it will define where the camera image is displayed.

### Tracking a Skeleton

The KinectSound uses OpenGL to superimpose certain *Kinect Skeleton* parts over the camera image. As mentioned in Chapter 3, the Kinect skeleton is composed of 20 core joints found in the human body. The Kinect SDK uses an enumerated type which contains the values between 0 and 19 to distinguish between the different joints. Figure 5.5 shows the core joints of the Kinect skeleton [100] as well as the enumerated type values for each joint.



| | |
|---|---|
| Hip Center | 0 |
| Spine | 1 |
| Shoulder Center | 2 |
| Head | 3 |
| Shoulder Left | 4 |
| Elbow Left | 5 |
| Wrist Left | 6 |
| Hand Left | 7 |
| Shoulder Right | 8 |
| Elbow Right | 9 |
| Wrist Right | 10 |
| Hand Right | 11 |
| Hip Left | 12 |
| Knee Left | 13 |
| Ankle Left | 14 |
| Foot Left | 15 |
| Hip Right | 16 |
| Knee Right | 17 |
| Ankle Right | 18 |
| Foot Right | 19 |

Figure 5.5 : The Kinect skeleton and enumerated type joint values

The KinectSound system uses the left arm and the right hand for menu selection and panning respectively, so these are the only sections of the skeleton that are displayed. The KinectSound system uses the methods in Listing 5.2 to convert the skeleton co-ordinates from displacement values into pixels. This code is part of the *GetSkeleton* method.

```
for (int i = 0; i < 19; i++){ //for each joint in the skeleton
   JointVec = skeleton.SkeletonPositions[i];
   NuiTransformSkeletonToDepthImage(JointVec, &xCoOrd, &yCoOrd);
   skele[0][i] = xCoOrd*2;
   skele[1][i] = yCoOrd*2;
}
```

Listing 5.2: Converting skeleton positions into 640x480 display co-ordinates.

When the Kinect detects a skeleton, it stores the joints shown in Figure 5.5 in a *Vector4* (4D vector) array called *SkeletonPositions*. A Vector4 structure stores the W, X, Y, and Z values. *W* is a scaling factor, and the *X, Y,* and *Z* values are the 3D co-ordinates [101]. *NuiTransformSkeletonToDepthImage* transforms the temporary W, X, Y, and Z values stored in *JointVec* into *X* and *Y* floating point values for image display. These values are stored in the *xCoOrd* and the *yCoOrd* variables, as they are passed by reference to this method. These co-ordinates are then stored in their respective positions in the array named *skele*. This is a 2 x 20 array which stores the X and Y display co-ordinates of each joint that comprises the Kinect skeleton. The values had to be multiplied by a factor of 2 before they were stored as the Kinect's skeleton tracking capability uses a 320 x 240 resolution and the KinectSound system is displaying to a 640x480 window.

Listing 5.3 shows the procedure for drawing the user's left arm, and the crosshair on the right hand. This code is contained within the *DrawSkeleton* method.

```
for (int j = NUI_SKELETON_POSITION_SHOULDER_LEFT; j < NUI_SKELETON_POSITION_HAND_LEFT; j++){
   glBegin(GL_LINES);
      glVertex2f(skele[0][j], skele[1][j]);
      glVertex2f(skele[0][j+1], skele[1][j+1]);
   glEnd();
}
glColor3f(1,0,0); //Change colour
glBegin(GL_LINES);   //Crosshair
   //Horizontal line
   glVertex2f(skele[0][NUI_SKELETON_POSITION_HAND_RIGHT]-10,
      skele[1][NUI_SKELETON_POSITION_HAND_RIGHT]);
   glVertex2f(skele[0][NUI_SKELETON_POSITION_HAND_RIGHT]+10,
      skele[1][NUI_SKELETON_POSITION_HAND_RIGHT]);
glEnd();
glBegin(GL_LINES);
   //Vertical line
   glVertex2f(skele[0][NUI_SKELETON_POSITION_HAND_RIGHT],
      skele[1][NUI_SKELETON_POSITION_HAND_RIGHT]-10);
   glVertex2f(skele[0][NUI_SKELETON_POSITION_HAND_RIGHT],
      skele[1][NUI_SKELETON_POSITION_HAND_RIGHT]+10);
glEnd();
glColor3f(0,1,0); //Change colour
```

```
23 glBegin(GL_POINTS); //Joints on arm
24     for (int j = NUI_SKELETON_POSITION_SHOULDER_LEFT; j <= NUI_SKELETON_POSITION_HAND_LEFT; j++){
25         glVertex2f(skele[0][j], skele[1][j]);
26     }
27 glEnd();
```

Listing 5.3: Drawing the user's left hand arm and right hand crosshair using OpenGL.

The code in listing 5.3 superimposes shapes over the Kinect image displayed in the window by using OpenGL. The KinectSound system uses two different OpenGL modes to draw the user's skeleton, these are:

1. GL_LINES - Joins a set of vertices with straight lines
2. GL_POINTS - Draws bullets on a set of vertices. The bullets may have their shape specified by the user.

The colour and weight of the bullets and lines are able to be changed via OpenGL functions. The OpenGL mode is specified as an argument in the *glBegin* function. This function along with the *glEnd* function delimit the OpenGL drawing procedure. The purpose of each OpenGL segment is as follows:

- The iteration in lines 1 - 6 use the OpenGL mode *GL_LINES* to draw a set of 3 straight lines which make up the users left arm.
- Line 7 changes the current drawing colour to red, as this is the colour used to draw the crosshair for the right hand display.
- Lines 8 - 21 use the *GL_LINES* mode twice to draw the crosshair as a separate vertical and horizontal line.
- Line 23 changes the colour to green, which is used for drawing the points along the user's arm.
- Lines 23 - 27 use the OpenGL mode *GL_POINTS* to draw points at the positions of each joint in the users left arm.

Figure 5.6 shows the outcome of the code in Listings 5.2 and  5.3.

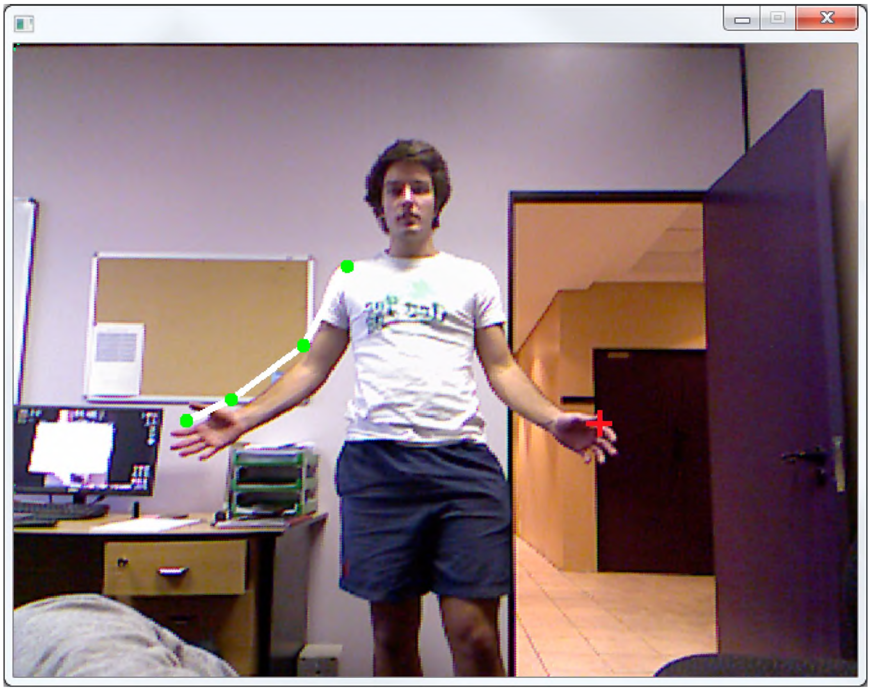


Figure 5.6 : The KinectSound system displaying the user's left arm, and right hand crosshair.

### 5.2.2 The Interfaces for a Device Free System

A user interface, in the context of the KinectSound system, is a group of buttons that are all displayed at the same time. This can be seen by the aggregation relationship in the class diagram (Figure 5.3). As a user navigates between interfaces, the groups of buttons being displayed will change according to which user interface the user has navigated to. The KinectSound system has 3 different user interfaces, these are:

1. **Home Interface** - the interface displayed upon startup. This user interface allows the user to navigate to the Track Selection Interface, clear track data, or shut down the system. The user may return to this interface at any time by selecting the *Home* option on any of the other interfaces

2. **Track Selection Interface** - allows the user to change the recording state and the muting state of the tracks. The user may do this by selecting a track

and then selecting either the *MUTE/UNMUTE SELECTED* or the *RECORD SELECT* options.

3. **Recording Interface** - allows the user to start and stop the recording and playback of the audio piece.

Figure 5.7 shows the Home Interface (A), the Track Selection Interface (B), and the Recording Interface (C). The interfaces in Figure 5.7 are shown with black backgrounds. The background for each interface would appear as the camera image showing the users right hand and left arm, as it is seen in Figure 5.6.

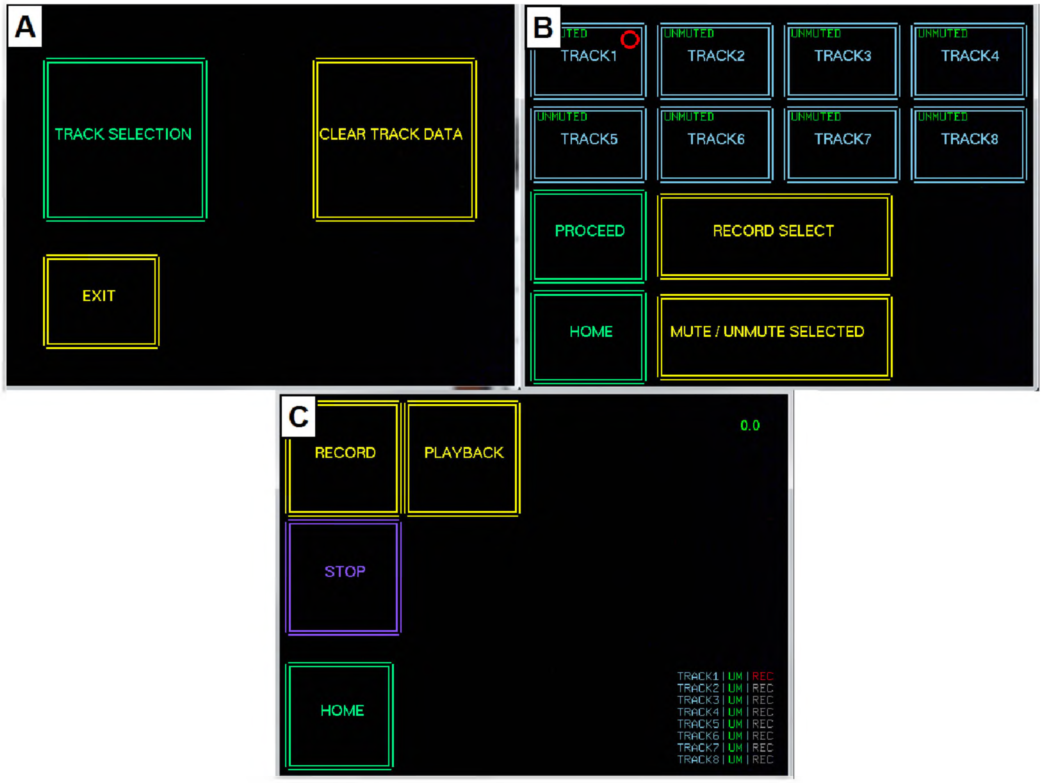


Figure 5.7 : The three interfaces used by the KinectSound system.

The KinectSound system is able to create instances of three button classes, which it is able to add to an interface. The button classes are:

- **Button Class** - This is the most basic button found within the KinectSound system. Instances of this class can be added to an interface for performing basic commands such as navigation between interfaces. The remaining button classes in the system contain the attributes and operations of this button class.

- **Track Button Class** - The Track Button objects have attributes that correspond to the tracks that make up the audio piece. These properties are the *Muting State* and the *Recording State*, and are displayed within the button. Each Track Button also has the ability to be selected. A Track Button will have to be selected before a user is able to change its recording or muting state. The Track Button will change colour when it is selected.

- **Record Interface Button Class** - The Record Interface Button objects have the capability to be *active*. This allows the user to change the state of the system between recording, playing, and idle. The button will change colour when it is active so that the user is able to see which button has been selected.

The *UserInterface* class provides a method called *AddButton* which allows a user to add one of the three button types to an interface. The *AddButton* method has been overloaded in order to determine which button to create. The function signatures for the *AddButton* functions are shown in Listing 5.4.

```
//Three overloaded methods to add different button types to an interface
//1. Function to add a Button
void AddButton(int xPos, int yPos, int controlLength, int controlWidth, char tName[], float *
    defaultColours, int controlGroupID);
//2. Function to add a Record Interface Button
void AddButton(int xPos, int yPos, int controlLength, int controlWidth, char tName[], float *
    defaultColours, int controlGroupID, bool defaultSelected);
//3. Function to add a Track Button
void AddButton(int xPos, int yPos, int controlLength, int controlWidth, char tName[], float *
    defaultColours, int controlGroupID, bool defaultMuteState, bool defaultSelectState);
```

Listing 5.4: Function signatures for adding buttons to an interface.

Each argument in the *AddButton* method defines one of the attributes of the button that it is creating. Not all of the buttons attributes are required in the signature as they can be given a default value (such as *clickThreshold*). The first *AddButton* function is for a standard button, the second is for a Record Interface Button, and the third is for a Track Button. A description of each argument is given below:

- **int** xPos - The horizontal position of the button.

- **int** yPos - The vertical position of the button.
- **int** controlLength - The length of the button.
- **int** controlWidth - The width (height) of the button.
- **char** tName[] - The name of the button, which will be displayed in the center of the button.
- **float** *defaultColours - A pointer to an array consisting of 3 floating point values. These values represent RGB values that are used to colour the button.
- **int** controlGroupID - This specifies which user interface the button is on. This value will be 1 for the Home Interface, 2 for the Track Selection Interface, and 3 for the Recording Interface. This value is used to create the unique *buttonID* associated with each button.
- **bool** defaultSelected - Required to create Recording Interface Buttons. It specifies the default selected state of the button.
- **bool** defaultMuteState - The first overloaded argument for creating a Track Button. Specifies the default muting state of the track.
- **bool** defaultSelectState - The second overloaded argument for creating a Track Button. Specifies the default selected state of a Track Button.

The *Kinect* object contains a list which stores the 3 instances of the *UserInterface* class that have been mentioned earlier. The instance of each user interface is created by the *Kinect* object, which also calls the functions to add the buttons to each interface. Each instance of the interface class has 3 different *Vectors* which store instances of the three different button types.

The interface and button storage structures are shown in Figure 5.8

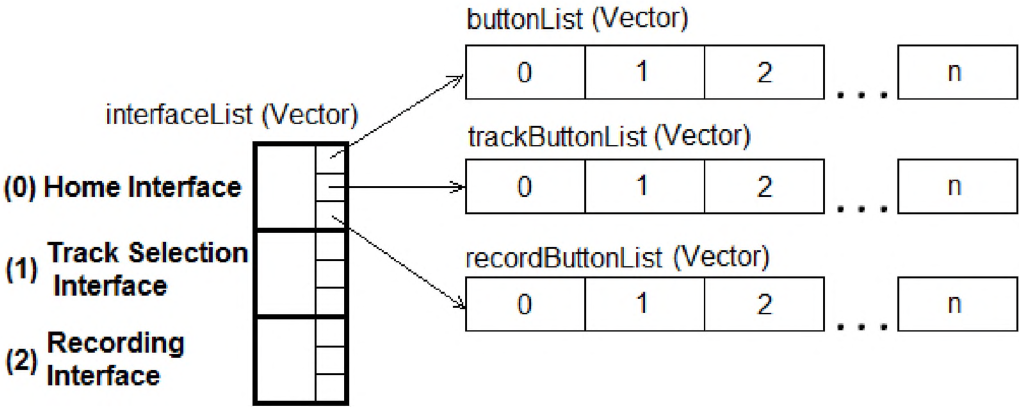


Figure 5.8 : A structure showing how interfaces and buttons are stored.

The *Kinect* class contains an integer variable which indicates the active interface. This is the current interface that will have its buttons displayed to the window, and will change when the user navigates between interfaces.
When the *Kinect* object creates the interfaces, to does the following steps for each interface:

1. Create an instance of an interface and add it to the interface list.

2. Add each button to the interface.

The buttons are added to an interface straight after it has been created so the *interfaceList.size()* function can be used to specify the *controlGroupID*. The order in which the interfaces are created is important in that it maintains the integrity of the *controlGroupID* and the process of creating the ID for each button. Listing 5.5 shows examples of the *AddButton* function being called.

```
//Initializing the interface
 float *d = new float[3];
 d[0] = 0.0/255.0f;d[1] = 255.0/255.0f;d[2] = 127.0/255.0f;
interfaceList.push_back(Interface("Home Interface")); //Add interface
 interfaceList.at(0).AddButton(50,480-50-112, 140, 112, "EXIT",d, interfaceList.size()); //Add button
//Add other interface buttons
 d[0] = 123.0/255.0f; d[1] = 205.0/255.0f; d[2] = 237.0/255.0f;
interfaceList.push_back(Interface("Track Selection Interface"));
 AddTrackToInterface(1, 10, 25, 140,90,"TRACK1",d, interfaceList.size(), 0, 0,1);
//Add other interface buttons
 d[0] = 255.0/255.0f; d[1] = 255.0/255.0f; d[2] = 0.0/255.0f;
interfaceList.push_back(Interface("Recording Interface"));
 interfaceList.at(2).AddButton(10,10, 140, 140, "RECORD",d, interfaceList.size(), 0);
//Add other interface buttons
//Other Initialization

//Function that adds a button to the interface and a track to the audiopiece
void Kinect::AddTrackToInterface(int interfaceNumber, int xPos, int yPos, int controlLength, int controlWidth, char tName[],float *defaultColours, int controlGroupID, bool defaultMuteState, bool defaultSelectState, int trackNo)
 {
   interfaceList.at(interfaceNumber).AddButton(xPos, yPos, controlLength,controlWidth,tName, defaultColours, controlGroupID, defaultMuteState, defaultSelectState);
  A->AddTrack(tName, trackNo);
 }
```

Listing 5.5: Calling the AddButton function.

All of the buttons are added to their respective interfaces upon system startup. When a Track Button is added to the interface, it is done by calling the *AddTrackToInterface* function. The reason for this is that when a Track Button is added, a *Track* must also be added to the *AudioPiece* class, which is done in this function along with the *AddButton* function. The attributes for each track button must remain the same as the attributes corresponding to their respective tracks which are contained in the *AudioPiece* class. These are the *Recording* and *Muting* attributes.

### 5.2.3 Interacting with the KinectSound system

The user uses his left hand to select a button. This is done by moving the left hand into the boundary of the button, and then moving the hand forward. Each button required 4 attributes in order to detect this selection. These are:

- **handWithin** - A boolean variable indicating whether the hand is within the boundaries of a button

- **zChecker** - A floating point value which stores the Z co-ordinates of the user's hand upon entering the boundary of a button
- **clickThreshold** - An integer variable which indicates the distance (in millimeters) the user is required to move their hand forward in order to signal a button click
- **switchState** - A boolean variable which states whether a button has been clicked

The procedure for determining a button click is shown using a flowchart in Figure 5.9. This procedure occurs every time a frame is received.

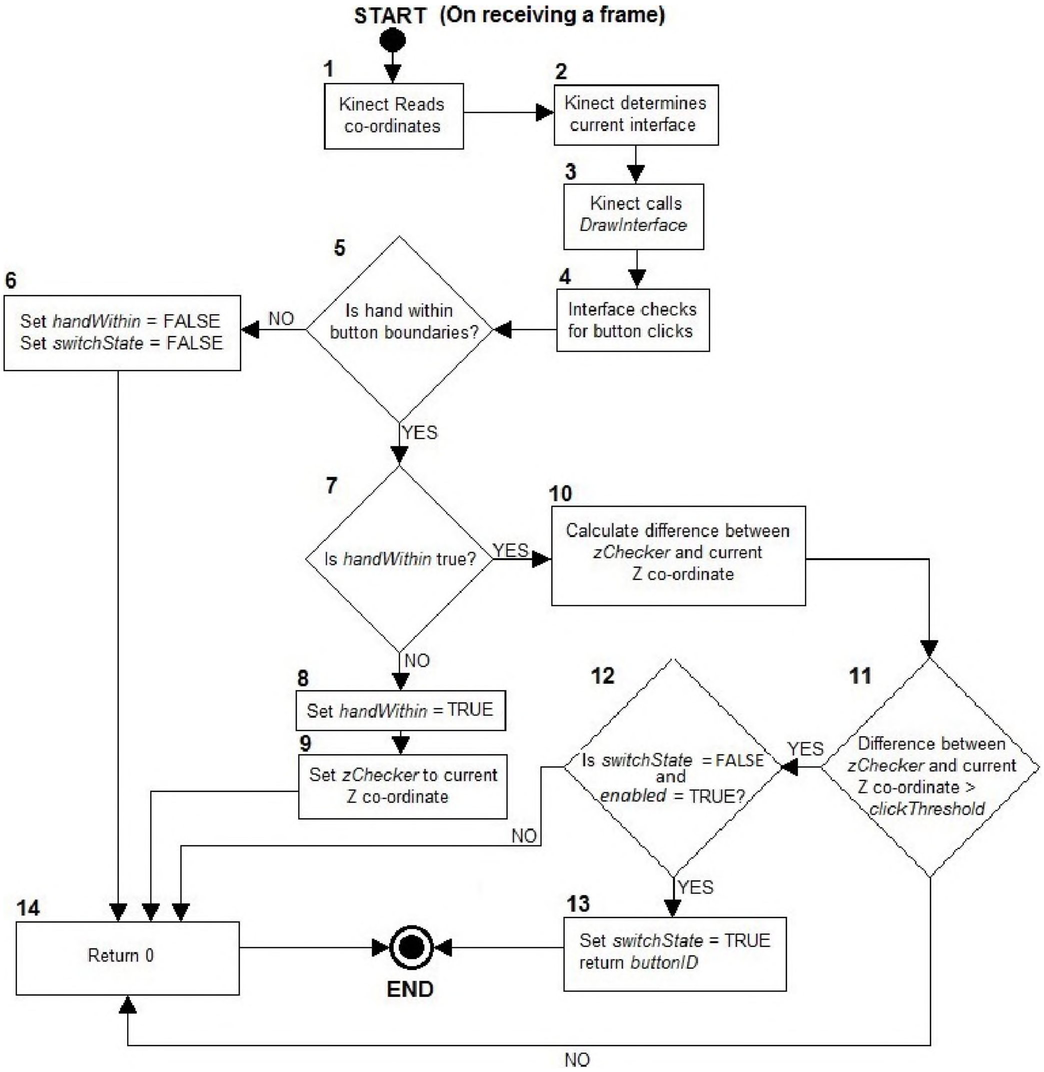

START (On receiving a frame)
1
Kinect Reads co-ordinates
2
Kinect determines current interface
3
Kinect calls DrawInterface
4
Interface checks for button clicks
5
Is hand within button boundaries?
NO
6
Set handWithin = FALSE
Set switchState = FALSE
YES
7
Is handWithin true?
YES
10
Calculate difference between zChecker and current Z co-ordinate
NO
8
Set handWithin = TRUE
9
Set zChecker to current Z co-ordinate
11
Difference between zChecker and current Z co-ordinate > clickThreshold
YES
12
Is switchState = FALSE and enabled = TRUE?
NO
YES
13
Set switchState = TRUE
return buttonID
14
Return 0
END
NO

Figure 5.9 : The flowchart showing how to determine a button click

Each step in the flowchart can be described in further detail as follows:

1. The Kinect determines the co-ordinates of the user's skeleton.
2. The active interface is determined by the reading the value of the *activeInterface* variable.
3. The *Kinect* class calls the *DrawInterface* function for the active interface. The current X, Y, and Z co-ordinates of the user's left hand are passed to this function.

4. The active interface iterates through the 3 different button lists and checks for a click on each button instance. The remaining steps in the flowchart happen for each button on the active interface.

5. A check is done to see whether the position of the user's left hand is inside the boundary of the button. Each button has an (X, Y) co-ordinate set for it's location, as well as a length and width. By using the current location of the user's hand, the system is able to determine whether or not the user's hand is within the boundaries of the button.

6. If the user's hand is not in the boundaries of the button, the variables *switchState* and *handWithin* are both set to false.

7. The system checks whether the value of *handWithin* is true or false. This condition will only result in being true when the users hand enters the boundaries of the button.

   The following 2 steps describe the events that take place upon the user's left hand entering the button's boundary.

8. The *handWithin* value is set to true.

9. The *zChecker* value is set to the current Z value of the user's left hand.

10. If *handWithin* is true, the system calculates the difference between the current Z value and the value of the *zChecker*.

11. The system checks whether the calculated difference is greater than the predefined *clickThreshold*

12. The values of *switchState* and *enabled* are checked. The value of *switchState* is required to be false and the value of *enabled* is required to be true for the system to signal a button click. The *switchState* check is done to prevent the system from signalling multiple button clicks once the *clickThreshold* is exceeded. If the user wishes to select the same button twice, they must remove their left hand from the button's boundaries which will set the value of *switchState* back

to false as shown in step 6. The *enabled* check is done so the user is unable to select buttons that have been disabled.

13. If all of the conditions are satisfied, the *CheckClick* function signals a button click by returning the identifier of the button that has been selected. The value of *switchState* is changed to true to prevent unnecessary button click signals.

14. If one of the conditions are not satisfied, the *CheckClick* function will return zero.

Listing 5.6 shows the programming code used for the *CheckClick* function.

```cpp
int Button::CheckClick(float coOrdX, float coOrdY, float coOrdZ){
    bool xIn = false;
    bool yIn = false;
    // Check X-co-ordinates
    if (coOrdX >= this->X && coOrdX <= ((this->X)+(this->length)))  xIn = true;
    else
    {
        handWithin = false;
        switchState = false;
        return 0;
    }
    // Check Y co-ordinates
    if (coOrdY >= this->Y && coOrdY <= ((this->Y)+(this->width)))  yIn = true;
    else
    {
        handWithin = false;
        switchState = false;
        return 0;
    }
    //When hand enters button boundary
    if (xIn && yIn && !handWithin)
    {
        handWithin = true;
        Zchecker = coOrdZ;
        return 0;
    }
    // If all the conditions are satisfied
    if (handWithin && Zchecker - coOrdZ > clickThreshold && !switchState && this->enabled == true)
    {
        switchState = true;
        return this->buttonID;
    }
}
```

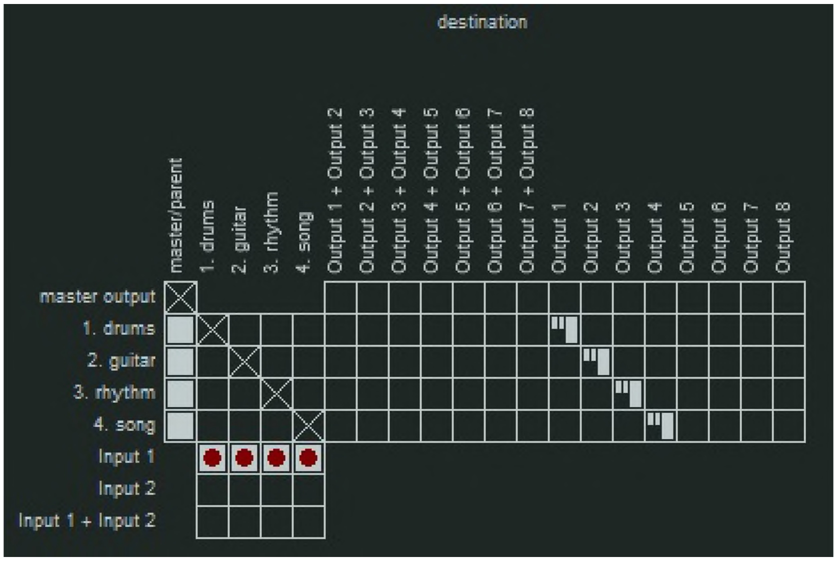
Listing 5.6: The *CheckClick* function.

## 5.3 Digital Audio Workstation (DAW) control

The DAW uses the Streamware ASIO driver [6] to transmit a maximum of 8 audio channels onto the AVB network. Although the ASIO driver is capable of transmitting a maximum of 64 channels, the endpoints are only capable of receiving 8. A form of interaction had to be established between the KinectSound system and the DAW to allow the remote playing and stopping of the audio piece. Internal MIDI control messaging was used to do this as it provided a simple method of DAW transport control.

### 5.3.1 Drivers and Channels

The DAW uses the Streamware ASIO drivers which are provided with the Echo network adaptor. The network adaptor is an AVDECC talker (Section 4.3.1) and AVDECC controller (Section 4.3.3) by default. The multicast stream between the workstation and the endpoints is configured using the Streamware software, which is also provided by Echo Audio. The channels for the stream are configured so that the ASIO outputs correspond to the stream channel numbers, as shown in Figure 5.10. For example: ASIO output 1 in the DAW corresponds to channel 1 in the multicast stream. This has to be done so that the track configuration in the KinectSound Track Selection interface corresponds to the configuration in the DAW.



destination

| | master/parent | 1. drums | 2. guitar | 3. rhythm | 4. song | Output 1 + Output 2 | Output 2 + Output 3 | Output 3 + Output 4 | Output 4 + Output 5 | Output 5 + Output 6 | Output 6 + Output 7 | Output 7 + Output 8 | Output 1 | Output 2 | Output 3 | Output 4 | Output 5 | Output 6 | Output 7 | Output 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| master output | ☒ | | | | | | | | | | | | | | | | | | | |
| 1. drums | ▨ | ☒ | | | | | | | | | | | ▮ | | | | | | | |
| 2. guitar | ▨ | | ☒ | | | | | | | | | | | ▮ | | | | | | |
| 3. rhythm | ▨ | | | ☒ | | | | | | | | | | | ▮ | | | | | |
| 4. song | ▨ | | | | ☒ | | | | | | | | | | | ▮ | | | | |
| Input 1 | | ● | ● | ● | ● | | | | | | | | | | | | | | | |
| Input 2 | | | | | | | | | | | | | | | | | | | | |
| Input 1 + Input 2 | | | | | | | | | | | | | | | | | | | | |

Figure 5.10 : The Reaper routing matrix.

The routing configuration shown in Figure 5.10 will route Track 1 (drums) to Output 1, Track 2 (guitar) to Output 2, and so on. The outputs shown in the DAW correspond to the 8 tracks available in the KinectSound Track Selection Interface. The user is also able to route 2 tracks to one output if they wish. The KinectSound system will treat this as a single track, in which all of the tracks' localization properties will be identical.

### 5.3.2 MIDI Control

As already stated, the KinectSound system uses MIDI messages to communicate with the DAW. These messages are sent to a MIDI driver known as *LoopBE Internal MIDI* [102], which allows applications within a workstation to send MIDI messages to each other. Figure 5.11 shows how the internal MIDI port allows communication between the KinectSound system and the DAW.

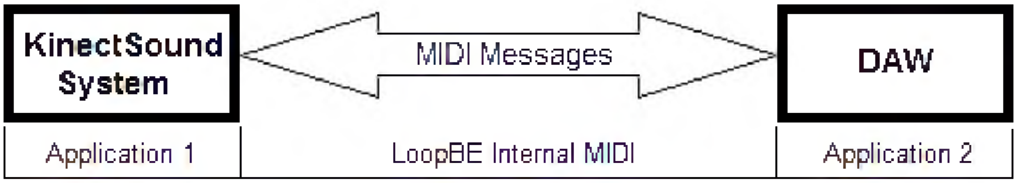


Figure 5.11 : Communication between the KinectSound system and the DAW using internal MIDI.

The internal MIDI software is automatically detected as a MIDI device by the DAW when it is installed. The KinectSound system uses MIDI control messages [103] to signal the DAW to start or stop playback. MIDI control messages are 3 byte MIDI messages which have the following properties:

1. **Status** - The first byte of the MIDI control message indicates the type of message and the channel that it is sent through. The KinectSound system sets this byte to *0*x*B0*, which indicates a MIDI control message on MIDI channel 1.

2. **Control Number** - The second byte of the MIDI control message indicates the control function that is used by the message. The KinectSound system sets this byte to *0*x*1A*, which is a value in the MIDI standard that currently has no control function and is reserved for future use [103].

3. **Value** - The third byte of the MIDI control message contains a value between 0 - 127 inclusive. This value is used to identify any further information about the control message. The KinectSound system will set this to *0*x*00* to signal a *Play* command, and *0*x*01* to signal a *Stop* command.

The hexadecimal values for each completed MIDI message that is sent by the KinectSound system will appear as follows:

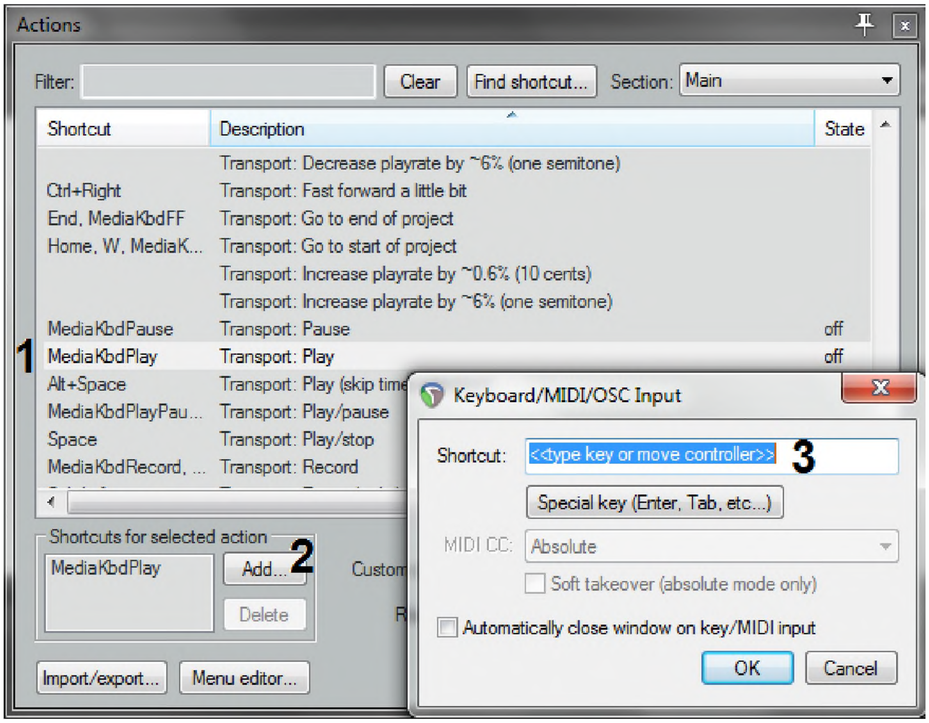
- **Play** - *0*x*B01A00*

- **Stop** - *0*x*B01A01*

The actions for these messages had to be configured in the DAW as shortcuts to perform *transport* commands. The transport commands in Reaper include actions such as Play, Pause, and Stop. These are configured by opening the action list (“?” hotkey) and doing the following:

1. Select the transport command that will have a shortcut added to it.
2. Click on the “Add” button in the “Shortcuts for selected action” region of the window.
3. Send the internal MIDI control message which will correspond to the selected transport action. The MIDI message’s status, control number, and value will appear in the “Shortcut” field of the dialog. Select “OK” to assign the shortcut to the transport command.

The 3 steps for assigning transport shortcuts are shown in Figure 5.12.



Filter:

Section: Main

| Shortcut | Description | State |
|---|---|---|
| | Transport: Decrease playrate by ~6% (one semitone) | |
| Ctrl+Right | Transport: Fast forward a little bit | |
| End, MediaKbdFF | Transport: Go to end of project | |
| Home, W, MediaK... | Transport: Go to start of project | |
| | Transport: Increase playrate by ~0.6% (10 cents) | |
| | Transport: Increase playrate by ~6% (one semitone) | |
| MediaKbdPause | Transport: Pause | off |
| MediaKbdPlay | Transport: Play | off |
| Alt+Space | Transport: Play (skip time | |
| MediaKbdPlayPau... | Transport: Play/pause | |
| Space | Transport: Play/stop | |
| MediaKbdRecord, ... | Transport: Record | |

Shortcuts for selected action: MediaKbdPlay

Keyboard/MIDI/OSC Input

Shortcut: <<type key or move controller>>

MIDI CC: Absolute

□ Soft takeover (absolute mode only)

□ Automatically close window on key/MIDI input

Figure 5.12 : The 3 steps for assigning transport command shortcuts

Listing 5.7 shows the initialization of the MIDI message class in the KinectSound system.

```
 1 #include <MMSystem.h>
 2 MidiCommands::MidiCommands(void)
 3 {
 4    MIDIOUTCAPS moc;
 5    CString loopBe, currentDevice;
 6    int loopID = -1;
 7    loopBe.Format("LoopBe Internal MIDI");
 8    for (int i = 0; i < midiOutGetNumDevs(); i++)
 9    {
10        // Get info about the next device
11       if (!midiOutGetDevCaps(i, &moc, sizeof(MIDIOUTCAPS)))
12       {
13          currentDevice.Format("%s",moc.szPname);
14          if (strcmp(currentDevice,loopBe) == 0)
15          {
16             loopID = i;
17             break;
18          }
19       }
20    }
21    if (loopID == -1) return;
22    midiOutOpen((LPHMIDIOUT) &MidiHandleDAW,loopID,NULL,0,0);
23    msgDAW.MidiByte[0] = 0xB0;
24    msgDAW.MidiByte[1] = 0x1A;
25    msgDAW.MidiByte[3] = 0x00;
26 }
```

Listing 5.7: Initializing the MIDI message class.

This function loops through all the MIDI devices that are found on the current workstation (8 - 20) and checks each one's name to see if it is *"LoopBE Internal MIDI"* (13 - 14). If it finds the LoopBE Internal MIDI device, it stores the ID of the device and exits the loop (16 - 17). If the loop ends and a LoopBE internal MIDI device has not been found, the function will exit as it does not have a valid MIDI device to connect to (21). If the Internal MIDI device was found, the KinectSound system will open it as an output device (22) and set the 1st byte of the MIDI message to *0*x*B0* (23), the MIDI Control Change command. The 2nd byte is set to *0*x*1A*, the reserved control command which is used by the KinectSound system (24). The 4th byte of this message is set to *0*x*00* (25) as it is unused by the KinectSound system. Listing 5.8 shows how the KinectSound system configures and sends the MIDI start\stop messages.

```
void *MidiCommands::MidiStartWrapper(void* wrapping)
{
   return ((MidiCommands*)wrapping)->MidiStart();
}
void *MidiCommands::MidiStart(void)
{
   msgDAW.MidiByte[2] = 0x00;
   Sleep(5000);
   midiOutShortMsg(MidiHandleDAW, msgDAW.MidiData);
   return 0;
}
void MidiCommands::MidiStop(void)
{
   msgDAW.MidiByte[2] = 0x01;
   midiOutShortMsg(MidiHandleDAW, msgDAW.MidiData);
}
```

Listing 5.8: Sending start and stop messages to the DAW.

As indicated in Section 5.1.1 there is a 5 second delay before recording or playback begins. This delay can be implemented via a *Sleep* command which necessitates a separate thread of execution. The POSIX Threads library was used for thread creation. This is an IEEE standard which provides an API for creating and manipulating threads [104]. Functions that are created on a new Pthread require a wrapper function as the Pthreads library doesn't allow thread creation on an object's member function. The wrapper function (which calls the *MidiStart* function) is shown in Listing 5.8. The wrapper function is called from the Kinect class via a *pthread_create* command, which creates a new thread for the duration of the function's execution.

## 5.4 Audio Encoding

Chapter 2 described the way in which the audio localization was encoded and stored using XML. The KinectSound system accesses the XML audio localization file and does the following:

- Reads from the file and temporarily stores the audio data to convert into mix levels for the endpoints.

- Makes changes to the data when a track is finished being recorded.

- Saves any changes in the audio localization back to the file.

The methods and mechanisms used to perform each of these tasks is described in further detail in the following subsections.

### 5.4.1 Restoring stored Co-ordinates

The KinectSound system uses the Document Object Model (DOM) for deserializing the audio localization data. The DOM is a cross platform API which interacts with objects in HTML, XHTML, and XML files [105]. The DOM uses the elements and attributes of an XML file and organizes them in a tree structure which allows easy access and manipulation. The KinectSound system uses the DOM to retrieve each *Track* element, and further retrieves each *TimedCoOrd* tag's attributes from each track. The attributes are then stored using a linked list implementation. The deserialization process occurs whenever the user selects the *Record* or *Playback* option in the Recording Interface.

A structure called *mixNode* was created within the Track class to store the attributes of every *timedCoOrd* tag. Each track contains a pointer to a mix node, which stores the attributes of a *timedCoOrd* tag and contains a pointer to the next mix node. This implementation is shown in Figure 5.13

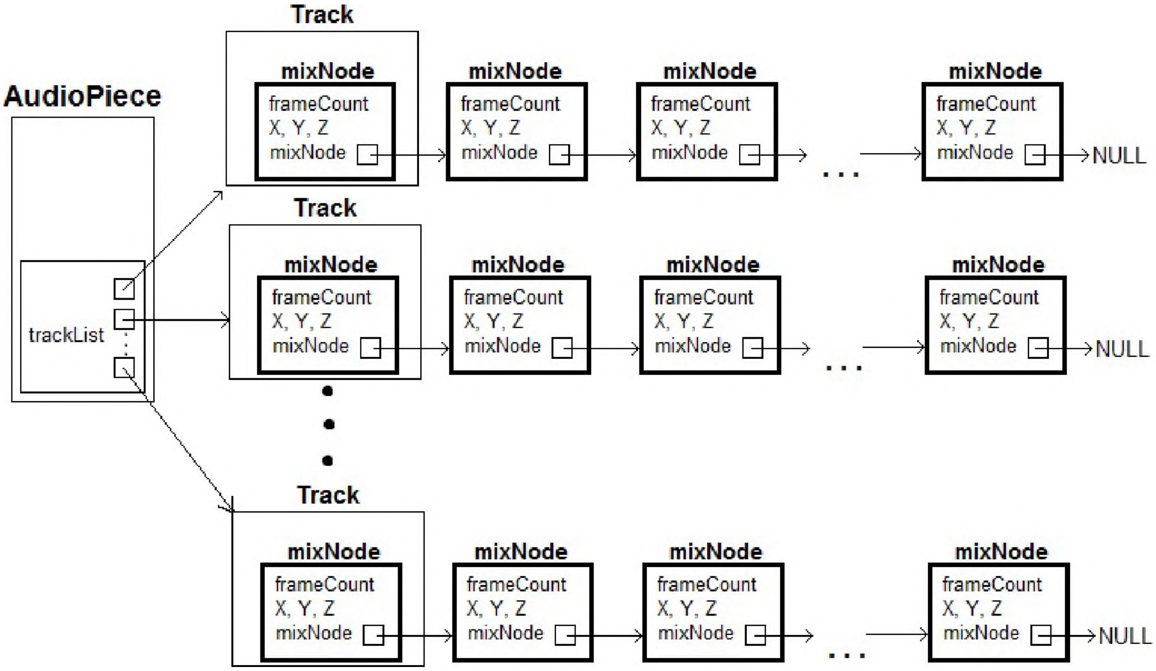


Figure 5.13 : The linked list implementation for storing audio data

The pseudocode for deserializing the audio data is shown in Listing 5.9.

```
DOM -> Initialize
Load <AudioXML>
DOM -> get "AudioPiece" node
DOM-> get number of "Track" nodes
FOR <number of tracks>
    Track->clear list
    DOM-> get all "timedCoOrd" nodes
    FOR <each timedCoOrd node>
        GET "frameCount"
        GET "X"
        GET "Y"
        GET "Z"
        Track -> Add Node <frameCount, X, Y, Z>
    ENDFOR
ENDFOR
```

Listing 5.9: The pseudocode for deserializing the audio data.

### 5.4.2 Saving current co-ordinates

The serialization of localization data in the linked lists to XML happens whenever the user selects the *Stop* option on the Track Recording interface. The sequence diagram for saving co-ordinates is shown in Figure 5.14, followed by the code for serialization in Listing 5.10:

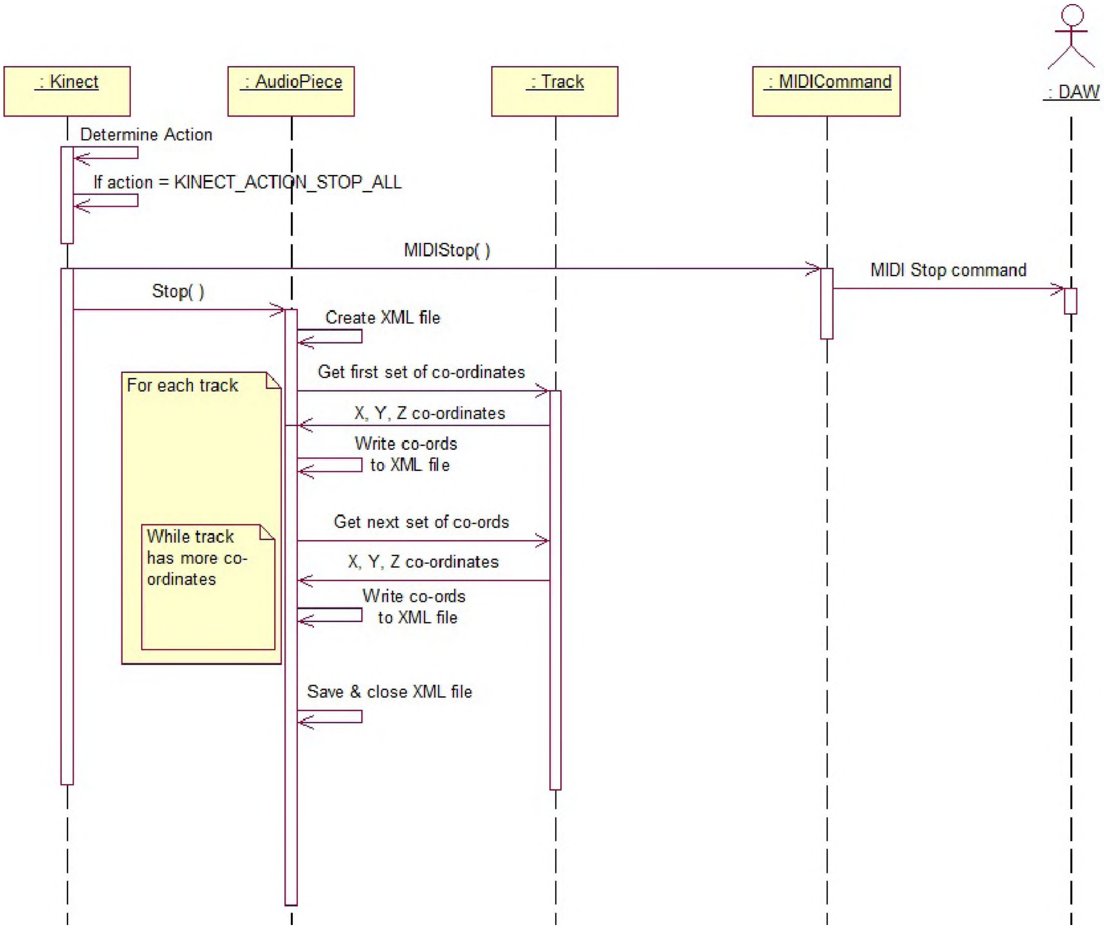


Figure 5.14 : The Sequence diagram for saving audio data

```
 1 void AudioPiece::Serialize(void)
 2 {
 3    FILE* file;
 4    CString filename, lineout;
 5    filename.Format("AudioData2.xml");
 6    file = fopen(filename,"w");
 7    fputs("<AudioPiece>\n", file);
 8    for(int trackLoop = 0; trackLoop < 8; trackLoop++)
 9    {
10       lineout.Format("\t<Track><!--TRACK%i-->\n", trackLoop+1);
11       fputs(lineout, file);
12       trackList.at(trackLoop)->ResetCurrent();
13       lineout.Format("\t\t<TimedCoOrd frameCount=\"%i\" X=\"%.3f\" Y=\"%.3f\" Z=\"%.3f\"/>\n",
              trackList.at(trackLoop)->GetCurrentFrameCount(),trackList.at(trackLoop)->GetCurrentX(
              false),trackList.at(trackLoop)->GetCurrentY(false),trackList.at(trackLoop)->GetCurrentZ(
              false));
14       fputs(lineout, file);
15       while(trackList.at(trackLoop)->LastNode() != true)
16       {
17          trackList.at(trackLoop)->GetNextNode();
18          lineout.Format("\t\t<TimedCoOrd frameCount=\"%i\" X=\"%.3f\" Y=\"%.3f\" Z=\"%.3f\"/>\n",
                 trackList.at(trackLoop)->GetCurrentFrameCount(),trackList.at(trackLoop)->GetCurrentX(
                 false),trackList.at(trackLoop)->GetCurrentY(false),trackList.at(trackLoop)->
                 GetCurrentZ(false));
19          fputs(lineout, file);
20       }
21       fputs("\t</Track>\n", file);
22    }
23    fputs("</AudioPiece>", file);
24    fclose(file);
25 }
```

Listing 5.10: The serializing of audio data and storing it in an XML file.

1. **(3 - 7)** - A *file* is declared and opened using the filename *AudioData2.xml*. The file is opened using the *write* (w) argument, which opens the file and clears its contents.

2. **(8 - 12)** - The function creates a loop which will iterate through each track (8), and within the track, iterate through each mix node and retrieve the co-ordinates. At the beginning of a track iteration, a Track tag is written to the file[1], and the *current* pointer for the linked list is reset (12), which makes it point to the head node for its respective list.

3. **(13 - 14)** - The function formats the *lineout* CString to contain the frameCount, X, Y, and Z values from the head node and write it to the file. The attributes

[1] This also writes the track number in as a comment

for the remaining nodes are formatted and written in the same way.

4. **(21)** - When the loop has reached the last node in the list, a closing *Track* tag is written to the file and serialization will begin for the next track.

5. **(23 - 24)** - When all of the iterations are complete, a closing *AudioPiece* tag is written to the file and the file is closed.

A similar process occurs when the user selects the *Clear Track Data* option. This does an iteration for each track and creates a single *timedCoOrd* tag using a frame count of 0 and the average of the X, Y, and Z endpoint positions for the respective attributes in the XML tag.

## 5.5 3D Audio Panning

The sound localization data for panning is read from an XML file, as explained in Section 5.4. This happens when the system is both in a recording and playback state. The following sections describe the data stored in the XML file, and how this is converted into endpoint specific mix levels. The way in which the KinectSound system communicates with Sketchup is also explained. This communication enables the KinectSound system to display the user's right hand position in three dimensions via Sketchup.

The sequence diagram for recording the track co-ordinates is shown in Figure 5.15. The sequence diagram for the playing process is similar to the recording process, but excludes the real-time audio panning which is shown as the last 4 steps in the recording diagram. The following subsections describe each step in the diagram in further detail.

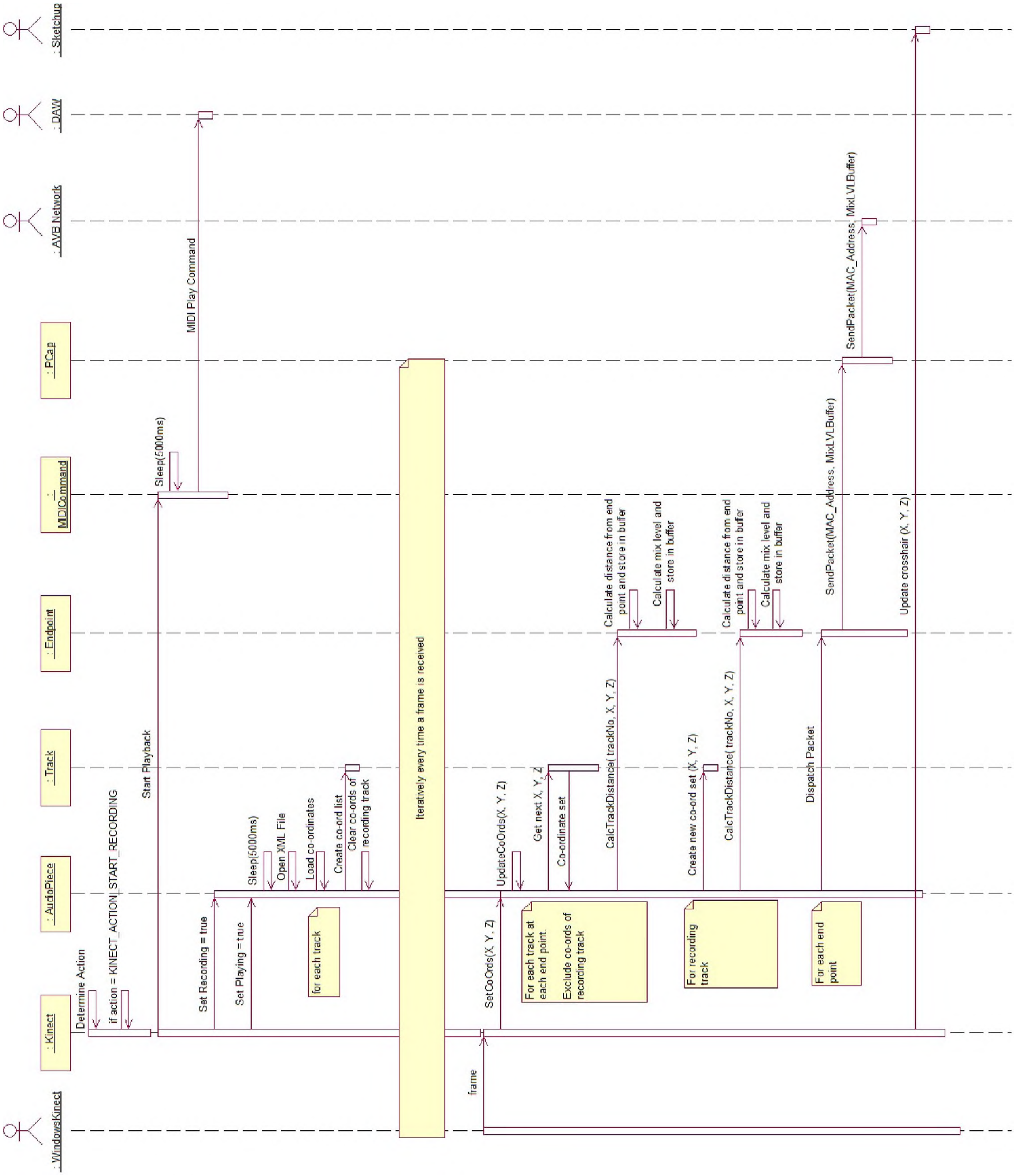

: WindowsKinect
: Kinect
: AudioPiece
: Track
: Endpoint
: MIDICommand
: PCap
: AVB Network
: DAW
: Sketchup
Determine Action
if action = KINECT_ACTION_START_RECORDING
Start Playback
Sleep(5000ms)
MIDI Play Command
Set Recording = true
Set Playing = true
Sleep(5000ms)
Open XML File
Load co-ordinates
Create co-ord list
Clear co-ords of recording track
for each track
Iteratively every time a frame is received
frame
SetCoOrds(X, Y, Z)
UpdateCoOrds(X, Y, Z)
Get next X, Y, Z
Co-ordinate set
For each track at each end point
Exclude co-ords of recording track
CalcTrackDistance( trackNo, X, Y, Z)
Calculate distance from end point and store in buffer
Calculate mix level and store in buffer
Create new co-ord set (X, Y, Z)
For recording track
CalcTrackDistance( trackNo, X, Y, Z)
Calculate distance from end point and store in buffer
Calculate mix level and store in buffer
Dispatch Packet
For each end point
SendPacket(MAC_Address, MixLVLBuffer)
SendPacket(MAC_Address, MixLVLBuffer)
Update crosshair (X, Y, Z)

Figure 5.15 : Recording the user's hand co-ordinates and dispatching the mix levels to the endpoints

### 5.5.1 Converting from Absolute Distance to Relative Distance

The Windows Kinect SDK reads the positions of the users skeleton joints as meter co-ordinate measurements relative to the Kinect's eye. The X co-ordinate is the horizontal displacement from the center of the Kinect's eye. The Y co-ordinate is the vertical displacement from the center of the Kinect's eye. Figure 5.16 shows the absolute X and Y displacement when the user is looking at the Kinect, labelled 'K' in the figure. The Z displacement is towards the reader, perpendicular to the page.

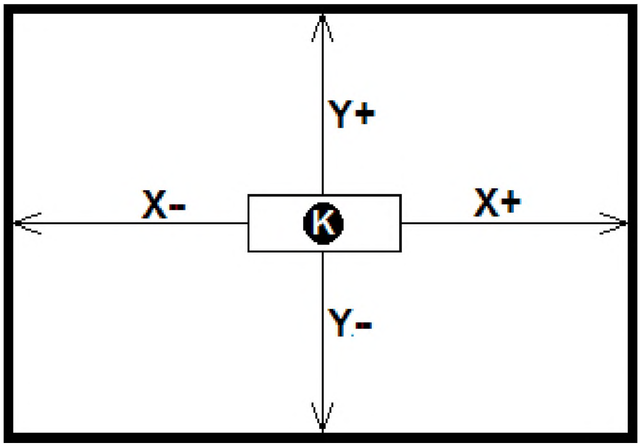


Figure 5.16 : The X and Y displacement from the Kinect's camera eye.

The Z co-ordinate is the displacement from the X / Y plane containing the Kinect eye to the user's joint. These co-ordinates require a scaling factor of 7 so the user is able to pan the sound with movements that are not excessive. The value used for the scaling factor was determined by means of experimentation. A scaling factor that is too high reduces the smoothness of the sound panning. If the scaling factor is too small, it would limit the panning towards the front of the room due to the Kinect's field of vision. When the Kinect cannot see a significant portion of the user's body, it loses tracking accuracy for the joints that it can see. Scaling the co-ordinates also allows the user to pan the sound to the front corners of the listening space which would be unreachable without scaling due to the Kinect's field of vision. An added advantage of scaling is that it allows the user to pan the sound source across the room quicker, as the required hand movement is reduced with the scaling factor. These distances have minimum and maximum values so that the user is not able to pan the sound beyond the listening space. Figure 5.17 shows the effects of scaling the co-ordinates. This figure shows the movement area marked as 'M', which limits the user to the boundaries of the panning space. If the scaling factor was too low, the front corners of the movement area wouldn't be within the Kinect's vision.

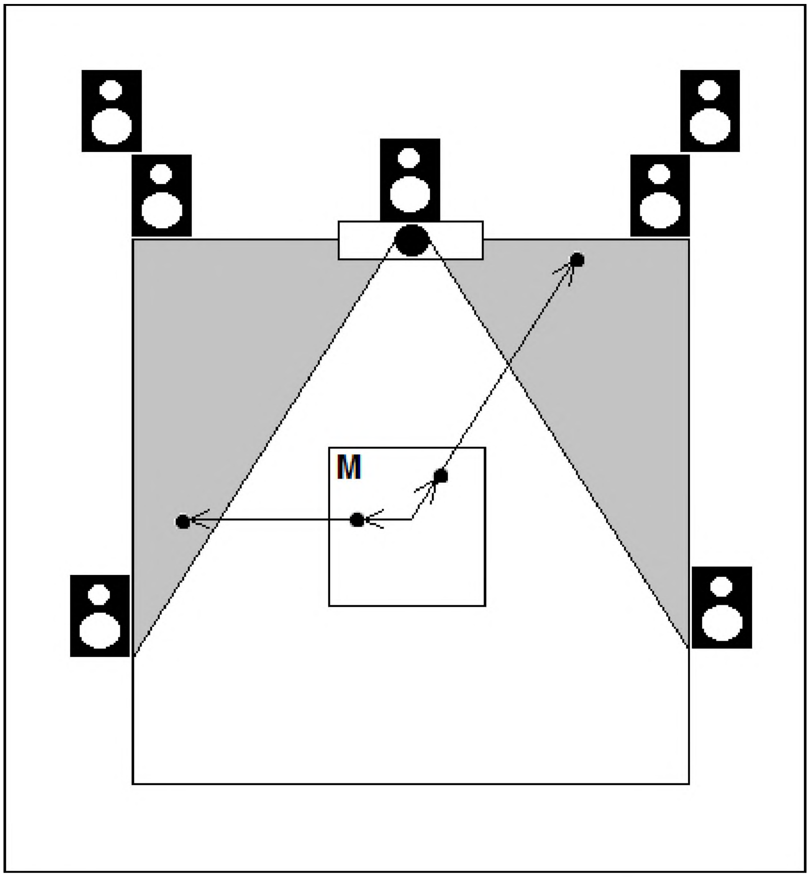


Figure 5.17 : The effects of scaling the Kinect's co-ordinates.

Once the distances are scaled, the KinectSound system sends a message with the scaled values to Sketchup. Sketchup then uses these values to display a crosshair at the position of the co-ordinates.

The X and Y co-ordinate values then have an offset of 1500mm and 1000mm added to them respectively, transforming them to distances from the KinectSound Origin, the bottom left corner at the front of the movement area where X = 0, Y = 0, and Z = 0. This is done so that the distance from the skeleton joint of the user's right hand to each endpoint may be calculated for localization purposes. The position of each endpoint is stored relative to the KinectSound system's origin. The minimum and maximum values for each axis in the KinectSound system are given in millimeters by:

- [ 0mm $\leq$ ( $X_s$ + 1500mm ) $\leq$ 3000mm ]
- [ 0mm $\leq$ ( $Y_s$ + 1000mm ) $\leq$ 2000mm ]
- [ 0mm $\leq$ $Z_s$ $\leq$ 3000mm ]

The origins and scaled ranges for the Kinect (A) and the KinectSound system (B) are shown in Figure 5.18. The origin used for the Sketchup display is similar to the Kinect's origin with the exception that it's Z value ranges between 1500mm and -1500mm.

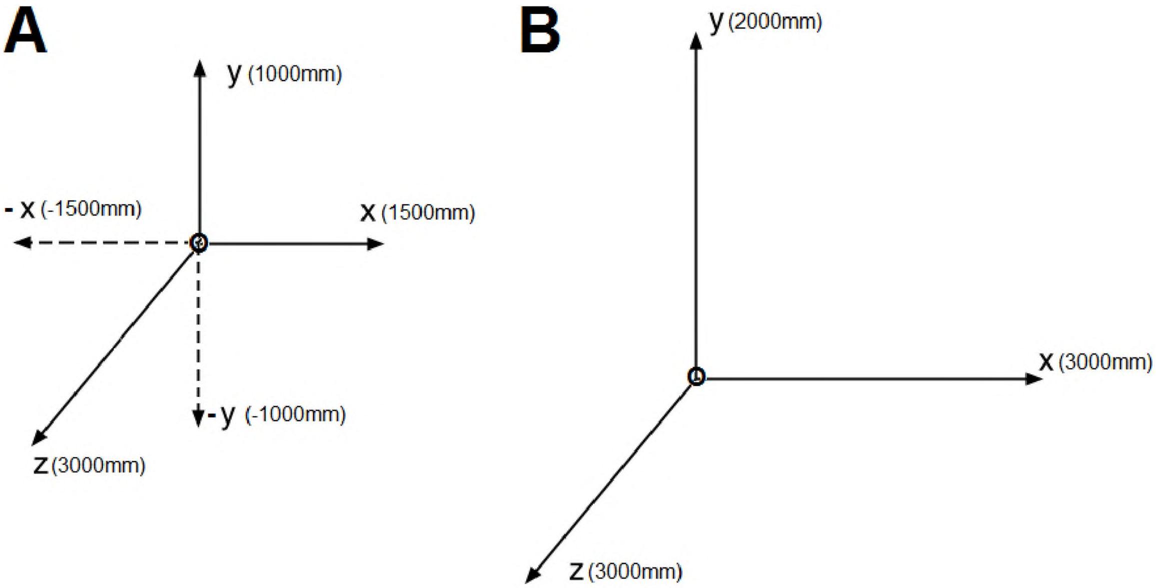


Figure 5.18 : The origins and ranges of the Kinect's origin (A) and the KinectSound system's origin (B).

The X and Y co-ordinates obtained from the Kinect API are scaled by applying a scaling factor of 7 to their current values, converting the co-ordinates to millimeters, and then adding the offset to the scaled value. If the scaled value is beyond the room's boundaries, then the maximum value is used.

The Z co-ordinate scaling has to have a pre-determined point between the user and the Kinect's eye which serves as the midpoint of the room along the Z axis. The mid-point value that is used in the KinectSound system is 1.5m from the Kinect's eye. With this distance, the Kinect provided accurate detection and the user is able to pan between all the areas of the listening environment. If this distance is increased, the scaling factor would also need to be increased to allow the user to pan between the minimum and maximum X and Y values (refer to Figure 5.16).

Examples of the the Z co-ordinate calculations are shown below, where $Z_a$ is the actual distance obtained from the Kinect API, and $Z_s$ is the scaled value of Z, both shown in meters. This allows the KinectSound system to convert a distance of the user from the Kinect's eye to a value between 0mm and 3000mm which is used for localizing sound:

When $Z_a = 1.5$m, $Z_s = 1.5$m, the Z-origin and center of the movement space
When $Z_a = 1.5\text{m}-\frac{1.5m}{7} = 1.286$m, $Z_s = 0$m, where 7 is the scaling factor
When $Z_a = 1.5\text{m}+\frac{1.5m}{7} = 1.714$m, $Z_s = 3.0$m, where 7 is the scaling factor

Figure 5.19 shows the movement zone (M) with the minimum and maximum values of both absolute ($Z_A$) and scaled ($Z_S$) values of Z.

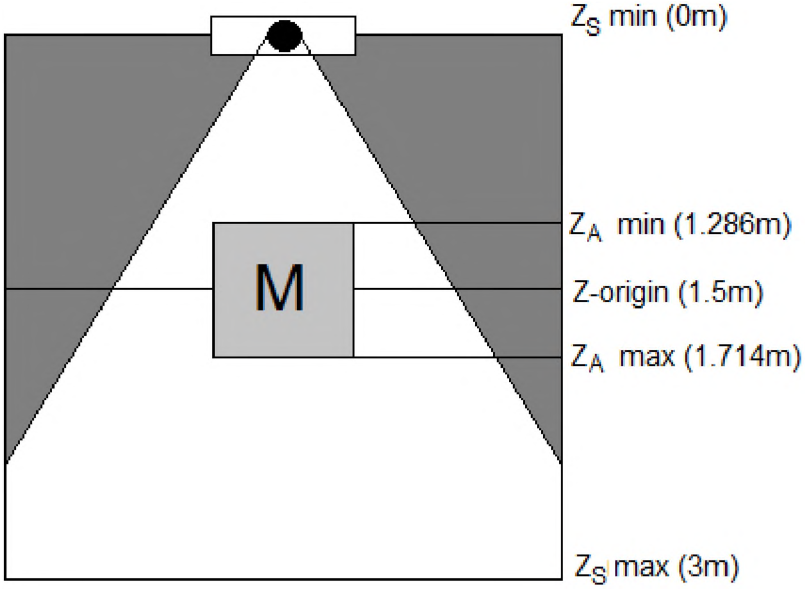


Figure 5.19 : The absolute and scaled values of Z used for panning.

The minimum and maximum values of the Z co-ordinate of the user's hand relative to the Kinect's eye are 1.286, and 1.714, and these are converted to values between 0mm and 3000mm. If the user's hand is beyond the minimum or maximum distance, the value of 0mm or 3000mm will be used, respectively. The following equation is used to calculate the scaled Z value ($Z_s$):

$$Z_s = \frac{Z_a - 1.286}{0.428} * 3000, \text{ where } [\ 1.286 \leq Z_a \leq 1.714\ ] \qquad \text{- Eq. 5.1}$$

The code for performing co-ordinate scaling for the user's right hand in the Kinect-Sound system is shown in Listing 5.11. The offset addition for the X and Y values is explained further in Section 5.5.2

```
1  JointVec = skeleton.SkeletonPositions[NUI_SKELETON_POSITION_HAND_RIGHT];
2  JointVec.x = min(max(JointVec.x*1000*7, -1500), 1500);
3  JointVec.y = min(max(JointVec.y*1000*7, -1000), 1000);
4  CString rightH;
5  rightH.Format("%i:%i:%i",(int)(JointVec.x),(int)(JointVec.y),
6      //Z - multiply by scaling factor of 7 and convert to mm, apply Sketchup display offset (1500*
           scaling factor)
7     min(1500,max(-1500,(((int)(JointVec.z*7*1000)-(1500*7))*-1))));
8  char* msg = rightH.GetBuffer(rightH.GetLength());
9  SC->SendDataTest(msg);  //Sketchup message sent
10 JointVec.z = min(max((((JointVec.z-1.286)/0.428)*3000), 0), 3000);
```

Listing 5.11: Scaling the right hand's co-ordinates.

The code in Listing 5.11 does the following:

1. Gets the current position of the user's right hand from the Kinect API (1).

2. Performs X and Y co-ordinate scaling and checks whether these values are within the boundaries and uses the maximum values if they are not(2, 3)

3. Declare a *CString* variable (4) which is formatted as a message (5-7) that will be sent to Sketchup. The Z value requires a display offset to be subtracted from it so the Kinect's co-ordinate origin (1.5m from the eye) corresponds to the center of the Sketchup venue being displayed.

4. The message generated by the *CString.format* function is converted to a character array named *msg*(8).

5. The *SendDataTest* function is called from the socket connections (SC) object which sends a string over a socket that Sketchup is listening on (9).

6. The value of Z is converted to the value used for localizing the audio by using the equation shown above(10).

Following the co-ordinate scaling, the offset values are added and the appropriate endpoint mix levels are calculated.

### 5.5.2 Converting Distance To Mix Levels

To calculate the mix levels for each track at an endpoint, the KinectSound system is required to calculate the distance between the sound source and each endpoint.

When the KinectSound system is in a recording or playing state, it will send control packets containing a set of mix levels (one for each track) to each endpoint every time it receives a frame from the Kinect. This ensures that the control packets are dispatched at regular intervals.

The KinectSound system creates an instance of the *Endpoint* class for each endpoint. Each instance of the Endpoint class contains the location of the endpoint as a distance in meters from the KinectSound Origin, and the Ethernet MAC address of the endpoint. This enables each instance of the Endpoint class to calculate the distance between the sound source and itself. Mix levels are calculated using this distance value and dispatched to the endpoint using the endpoint's MAC address. The procedure for calculating the mix levels can be grouped into two sub-functions. Each one is explained below with the programming code that perform each function.

### Function 1 - Update current co-ordinates using latest Kinect frame

When the KinectSound system receives a frame from the Kinect, the X, Y, and Z co-ordinates in the *AudioPiece* object are updated. Asynchronous transmissions weren't supported by the system so the co-ordinates were updated on receipt of a Kinect frame. The Audiopiece object then uses its current co-ordinates to calculate the mix levels and dispatches the control packet if it is in a recording or playback state. The code for updating the current co-ordinates is shown in Listing 5.12

```
/*      Kinect Class      */
   A->SetCoOrdinates(JointVec.x,JointVec.y,JointVec.z);
/*      AudioPiece Class      */
void AudioPiece::SetCoOrdinates(float x, float y, float z)
{
   this->X = x;
   this->Y = y;
   this->Z = z;
   //Further calls to calculate distances and dispatch packets
}
```

Listing 5.12: The call from the *Kinect* class to update the current co-ordinates.

When the KinectSound system is in the recording state it is simultaneously in the playing state. When the user is recording a track, all of the other tracks will be playing.

### Function 2 - Calculate the distances between the sound sources and endpoints and convert them to mix levels

When the *AudioPiece* object receives a new set of co-ordinates at the start of a frame, it will:

1. Calculate, for each endpoint, the distance from the sound source location to the endpoint for each track at that frame count using the stored data from the track linked lists.

2. If the system is in a recording state, it will use the current right hand co-ordinates as the source location for the track that has been chosen for recording. It will add another node to the track linked list with the new data. This data contains the latest frame count, followed by the current X, Y, and Z co-ordinates of the user's right hand.

The code for this procedure is shown in Listing 5.13

```
//AudioPiece class
void AudioPiece::SetCoOrdinates(float x, float y, float z)
{
   //Update co-ordinates code
      :
      :
      :
      :
   if(this->playing)
   {
      for (int trackLoop = 0; trackLoop < trackList.size(); trackLoop++)
      {
         trackFactorSums[trackLoop] = 1;
         for (int EPLoop = 0; EPLoop < endpoints.size(); EPLoop++)
         {
            if ((trackLoop == recordingTrack-1) && (this->recording == true))
            {
               trackList.at(recordingTrack-1)->NewNode(frameCount,
                  (this->X)+1500.0,
                  (this->Y)+1000,
                  this->Z);
               frameCount++;
               trackFactorSums[trackLoop] += endpoints.at(EPLoop)->CalcTrackDistance(
                  recordingTrack-1,((this->X)+1500.0)/1000,
                  ((this->Y)+1000)/1000, (this->Z)/1000);
               continue;
            }
            trackFactorSums[trackLoop] += endpoints.at(EPLoop)->CalcTrackDistance(
               trackLoop, trackList.at(trackLoop)->GetCurrentX(true)/1000,
               trackList.at(trackLoop)->GetCurrentY(true)/1000,
               trackList.at(trackLoop)->GetCurrentZ(true)/1000);
         }
33          for (int EPLoop = 0; EPLoop < endpoints.size(); EPLoop++)
34          {
35             endpoints.at(EPLoop)->CalcMixLevel(trackLoop, sqrt(trackFactorSums[trackLoop]));
36          }
37       }
38       for (int EPLoop = 0; EPLoop < endpoints.size(); EPLoop++)
39       {
40          endpoints.at(EPLoop)->DispatchPacket();
41       }
42       for (int i = 0; i < 8; i++)
43       {
44          if ((i == recordingTrack-1) && (this->recording == true)) continue;
45          trackList.at(i)->GetNextNode();
46       }
47    }
48 }
49 //Endpoint class
50 double Endpoint::CalcTrackDistance(int trackNum, float X, float Y, float Z)
51 {
52    distanceBuffer[trackNum] = sqrt(
53       pow(fabs(location[0] - X),2)+
54       pow(fabs(location[1] - Y),2)+
55       pow(fabs(location[2] - Z),2));
56    return (1 / pow(1 + distanceBuffer[trackNum], 4));
57 }
58 
59 void Endpoint::CalcMixLevel(int trackNum, double factorSum)
60 {
61    mixLevelBuffer[trackNum] = 1 / (pow(1+distanceBuffer[trackNum],2)*factorSum);
62 }
```

Listing 5.13: Calculating the distances between the sound locations of each track at each endpoint.

The code in Listing 5.13 determines whether the system is in a playing state (line 5) and if it is, does the following:

1. Iterates through each track at each endpoint (11, 14) to determine the distance between the sound source and the endpoint, and its respective mix level.

2. A *trackFactorSums* array is used (13) to store the factor for each track that is used to keep the sound energy at a constant level.

3. Lines 16 - 24 are executed when the system is also in a recording state and localize the sound source in real time if it is. This will create a new node in the linked list (18 - 21) and store the frame count as well as current co-ordinates of the track being recorded. The X and Y co-ordinates have an offset added to them to take the KinectSound system's origin into account. The frame count is

then incremented(22). The *CalcTrackDistance* member function of the *Endpoint* object is called (23 - 25) for the recording track. This function calculates the distance between a given track and the endpoint that it is associated with and stores it in a buffer (52 - 55). It then returns a value (56) which is added to the *trackFactorSums* array for the track being recorded.

4. Lines 28 - 31 perform the same distance and factor calculations as mentioned above for the tracks that are not being recorded.

5. With the constant energy factor for each track calculated, the KinectSound system calls the *CalcMixLevel* function for each endpoint (33 - 36). This calculates the amplitude factor for a track in an endpoint by implementing the formula (61) described in Section 2.3.4.

### Dispatch of control packets

Upon receiving each successive Kinect frame, the control packets for each endpoint are filled with the amplitude values for each track and dispatched to their respective endpoints. The tracks will be mixed using these values at the endpoints. Listing 5.13 shows the calculation call to dispatch the control packets to each endpoint (38 - 41). The *DispatchPacket* member function calls the *SendPacket* member function of the *PCapFunctions* class and supplies it with the destination MAC address of the packet, and the mix level buffer which used to populate the payload fields of the packet. The mix level buffer is populated with the latest set of mix levels, as described in the previous two functions. The *SendPacket* function is shown in Listing 5.14.

```
int PcapFunctions::SendPacket(u_char *destBoard, u_char *mixValues)
{
   //Create the packet
   u_char packet[50];

   // set MAC destination address to supplied endpoint address
   for(int i = 0;i <=5; i++) packet[i] = destBoard[i];

   // set mac source address to Echo card
   packet[6] = 0x00;
   packet[7] = 0x14;
   packet[8] = 0x86;
   packet[9] = 0x00;
   packet[10] = 0x01;
   packet[11] = 0x44;

   //Set the protocol type to 1722 AVB
   packet[12] = 0x22;
   packet[13] = 0xf0;

   //Set control indicator and AVBTP subtype
   packet[14] = (0x80 | 0x7b);

   //Set AVBTP stream ID, AVBTP Version, and message type
   packet[15] = (0x00 | 0x00 | 0x00);

   // Other static assignments //

   //Set the target GUID (Atterotech Board)
   packet[18] = destBoard[0];
   packet[19] = destBoard[1];
   packet[20] = destBoard[2];
   packet[21] = 0xff;
   packet[22] = 0xfe;
   packet[23] = destBoard[3];
   packet[24] = destBoard[4];
   packet[25] = destBoard[5];

   //Set the controller GUID (
   packet[26] = 0x00;
   packet[27] = 0x14;
   packet[28] = 0x86;
   packet[29] = 0xff;
   packet[30] = 0xfe;
   packet[31] = 0x00;
   packet[32] = 0x01;
   packet[33] = 0x44;

   // Populate the payload
   for (int i = 42; i <= 49; i++) packet[i] = mixValues[i-42];

   //Send the packet
   pcap_sendpacket( adapterHandle, // the adapter handle
      packet, // the packet
      50) // the length of the packet
}
```

Listing 5.14: The *SendPacket* function contained in the PcapFunctions object.

The *SendPacket* function does the following:

1. Creates a “packet” as an array of 50 unsigned characters whcih represent bytes in the packet (4).

2. Uses the supplied MAC address to set the destination MAC address (7).

3. Assign the MAC address of the Echo Card as the source MAC address (10 - 15)

4. Set the protocol type to 1722 AVB (18 - 19)

5. Set the control indicator to true and the AVBTP subtype to AECP (22). The first bit of this byte represents the control indicator (0x80) and the remaining 7 bits represent the AVBTP subtype (0x7b).

6. The 16th byte (25) of the packet contains a valid stream flag (1st bit), an AVBTP version (bits 2 - 4), and a message type (last 4 bits). These are all set to zero which indicates that this packet isn’t part of an AVB stream, and that the message type is an AEM command.

7. Assigns the target GUID (30 - 37) and the source GUID (40 - 47) by using their respective MAC addresses.

8. Uses the supplied mix levels to populate the payload fields (50).

9. Sends the packet via a preselected network adaptor (The Echo card) to the network (53 - 55). This Echo card is selected as the default adaptor on startup.

### 5.5.3 Sketchup Display

Sketchup is a 3D modelling tool originally created by Google and is now owned by Trimble [96]. It allows a user to create 3D object models using a set of predefined Sketchup Tools. 3D objects in Sketchup are referred to as *entities*. Sketchup also allows a user to view models from many different aspects by moving the camera around using the mouse or camera tools.

As well as 3D modelling, Sketchup provides an API in the form of “Plugins” which are written by a user and loaded by Sketchup upon startup. Plugins are segments of

Sketchup Ruby code which are saved as files with 'rb' extensions, indicating they are Ruby files. Plugins extend the functionality of Sketchup by:

- Simplifying multi-step operations
- Allowing external operations
- Perform automated actions upon startup

The KinectSound system uses Sketchup to display the location of the user's right hand in it's 3D environment in real-time. The KinectSound system used a TCP socket to establish a connection with Sketchup. Sketchup provides a socket class called SKSocket (Sketchup Socket) which allows basic socket programming functionality, however it is not officially supported and documented [106]. The KinectSound system sends the co-ordinate values in millimeters as a colon separated string containing the scaled X, Y, and Z values respectively.

The origin for the Sketchup model is the center of the panning environment, therefore the co-ordinate values of the crosshair in Sketchup differ from the co-ordinates used for the sound source panning as described in Section 5.5.1. The Kinect plugin for sketchup receives the string mentioned above over the socket and uses the X, Y, and Z values to update the crosshair's position. The flowchart for updating the crosshair is shown in Figure 5.20

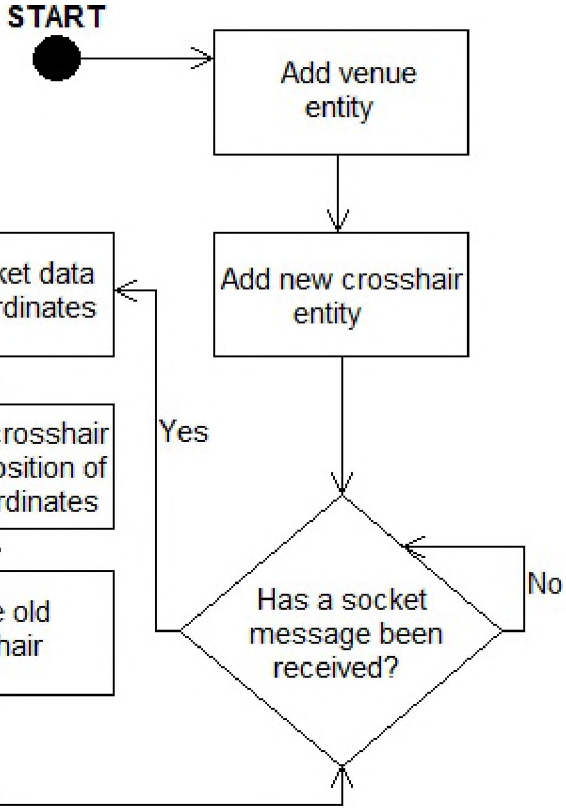

START
Add venue entity
Add new crosshair entity
Split socket data into co-ordinates
Add new crosshair entity at position of new co-ordinates
Delete old crosshair
Yes
No
Has a socket message been received?

Figure 5.20 : The Sketchup procedure for updating the crosshair's position

## 5.6 Ethernet AVB for KinectSound

The KinectSound system uses AVDECC for connecting talker streams to listener endpoints. During implementation, the Extreme bridge was used as a diagnostic tool. This section describes how the Extreme ethernet bridge is used as a diagnostic tool for AVB, and also describes the embedded implementation of the audio mixing on the XMOS endpoints.

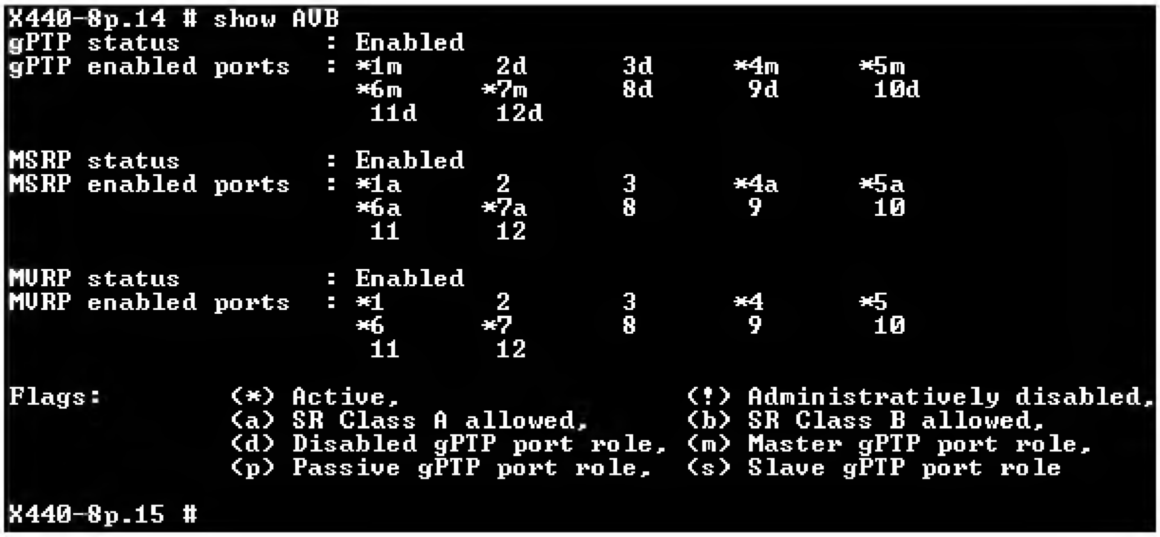
### 5.6.1 Extreme Networks Bridge

Section 4.4.2 has explained the use of the Extreme Networks Ethernet bridge in the Ethernet AVB implementation of the KinectSound system. The default Extreme Operating System (XOS) on the Bridge did not provide an AVB capability. The XOS was updated so the AVB capability could be enabled. This can be done using terminal emulator software and connecting the workstation to the bridge via a Serial to Ethernet connection.

Once the connection to the bridge is established, there are two commands which are used to monitor the network status of the bridge. These are the "*Show AVB*" command, and the "*Show port*" command. The function and usage of these commands is described in further detail in the following subsections.

### AVB status

The AVB status is viewed by typing the command "*show AVB*" as a terminal command. The AVB status shows whether the generalized precision time protocol (gPTP), Multiple Stream Reservation Protocol (MSRP), and Multiple VLAN Registration Protocol (MVRP) are enabled and functional. These protocols were described in Section 4.2.1. If these protocols are enabled, the terminal will furthermore show the ports on the bridge that are using the respective protocols. Figure 5.21 shows the terminal output of a *show AVB* command.



```
X440-8p.14 # show AVB
gPTP status          : Enabled
gPTP enabled ports   : *1m      2d      3d      *4m     *5m
                       *6m     *7m      8d       9d      10d
                        11d     12d

MSRP status          : Enabled
MSRP enabled ports   : *1a      2       3       *4a     *5a
                       *6a     *7a      8        9       10
                        11      12

MVRP status          : Enabled
MVRP enabled ports   : *1       2       3       *4      *5
                       *6      *7       8        9       10
                        11      12

Flags:        (*) Active,                    (!) Administratively disabled,
              (a) SR Class A allowed,        (b) SR Class B allowed,
              (d) Disabled gPTP port role,   (m) Master gPTP port role,
              (p) Passive gPTP port role,    (s) Slave gPTP port role

X440-8p.15 #
```

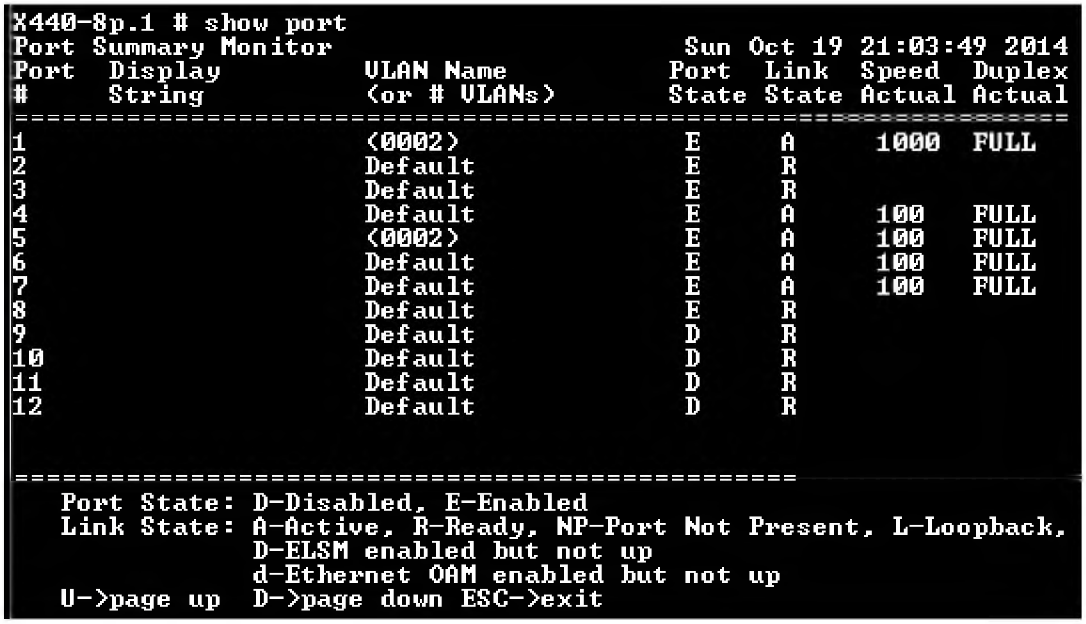
Figure 5.21 : Terminal output of a *show AVB* command.

Figure 5.21 shows that there are AVB capable devices connected to ports 1 (Echo

Card), 4, 5, 6, and 7 (XMOS endpoints)[2]. This command is used as a diagnostic tool to detect which (if any) devices are not being detected by the AVDECC controller, which in this case is the Echo Card connected to Port 1.

### Port Status

The port status of the bridge is viewed by typing the command "*show port*" as a terminal command. The port status shows the information about each port on the bridge which is the VLAN Name, state, link speed, and duplex information. Figure 5.22 shows the terminal output of a *show port* command.



```
X440-8p.1 # show port
Port Summary Monitor                           Sun Oct 19 21:03:49 2014
Port   Display          VLAN Name            Port  Link  Speed  Duplex
#      String           (or # VLANs)         State State Actual Actual
=======================================================================
1                       (0002)                E     A     1000   FULL
2                       Default               E     R
3                       Default               E     R
4                       Default               E     A     100    FULL
5                       (0002)                E     A     100    FULL
6                       Default               E     A     100    FULL
7                       Default               E     A     100    FULL
8                       Default               E     R
9                       Default               D     R
10                      Default               D     R
11                      Default               D     R
12                      Default               D     R


=================================================
   Port State: D-Disabled, E-Enabled
   Link State: A-Active, R-Ready, NP-Port Not Present, L-Loopback,
               D-ELSM enabled but not up
               d-Ethernet OAM enabled but not up
   U->page up  D->page down ESC->exit
```

Figure 5.22 : Terminal output of a *show port* command.

This command shows the 12 ports contained on the X440-8p bridge, of which only 8 are usable by the current configuration. This excludes ports 9 to 12 as they are unpopulated combo ports and do not contain RJ45 jacks. The unpopulated ports are

[2] The XMOS endpoints that are usually connected to ports 2, 3, and 8 were not connected during this screenshot.

disabled and show a 'D' under the *port state* column. By looking at the *Speed Actual* and *Duplex Actual* columns, a user is able to see which ports contain connections to active devices. The *VLAN* column shows the name (or number) of the VLAN that the audio stream is being transported across. The Echo Card (connected to Port 1) is always on the VLAN, and any devices that it is streaming to will be on the same VLAN. Figure 5.22 shows the Echo Card only streaming to a single device, which is connected to Port 5.

### 5.6.2 XMOS Attero Tech Endpoint Implementation

The XMOS endpoints are required to do the processing and mixing of the audio samples to satisfy the distributed processing requirement. The processing component on each endpoint performs the following tasks:

1. Receive and process Ethernet AVB data packets and AVDECC control packets.
2. Perform amplitude adjustment of the 8 audio channels by applying a mix ratio to each one.
3. Mix the adjusted channels and send the resulting sample to a speaker for presentation.

The endpoints provide an Ethernet AVB implementation which is able to process up to 8 channels of audio at 48KHz and transmit them to the $I^2S$ interface [93]. Figure 5.23 shows a high level view of the audio processing within a listener endpoint.

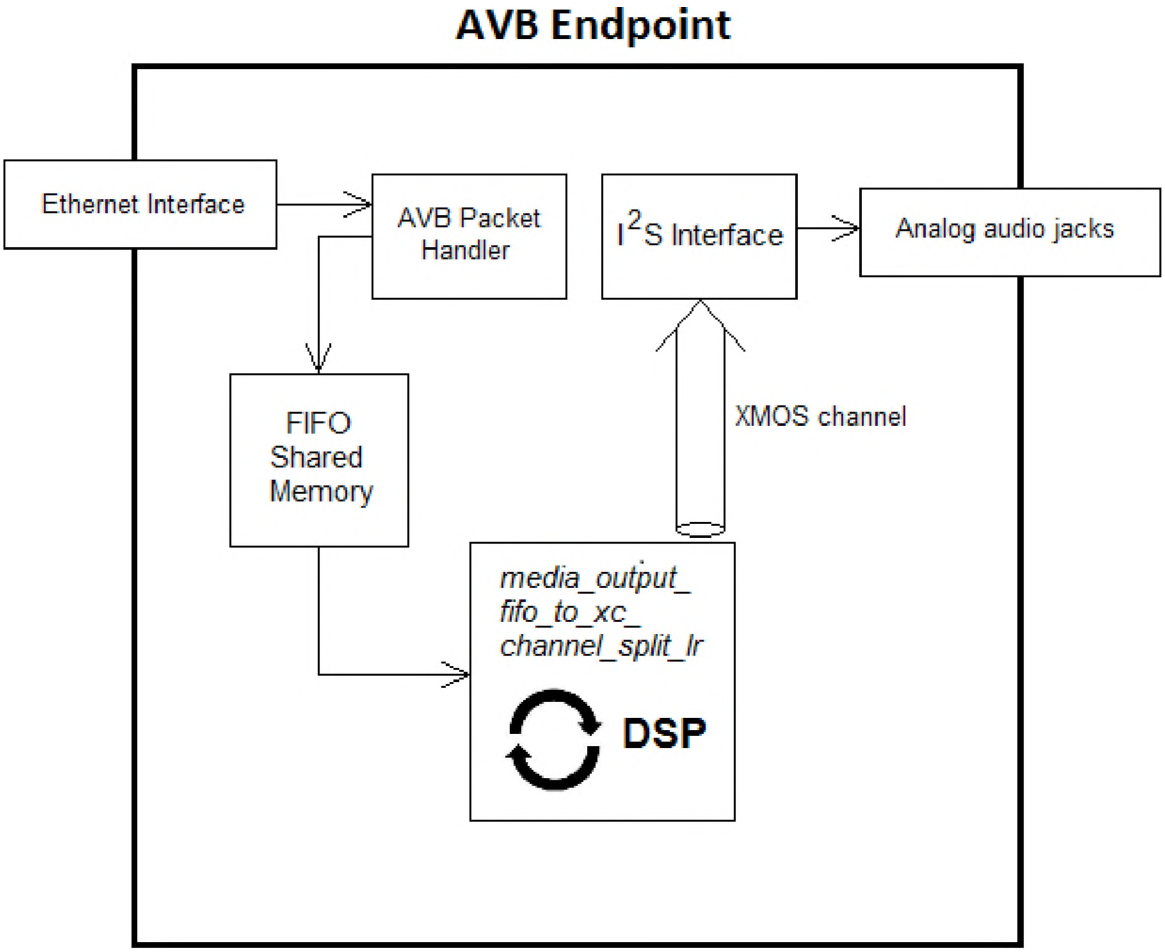


Figure 5.23 : Audio processing within a listener endpoint

The I$^2$S interface is located on processing tile 0 of the XS1 microprocessor and is initialized upon startup. The I$^2$S interface uses buffers to temporarily store any output audio samples that it has received over the XMOS channel, which it then sends to the output component to be presented at the speaker. The output audio data samples received from the AVB network are stored in shared memory FIFOs and can be extracted via a function call *media_output_fifo_pull_sample*. The extraction call is made in the *media_output_fifo_to_xc_channel_split_lr* function. The *media_output_fifo_to_xc_channel_split_lr* function sends the samples to the I$^2$S interface over an XMOS channel.

The AVB implementation also provides a function for handling IEEE 1722.1 connection management and control packets. This enables the connection of a stream between the endpoint and another device on the network.

The firmware on each endpoint was modified to fit the requirements of the KinectSound system. This required the following modifications:

- **DSP functionality** - The *media_output_fifo_to_xc_channel_split_lr* function is enhanced to provide Digital Signal Processing (DSP). The DSP consists of an amplitude adjustment for each sample, and the mixing of the 8 adjusted samples so they are sent out as a single channel. The amplitude is adjusted according to a set of mix levels that are stored within the function. The stored mix levels are updated by a set of mix levels received over an XMOS channel.

- **Disabled channel selection** - The initial AVB firmware received 8 channels of audio in a stream, but would only output 2 of them. The user was able to change which channel pair was sent to the output by pressing the *channel select* GPIO button located on the endpoint. This needed to be disabled as the KinectSound system required all the channels to be mixed and output over a single channel.

- **Control packet handler** - A handler function was added to the 1722.1 control packet handler within the *demo* function on tile 1. This function is called when an AVDECC control packet containing an *AEM_COMMAND* is received. Each AEM command network packet contains a set of 8 mix levels. The mix levels are extracted and used to update the set of mix levels within the DSP function.

The DSP and control functions run in parallel and use XMOS channels to send data to each other. Figure 5.24 shows the data flow between the *media_output_fifo_to_xc_channel_split_lr* (Tile 1) and *demo* (Tile 0) functions which were modified for the KinectSound system. Each element of the data flow is labelled and described below the figure:

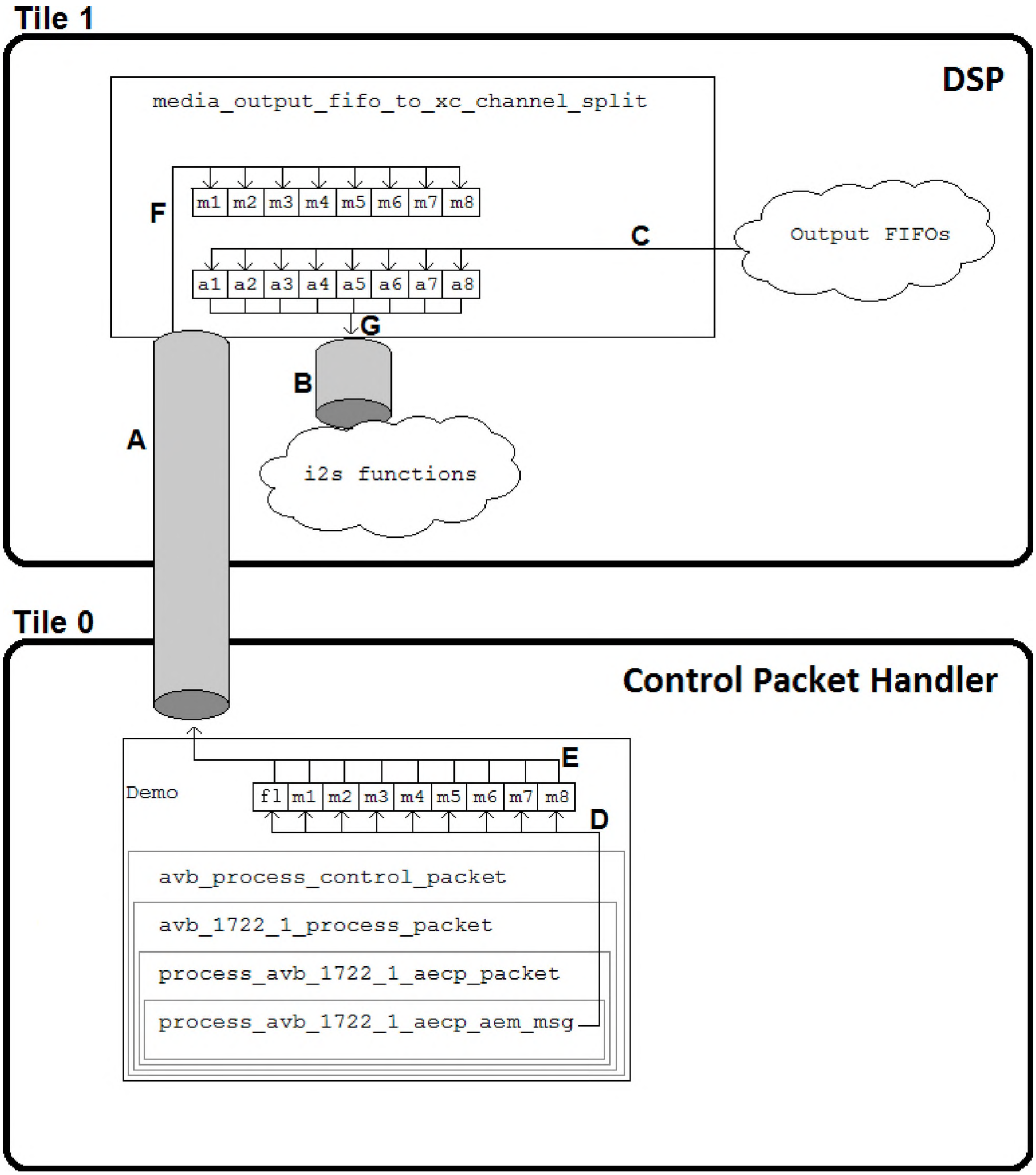

Tile 1
DSP
media_output_fifo_to_xc_channel_split
F
m1
m2
m3
m4
m5
m6
m7
m8
C
Output FIFOs
a1
a2
a3
a4
a5
a6
a7
a8
G
B
A
i2s functions
Tile 0
Control Packet Handler
E
Demo
f1
m1
m2
m3
m4
m5
m6
m7
m8
D
avb_process_control_packet
avb_1722_1_process_packet
process_avb_1722_1_aecp_packet
process_avb_1722_1_aecp_aem_msg

Figure 5.24 : The data flow between the modified functions and their respective processing tiles.

A. The channel between the two tiles is used to send the set of mix levels extracted from the control packet to the DSP function. The DSP function stores the latest set of mix levels it has received.

B. The channel between the DSP function and the I$^2$S functions is used to send the mixed audio data from the DSP function for presentation.

C. The *media_output_fifo_pull_sample* function call which updates the set of audio samples. This happens at the beginning of the *media_output_fifo_to_xc_channel_split_lr* function so that it is able to acquire the latest set of audio samples before beginning the DSP.

D. When the demo function calls the *avb_process_control_packet* function, it passes it an array by reference. This array's values are updated when the mix levels are extracted from the control packet. The array also contains a flag (fl) which indicates when the mix levels are ready to be sent across the channel to the DSP function. The payload extraction is a result of a series of function calls from the *demo* function, where each function makes the next call depending on an attribute within the packet's header. Each function is described below in the order that they are called. The attribute shown in brackets at the end of each function's description is the attribute identified in the KinectSound system's packet headers which determines the next call.
- *demo*, receives incoming packets and tests for AVB control packets (Control indicator).
- *avb_process_control_packet*, checks what type of control is implemented in the packet (IEEE 1722.1).
- *avb_1722_1_process_packet*, uses the *AVBTP Subtype* field to determine which AVDECC sub protocol is being called (AECP).
- *process_avb_1722_1_aecp_packet*, determines the purpose of the message by checking the *Message Type* field (AEM_COMMAND).
- *process_avb_1722_1_aecp_aem_msg*, extracts the packet's payload and updates the flag and values in the array.

E. When the flag is set to *MIX_LEVELS_READY*, the demo function sends the flag's value and the mix level values through the channel to the DSP function.

F. The DSP function updates its set of mix values when it receives a *MIX_LEVELS_READY* flag from the channel.

G. When the audio samples have been mixed, they are sent through the channel to the I$^2$S functions for presentation.

The programming code for each of the modifications is given below, along with further explanation.

### 5.6.2.1 Control Packet Handler

The control packet functionality for the AEM command *SET_MIXER* is implemented to extract the set of 8 mix levels from the control packet. This happens in the *demo* function. These mix level values are then sent over an XMOS channel to the DSP function. Figure 5.24 shows the function calls in the *demo* function which are described earlier. Several checks are done for attributes in each of a packet's layered headers in order to determine which function to call as explained above. The lowest level function call, which is the *process_avb_1722_1_aecp_aem_msg* function is responsible for extracting the mix levels from the packet's payload. Listing 5.15 shows the code contained in the demo function on tile 0 that is responsible for initiating the mix level extraction and sending the mix levels over the XMOS channel for the DSP function on tile 1.

```
//in demo function
select
{
   case avb_get_control_packet(c_rx, buf, nbytes): //Check for AVDECC control packet
   {
      avb_process_control_packet(buf, nbytes, c_tx, mixLevels); //First call to process the control packet
      if (mixLevels[0] == MIX_CHANNEL_READY) //Check if there is a new set of values
      {
         c_mix_channel <: MIX_CHANNEL_READY; //Send flag through XMOS channel
         for (int i = 1; i < 9; i++)   c_mix_channel <: mixLevels[i]; //Send mix levels through XMOS channel
      }
      mixLevels[0] = MIX_CHANNEL_IDLE; //Set flag back to idle
      break;
   }
   //Other select events
}
```

Listing 5.15: The demo function on the endpoints.

The select statement checks for the receipt of control packets. It may also include checks for events such as GPIO interaction. The case statement (4) shown in Listing 5.15 shows the events that occur when a control packet is received, these are:

1. The *avb_process_control_packet* function (6) determines the type of packet and will call any further functions depending on the type. This function is supplied with an array, *mixLevels*, which is passed by reference. This array contains 9 integer values that consist of a flag in position 0, followed by 8 mix level values. When the mix levels are extracted from the control packet, the array's flag is set to *MIX_CHANNEL_READY* by the extraction function and the remaining values are set to the mix levels extracted from the packet.

2. The *demo* function performs a check for a *MIX_CHANNEL_READY* in the first element of the array (7).

3. If this returns true, it will transmit the *MIX_CHANNEL_READY* value and the 8 mix level values over the channel (9 - 10).

4. Once this has been completed, the flag value in the array is set back to *MIX_CHANNEL_IDLE* (12).

### 5.6.2.2 Digital Signal Processing (DSP)

The DSP is done in the *media_output_fifo_to_xc_channel_split_lr* function before the sample is sent to the I$^2$S buffers. The code for this function is separated and explained in different subsections.

#### 5.6.2.2a Acquire Mix Levels

Listing 5.16 shows the select statement that is used to determine if there is a new set of mix levels to update the set that is currently stored in the function.

```
while (1) {
   select{
   case c_mix_channel :> flag: //If there is a flag value in the channel
      //Extract the first mix level from the channel and store it
      c_mix_channel :> fifoMixVals[0];
         :
         :
      //Update the last value which is also the last value in the channel
      c_mix_channel :> fifoMixVals[7];
               //Fallthrough to DSP after update
   default:
   //DSP
   }
}
```

Listing 5.16: The *select* statement for determining if an update is required.

The function uses a case statement to determine if there is a value in the *c_mix_channel*, the XMOS channel used to transfer the mix levels between the functions. The "*:>*" operator is used to read data from a channel. If a value is present in the channel, it will be the flag value which is always sent first. This value can be discarded as the flag's value only serves as an indicator that 8 mix level values are currently in the channel waiting for extraction. A further 8 values are then read from the channel and stored in the *fifoMixVals* array. This array stores the mix levels for the respective 8 tracks. If a flag is not in the channel, the select statement will go to the DSP section by default.

### 5.6.2.2b Audio Sample Mixing

The DSP section of this function is shown in Listing 5.17:

```
1  while (1) {
2    select{
3      //Mix level update (explained above)
4      default:
5        samples_out :> timestamp; //Get the timestamp from the channel
6        mo_ts = timestamp;
7
8        for (int i=0;i<8;i++) { //Loop to get samples from FIFO
9          unsigned sample;
10         sample = media_output_fifo_pull_sample(output_fifos[i],timestamp);
11         samples[i] = sample;
12       }
13       for (int i=0;i<8;i++){ //Bit shifting to preserve sign
14         samples[i] = samples[i] << 8;
15         samples[i] = samples[i] >> 8;
16         samples[i] = (samples[i]*fifoMixVals[i])/100;
17       }
18       //Mix the samples
19       sampleout = (samples[0])+(samples[1])+(samples[2])+(samples[3])+(samples[4])+(samples[5])
             +(samples[6])+(samples[7]);
20       samples_out <: sampleout; //Send sample out over first channel
21       for (int i=0;i<7;i++) samples_out <: 0; //Send null values for other 7 channels
22     break;
23   }
24  }
```

Listing 5.17: The DSP on audio samples.

Given below is a stepwise explanation of the code:

1. A timestamp from the I$^2$S channel is received and stored (5 - 6). This timestamp is required to get a set of samples from the output FIFO.

2. The function uses a for loop to sequentially extract samples from the output FIFO and store them in the *samples* array. The function for extracting a single sample from the output FIFO returns the sample as an unsigned integer. These are 24-bit values which are represented using 32 bits [93].

3. An arithmetic bit-shift is then performed on each sample which preserves the sign, making each a valid integer value. This allowed the mix values to be added, as they are all signed integers. Each mix level is given as a percentage (1-100) which is used to modify the current sample values to get the final result.

4. A *linear mix* of the modified samples is then sent to the I$^2$S channel for presentation (19/20).

5. A further 7 values of 0 are sent (21) to the I$^2$S channel to indicate that the remaining 7 I$^2$S channels must remain silent as the final mixed product is played over channel 1.

## 5.7 Chapter Summary

This chapter has described the design approach to implementing the KinectSound system. This approach produced the following artefacts:

- Requirement Specification
- Use Case Diagrams
- Class Diagram
- Sequence Diagrams

The remaining sections in this chapter focused on the implementation of the core functionality of the KinectSound system which enabled 3D audio panning across a distributed processing system. The core functionality was described in the following sections:

- **Device Free HCI** - This section explained the features of the Window's Kinect and its ability to detect a skeleton. This ability is used to provide a user with a way to interact with the computer. Several *User Interfaces* are created by the KinectSound system using OpenGL to allow the user an easy way of interacting with the KinectSound system.

- **DAW Control** - This section explained how the DAW uses ASIO drivers and channel routing to send a set of tracks onto an Ethernet AVB network. It also describes how the KinectSound system is able to have control over the DAW via MIDI messages. These were used to start and stop the DAW's audio playback.

- **Audio Encoding** - This section explained the mechanisms used in the KinectSound system to store the audio data that is read from the external file. The audio data is stored in an external XML file which the KinectSound system locates and deserializes into a linked list implementation when it enters a recording or playback state. When the user stops recording or playing the audio, the KinectSound system serializes the audio data from the linked lists and saves it to a new XML file. This updates any recording changes that have been made.

- **3D Audio Panning** - This section explained how the KinectSound system uses the user's right hand co-ordinates from the Kinect to create a point in the 3D environment. These co-ordinates were converted to two different values which are used for locating the sound, and for displaying the position of the user's right hand in Sketchup. The KinectSound system uses a Distance Based Amplitude Panning (DBAP) method to calculate the mix levels for each track at each endpoint. The mix levels are sent to each endpoint via AVB control packets. These packets are sent at regular intervals which occur every time a frame is received from the Kinect.

- **Ethernet AVB for KinectSound** - This section described how the Extreme Networks Bridge is used as a diagnostic tool and how the KinectSound system uses AVDECC for the connection management and control of each endpoint. The Extreme Networks Bridge has an operating system (XOS) which allows the user to view and change certain attributes of the switch. This can be done through terminal emulator software with the Bridge connected to an RS232 port.

  This section also described the audio processing component of each endpoint. The endpoints use two separate tiles for processing the audio data and control data. The tiles are able to send data between each other by using XMOS channels. Each endpoint uses a stored set of mix levels to perform amplitude adjustment on the sets of audio samples that are received from the network. The stored mix levels are updated when an AVDECC AECP AEM command packet is received, which contains a new set of 8 mix levels.

# Chapter 6

# System Testing

The functionality and usability of the KinectSound system was tested against a commercially available system to see how well it compared. This was done by gathering a sample user base, all with different background experience in HCI, computer programming, and/or sound production. These users were to test the KinectSound system against the Spatial Audio Designer (S.A.D) mentioned in Chapter 3.

The KinectSound system used the standard inverse square law for sound localization as the need for energy normalization was only determined after the listening tests were conducted.

There were both qualitative and quantitative measurements taken in order to compare each system. Using these measurements, a comparison of the *Five E's* of usability (Chapter 3) can be made. These comparisons are discussed further in the chapter.

## 6.1 The Testing Process

Before commencing the test, each user was provided with a summarized version of the user manuals for each system, shown in Appendix 8.4[1]. These contained a detailed description of how to perform the actions required for the two different tasks. The user had to read each manual and then perform two tasks with each system. Upon the completion of both tasks, the user was required to answer a set of questions and provide any additional feedback which related to their experience with each system. The feedback forms were formulated by the tester [107] [108] and given to the user before the test.

While the user was completing each task, their actions were observed as well as having

[1] The full KinectSound system user manual is shown in Appendix 8.5

the tasks timed. The timing data gave an indication of how easily the tasks were able to be accomplished given that the user had never seen either of the systems before. The tests were simplified in that the user was not required to perform any technical tasks, such as configuring audio streams and the DAW. The users were also informed about which system they were going to test first. This changed with each test so that half of the users had tested the S.A.D system first, while the other half tested the KinectSound system first. This measure was taken to ensure that the results weren't skewed in favour of either system.

### 6.1.1 The Tasks

Task 1 required the user to move a sound source within a three dimensional space. The audio piece used for testing consisted of four tracks loaded into the Reaper DAW. The tracks could be played, stopped, and muted from both the DAW and the KinectSound system. The tracks were listed as:

1. Drums
2. Guitar
3. Rythm
4. Vocals

The user was required to mute the *Rythm* track (Track 3) and select the *Vocals* track (Track 4) for recording the sound source locations. Once this track setup had been achieved, the user was able to begin recording. The task required the sound source to be located at the following locations for roughly 7 seconds each (with exception to part 9, as it requires a different time):

1. Middle of the room
2. Front Left speaker
3. Front Right speaker

4. Front Right Height speaker
5. Front Left Height speaker
6. Back Left speaker
7. Front Center speaker
8. Back Right speaker
9. 20 seconds of any movement around the room

The user was able to use the time displayed by the DAW or the KinectSound system's recording interface in order to time the movements. The locations listed above allowed the user to move the sound source to all the speakers in the current configuration (Figure 2.8). The 20 seconds of free movement allowed the user to test any specific movements that they wished to. Some of the users wished to test a system for a second time in order to test a certain capability such as panning between speakers or the playback of their recorded track, as this wasn't included in task 1.

Task 2 required the users to reset the recorded co-ordinates of the system so that the whole audio piece was to play from the center of the room. This was done by simply deleting any recorded data from the previous task, and in the case of the S.A.D, moving the *Vocals* track to the center of the listening space.

### 6.1.2 Qualitative Questionnaire

The qualitative questionnaire was provided to the users before testing, and contained questions which covered various broad areas of the system, which pertain to the *Five E's* of system usability. Each question asked the user to give a rating between 0 (least) and 10 (most). The questionnaire consisted of 8 questions, listed below:

1. How complicated did the system look at first sight?
2. How hesitant were you to start using the system?
3. How well did the manuals describe the procedures?

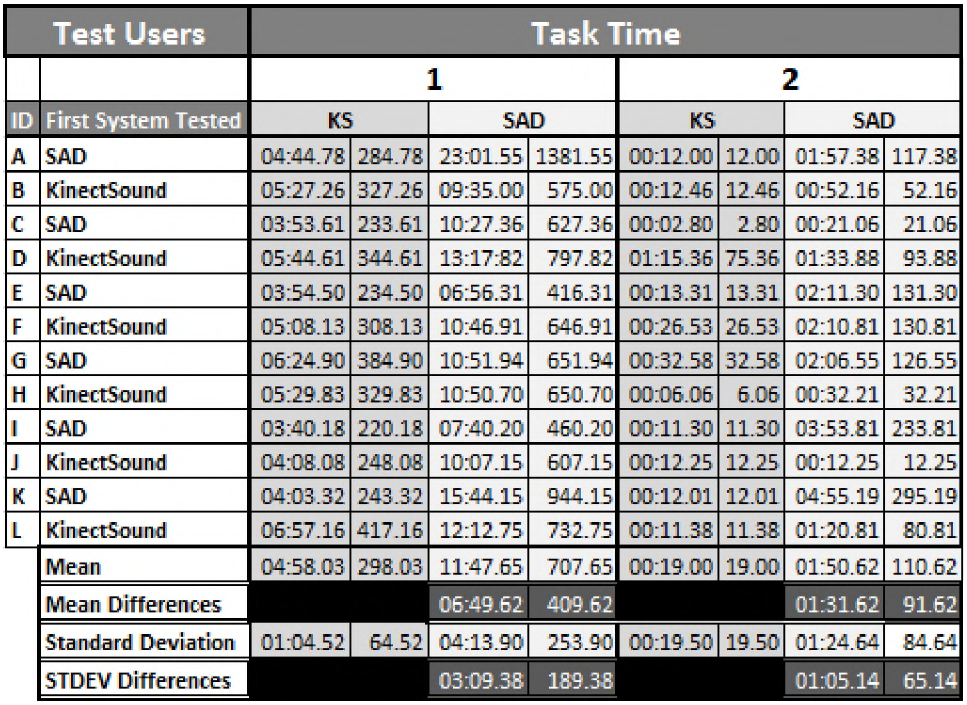
4. How easy was it to accomplish Task 1 (record a track)?
5. How easy was it to accomplish Task 2 (reset tracks to default locations)?
6. How easy was it to pan between the surround speakers?
7. How satisfied were you with the final sound product (mixed sound from each speaker)?
8. Please rate the quality of the audio coming from the speakers (noise, sound glitches, etc.)?

Questions 1 and 2 cover the *Engaging* component of system usability. The ratings given by these questions give an indication of the user's first impression of the system. Questions 3, 4, and 5 cover the *Easy to learn* component of system usability. These questions allowed the user to give feedback about how intuitively simple each task was to perform, as well as how well the manual described each task. Systems that are less intuitively easy to use often require a detailed set of instructions before the user is able to use them. Questions 6 and 7 cover the *Effectiveness* component of system usability. Effectiveness is regarded as the most important component as it pertains to task completion. Sound source panning and appropriate mixing at the endpoints leading to localization were the goals of the system, and this is what these questions refer to. Question 8 covers the *Error tolerance* component of system usability. Errors in systems of this nature would typically relate to sound reproduction errors.

## 6.2 Quantitative results

The quantitative results consisted of the times taken by the users to complete each task for each system. The time was started from the user's first interaction with the system, and stopped when they had completed the task. Once the user had completed Task 1, they were asked to briefly wait before they began Task 2. The time for Task 2 was taken from the state the system was in after panning to the time it took the user to reset the position of the recorded tracks. These results are a measure of the *Efficiency* component of system usability, as this refers to the time taken to complete a specific task.

Table 6.1 shows the quantitative results from the user tests.



| Test Users | | Task Time | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | 1 | | | | 2 | | | |
| ID | First System Tested | KS | | SAD | | KS | | SAD | |
| A | SAD | 04:44.78 | 284.78 | 23:01.55 | 1381.55 | 00:12.00 | 12.00 | 01:57.38 | 117.38 |
| B | KinectSound | 05:27.26 | 327.26 | 09:35.00 | 575.00 | 00:12.46 | 12.46 | 00:52.16 | 52.16 |
| C | SAD | 03:53.61 | 233.61 | 10:27.36 | 627.36 | 00:02.80 | 2.80 | 00:21.06 | 21.06 |
| D | KinectSound | 05:44.61 | 344.61 | 13:17:82 | 797.82 | 01:15.36 | 75.36 | 01:33.88 | 93.88 |
| E | SAD | 03:54.50 | 234.50 | 06:56.31 | 416.31 | 00:13.31 | 13.31 | 02:11.30 | 131.30 |
| F | KinectSound | 05:08.13 | 308.13 | 10:46.91 | 646.91 | 00:26.53 | 26.53 | 02:10.81 | 130.81 |
| G | SAD | 06:24.90 | 384.90 | 10:51.94 | 651.94 | 00:32.58 | 32.58 | 02:06.55 | 126.55 |
| H | KinectSound | 05:29.83 | 329.83 | 10:50.70 | 650.70 | 00:06.06 | 6.06 | 00:32.21 | 32.21 |
| I | SAD | 03:40.18 | 220.18 | 07:40.20 | 460.20 | 00:11.30 | 11.30 | 03:53.81 | 233.81 |
| J | KinectSound | 04:08.08 | 248.08 | 10:07.15 | 607.15 | 00:12.25 | 12.25 | 00:12.25 | 12.25 |
| K | SAD | 04:03.32 | 243.32 | 15:44.15 | 944.15 | 00:12.01 | 12.01 | 04:55.19 | 295.19 |
| L | KinectSound | 06:57.16 | 417.16 | 12:12.75 | 732.75 | 00:11.38 | 11.38 | 01:20.81 | 80.81 |
| | Mean | 04:58.03 | 298.03 | 11:47.65 | 707.65 | 00:19.00 | 19.00 | 01:50.62 | 110.62 |
| | Mean Differences | | | 06:49.62 | 409.62 | | | 01:31.62 | 91.62 |
| | Standard Deviation | 01:04.52 | 64.52 | 04:13.90 | 253.90 | 00:19.50 | 19.50 | 01:24.64 | 84.64 |
| | STDEV Differences | | | 03:09.38 | 189.38 | | | 01:05.14 | 65.14 |

Table 6.1 : The times taken for each user to complete both tasks

The users were labelled *A*-*L* as their identities were to remain anonymous. For each time measurement the values are shown in both minutes and seconds for readability and graphing purposes respectively. Shown at the bottom of the figure is the mean time to complete a task, the difference between the means (shown under the S.A.D column), the standard deviation, and the difference between the standard deviations. A visual representation of the times provide a simpler way to compare the times taken to complete the tasks. Figure 6.1 displays a graphical representation of the times taken. This shows the mean time taken for each task as well as the standard deviation for each task. These graphs display the time in seconds.

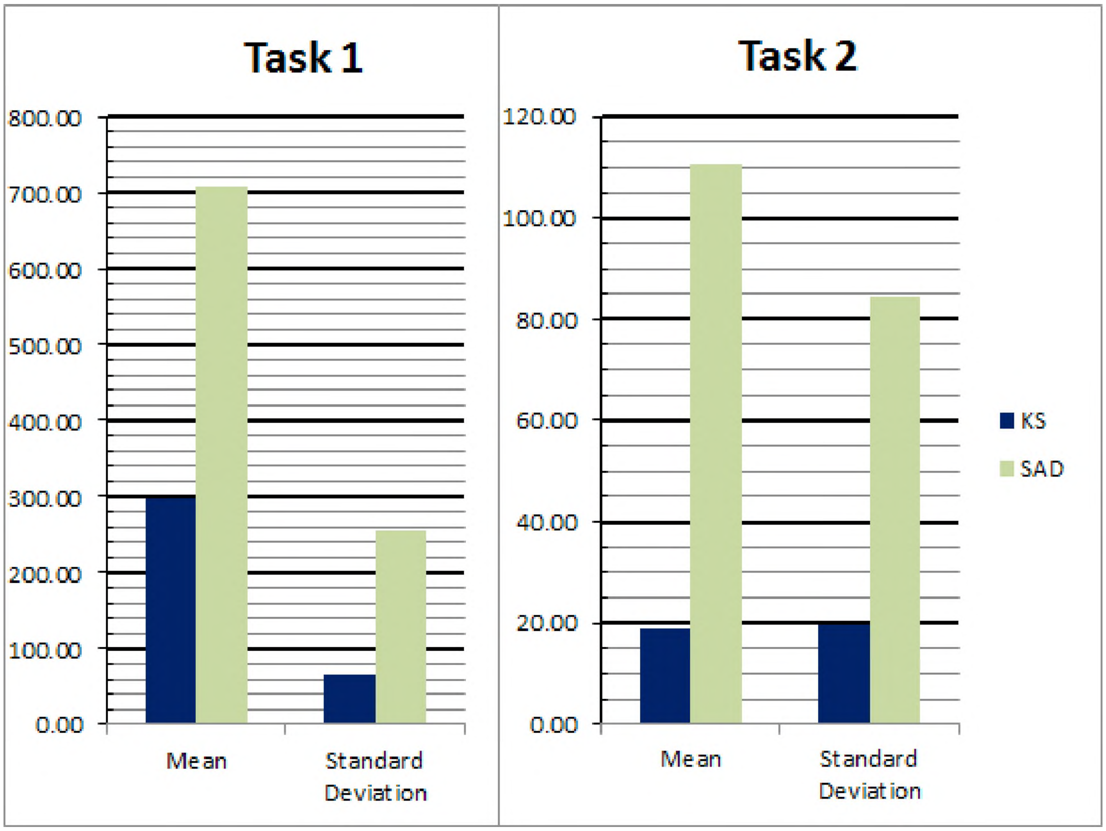


Figure 6.1 : The means and standard deviations of the task times

The results in Figure 6.1 show that the time taken for each task was significantly less when the user was using the KinectSound system to perform the task. The majority of the users struggled to find specific controls on the DAW that were required for the recording process outlined in Task 1 when using the S.A.D. Some users got completely stuck at certain points and had to refer back to the manuals and/or request help from the tester. Users that were more familiar with DAWs and/or had used Reaper before were much faster in completing the tasks. The times for both tasks varied significantly more when using the S.A.D as opposed to the KinectSound system. This can be shown by the *Standard Deviation* column in Figure 6.1. The standard deviation for each task provides a fairly accurate measure of how intuitive each system is due to the difference in mean times. The time taken for the tasks using the S.A.D have a larger variation, which is influenced by factors such as past experience and confidence using the system. Users that had no previous DAW experience or were hesitant to click on any DAW controls took more time to complete each task than users with previous experience. The KinectSound system provides a simple interface which allows users a fast as well as interactive way of learning the system. This provided a more consistent mean, resulting in a lower standard deviation.

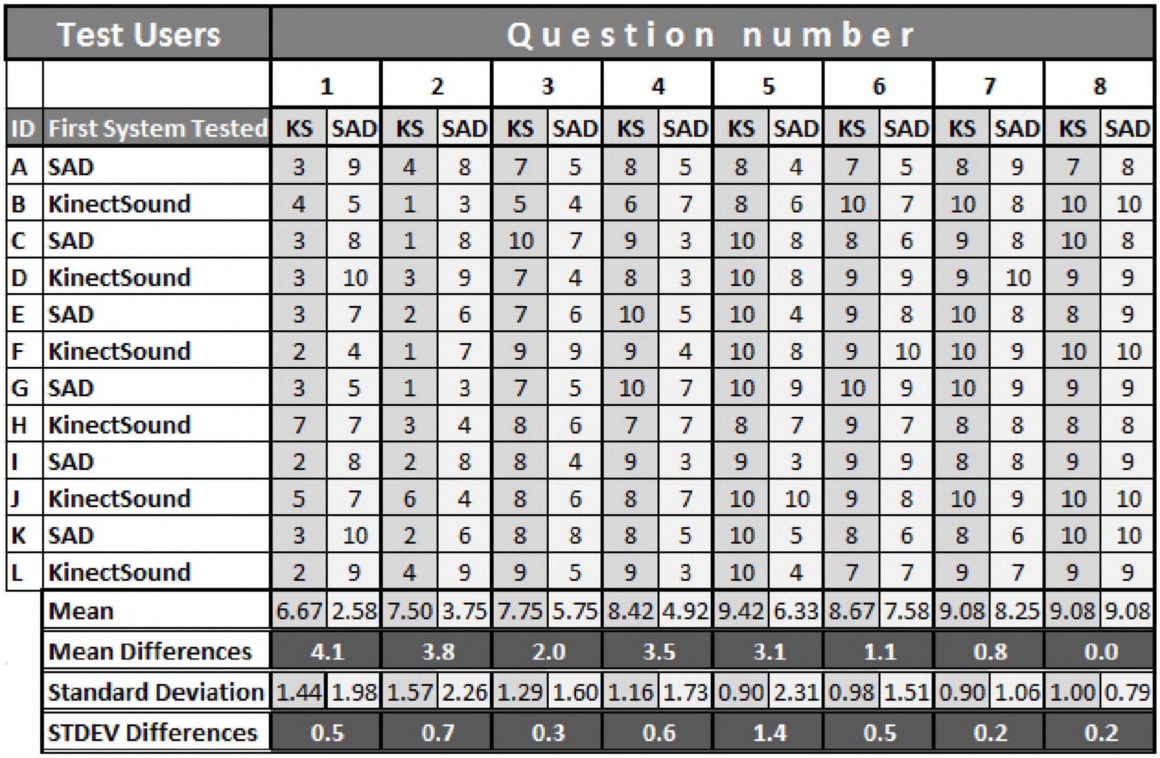
Although none of the users had used S.A.D before, the fact that some were familiar with the controls on the interface of a DAW meant that they were able to easily find the controls required to perform the task. This reduced their time significantly compared to the people that had to refer back to the manual. Since the KinectSound system was completely unseen and relatively intuitive, most of the users were able to complete the tasks with similar times, with the standard deviation just exceeding 60 seconds for Task 1 and being just under 20 seconds for Task 2.

## 6.3 Qualitative results

The qualitative results were derived from the questionnaire and feedback given by the users. The results of the questionnaire have been consolidated and are shown in Table 6.2. Shown at the bottom of Table 6.2 are the mean values, the standard deviations, and the differences between the means and standard deviations of each system. The first two questions of the questionnaire investigate the complexity of the user interface, and the users' hesitance at using the system. In these questions a low score is a positive indicator. In order to keep the graph representations consistent, the *mean* value shown for questions 1 and 2 of both systems in Figure ?? is given by *11* - $\bar{x}$, where $\bar{x}$ is the mean.

Figure 6.2 graphs the mean values of the two systems for each question. This visual representation of the data allows an easier comparison of the feedback for each system.

The quantitative results show that the KinectSound system has better scores than S.A.D for each question. The standard deviation for the KinectSound system was lower than the score of S.A.D for each question, which shows that the KinectSound system's scores are consistently better and that the mean values were not influenced by any outliers. The three greatest differences in the standard deviation values were for questions 2 (hesitancy to use the system), and questions 4 and 5 (ease of task completion). These values were influenced the most by user's with DAW experience as opposed to those that did not have any previous experience. The results from these questions show that the design considerations taken before building the KinectSound system's interface have provided a better user experience, thus influencing the positive scores obtained. Questions 7 and 8 referred to the final mixed audio in which the KinectSound system scored slightly higher for Question 7, and an equal score for Question 8. These scores indicate that the immersive experience from the KinectSound system is slightly better than that of a commercially available system. The results from Questions 7 and 8 also show that the distance based amplitude panning method used to localize sound sources works successfully.



| Test Users | | Question number | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 1 | | 2 | | 3 | | 4 | | 5 | | 6 | | 7 | | 8 | |
| ID | First System Tested | KS | SAD | KS | SAD | KS | SAD | KS | SAD | KS | SAD | KS | SAD | KS | SAD | KS | SAD |
| A | SAD | 3 | 9 | 4 | 8 | 7 | 5 | 8 | 5 | 8 | 4 | 7 | 5 | 8 | 9 | 7 | 8 |
| B | KinectSound | 4 | 5 | 1 | 3 | 5 | 4 | 6 | 7 | 8 | 6 | 10 | 7 | 10 | 8 | 10 | 10 |
| C | SAD | 3 | 8 | 1 | 8 | 10 | 7 | 9 | 3 | 10 | 8 | 8 | 6 | 9 | 8 | 10 | 8 |
| D | KinectSound | 3 | 10 | 3 | 9 | 7 | 4 | 8 | 3 | 10 | 8 | 9 | 9 | 9 | 10 | 9 | 9 |
| E | SAD | 3 | 7 | 2 | 6 | 7 | 6 | 10 | 5 | 10 | 4 | 9 | 8 | 10 | 8 | 8 | 9 |
| F | KinectSound | 2 | 4 | 1 | 7 | 9 | 9 | 9 | 4 | 10 | 8 | 9 | 10 | 10 | 9 | 10 | 10 |
| G | SAD | 3 | 5 | 1 | 3 | 7 | 5 | 10 | 7 | 10 | 9 | 10 | 9 | 10 | 9 | 9 | 9 |
| H | KinectSound | 7 | 7 | 3 | 4 | 8 | 6 | 7 | 7 | 8 | 7 | 9 | 7 | 8 | 8 | 8 | 8 |
| I | SAD | 2 | 8 | 2 | 8 | 8 | 4 | 9 | 3 | 9 | 3 | 9 | 9 | 8 | 8 | 9 | 9 |
| J | KinectSound | 5 | 7 | 6 | 4 | 8 | 6 | 8 | 7 | 10 | 10 | 9 | 8 | 10 | 9 | 10 | 10 |
| K | SAD | 3 | 10 | 2 | 6 | 8 | 8 | 8 | 5 | 10 | 5 | 8 | 6 | 8 | 6 | 10 | 10 |
| L | KinectSound | 2 | 9 | 4 | 9 | 9 | 5 | 9 | 3 | 10 | 4 | 7 | 7 | 9 | 7 | 9 | 9 |
| | Mean | 6.67 | 2.58 | 7.50 | 3.75 | 7.75 | 5.75 | 8.42 | 4.92 | 9.42 | 6.33 | 8.67 | 7.58 | 9.08 | 8.25 | 9.08 | 9.08 |
| | Mean Differences | 4.1 | | 3.8 | | 2.0 | | 3.5 | | 3.1 | | 1.1 | | 0.8 | | 0.0 | |
| | Standard Deviation | 1.44 | 1.98 | 1.57 | 2.26 | 1.29 | 1.60 | 1.16 | 1.73 | 0.90 | 2.31 | 0.98 | 1.51 | 0.90 | 1.06 | 1.00 | 0.79 |
| | STDEV Differences | 0.5 | | 0.7 | | 0.3 | | 0.6 | | 1.4 | | 0.5 | | 0.2 | | 0.2 | |

Table 6.2 : User feedback from the questionnaire

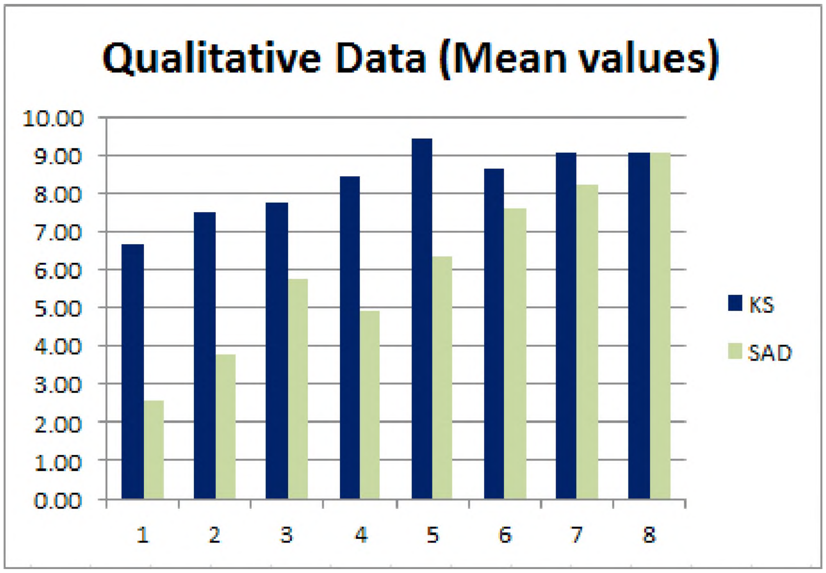


Figure 6.2 : Graphical representation of the questionnaire

## 6.4 Further User Feedback

Each user was given the opportunity to provide any personal feedback or recommend any changes that they thought would improve either of the systems. All of the feedback given by the users falls within the following subsections:

### S.A.D - Complicated DAW Interface

Almost all of the users found that the interface for the DAW was too complicated. The users generally struggled to find small controls such as *Audio Effects* and *Track Automation*. Both controls are essential for recording using the S.A.D plugin. A DAW is typically complex and contains many different controls, a requirement for the range of tasks it can perform. The KinectSound system does not require any knowledge of a DAW as the tasks required by the DAW for the KinectSound system are done using the MIDI commands sent from the KinectSound system.
Users that had previous experience with recording and DAWs finished task 1 faster than the rest of the users. These users were also able to find the required controls that were mentioned in the manual relatively easily. The manual for S.A.D (shown in

the Appendix 8.4) shows the locations of each control required to perform the tasks, as well as screenshots of where they are located. This issue could have been a result of the users not reading the manuals properly.

### S.A.D - Relation Between Track 1 and Track 4

When recording Track 4 sound source movements in S.A.D, all of the audio tracks have to be routed and mixed through track 1. *Track Automation* has to be altered on track 4 in the DAW in order to record the sound movements, while track 3 has to be muted by selecting the *Mute* option in the S.A.D plugin. The S.A.D plugin is considered as an additional *Audio Effect* of Track 1 by the DAW. The majority of the users did not perform these two actions correctly and used the DAW *Mute* control to mute track 3. Due to the fact that this track was routed through track 1, the DAW mute control had no effect and the track would still play. Out of all the users that made this error, a lot of them realized it and rectified it when they opened the S.A.D *Mix* module, where it shows the sound sources and the correct mute controls.

The KinectSound system eliminates this complexity by providing a simple method of changing the muting state and recording state of a track. Each track's muting and recording state can only be changed in the *Track Selection Interface*. This interface, as well as the *Recording Interface*, show the muting and recording state of each track. These are both clearly shown and easily observable by a user.

### Manual Descriptions

It was evident that many users did not read the manuals thoroughly. Users were given as much time as they needed in order to read the manuals thoroughly before starting the tasks. Many of the users claimed to have read the manuals and were ready to start the tasks in a much faster time than it should have taken to read the manuals. This caused the users to get stuck whilst performing a task and having to refer back to the manuals. The timing for each task began when they started the task, so referring back to the manual whilst in the middle of a task increased the time taken for them to complete the task. This happened more often with S.A.D as the manual was longer and the interface proved to be less intuitive. When the users had to refer back to the manual, they often had to refer back to the task's instructions, which used up more time. The KinectSound system provided intuitive interfaces and clear controls in order to prevent the users from having to refer back to the manual. Some of the users still referred back to the task's instructions when using the KinectSound system, which did not influence the time as much as referring to the manual. Some of the users suggested that the manuals be replaced with short video clips showing how to perform the different operations.

### KinectSound system - Interface Control Selection

The KinectSound system's button selection triggers an action when a user's hand has moved into a control, and moved forward by a predefined click threshold (20cm) whilst in the control. This is easy for users to do by moving their hand horizontally into the control, and then forward once their hand is in the control's boundaries. Many of the users testing the system tended to move their hand horizontally and forward at the same time. This resulted in their hand not moving forward the required distance once it had entered the control, and not triggering the control they wished to. This required them to step further forward, or move their hand out of the control and re-attempt selecting the control.

This error could be eliminated by explaining the button selection procedure in further detail in the manual. The distance used to determine a button selection (20cm) could also be reduced to enable easier button selection. However, there is a trade-off here. When users move across a control in attempt to reach another control on interfaces where there are a lot of controls, such as in the *Track Selection Interface*, they may select a control that they did not want to select. The unwanted button selection still happened with a click threshold of 20cm, although it was a rare occurrence.

### KinectSound - Crosshair Jitter

At times Kinect's co-ordinates displayed jitter. This occurred when the co-ordinates of the user's hand, as detected by the kinect, suddenly moved a small distance in a completely random direction from the user's hand and then back to the correct location. This jitter would sometimes happen when the user's hands were pointing directly at the Kinect, or approached an area where the Kinect was unable to detect them (such as behind the user's back). The jitter would usually go unnoticed. However the co-ordinates in this system were scaled by a factor of 7, and this resulted in slightly larger jitter in the range of approximately 50mm - 150mm. This was noticable on the 3D display, but was inaudible when recording as it was not large enough to result in an audible difference.

### KinectSound - Crosshair 3D view

The 3D view was provided in order to help the users identify the location of their hand. The disadvantage of a single 3D display is that it only enables a single camera view. Figure 6.3 shows two different crosshair locations that appear to be in the same place from the camera's perspective when viewed by the user.

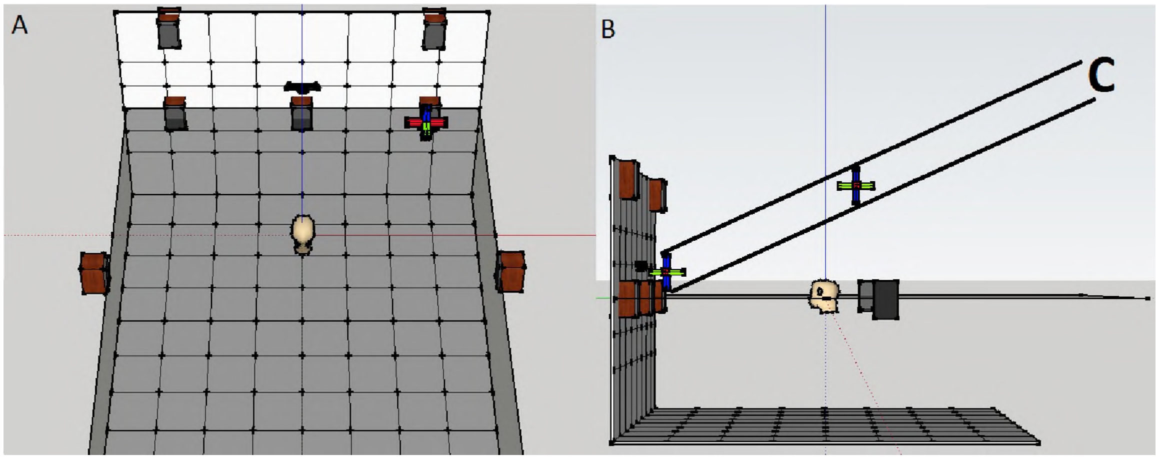


Figure 6.3 : A Sketchup camera view error

Figure 6.3A shows the default camera view for the KinectSound system's 3D display, with a crosshair appearing to be on the front right speaker. Figure 6.3B shows a side view of the crosshair locations, which appear in the same place to a user if the camera position is at $C$. This error can be prevented by observing the right hand's co-ordinates in the given dialog box. A lot of users did not do this, and ended up making sound localization errors. This problem could be fixed by providing several 3D views of the localization area from different perspectives. This could not be done as Sketchup does not provide a way of viewing a single model from different perspectives at the same time. There could have been a further extension to the Sketchup plugin to vary the size of the crosshair as the height of the sound source changed. This could improve the user's idea of the sound source height.

### Dynamic Panning

The main advantage of panning using the KinectSound system was the dynamic panning. This refers to the ability to move the sound source through all 3 axes in a single movement. Although the S.A.D was able to localize the sound in the same region as the KinectSound system, it had to be done using two different movements. Certain sound movements could not be done using S.A.D such as moving the sound source between opposite corners of the movement area, which required movement through all 3 axes. These movements were simple to record with the KinectSound system as it allowed the user to move their hand in any direction they wished.

## 6.5 Chapter Summary

This chapter has described the testing methods used in order to compare the KinectSound system, and New Audio Technology's Spatial Audio Designer. The testing group consisted of 12 users of whom each had at least some experience in one or more of the following: sound production, computer programming, or user interfaces. The test included both qualitative and quantitative feedback. The quantitative results were a measure of the time taken to complete two tasks. The qualitative data was derived from a questionnaire as well as any additional feedback from the user.

Both quantitative and qualitative results indicated that the KinectSound system is more intuitive and satisfies four of the *Five E's* of system usability to a greater degree than the S.A.D. Question 8 had an identical score for the two systems. This question referred to the quality of sound while the system was in a recording or playback state and is associated with the *Error tolerance* component of system usability.

Although the need for energy normalization was only determined after the listening tests were conducted, the results show that localization was accurate for a room of this size with the KinectSound system's speaker layout without the need for energy normalization.

# Chapter 7

# Conclusion

This thesis has described the state of the art with regard to surround sound and immersive sound. Furthermore it has described the conceptualization and implementation of a system, the KinectSound system. This system was created with the goal of enhancing the current state of the art, both with regard to user control and sound processing capability

The KinectSound system's implementation was evaluated by a sample group of users that had a varied level of experience with audio recording software. The feedback from the users, both qualitative and quantitative, provided a comprehensive comparison of the KinectSound system's usability and performance against a commercially available sound localization system.

This section provides a summary of each chapter and its relevance to the KinectSound system as well as a discussion of the research question, the research objectives and the extent to which the objectives have been fulfilled.

## 7.1 Chapter Summaries

Chapters 2 - 4 each focused on an area of research that pertained to a key area of the KinectSound system. This was required in order to formulate an optimal set of requirements which were followed during implementation, described in Chapter 5 and tested in Chapter 6

**Chapter 2** - focused on state of the art surround sound systems. Various speaker configurations were introduced that ranged from the standard 2.1 stereo configuration to the Auro3D 9.1 speaker configuration. This chapter also discussed various techniques for localizing sound, including distance based amplitude panning (DBAP) with a constant sound energy, the panning technique chosen to be used by the KinectSound system.

**Chapter 3** - provided a benchmark of usability metrics that measure the user friendliness and functionality of computer systems. This chapter also discussed various audio panning software and highlighted the need for a graphical representation of 3D sound localization.

**Chapter 4** - presented the distributed approach to immersive and surround sound processing. This gave a brief overview of audio networking technologies and the reason as to why Ethernet AVB was chosen as the networking technology used for the KinectSound system. Ethernet AVB was discussed further along with the transport and control protocols that it utilizes. This chapter also describes the hardware used in the networking configuration of the KinectSound system.

**Chapter 5** - provides further information about the system design and implementation of the KinectSound system. The core features of the KinectSound system were described along with their respective programming code. The core features are:

- Device free HCI
- Digital Audio Workstation (DAW) control
- Audio encoding
- 3D audio panning
- Ethernet AVB for the KinectSound system

**Chapter 6** - described the steps taken in testing the implementation with a sample group of users. The quantitative and qualitative results are shown in this chapter along with a discussion and user feedback.

## 7.2 Review of the Research

### 7.2.1 The Research Question

The primary goal of the research described in this thesis was to create and then evaluate the effectiveness of a gesture controlled immersive sound system with distributed localization processing.

This goal was approached by first investigating knowledge areas related to the desired system. The system was then designed, implemented, and tested against a commercial system. The evaluation used a set of criteria which was presented to users in order to evaluate the system.

The use of *intelligent endpoints* proved to be successful as they provided a way of partitioning and reallocating the mix level processing. The endpoints were able to be added incrementally, which removed the limitation of a preset speaker configuration. The endpoints used in the implementation did however provide two limitations, which were the limited programming code space, and the lack of power over Ethernet (PoE). These limitations point to future work and are discussed in further detail in Section 7.3.

The user feedback showed that gesture control was successful in doing the following

- Providing a simplified, interactive means of controlling the system.
- Allowing the user to move the sound source through 3 dimensions simultaneously.
- Gain a better understanding of where the sound was being located in the room.

### 7.2.2 Research Objectives

This section describes the degree to which the research objectives were achieved, as stated in the introduction of this thesis

- The speaker configuration used for the KinectSound system was a variation on the Auro3D 9.1 and the Dolby 5.1 configuration. The configuration contained 5 speakers on the same level as the user, and two speakers above the right and left front speakers to incorporate height. This configuration was selected in order to approximate an immersive sound experience given the current hardware limitations.

- The distribution of processing to the endpoints was implemented as part of a modular approach to building the KinectSound system. A total of 7 endpoints were used, one for each speaker, as this was the maximum amount that were able to be connected to the Extreme Ethernet bridge. The current firmware on the endpoints was modified to meet the distance based amplitude panning requirements as laid out in the specification. The user feedback relating to the surround sound experience, and the quality of audio produced by the mixing performed at each endpoint proved that they are capable of mixing audio to a commercially acceptable standard.

- Device free control of the system was implemented via several user interfaces. Users are able to interact with these interfaces using left hand movements. As this was a concept that none of the users had experienced before, it took them a short amount of time to adjust to the mechanics of the device free interaction. From evaluation feedback, the interfaces proved to be more intuitive than a commercially available sound localization system. The goal of device free interaction was achieved in that the user was able to perform all the recording, playback, and shutdown actions without having to touch the keyboard or mouse. The 3D panning was successfully implemented and tested by the users. The feedback given by the users suggested that this approach was simpler and more effective than using a mouse to control localization via two different perspectives of the listening environment.

- The user testing was carried out using a sample base of 12 different users who performed the same tasks with the KinectSound system and the commercially available sound panning software (Spatial Audio Designer). Quantitative results were obtained from the tests by measuring the times taken to complete each task using the two systems. The qualitative results included a questionnaire as well as further feedback given by each user. These results were consolidated and overall provided a positive comparative evaluation of the KinectSound system as a tool for 3D sound localization.

## 7.3 Limitations and Future Work

This section describes the limitations that were encountered during the implementation of the system and how these could possibly be overcome through further research.

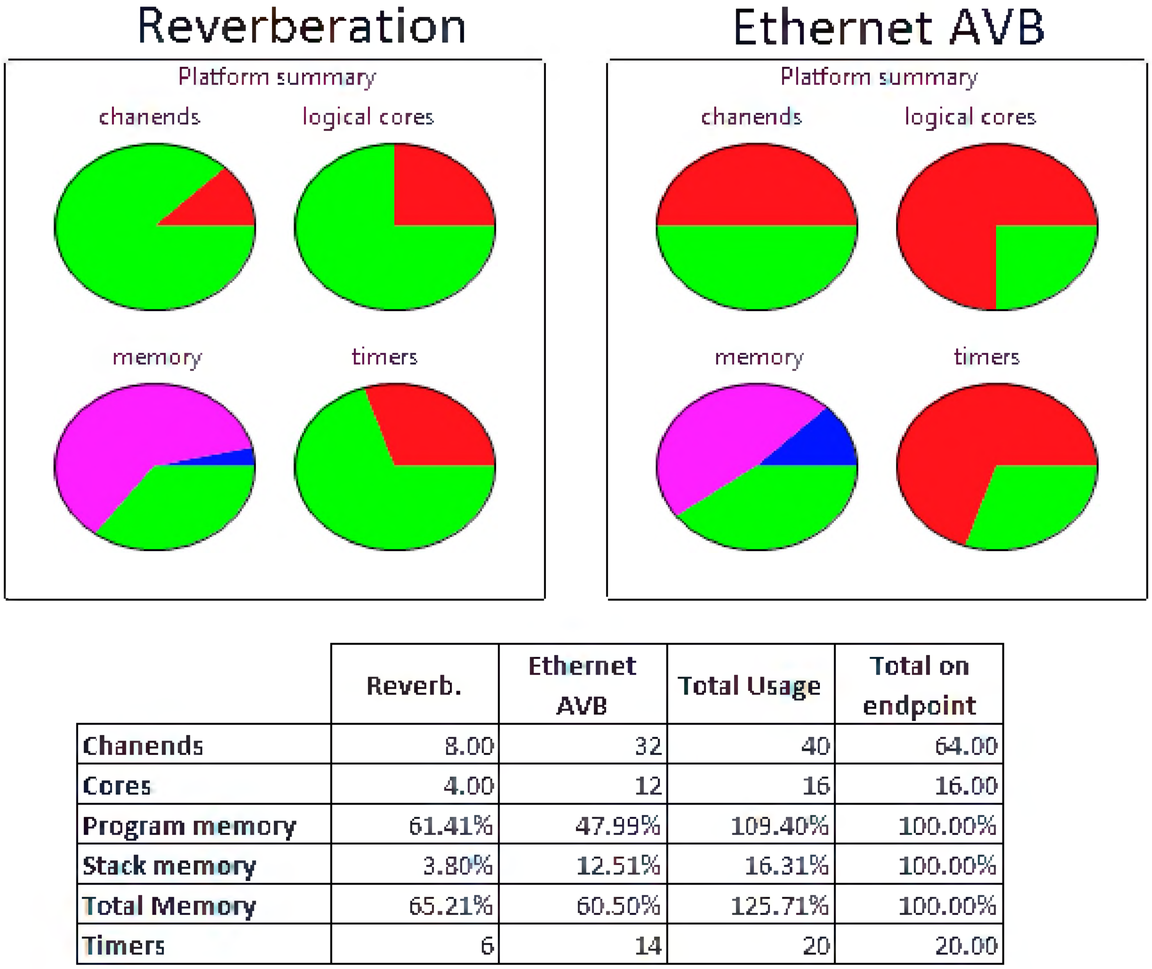
### 7.3.1 Improve Sketchup Display

Figure 6.3 shows how Sketchup's crosshair display may appear at a different place to what the user perceives. The Sketchup plugin for the KinectSound system could be further improved to add features such as a colour variation of the crosshair which would provide a more accurate representation of the crosshair's position.

A further extension to the KinectSound system's Sketchup display would be to add the positions of the recorded tracks to Sketchup. This would allow the user to view the current points in the 3D space where the already tracks are being localized. It would also require the KinectSound system to send through the co-ordinates of each track over the TCP socket just before the mix levels are calculated.

### 7.3.2 Added Reverberation

The use of reverberation at the endpoints of the immersive sound system could enhance the immersive sound experience. Reverberation firmware was implemented and tested during the implementation of this project, however the endpoints that were used were unable to provide reverberation and the Ethernet AVB listener capabilities on the same endpoint. This limitation was due to a memory limitation of the endpoints. Figure 7.1 shows the resource usage on an endpoint for the reverberation and Ethernet AVB implementations.



| | Reverb. | Ethernet AVB | Total Usage | Total on endpoint |
|---|---|---|---|---|
| Chanends | 8.00 | 32 | 40 | 64.00 |
| Cores | 4.00 | 12 | 16 | 16.00 |
| Program memory | 61.41% | 47.99% | 109.40% | 100.00% |
| Stack memory | 3.80% | 12.51% | 16.31% | 100.00% |
| Total Memory | 65.21% | 60.50% | 125.71% | 100.00% |
| Timers | 6 | 14 | 20 | 20.00 |

Figure 7.1 : The resource usage on XMOS endpoints

Figure 7.1 shows that if the memory on these endpoints is increased by 25.71%, both the reverberation and Ethernet AVB implementations could be used on a single endpoint.

### 7.3.3 Power over Ethernet (PoE)

The endpoints used in this implementation did not provide the Power PoE capability and thus each endpoint required an external power source. The Extreme Ethernet bridge used in the implementation provided a PoE capability, and using this capability would improve the modularity of the system by removing the requirement of external power. Each endpoint requires an Ethernet cable to receive the audio, so using Ethernet to power the devices would simplify configuration.

### 7.3.4 Test the Device Limitations

The KinectSound system's implementation encountered no problems with the distribution of audio to 7 endpoints. The Extreme Ethernet bridge that was used was an 8 port switch which only allowed 7 endpoints to be used. Further testing could be done in order to determine the maximum number of endpoints that the system could distribute audio to. An increased number of endpoints would mean that more speakers could be added to the system, which would in turn increase the overall user experience.

### 7.3.5 Further Testing

The user base that tested the KinectSound system consisted of 12 users. The accuracy of the results obtained from testing the system would increase as the pool of users was increased. The majority of the users did not have any professional sound recording experience. Further testing by a large group of audio professionals would enable a better understanding of how the KinectSound system compares to the current state of the art audio recording systems.

# Bibliography

[1] Dolby. (2013) Home theatre speaker guide. Dolby. [Online]. Available: http://www.dolby.com/us/en/consumer/setup/connection-guide/home-theater-speaker-guide/index.html

[2] F. Rumsey, D. Griesinger, T. Holman, M. Sawaguchi, G. Steinke, G. Theile, and T. Wakatuki., "Multichannel surround sound systems and operations," Audio Engineering Society, Tech. Rep., 2001.

[3] F. Rumsey, *Spatial Audio*, F. Rumsey, Ed. Focal Press, 2001.

[4] W. V. Baelen, T. Bert, B. Claypool, and T. Sinnaeve, "Auro3d - a new dimension in cinema sound," *Barco*, vol. 1, pp. 1–11, 2011.

[5] F. Melchior, C. Pike, M. Brooks, and S. Grace, "On the use of a haptic feedback device for sound source control in spatial audio systems," in *Audio Engineering Society 134th Convention, Rome*. AES, May 2013.

[6] Echo. (2014) Echo nic-1. Echo Audio. [Online]. Available: http://echoavb.com/products/streamware-nic-1

[7] F. Otten, "Network simulation for professional audio networks," Ph.D. dissertation, Rhodes University, June 2014.

[8] M. Neukom, "Ambisonic panning," in *Audio Engineering Society 123rd Convention, New York*. AES, October 2007.

[9] J. Barker, A. de Cheveign, D. P. W. Ellis, A. S. Feng, M. Goto, D. L. Jones, K. Palomki, and R. Stern, *Computational Auditory Scene Analysis*, D. Wang and G. Brown, Eds. Wiley Interscience, 2005.

[10] V. Pulkki, "Virtual sound source positioning using vector base amplitude panning," *Journal of the Audio Engineering Society*, vol. 45, no. 6, pp. 456–466, 1997.

[11] T. Lossius, P. Baltazar, and T. de la Hogue, "Dbap - distance-based amplitude panning," in *International Computer Music Conference (ICMC), Montreal.* ICMC, August 2009.

[12] D. Kostadinov, J. Reiss, and V. Mladenov, "Evaluation of distance based amplitude panning for spatial audio," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), Dallas.* ICASSP, March 2010.

[13] (2014) Spatial audio designer. New Audio Technology. [Online]. Available: http://www.newaudiotechnology.com/products/spatial-audio-designer/

[14] (2014, December) Ieee 802.3 ethernet working group. IEEE. [Online]. Available: http://www.ieee802.org/3/

[15] (2014) Kinect for windows sensor components and specifications. Microsoft. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131033.aspx

[16] M. Hedges and R. Foss, "The use of gesture recognition to select and distribute audio across a network," in *South African Telecommunications, Networking, and Applications Conference (SATNAC), Stellenbosch.* SATNAC, September 2013.

[17] ——, "Utilizing gesture recognition and ethernet avb for distributed surround sound control," in *Audio Engineering Society 137th Convention, Los Angeles.* AES, October 2014.

[18] T. Holman, *Surround Sound - up and running*, 2nd ed., T. Holman, Ed. Focal Press, 2008.

[19] (1881, December) Scientific american - the telephone at the paris opera. Scientific American. [Online]. Available: http://earlyradiohistory.us/1881opr.htm

[20] H. Robjohn. (1997, February) Stereo microphone techniques explained. Sound on Sound. [Online]. Available: https://www.soundonsound.com

[21] M. Miller. (2004, September) The history of surround sound. Pearson. [Online]. Available: http://www.quepublishing.com/articles/article.aspx?p=337317

[22] T. Holman, *Sound for Film and Television*, F. Press, Ed. Oxford, 1997.

[23] (2011, April) Understanding surround sound formats. Crutchfield. [Online]. Available: http://www.crutchfield.com/S-iyORSHAIY2P/learn/learningcenter/home/hometheater_surround.html

[24] R. Silva. (2013) 5.1 vs 7.1 channel home theater receivers - which is right for you? About.com. [Online]. Available: http://hometheater.about.com/od/hometheateraudiobasics/qt/5-1vs7-1diff.htm

[25] (2013, March) Guide to home theater speaker placement - understanding speaker placement in multichannel audio. Audyssey. [Online]. Available: http://www.practical-home-theater-guide.com/home-theater-speaker-placement.html

[26] J. Middlebrooks and D. Green, “Sound localization by human listeners,” Research Paper, 1991.

[27] C. Denison. (2014, December) Ultimate surround sound guide: Different formats explained. Digital Trends. [Online]. Available: http://www.digitaltrends.com

[28] G. Potard, “3d-audio object oriented coding,” Ph.D. dissertation, University of Wollongong, 2006.

[29] K. Lee, S. Jo, and D. Kim, “3d object rendering into 5.1 surround system,” in *Audio Engineering Society 136th Convention, Berlin*. AES, April 2014.

[30] *Audio Definition Model - Metadata Specification*, Online, European Broadcasting Union Specification 1.0, January 2014.

[31] (2013, April) Authoring for dolby atmos cinema sound manual. Manual. Dolby Laboratories. [Online]. Available: http://www.dolby.com/

[32] Barco home. Barco. [Online]. Available: http://www.barco.com/en/

[33] B. V. Daele and W. V. Baelen. (2011, November) Auro3d octopus codec - principles behind a revolutionary codec. Auro Technologies. [Online]. Available: www.auro-3d.com

[34] Mpeg - the moving picture experts group. MPEG. [Online]. Available: http://mpeg.chiariglione.org/

[35] J. Herre, J. Hilpert, A. Kuntz, and J. Plogsties, "Mpeg-h audio - the new standard for universal statial/3d audio coding," Audio Engineering Society, Tech. Rep., December 2014.

[36] S. Fug, A. Holzer, C. Bor, C. Ertel, M. Kratschmer, and J. Plogsties, "Design, coding and processing of metadata for object-based interactive audio," in *Audio Engineering Society 137th Convention, Los Angeles.* AES, October 2014.

[37] About dts. Digital Theater Systems. [Online]. Available: http://www.dts.com/corporate/about-dts.aspx

[38] T. Nicolakis. (2015, January) Dts announces dts:x immersive surround sound format. Audioholics. [Online]. Available: http://www.audioholics.com/audio-technologies/dts-x-immersive-surround-sound

[39] SMPTE. (2013, March) Smpte creates new technology committee dedicated to cinema sound. Online. Society of Motion Picture & Television Engineers. [Online]. Available: https://www.smpte.org/news-events/news-releases/

[40] B. Vessa. (2014, May) 25css interoperability of immersive sound systems in digital cinema. Online. Society of Motion Picture & Television Engineers. [Online]. Available: https://kws.smpte.org/kws/public/projects/

[41] M. Morrell and J. Reiss, "A comparative approach to sound localisation within a 3d sound field," in *Audio Engineering Society 126th Convention, Munich.* AES, May 2009.

[42] S. Brown. (2004, October) Simon's graphics blog - spherical harmonics basis functions. [Online]. Available: http://www.sjbrown.co.uk/2004/10/16/spherical-harmonic-basis/

[43] (2008) Hoa technical notes - b-format. Blue Ripple Sound. [Online]. Available: http://www.blueripplesound.com/b-format

[44] J. J. Hofmann. (2005, September) Ambisonics tutorial. Sonic Architecture. [Online]. Available: http://csounds.com/resources

[45] D. Griesinger, "Stereo and surround panning in practice," in *Audio Engineering Society 112th Convention, Munich.* AES, April 2002.

[46] F. Keiler and J.-M. Batke, "Evaluation of virtual source localization using 3d loudspeaker setups," in *Audio Engineering Society 128th Convention, London.* AES, May 2010.

[47] S. Haw, "The x170 protocol as a vehicle for 3d sound control," Thesis, November 2011.

[48] C. Dewey and J. Wakefield, "A guide to the design and evaluation of new user interfaces for the audio industry," in *Audio Engineering Society 136th Convention, Berlin.* AES, April 2014.

[49] J. Sauro. (2011, November) 10 essential usability metrics. MeasuringU. [Online]. Available: https://www.measuringu.com/blog/essential-metrics.php

[50] A. Dillon, *Usability Evaluation*, W. Karwowski, Ed. Taylor and Francis, 2001.

[51] W. Quesenbery, "What does usability mean: Looking beyond ease of use," in *Society for Technical Communication, 48th Annual Conference*, 2001.

[52] About the vst standard. Steinberg. [Online]. Available: https://www.steinberg.net/en/company/technologies/vst3.html

[53] (2014) Anymix pro - the surround maker. IOSONO. [Online]. Available: http://www.iosono-sound.com/vstaax-plug-ins/

[54] Nuage surround panner. JL Cooper Electronics. [Online]. Available: https://jlcooper.com/

[55] A new platform for unprecedented post-production productivity. Yamaha Commecial Audio, Steinberg. [Online]. Available: http://www.yamahaproaudio.com/

[56] Nuendo 6.5 - advanced post and audio production system. Steinberg. [Online]. Available: https://www.steinberg.net

[57] The leap motion controller. Leap Motion Inc. [Online]. Available: https://www.leapmotion.com/

[58] J. Ratcliffe, "Motionmix: A gestural audio mixing controller," in *Audio Engineering Society 137th Convention, Los Angeles.* AES, October 2014.

[59] (2015) Open source computer vision. Itseez. [Online]. Available: http://opencv.org/

[60] L. Srinivasan. (2013, April) Hand tracking and gesture detection (opencv). [Online]. Available: http://s-ln.in/2013/04/18/hand-tracking-and-gesture-detection-opencv/

[61] Kinect for windows sensor components and specifications. Microsoft. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131033.aspx

[62] D. Lau. (2013, November) The science behind kinects or kinect 1.0 versus 2.0. Gamasutra. [Online]. Available: http://www.gamasutra.com/

[63] Kinect for windows. Microsoft. [Online]. Available: http://www.microsoft.com/en-us/kinectforwindows/

[64] W. Zeng. (2012) Microsoft kinect sensor and its effect. IEEE. [Online]. Available: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6190806

[65] K. Gross, "Audio networking: Applications and requirements," in *Journal of the Audio Engineering Society*, vol. 54, no. 1/2. AES, February 2006, pp. 62–66.

[66] M. Teener. Technical introduction to ieee 1394. Presentation. IEEE. [Online]. Available: http://www.ieee802.org/802_tutorials/

[67] D. Anderson, *FireWire System Architecture (2Nd Ed.): IEEE 1394a.* Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[68] C. Tindel and B. Pietsch, "Ieee 1394 and linux - the firewire serial bus and its implementation," Online, February 2000. [Online]. Available: http://www.tindel.net/Firewire/firewire.html

[69] J. Canosa, "Fundamentals of firewire," Online, November 2000. [Online]. Available: http://public.fh-wolfenbuettel.de/~bermbach/research/firewire/files/basics.pdf

[70] N. Chikwamba and R. Foss, "Enhancing end-user capabilities in high speed audio networks," in *Audio Engineering 123rd Convention, New York*. AES, October 2007.

[71] F. Culas, "Ethersound es-giga system transport preliminary information," EtherSound, Tech. Rep., December 2006.

[72] Ethersound es-giga system transport. EtherSound. [Online]. Available: http://www.ethersound.com/download/files/cutsheetESgiga.pdf

[73] C. Doering, "The ethersound standard - ethernetworking professional audio," Brochure, March 2006, dynamic Market Systems.

[74] *EtherSound Setup Guide*, Yamaha and Commercial Audio, April 2010.

[75] "Overview - an introduction to the es-100 technology," Document, August 2006, rev. 3.0b.

[76] S. Schmitt and J. Cronemeyer. Audio over ethernet: There are many solutions but which one is best for you? Online Paper. DSPECIALISTS. [Online]. Available: http://www.dspecialists.com/

[77] (2014) About cobranet and frequently asked questions. Cirrus Logic. [Online]. Available: http://www.cobranet.info/support/

[78] B. Klinkradt and R. Foss, "A comparative study of mlan and cobranet technologies and their use in the sound installation industry," in *Audio Engineering Society 114th Convention, Amsterdam.* AES, March 2003.

[79] D. Harrington, R. Presuhn, and B. Wijnen. (2002, December) An architecture for describing simple network management protocol (snmp) management frameworks. The Internet Engineering Task Force. [Online]. Available: https://www.ietf.org/

[80] J. Dibley, "An investigation of the xmos xs1 architecture as a platform for development of audio control standards," Masters Thesis, Rhodes University, October 2013.

[81] (2013) Dante. Biamp Systems. [Online]. Available: http://support.biamp.com/Tesira/Control/Dante

[82] (2014, July) Dante networking guide. Peavey Electronics Corporation. [Online]. Available: http://www.peaveyoxford.com/kc/Dante%20Networking%20Guide.pdf

[83] IEEE, *IEEE Standard for Layer 2 Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks*, Draft Standard, IEEE Standard, February 2014.

[84] *AES standard for audio applications of networks - High-performance streaming audio-over-IP interoperability*, Audio Engineering Society Standard, March 2014.

[85] K. Nichols, S. Blake, F. Baker, and D. Black. (1998, December) Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers. The Internet Engineering Task Force. [Online]. Available: https://www.ietf.org/rfc/rfc2474.txt

[86] Home of the audio engineering society. AES. [Online]. Available: http://www.aes.org/

[87] *IEEE Standard for Local and Metropolitan area networks - Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol*, Document Standard, IEEE Std., September 2010.

[88] *IEEE Standard for Local and Metropolitan area networks - Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams*, Document Standard, IEEE Std., January 2010.

[89] G. Garner. (2008, September) Ieee 1588 version 2. IEEE. [Online]. Available: http://www.ieee802.org/

[90] *IEEE Standard for Local and Metropolitan area networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks*, Document Standard, IEEE Std., March 2011.

[91] *IEEE Standard for Device Discovery, Connection Management, and Control Protocol for IEEE 1722 BasedDevices*, Document Standard, IEEE Std., 2013.

[92] (2000, January) Steinberg asio 2.0. Steinberg. [Online]. Available: http://comp.ist.utl.pt/

[93] XMOS. (2014) Xmos avb audio endpoint reference design. XMOS. [Online]. Available: https://www.xmos.com

[94] (2007) Winpcap documentation - 4.0.2. CACE Technologies. [Online]. Available: http://www.winpcap.org/docs/

[95] N. Morrow and D. Mitchell, "Ieee1394 common isochronous packet (cip) enhancements for host controllers," June 2002, uS Patent 6,405,275. [Online]. Available: http://www.google.com/patents/US6405275

[96] (2013) Sketchup - the easiest way to draw in 3d. Trimble Navigation Limited. [Online]. Available: http://www.sketchup.com/

[97] G. Booch, J. Rumbaugh, and I. Jacobson, *The unified modeling language user guide*. Pearson Education India, 1999.

[98] IBM. Rational rose modeler. The International Business Machines Corporation (IBM). [Online]. Available: http://www-03.ibm.com/software/products/en/rosemod/

[99] Kinect sdk c++ - 1. kinect basics. Princeton University. [Online]. Available: http://www.cs.princeton.edu/

[100] Tracking users with kinect skeletal tracking. Microsoft. [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131025.aspx

[101] Vector4 structure. Microsoft. [Online]. Available: http://msdn.microsoft.com/en-us/library/windows/desktop/bb324268%28v=vs.85%29.aspx

[102] (2012) Loopbe1 - a free virtual midi driver. Nerds.de - Audio & MIDI Particles. [Online]. Available: http://www.nerds.de/en/loopbe1.html

[103] (2014) Midi messages. MIDI Manufacturers Association. [Online]. Available: http://www.midi.org/techspecs/midimessages.php

[104] (1996, October) Ieee standards interpretations for ieee std 1003.1c-1995 ieee standard for information technology–portable operating system interface (posix) - system application program interface (api) amendment

2: Threads extension (c language). Standard. IEEE. [Online]. Available: http://standards.ieee.org/findstds/interps/1003-1c-95_int/

[105] A. van Kesteren, A. Gregor, A. Russell, and R. Berjon. (2014, July) W3c dom4. World Wide Web Consortium (W3C). [Online]. Available: http://www.w3.org/TR/dom/

[106] J. Foltz. (2012, May) Class: Sksocket. [Online]. Available: http://www.rubydoc.info/github/jimfoltz/SketchUp-Ruby-API-Doc/master/SKSocket

[107] T. Tullis and W. Albert, *Measuring the User Experience: Collecting, Analyzing, and Presenting Usability Metrics*, M. Kaufmann, Ed. Morgan Kaufmann, 2008.

[108] E. Goodman, M. Kuniavsky, and A. Moed, *Observing the User Experience: A Practitioner's Guide to User Research*, M. Kaufmann, Ed. Morgan Kaufmann, 2012.

[109] *Specification of the Broadcast Wave Format (BWF)*, Online, European Brodcasting Union Specification 2.0, May 2011.

# Appendix 8.1

# The Complete EBU Layout

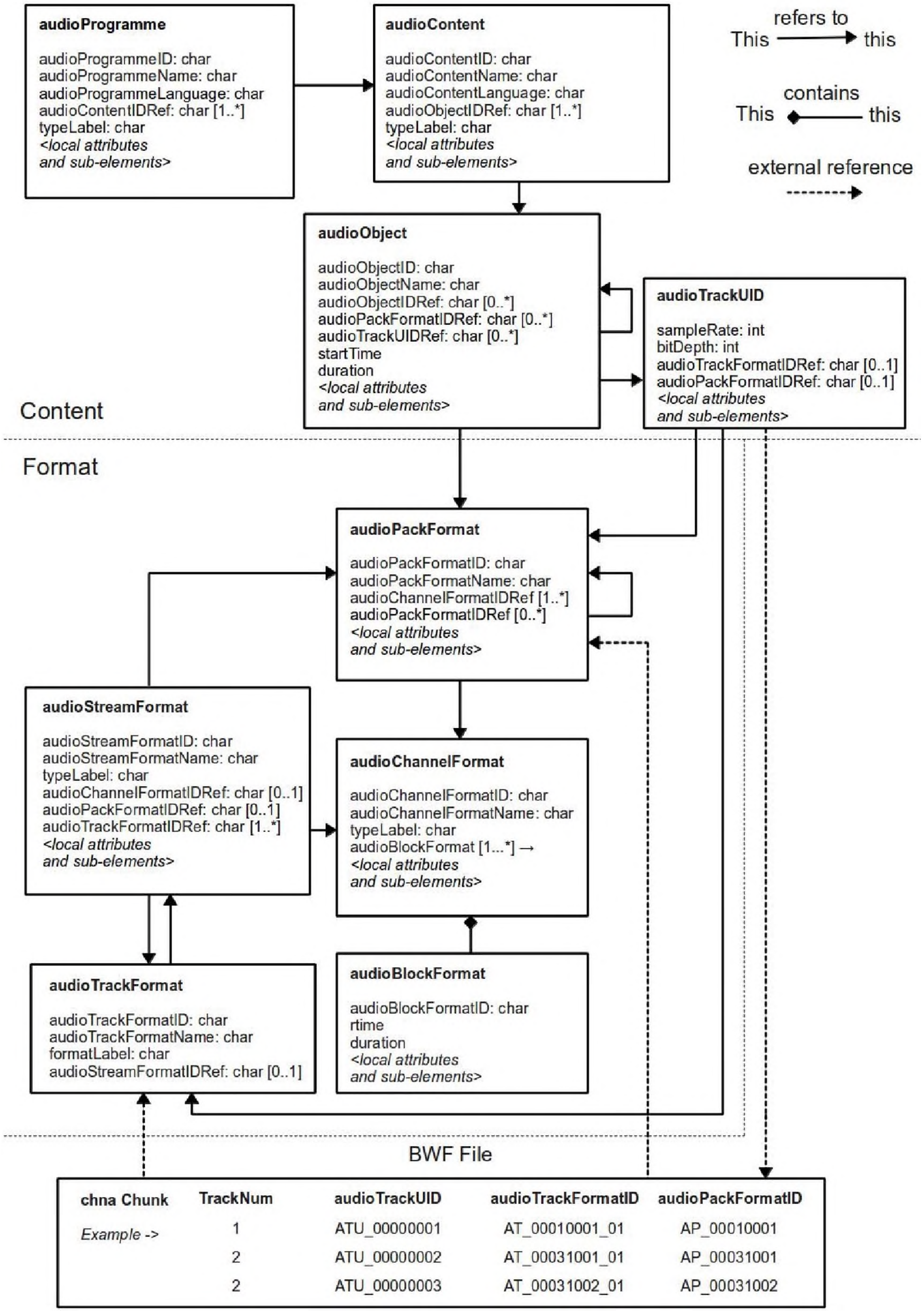


Figure 8.1: The UML layout of the EBU specification [30]

The specification uses *Broadcast Wave Format* [109] (BWF) files as an example for the audio, as shown at the bottom of Figure 8.1. Due to the fact that a BWF file may have multiple tracks, each track is accompanied by at least one *<chna> chunk*. Each *<chna> chunk* contains numbers which correspond to the track which it is describing. The reason that the list may be longer than the maximum number of tracks is that a track may be redefined at a different time during the audio playback [30]. Each track has an *audioTrackUID* which is a unique ID that is assigned to each track.

### The *audioTrackFormat*

The *audioTrackFormat* describes the format that the audio data is in. This is a requirement which allows the audio renderer to decode the signal correctly. The *audioTrackFormat* is referred to from the *audioStreamFormat* class, which is responsible for identifying the combination of tracks required to decode the track data. Shown in Table 8.1 are the attributes and sub elements contained within the *audioTrackFormat* class. Listing 8.1 shows sample XML code of a single *audioTrackFormat* block. This block specifies an audio track for the Front Right speaker of a system. It is linked to an *audioStreamFormat* block with an ID of *AS00010002*.

| **Attribute** | **Description** | **Example** |
|---|---|---|
| audioTrackFormatID | ID of the audio track | AT_00010001_02 |
| audioTrackFormatName | Name of the audio track | PCM_FrontRight |
| formatLabel | Descriptor of the format | 0001 |
| formatDefinition | Description of the format | PCM |
| **Sub Element** | **Description & Quantity** | **Example** |
| audioStreamFormatIDRef | Reference to an *audioStreamFormat* [0 or 1] | AS00010002 |

Table 8.1 : The *audioTrackFormat* attributes and sub elements

```
<audioTrackFormat audioTrackFormatID="AT_00010001_02" audioTrackFormatName="PCM_FrontRight"
formatLabel="0001" formatDefinition="PCM">
   <audioStreamFormatIDRef>AS_00010002</audioStreamFormatIDRef>
</audioTrackFormat>
```

Listing 8.1: *audioTrackFormat* sample code

### The *audioStreamFormat*

The main purpose of the *audioStreamFormat* is to establish a relationship between the *audioTrackFormat*, and the *audioPackFormat* or *audioChannelFormat*. The main use of the *audioStreamFormat* is to create a decodable signal which covers several *audioChannelFormats* from encoded tracks where several *audioTrackFormats* have been used. Table 8.2 shows the attributes and sub elements of the *audioStreamFormat* and listing 8.2 shows corresponding sample code. This code creates an *audioStreamFormat* with an ID of *AS_00010002* for a Front Right speaker. The ID for this stream is the same ID used as the reference in listing 8.1, this is an example of how block formats in this model are able to reference one another. The sub elements in this audio stream contain references to a track, and channel respectively.

| **Attribute** | **Description** | **Example** |
|---|---|---|
| audioStreamFormatID | ID of the audio stream | AT_00010002 |
| audioStreamFormatName | Name of the audio stream | PCM_FrontRight |
| formatLabel | Descriptor of the format | 0001 |
| formatDefinition | Description of the format | PCM |
| **Sub Element** | **Description** | **Example** |
| audioChannelFormatIDRed | Reference to an *audioChannelFormat* | AC_00010003 |
| audioPackFormatIDRef | Reference to an *audioPackFormat* | AP_00010002 |
| audioTrackFormatIDRef | Reference to an *audioTrackFormat* | AT_00010001_01 |

Table 8.2 : The *audioStreamFormat* attributes and sub elements

```
<audioStreamFormat  audioStreamFormatID=”AS_00010002”  audioStreamFormatName=”PCM_FrontRight”
 formatDefinition=”PCM”  formatLabel=”0001”>
    <audioTrackFormatIDRef>AT_00010001_02</audioTrackFormatIDRef>
    <audioChannelFormatIDRef>AC_00010003</audioChannelFormatIDRef>
</audioStreamFormat>
```

Listing 8.2: *audioStreamFormat* sample code

### The *audioChannelFormat*

The *audioChannelFormat* contains a sequence of audio samples where actions such as movement may occur. These actions are divided into timed chunks which are defined by the *audioBlockFormat*. Shown in Table 8.3 are the attributes and sub elements of the *audioChannelFormat*.

| **Attribute** | **Description** | **Example** |
|---|---|---|
| audioChannelFormatID | The ID of the channel | AC_00010003 |
| audioChannelFormatName | The name of the audio channel | FrontRight |
| typeLabel | Descriptor of the type of channel | 0002 |
| typeDefinition | Description of the type of channel | Matrix |
| **Sub Element** | **Description & Quantity** | **Attributes** |
| audioBlockFormat | A timed division of the channel containing metadata | See table 8.5 |
| frequency | Sets a high or low cut off frequency for the audio | typeDefinition= “lowPass” or “highPass” |

Table 8.3 : The *audioChannelFormat* attributes and sub elements

The *typeDefinition* field specifies the type of audio it is describing. There are five possible definitions which are shown in Table 8.4.

| Type def. | Label | Audio Type | Audio Description |
|---|---|---|---|
| DirectSpeakers | 0001 | Channel-based | Each channel feeds a speaker directly |
| Matrix | 0002 | Channel-based | Channels are matrixed together |
| Objects | 0003 | Object-based | Channels represent audio objects which include sound location information |
| HOA | 0004 | Scene-based | Used for ambisonics and Higher Order Ambisonics |
| Binaural | 0005 | Binaural | Audio playback through a stereo headset |

Table 8.4 : The five different type definitions available

Shown in Listing 8.3 is sample code for the *audioChannelFormat*. This example describes a direct channel fed to the Front Right speaker. The position of the speaker is described within the *audioBlockFormat* tag.

```
<audioChannelFormat audioChannelFormatID="AC_00010003" audioChannelFormatName="FrontRight"
typeDefinition="DirectSpeakers">
   <audioBlockFormat <!--audioBlockFormat attributes>>
   <!--audioBlockFormat co-ordinate information>
   </audioBlockFormat>
   <!--other audioBlockFormats>
</audioChannelFormat>
```

Listing 8.3: *audioChannelFormat* sample code.

### The *audioBlockFormat*

The *audioBlockFormat* represents a sequence of *audioChannelFormat* samples. The *audioBlockFormat* contains several fixed parameters, these parameters will differ depending on the *typeDefinition* field in the *audioChannelFormat*. The attributes of the *audioBlockFormat* are shown in Table 8.5.

| Attribute | Description | Example |
|---|---|---|
| audioBlockFormatID | Id of the audio block | AB_00010001_00000001 |
| rtime | The starting time of the audio block | 00:01:00.00000 |
| duration | The time duration of the block | 00:02:00.00000 |

Table 8.5 : The *audioBlockFormat* attributes

Listing 8.4 shows a channel with two different *audioBlockFormats*. The *typeDefinition* for these blocks has been set to *Objects*, which describe the sound location for this channel as an azimuth, elevation, and distance. The azimuth and elevation are measure in degrees while the distance is a normalized value between $\left[0..1\right]$ from the origin. The first audio block describes the position for 5 seconds. This can be seen from the *rtime* parameter being set to zero (the beginning of the audio signal) and the *duration* parameter set to 5 seconds. The next audio block's position is described from the 5th second, for a duration of 10 seconds. The azimuth, elevation, and distance can be seen in these audio blocks with a slight change between the two blocks.

```
<audioChannelFormat audioChannelFormatID="AC_00031001" audioChannelFormatName="Car1"
typeLabel="0003" typeDefinition="Objects">
   <audioBlockFormat audioBlockFormatID="AB_00031001_00000001" rtime="00:00:00.00000"
   duration="00:00:05.00000">
      <position coordinate="azimuth">-22.5</position>
      <position coordinate="elevation">5.0</position>
      <position coordinate="distance">1.0</position>
   </audioBlockFormat>
   <audioBlockFormat audioBlockFormatID="AB_00031001_00000002" rtime="00:00:05.00000"
   duration="00:00:10.00000">
      <position coordinate="azimuth">-45</position>
      <position coordinate="elevation">20</position>
      <position coordinate="distance">0.75</position>
   </audioBlockFormat>
   <!--more audio blocks>
</audioChannelFormat>
```

Listing 8.4: Two *audioBlockFormat* blocks as they would appear in the code

Figure 8.2 shows the two different audio locations described in the listing. Figure 8.2A shows the source at an azimuth of $-22.5°$, an elevation of $5°$, and a distance of 1.0. Figure 8.2B shows the source at an azimuth of $-45°$, an elevation of $205°$, and a distance of 0.75.

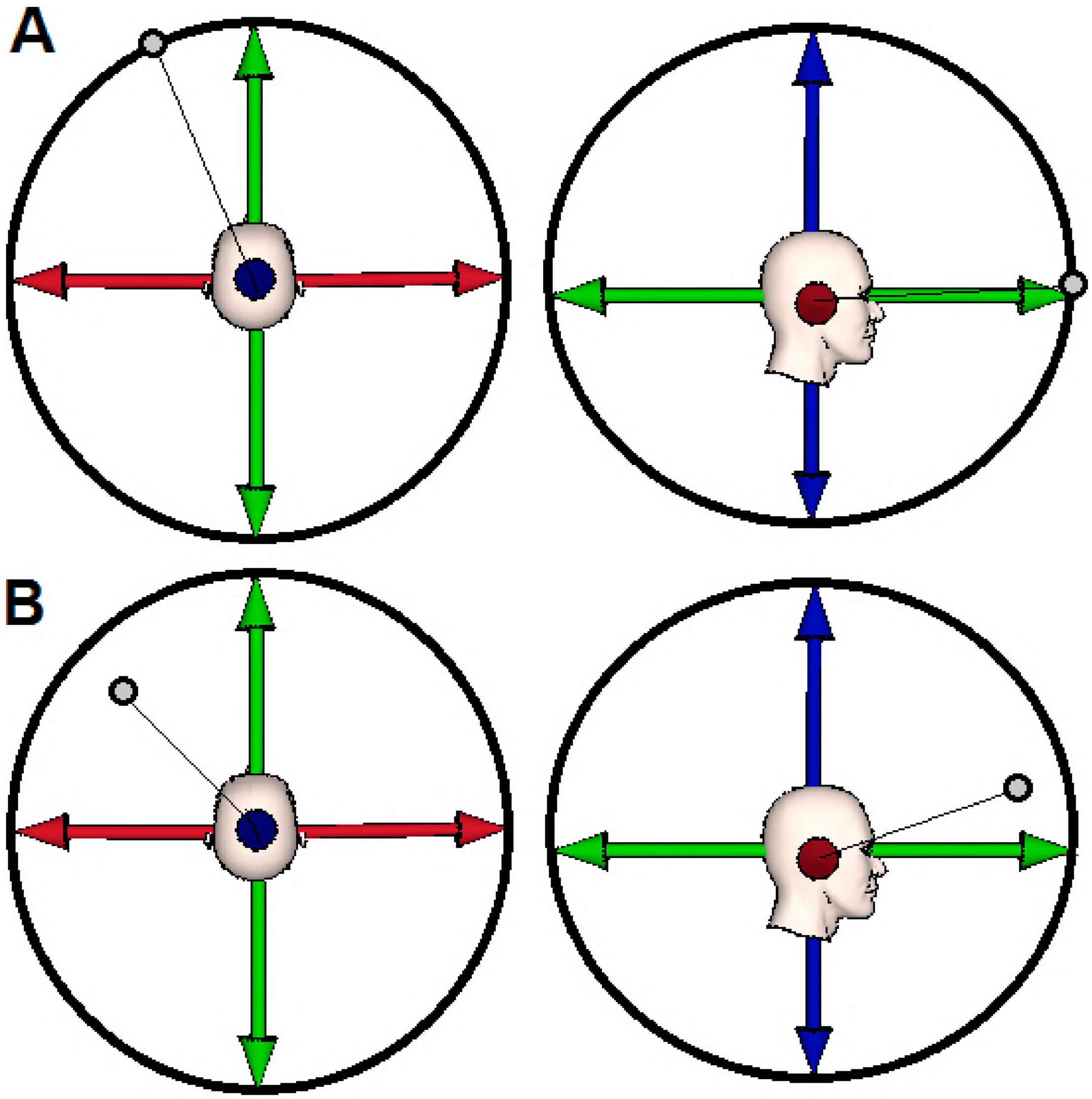


Figure 8.2 : Two different locations of sound sources, as described in the previous code listing

### The *audioPackFormat*

The *audioPackFormat* groups different channels together that are of the same type. If the audio file has many channels that are of the same type, only one *audioPackFormat* block would be required, which would contain references to the channels. Shown in

Table 8.6 are the attributes of the *audioPackFormat*.

| **Attribute** | **Description** | **Example** |
|---|---|---|
| audioPackFormatID | ID of the audio pack | AP_00010001 |
| audioPackFormatName | Name of the audio pack | Stereo |
| typeLabel | Descriptor of the channel type | 0001 |
| typeDefinition | Description of the channel type | DirectSpeakers |
| importance | Importance of the track rated from 0 - 10. This allows a renderer to discard packs with an importance below a pre-defined threshold | 7 |
| **Sub Element** | **Description & Quantity** | **Example** |
| audioChannelFormatIDRef | Reference to an *audioChannelFormat*, [0 .. *] | AC_00010001 |
| audioPackFormatIDRef | Reference to an *audioPackFormat*, [0 .. *] | AP_00010002 |
| absoluteDistance | Absolute maximum distance in meters, [0 or 1] | 3.9 |

Table 8.6 : The *audioPackFormat* attributes

The *absoluteDistance* sub element specifies the maximum distance a speaker, sound object, or channel may appear from the origin. These objects will contain a normalized distance between 0 and 1 which can be applied to the absolute distance to find out the actual distance. Shown in Listing 8.5 is sample code of an *audioPackFormat* block.

```
<audioPackFormat audioPackFormatID="AP_00040001" audioPackFormatName="DIRECT_STEREO"
typeLabel="0001" typeDefinition="DirectSpeakers">
    <audioChannelFormatIDRef>AC_00010001</audioChannelFormatIDRef>
    <audioChannelFormatIDRef>AC_00010002</audioChannelFormatIDRef>
    <absoluteDistance>5.0</absoluteDistance>
</audioPackFormat>
```

Listing 8.5: An example of an *audioPackFormat* block

# Appendix 8.2

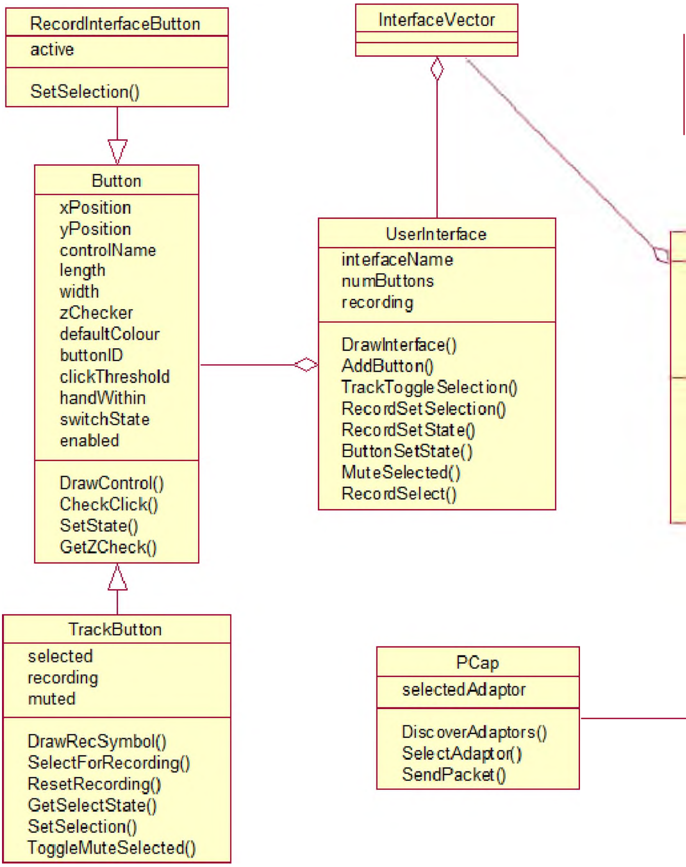
## The Sketchup plugin for the KinectSound system

```
Sketchup.active_model.active_entities.erase_entities Sketchup.active_model.active_entities[0]
model = Sketchup.active_model
ven_def = Sketchup.active_model.definitions.load("C:\\Program Files (x86)\\SketchUp\\SketchUp
      2013\\Components\\Audio\\Venue.skp")
ven_location = Geom::Point3d.new 0.mm,0.mm,0.mm
transform = Geom::Transformation.new ven_location
entities = Sketchup.active_model.active_entities
instance = entities.add_instance ven_def, transform

def Connect ( )
   SKSocket.disconnect
   puts "Connecting to 127.0.0.1 (LOCALHOST) on port 30456"
   SKSocket.connect "127.0.0.1" , 30456
   puts "Connection established"
   SKSocket.add_socket_listener{|recvData| socket_listener(recvData)}
end

def SendData(data)
   SKSocket.write data
end

def socket_listener(sockData)
   puts sockData
   codeArr = sockData.split(":")
   crosshair_def = Sketchup.active_model.definitions.load("C:\\Program Files (x86)\\SketchUp\\
        SketchUp 2013\\Components\\Audio\\Crosshair.skp")

   #Moving the crosshair on all 3 axes
   crosshair_location = Geom::Point3d.new codeArr[0].to_f.mm-150.mm,codeArr[2].to_f.mm-150.mm,
        codeArr[1].to_f.mm-150.mm

   transform = Geom::Transformation.new crosshair_location
   entities = Sketchup.active_model.active_entities
   instance = entities.add_instance crosshair_def, transform
   Sketchup.active_model.active_entities.erase_entities Sketchup.active_model.active_entities[1]

end

def init()
   crosshair_def = Sketchup.active_model.definitions.load("C:\\Program Files (x86)\\SketchUp\\
        SketchUp 2013\\Components\\Audio\\Crosshair.skp")
   crosshair_location = Geom::Point3d.new -150.mm,-150.mm,-150.mm
   transform = Geom::Transformation.new crosshair_location
   entities = Sketchup.active_model.active_entities
   instance = entities.add_instance crosshair_def, transform

   eye = [-0.156, -297.18, 297.77]
   target = [-0.272, 407.261, -381.39]
   up = [-0.00, 0.684, 0.729]
    my_camera = Sketchup::Camera.new eye, target, up
    view = Sketchup.active_model.active_view
    view.camera = my_camera
end

def cameraStats
   view = Sketchup.active_model.active_view
   camera = view.camera
   eye = camera.eye
   target = camera.target
   up = camera.up
   #UI.messagebox "Eye: " + eye.to_s + "\nTarget: " + target.to_s + "\nUp: " + up.to_s
```

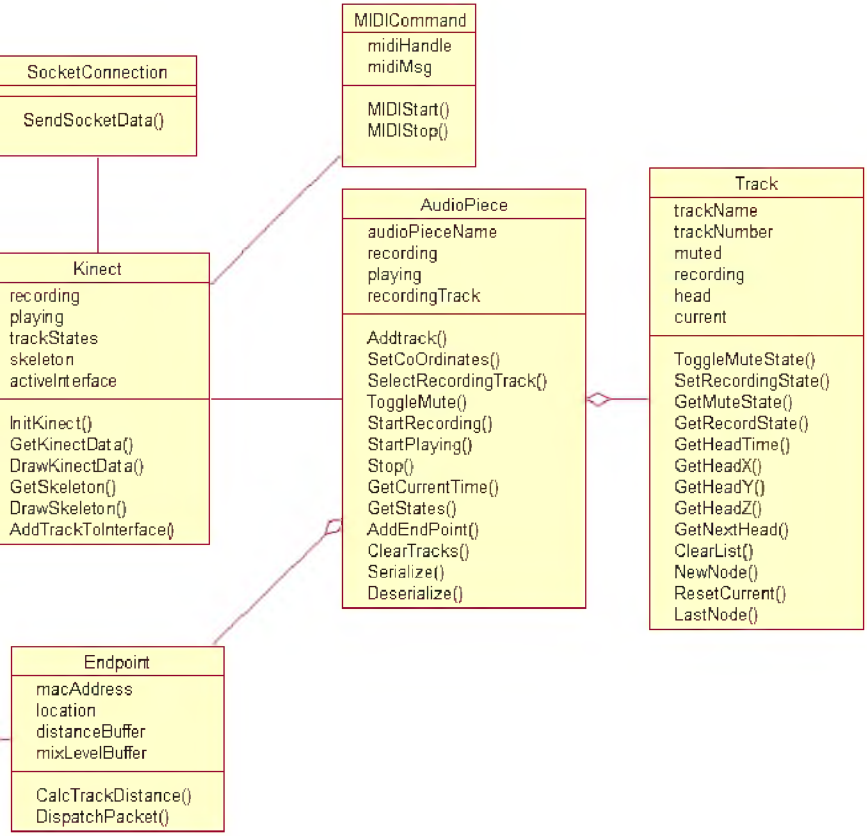
```
    UI.messagebox "Eye: " + (eye[0]*1).to_s + ", " + (eye[1]*1).to_s + ", " + (eye[2]*1).to_s +
    "\nTarget: " + (target[0]*1).to_s + ", " + (target[1]*1).to_s + ", " + (target[2]*1).to_s +
    "\nUp: " + (up[0]*1).to_s + ", " + (up[1]*1).to_s + ", " + (up[2]*1).to_s
end

Connect()
init()
```

# Appendix 8.3

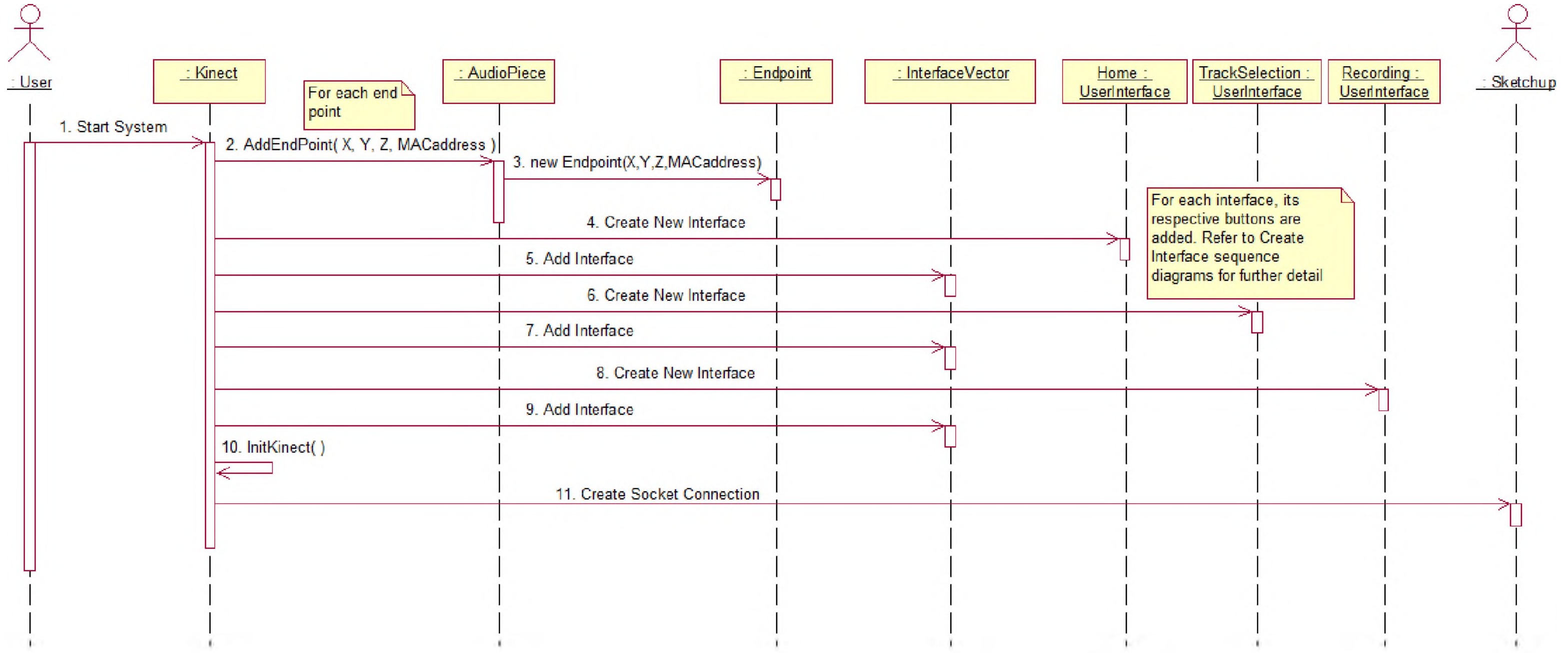
# Class diagram and sequence diagrams

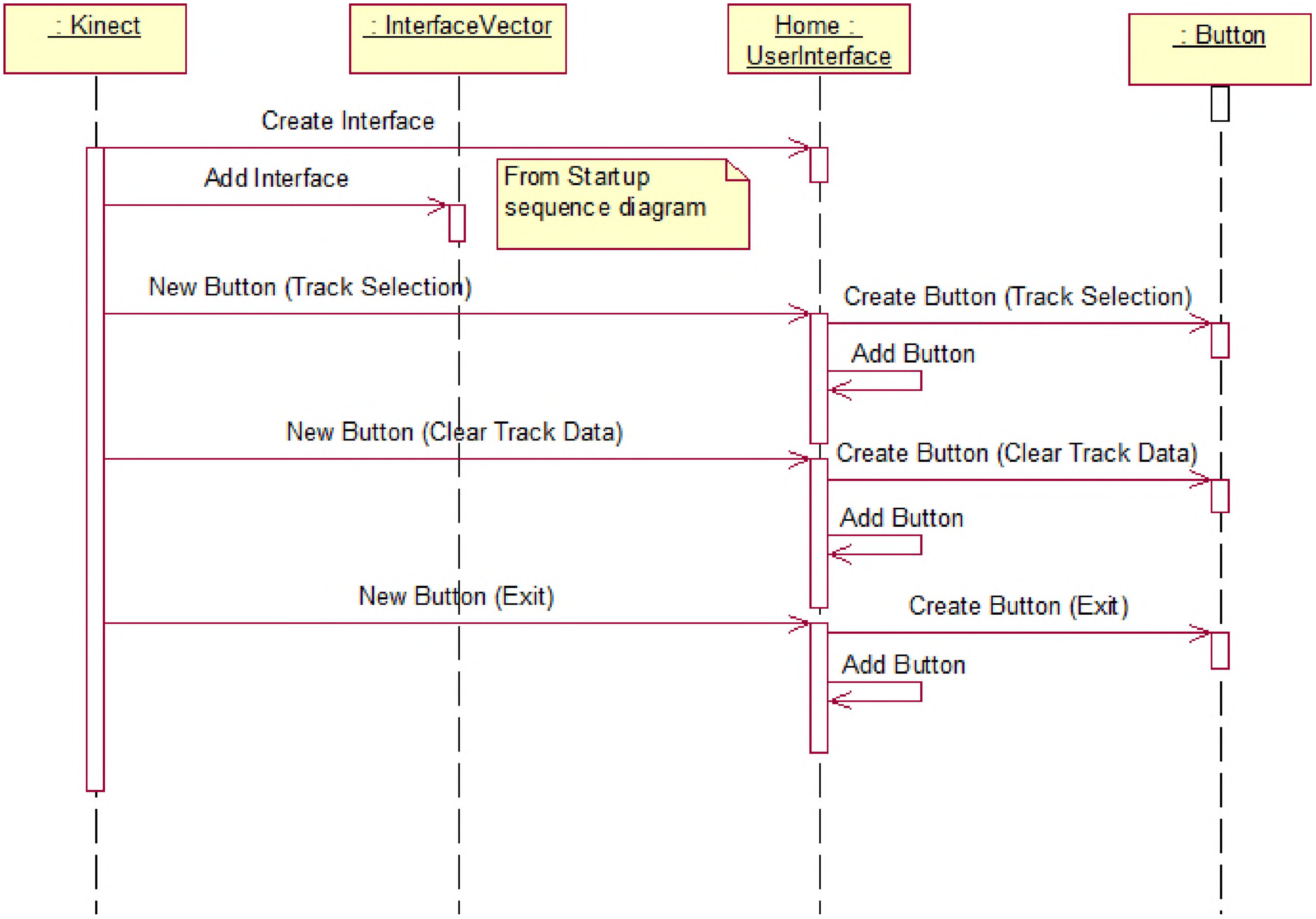


The class diagram

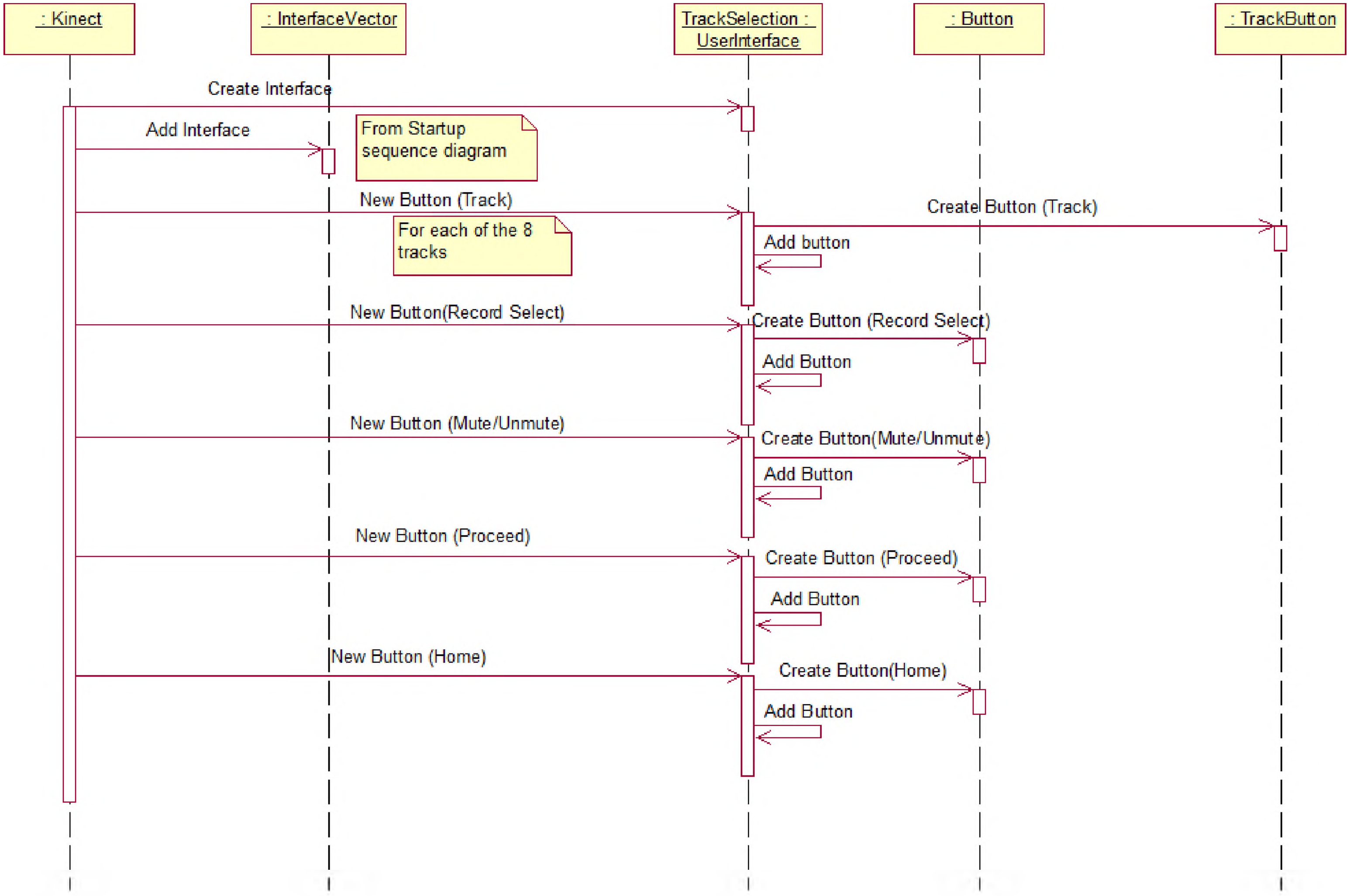

MIDICommand
midiHandle
midiMsg
MIDIStart()
MIDIStop()
SocketConnection
SendSocketData()
Kinect
recording
playing
trackStates
skeleton
activeInterface
InitKinect()
GetKinectData()
DrawKinectData()
GetSkeleton()
DrawSkeleton()
AddTrackToInterface()
AudioPiece
audioPieceName
recording
playing
recordingTrack
Addtrack()
SetCoOrdinates()
SelectRecordingTrack()
ToggleMute()
StartRecording()
StartPlaying()
Stop()
GetCurrentTime()
GetStates()
AddEndPoint()
ClearTracks()
Serialize()
Deserialize()
Track
trackName
trackNumber
muted
recording
head
current
ToggleMuteState()
SetRecordingState()
GetMuteState()
GetRecordState()
GetHeadTime()
GetHeadX()
GetHeadY()
GetHeadZ()
GetNextHead()
ClearList()
NewNode()
ResetCurrent()
LastNode()
Endpoint
macAddress
location
distanceBuffer
mixLevelBuffer
CalcTrackDistance()
DispatchPacket()

Sequence diagram - Startup

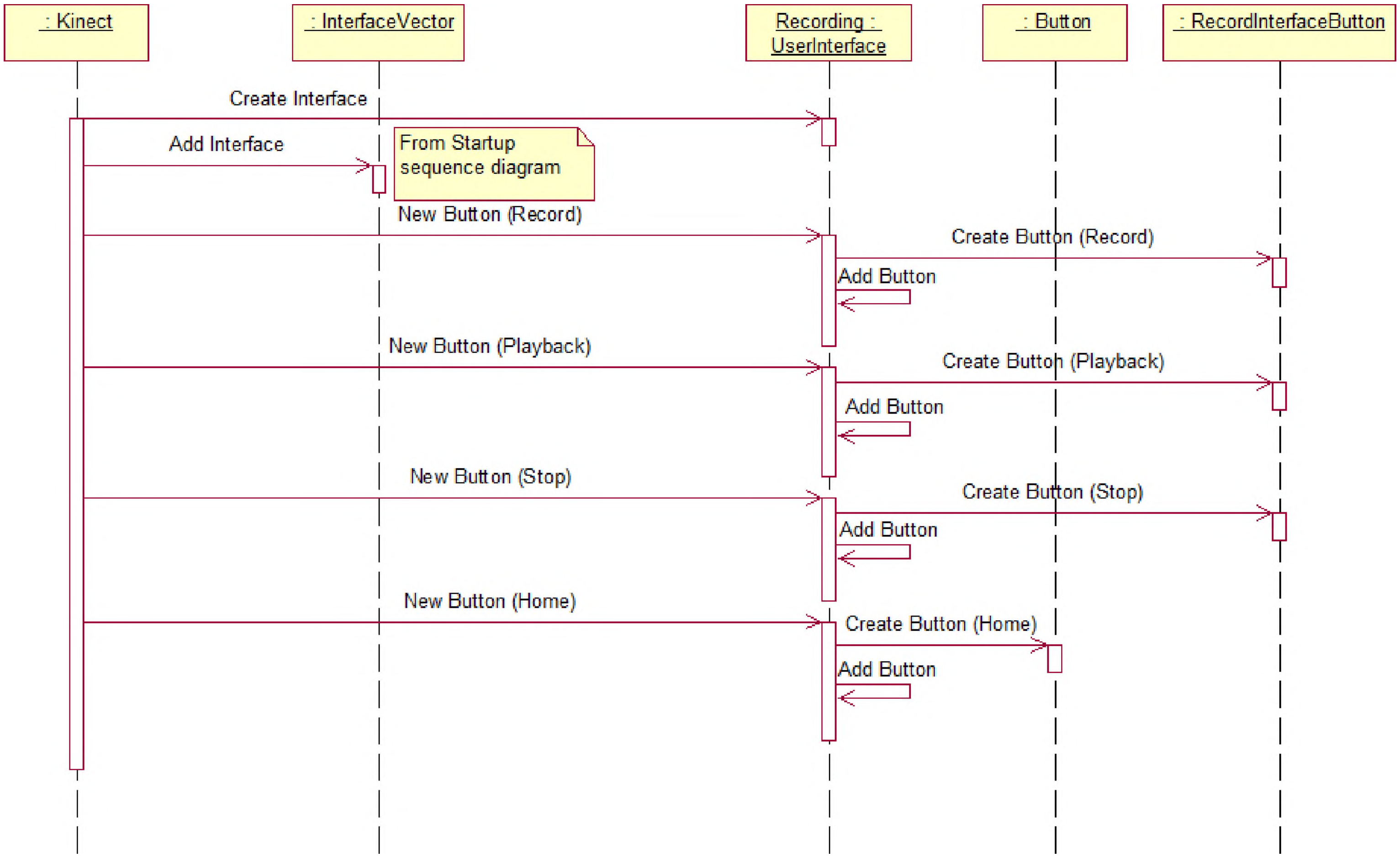

: User
: Kinect
For each end point
: AudioPiece
: Endpoint
: InterfaceVector
Home : UserInterface
TrackSelection : UserInterface
Recording : UserInterface
: Sketchup
1. Start System
2. AddEndPoint( X, Y, Z, MACaddress )
3. new Endpoint(X,Y,Z,MACaddress)
For each interface, its respective buttons are added. Refer to Create Interface sequence diagrams for further detail
4. Create New Interface
5. Add Interface
6. Create New Interface
7. Add Interface
8. Create New Interface
9. Add Interface
10. InitKinect( )
11. Create Socket Connection


: Kinect
: InterfaceVector
Home : UserInterface
: Button
Create Interface
Add Interface
From Startup sequence diagram
New Button (Track Selection)
Create Button (Track Selection)
Add Button
New Button (Clear Track Data)
Create Button (Clear Track Data)
Add Button
New Button (Exit)
Create Button (Exit)
Add Button

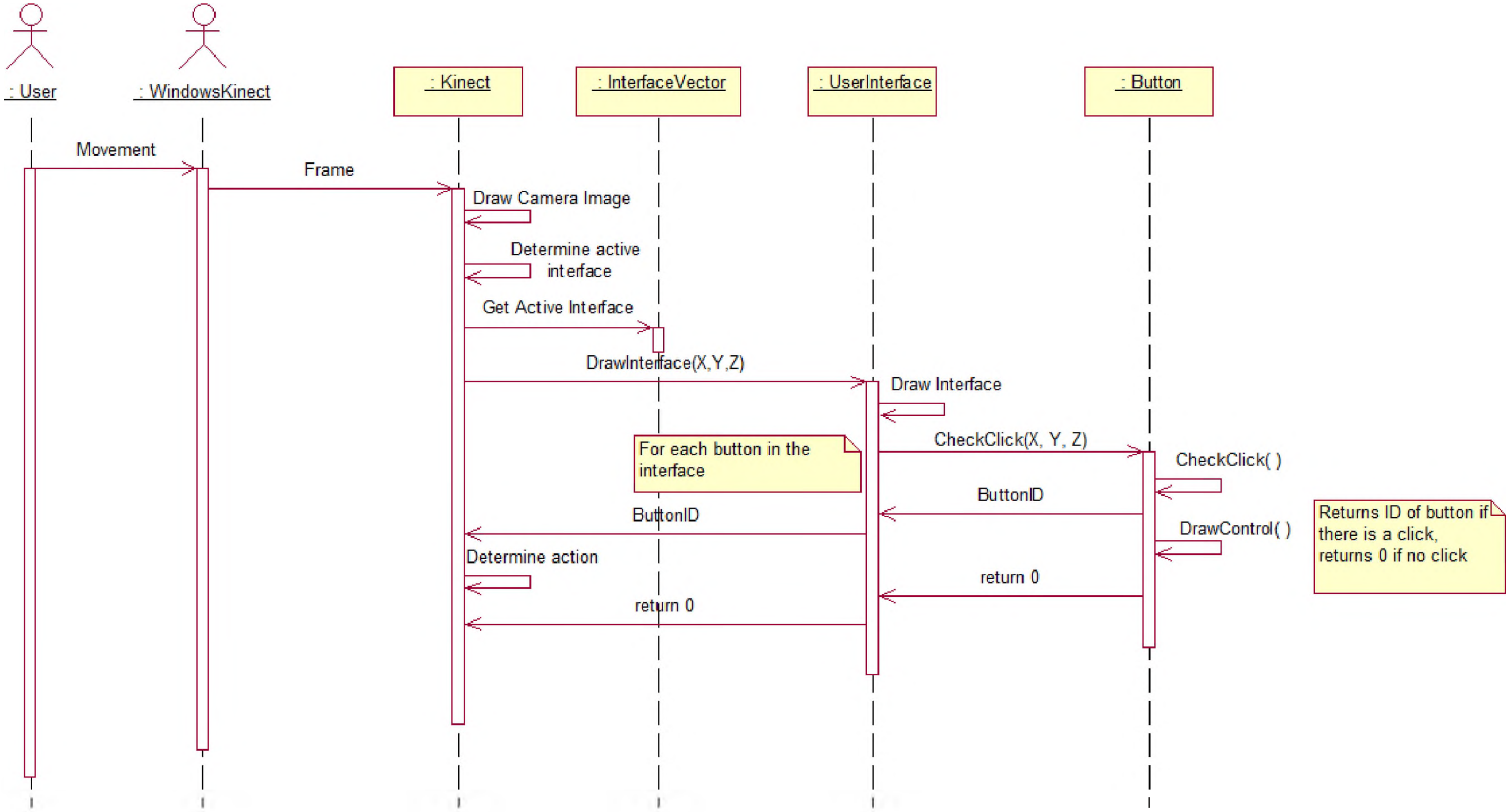
Sequence diagram - Create home interface



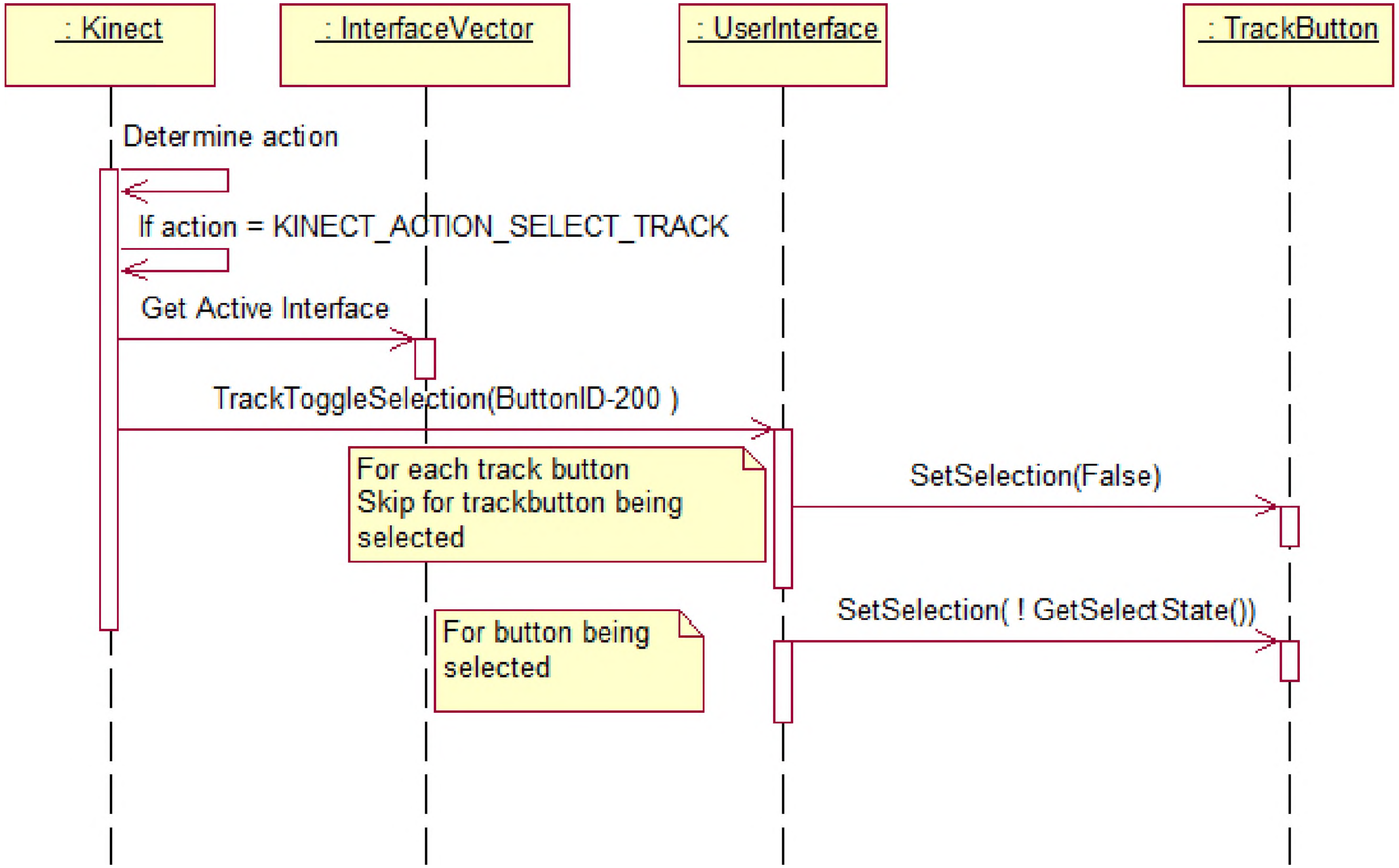
Sequence diagram - Create track selection interface



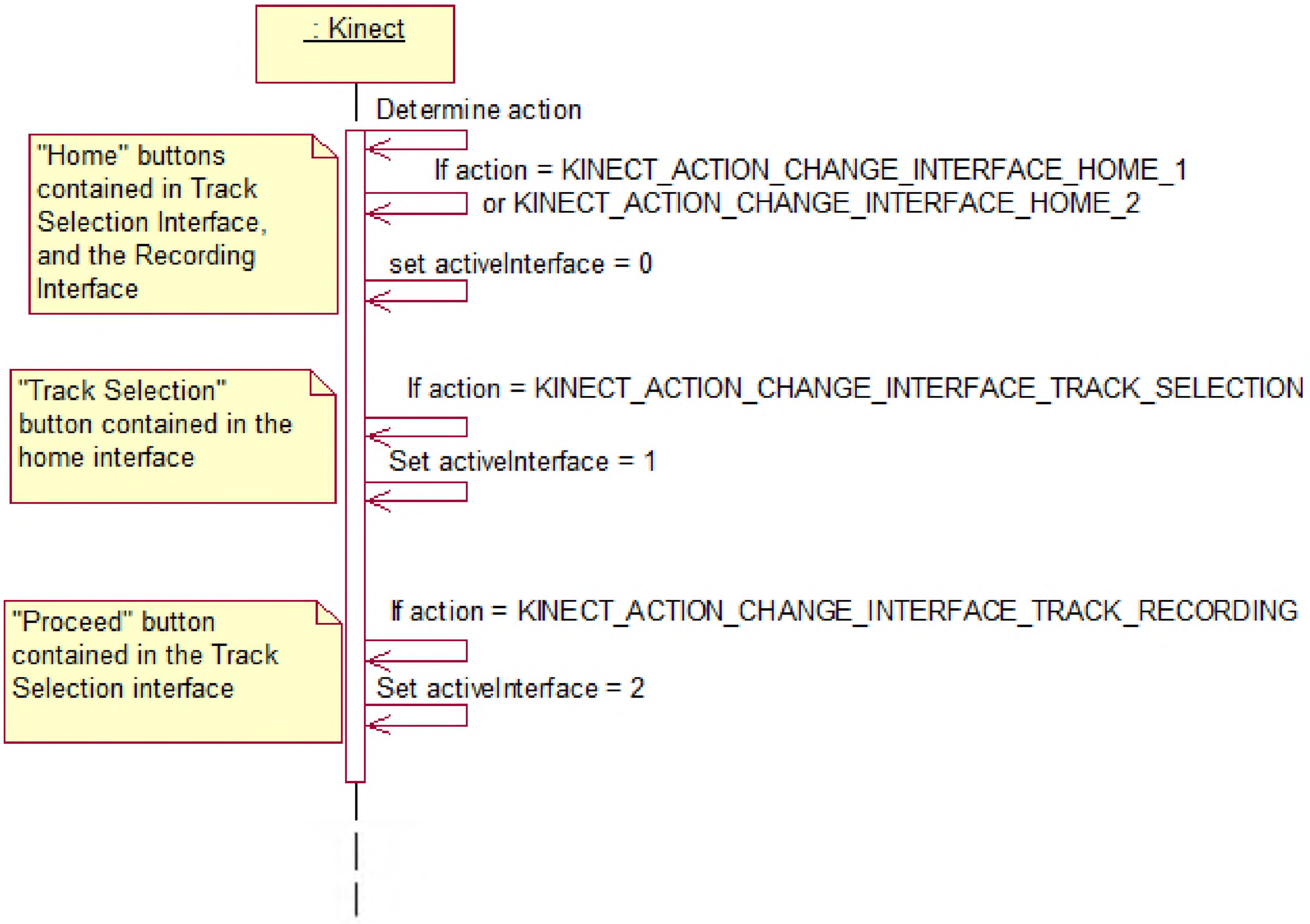
Sequence diagram - Create recording interface



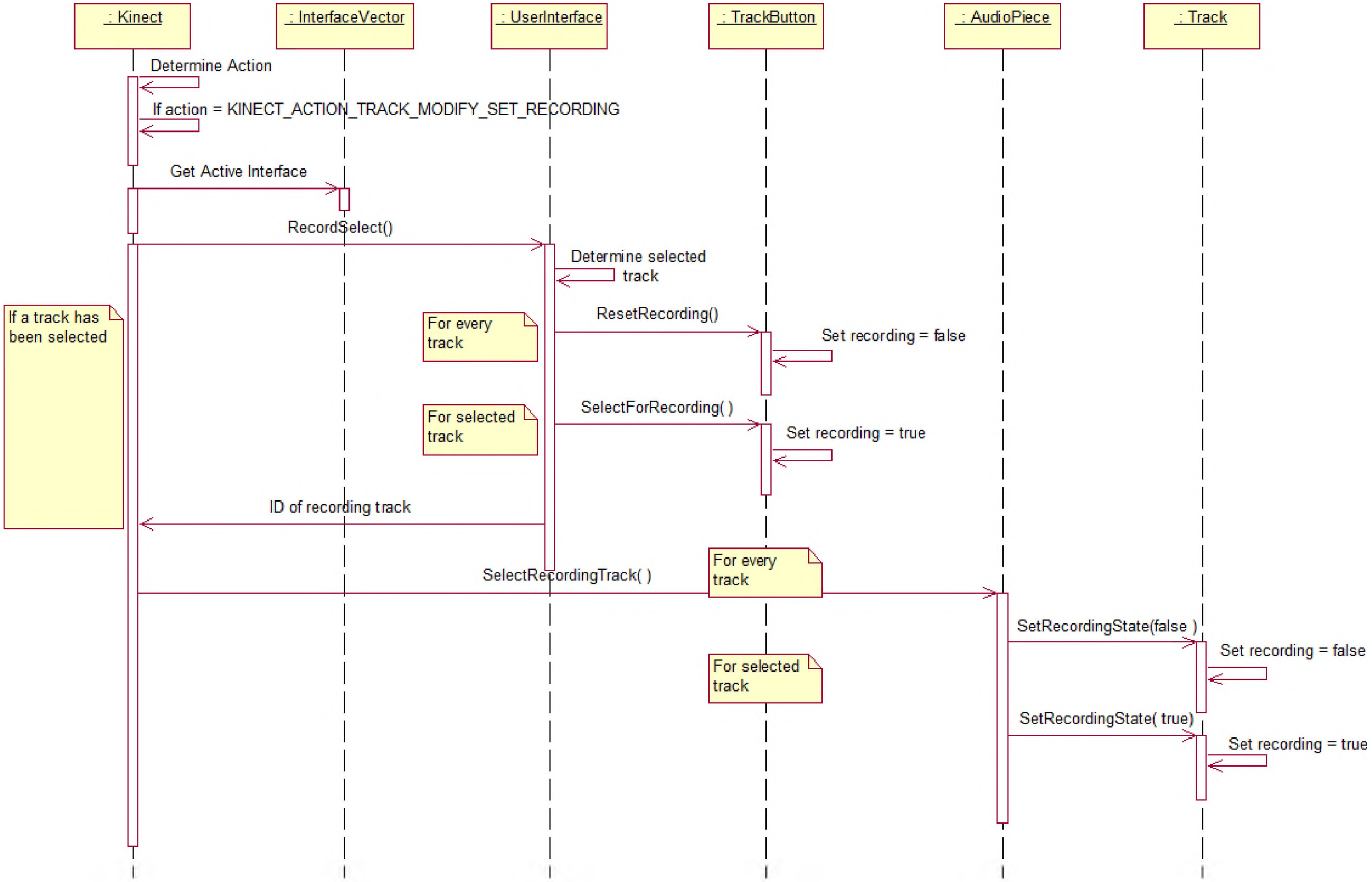
Sequence diagram - Determine task selection



Sequence diagram - Determine track selection

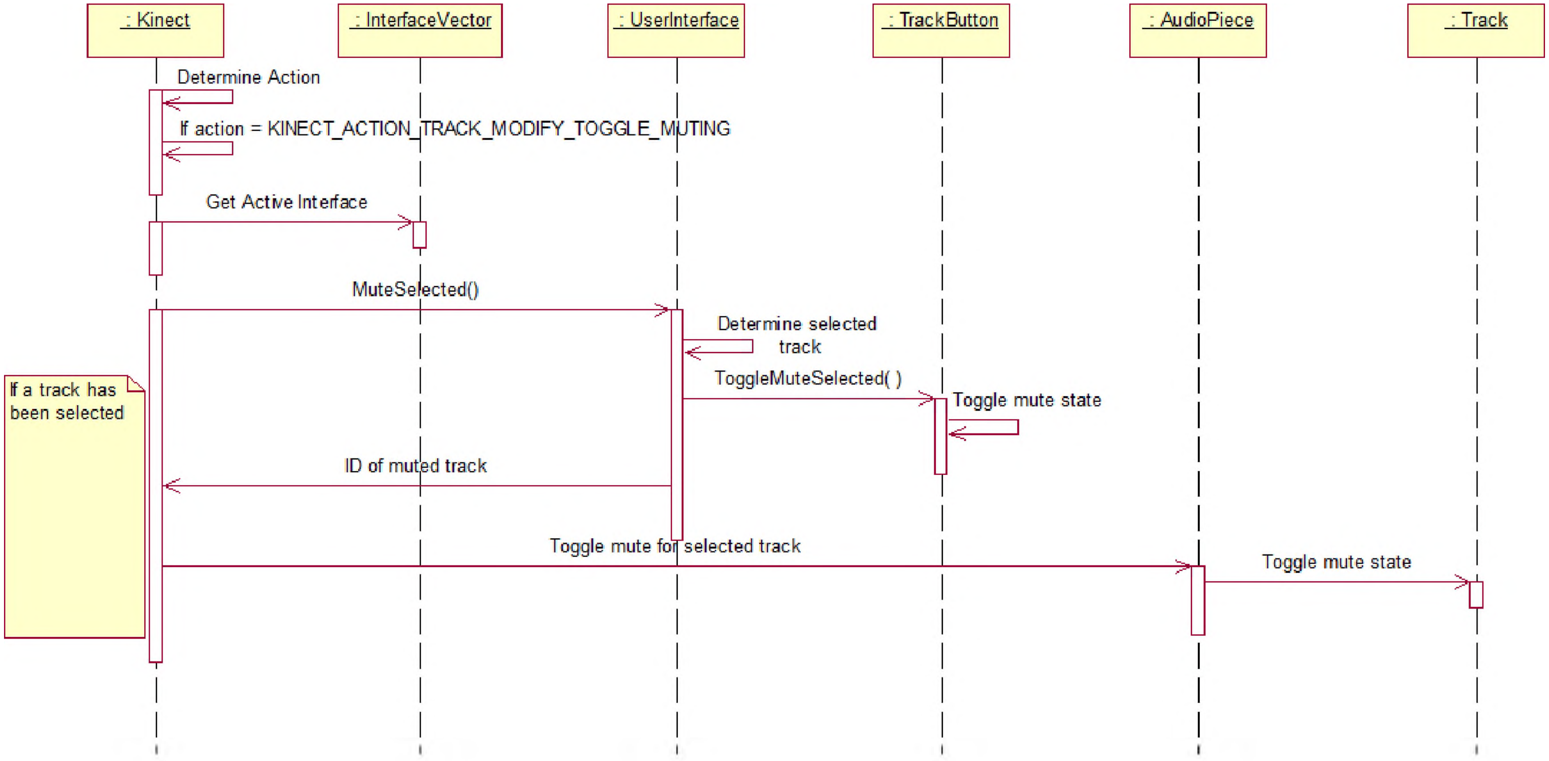

: Kinect
Determine action
"Home" buttons contained in Track Selection Interface, and the Recording Interface
If action = KINECT_ACTION_CHANGE_INTERFACE_HOME_1
or KINECT_ACTION_CHANGE_INTERFACE_HOME_2
set activeInterface = 0
"Track Selection" button contained in the home interface
If action = KINECT_ACTION_CHANGE_INTERFACE_TRACK_SELECTION
Set activeInterface = 1
"Proceed" button contained in the Track Selection interface
If action = KINECT_ACTION_CHANGE_INTERFACE_TRACK_RECORDING
Set activeInterface = 2

Sequence diagram - Change interface

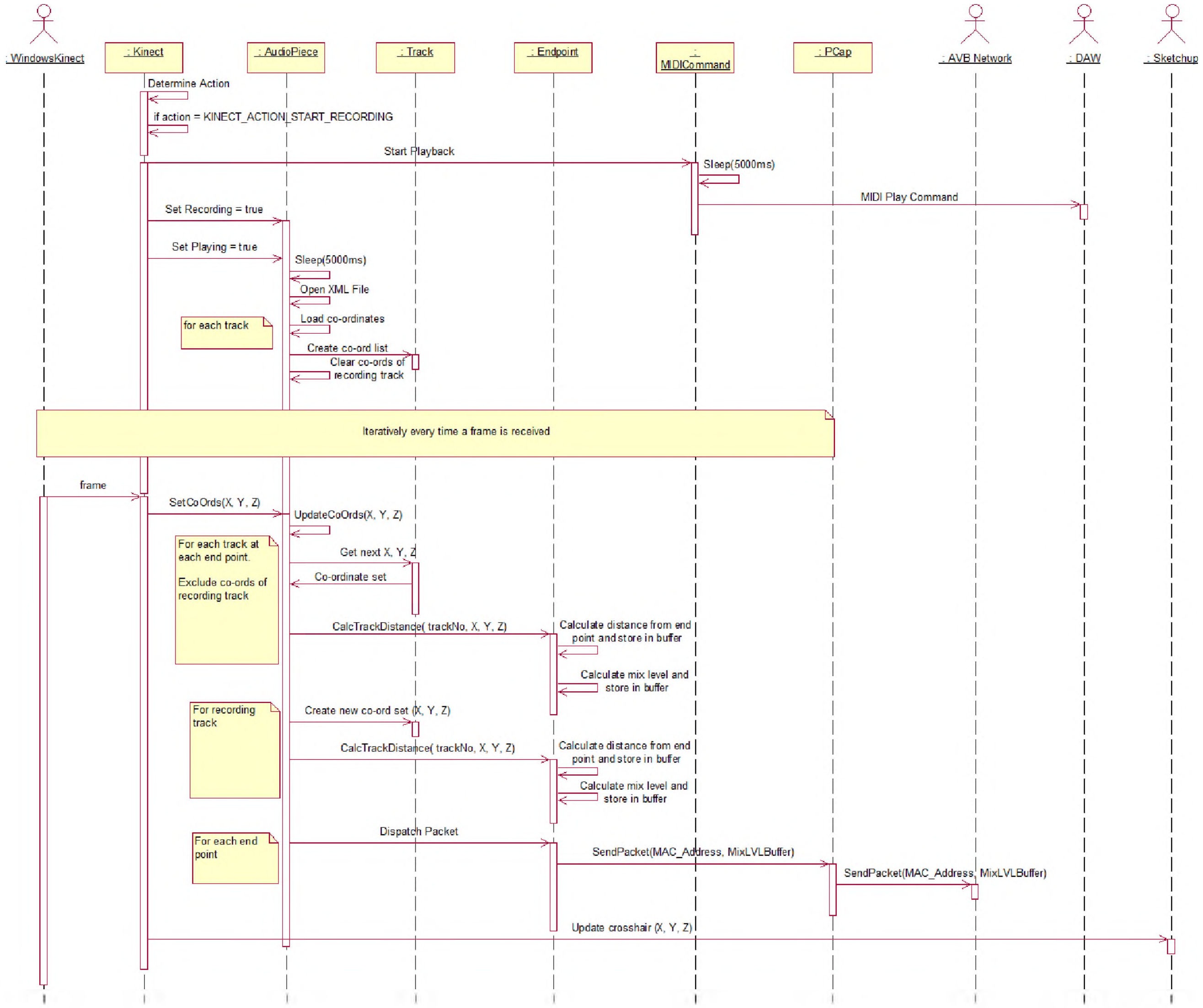


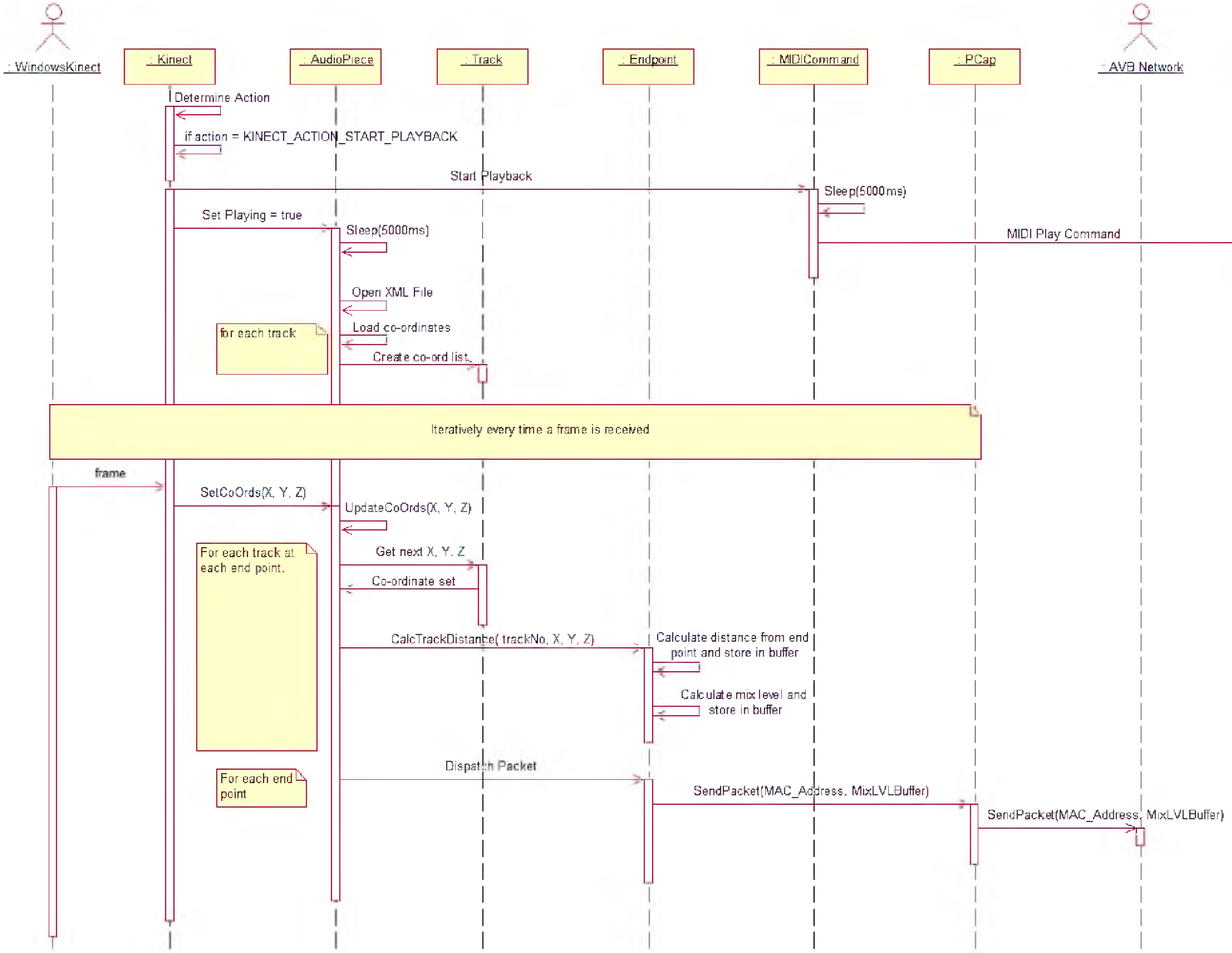
Sequence diagram - Arm a track for recording

Sequence diagram - Mute a track

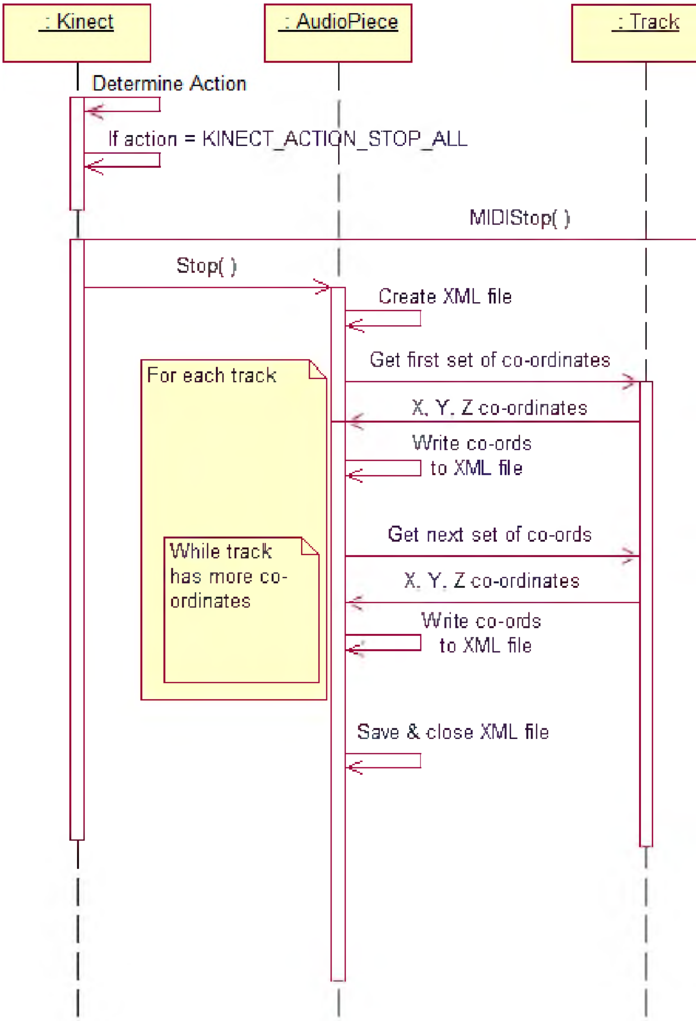


Sequence diagram - Recording a track

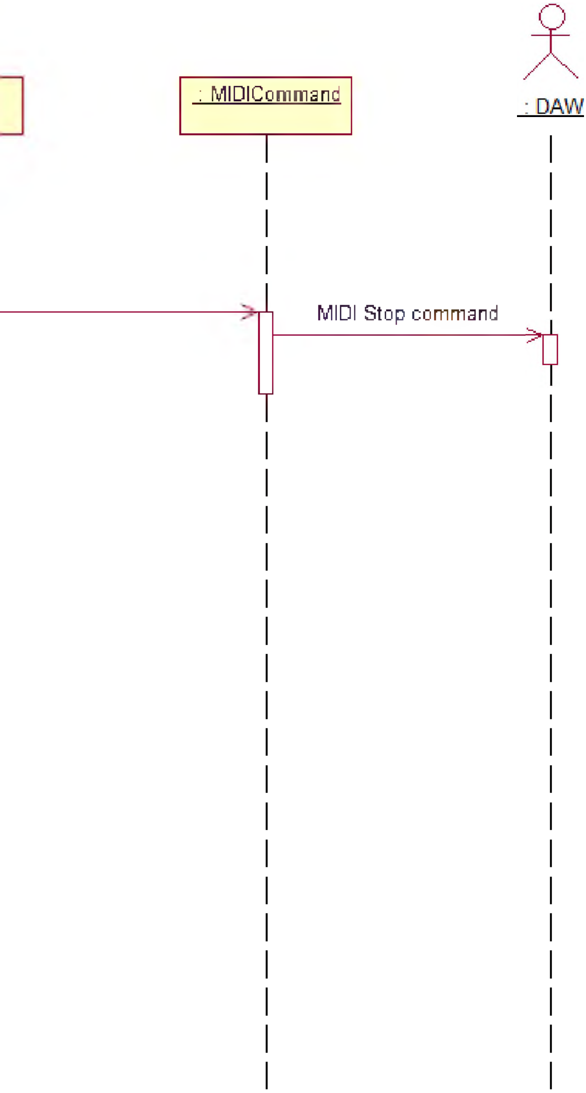

: WindowsKinect
: Kinect
: AudioPiece
: Track
: Endpoint
: MIDICommand
: PCap
: AVB Network
: DAW
: Sketchup
Determine Action
if action = KINECT_ACTION_START_RECORDING
Start Playback
Sleep(5000ms)
MIDI Play Command
Set Recording = true
Set Playing = true
Sleep(5000ms)
Open XML File
Load co-ordinates
for each track
Create co-ord list
Clear co-ords of recording track
Iteratively every time a frame is received
frame
SetCoOrds(X, Y, Z)
UpdateCoOrds(X, Y, Z)
For each track at each end point.
Exclude co-ords of recording track
Get next X, Y, Z
Co-ordinate set
CalcTrackDistance( trackNo, X, Y, Z)
Calculate distance from end point and store in buffer
Calculate mix level and store in buffer
For recording track
Create new co-ord set (X, Y, Z)
CalcTrackDistance( trackNo, X, Y, Z)
Calculate distance from end point and store in buffer
Calculate mix level and store in buffer
For each end point
Dispatch Packet
SendPacket(MAC_Address, MixLVLBuffer)
SendPacket(MAC_Address, MixLVLBuffer)
Update crosshair (X, Y, Z)

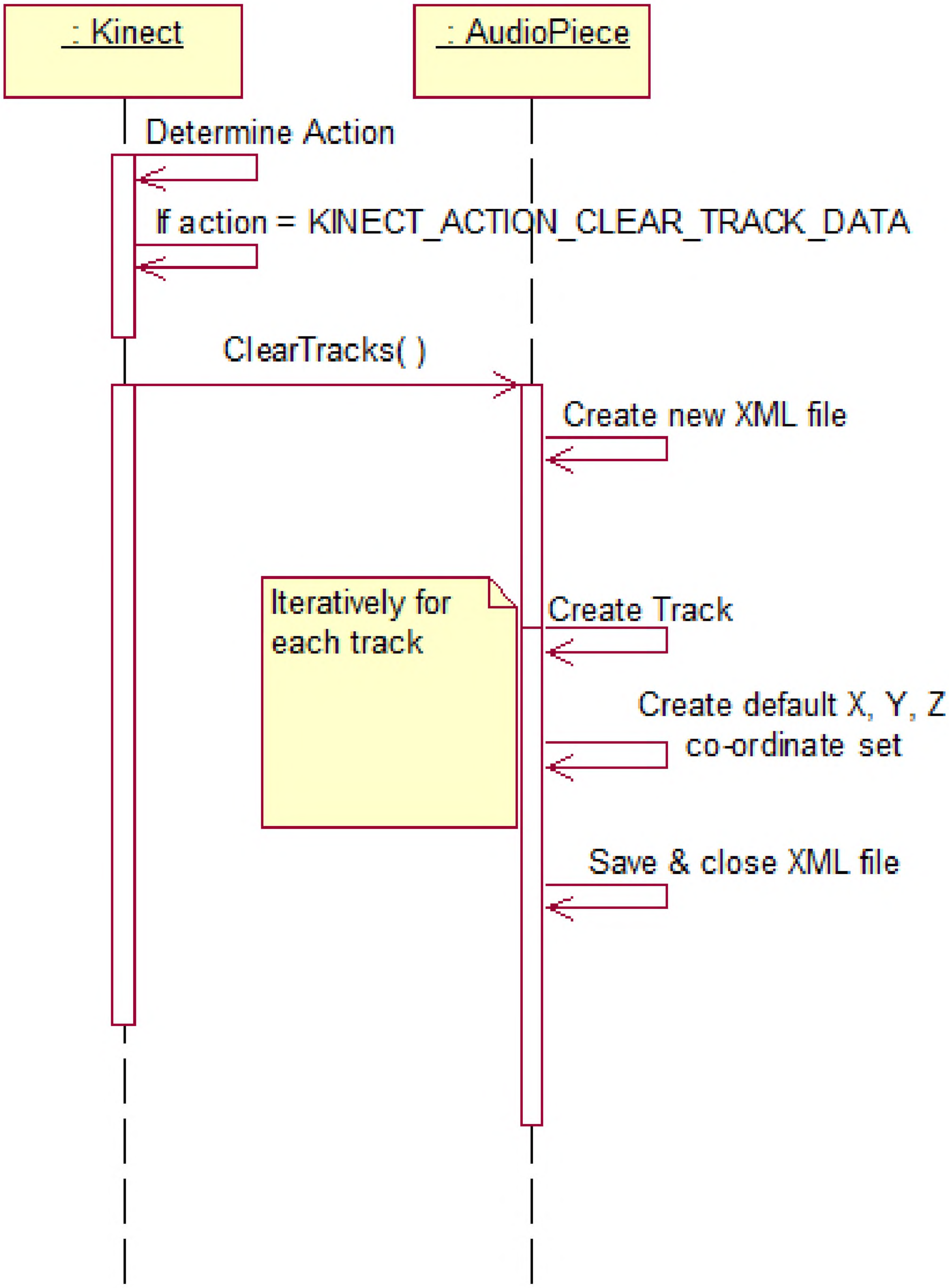


Sequence diagram - Playing a track



Sequence diagram - Saving a track's co-ordinates


: MIDICommand
: DAW
MIDI Stop command


: Kinect
: AudioPiece
Determine Action
If action = KINECT_ACTION_CLEAR_TRACK_DATA
ClearTracks( )
Create new XML file
Iteratively for each track
Create Track
Create default X, Y, Z co-ordinate set
Save & close XML file

Sequence diagram - Resetting a track's co-ordinates

# Appendix 8.4

# Summarized manuals for system testing

# Recording with KinectSound

## Selecting options

The KinectSound interface shows the current menu options available superimposed over the camera image. The Kinect displays the user's left arm (green squares joined with white lines) and right hand (red crosshair). The menu options from all the interfaces have black backgrounds so the user is able to read the text within the control. An example of this is shown in the figure below.

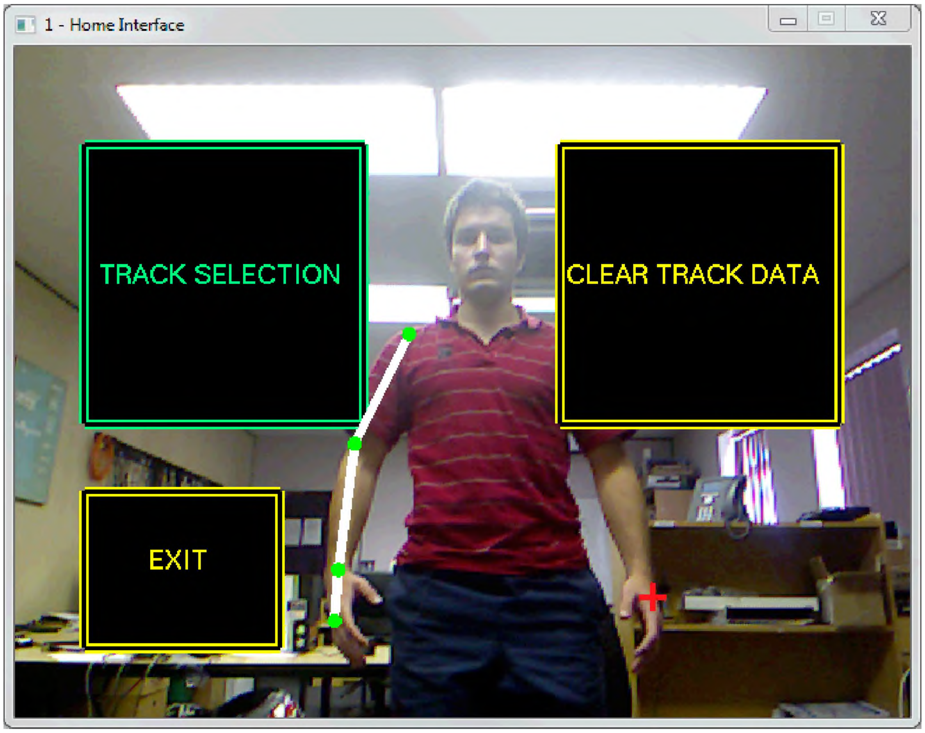


Figure 1: The Kinect detecting a user

The user is able to select any of the menu options by moving their left hand into the menu item and moving it forward as if pressing a button.

## Recording tracks

Upon startup, the KinectSound interface will display the home interface. The user must then navigate to the *Track Selection Interface* where they are able to change track settings before proceeding to record. The layout of the track selection interface is shown below.

Each track is shown in pale blue. When a track has been selected, the outline

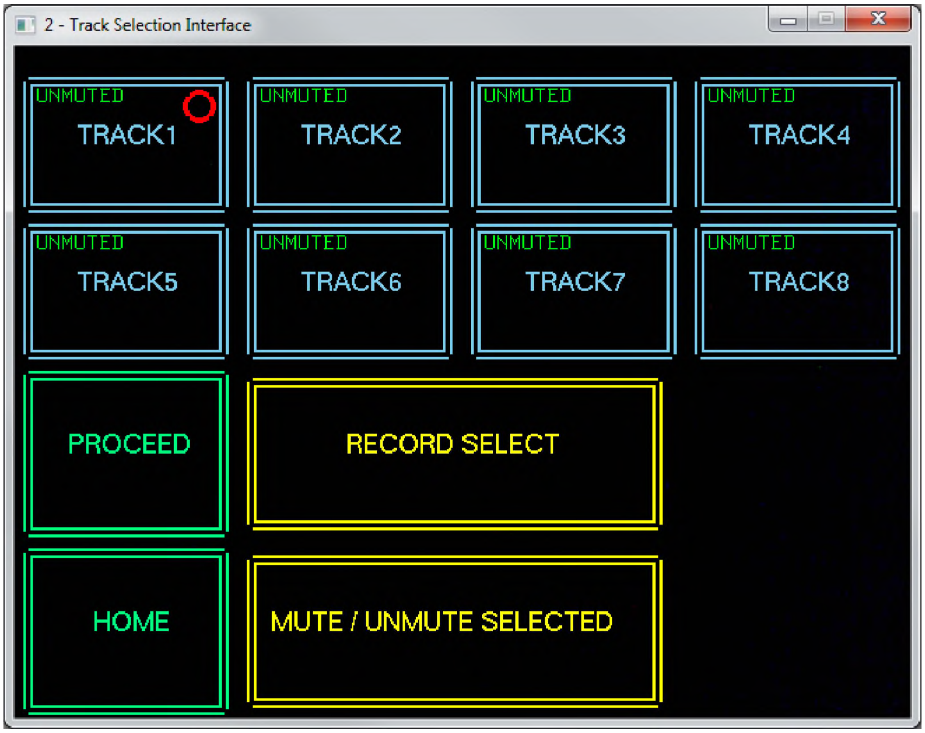


Figure 2: The Track Selection Interface

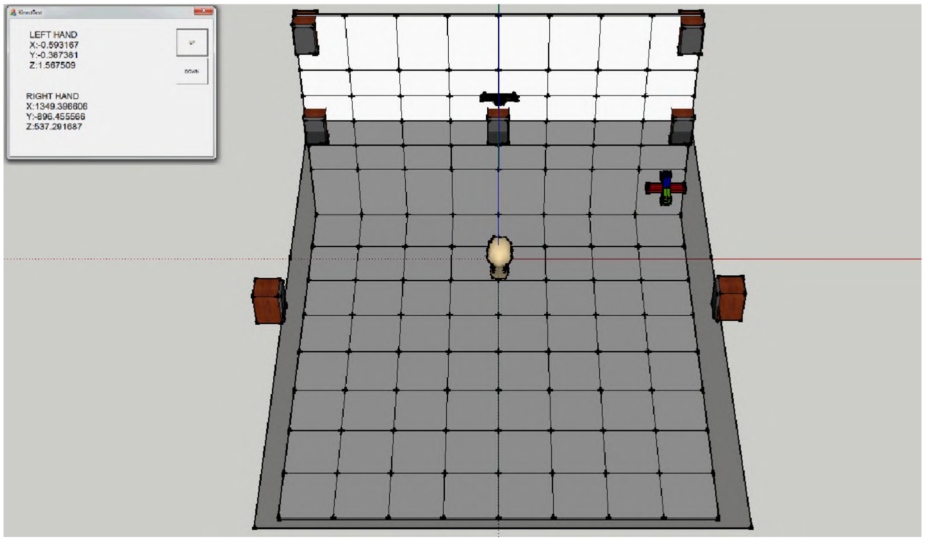
will be shown in green. A selected track can either have its muting state changed or its recording state changed by selecting the *Mute/Unmute Selected* or *Record Select* options respectively. Only one track may be selected for recording at a time. The track that has been selected for recording will have a red circle shown at the top right of it's boundary. The muting state of each track is shown at the top left of the tracks boundary either in red or green, for muted or unmuted respectively. The user may then proceed to the *Recording Interface* by selecting the *Proceed* option.
The *Recording Interface* has options to do the following:

- Record - Begins recording the current position of the user's hand and applies it to the track that the user has chosen to record. All other tracks will be played using their stored co-ordinates. If a user stops recording before the end of a track, the system will assume their last hand position as the co-ordinates used for the rest of the track.
- Playback - Plays the tracks back using their stored co-ordinates
- Stop - Stops the recording or playback

If a track has been muted, then it will not be played back during recording or playback. If a track has been muted but is chosen for recording, then it will still be played whilst the user is recording. All tracks that have not been recorded will have their sound sources as default, which is the center of the room.

## 3D viewing

The user is able to identify their right hand location via co-ordinates or 3D display. The co-ordinates of the user's right hand are shown in a windows dialog box, the 3D display is shown via Sketchup. Shown below are the dialog and Sketchup display.



LEFT HAND
X:-0.593167
Y:-0.387381
Z:1.567509

RIGHT HAND
X:1349.396606
Y:-896.455566
Z:537.291687

Figure 3: 3D viewing of the user's right hand

Sketchup contains a translucent plane at the height of the surround speakers. This allows the user to get a better sense of the crosshair's height. Sketchup also displays the *Ideal listening position* as a user's head in the center of the room.

## Resetting co-ordinates

A user is able to clear any stored data by selecting the *Clear Track Data* option from the home interface. The system will briefly pause to clear any stored data. This will reset all co-ordinates to the center of the room, moving the sound locations to the ideal listening position which is where the user's head is shown in Sketchup.

# Recording with the Spatial Audio Designer (SAD)

The Spatial Audio Designer (SAD) operates as an effect plugin in a Digital Audio Workstation (DAW), the digital audio workstation used for this will be Reaper. The DAW has been set up with the SAD effect already added on, ready for recording. The SAD can be accessed by clicking on the *FX* option of Track 1, and then the *VST: 3D-SAD Mix (New Audio Technology)* option. Shown below in figure 4 is how a user is able to access the SAD.

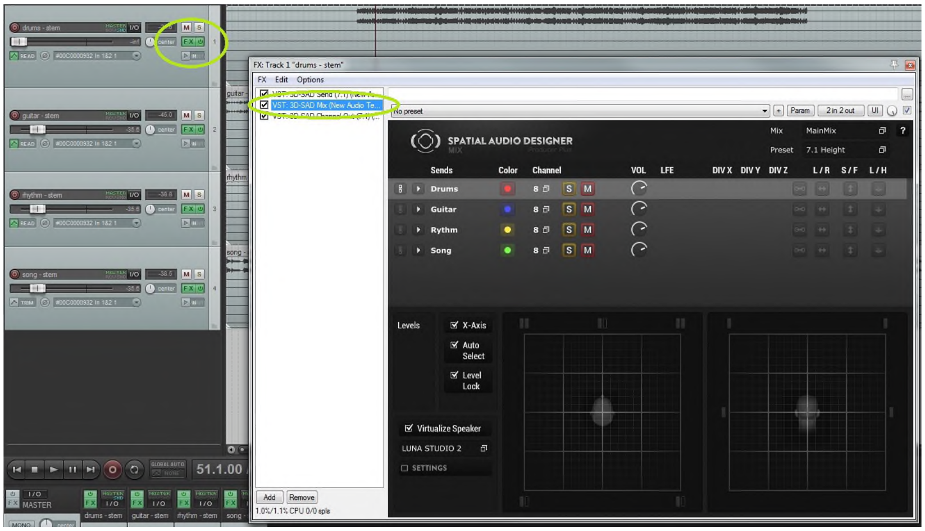
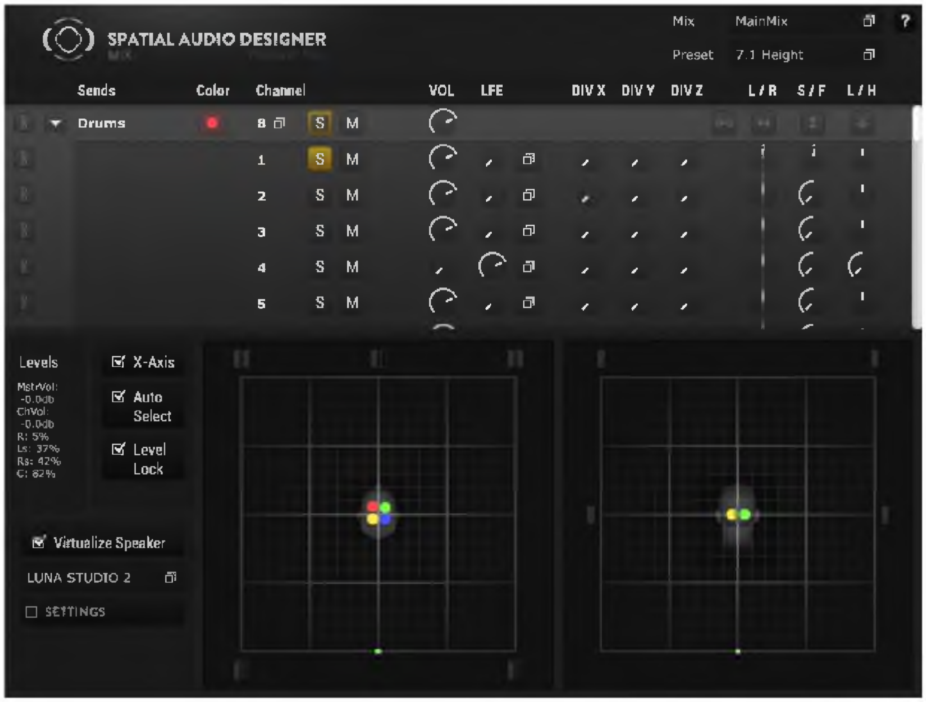


Figure 4: Accessing the SAD in Reaper

The user needs to click on the drop down buttons for each track in order for them to be enabled. Each track has 8 channels for each of the speakers, the mute and solo of the channels for each track can be controlled via the *Channel 1* mute/solo button. Shown below in figure 5 is what the SAD should look like before recording.



Mix MainMix

Preset 7.1 Height

Levels

MstrVol:
-0.0db
ChVol:
-0.0db
R: 5%
Ls: 37%
Rs: 42%
C: 82%

☑ X-Axis

☑ Auto Select

☑ Level Lock

☑ Virtualize Speaker

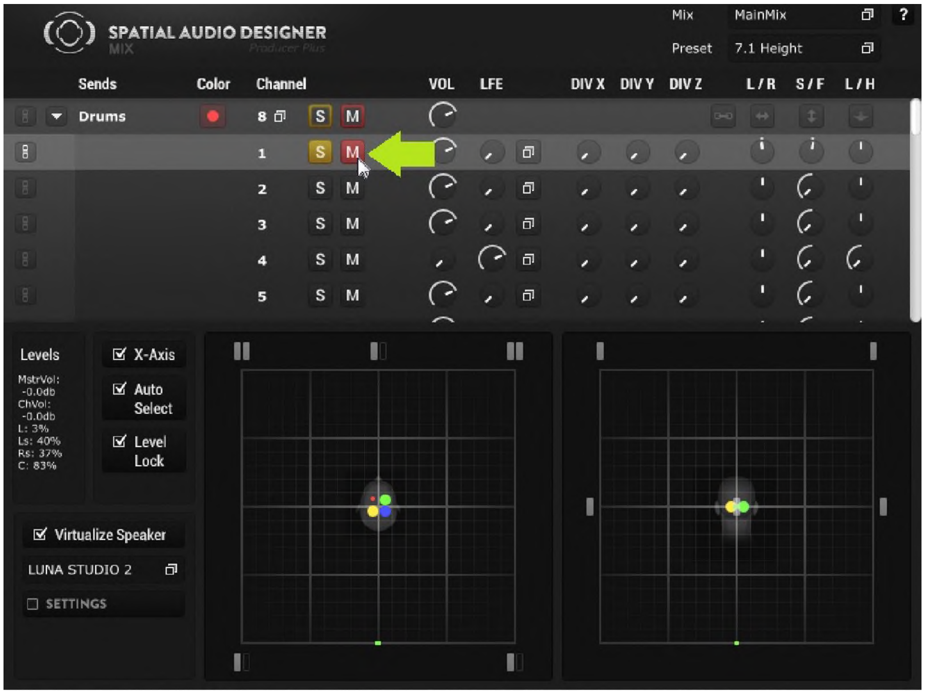
LUNA STUDIO 2

☐ SETTINGS

Figure 5: The SAD before the user begins recording it

## Muting\Unmuting tracks

If a user wishes to mute tracks within the SAD, this can be done by selecting the *M* button located next to *Channel 1* of the track they wish to mute. Shown below in fiigure 6 is how a user would mute a track.



Mix MainMix

Preset 7.1 Height

Levels
MstrVol: -0.0db
ChVol: -0.0db
L: 3%
Ls: 40%
Rs: 37%
C: 83%

☑ X-Axis

☑ Auto Select

☑ Level Lock

☑ Virtualize Speaker

LUNA STUDIO 2

☐ SETTINGS

Figure 6: Muting a track in the SAD

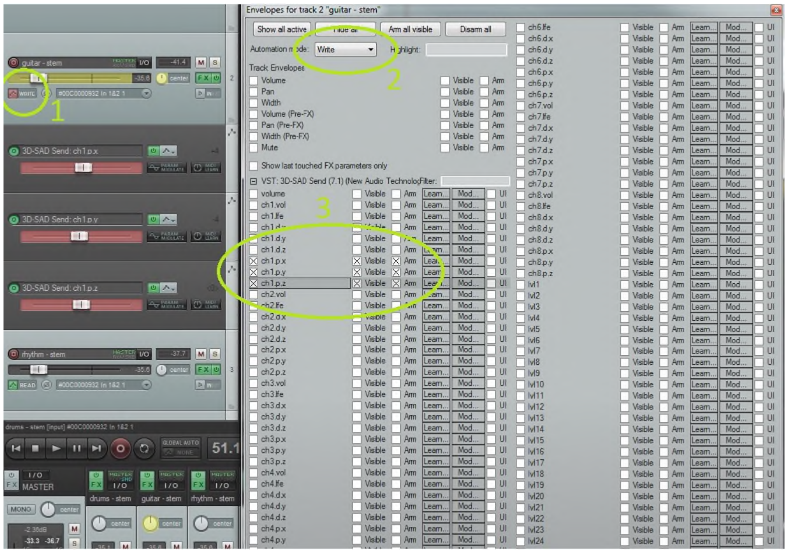
Once a track has been muted, the dot representing the track within the environment will appear a lot smaller. In the figure above, the red dot represents the muted track and has appeared a lot smaller.

## Recording Tracks

A user is able to arm a track for recording by doing the following:

1. Select the *Track Envelopes/Automation* of the track you wish to record in the DAW
2. Change the automation to *Write*
3. Check all the boxes for *ch1.p.x, ch1.p.y* and *ch1.p.z*

These steps for arming a track are shown in figure 7 below



Envelopes for track 2 "guitar - stem"

Automation mode: Write

☒ ch1.p.x ☒ Visible ☒ Arm

☒ ch1.p.y ☒ Visible ☒ Arm

☒ ch1.p.z ☒ Visible ☒ Arm

Figure 7: Arming a track for recording

Once a track has been armed for recording, the user may begin recording by selecting the *Play* control in the DAW. The user can move the armed track around in the 3D space provided by the SAD. The automation will be recording the X, Y, and Z parameters while the user moves them around, this can be seen in the DAW as shown in the figure below.

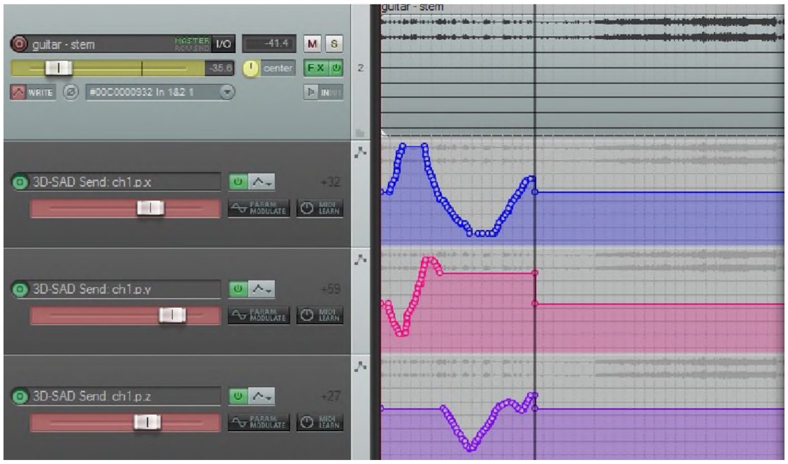


Figure 8: The X, Y, and Z parameters having their values changed whilst the user is recording

The user is able to stop recording by selecting the *Stop* button on the DAW. If the user wishes to keep the last positioned co-ordinate for the rest of the track, they may select the last one on the recorded parameters and delete it. Shown below is what the tracks should look like after doing this.

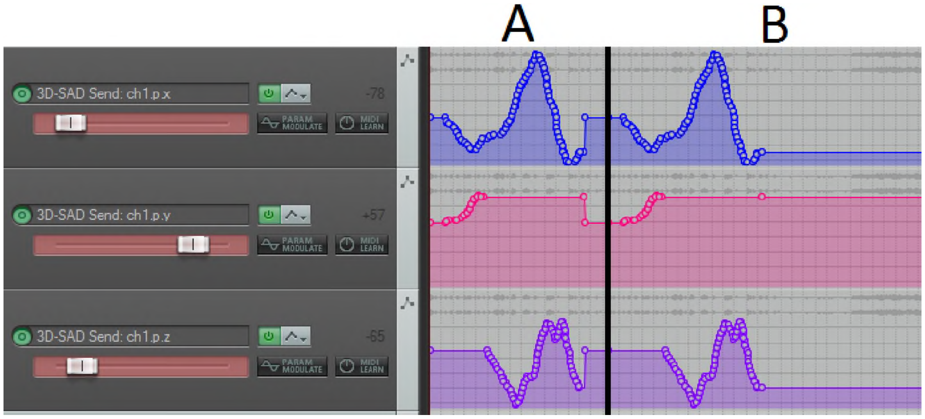


Figure 9: A. Before deleting the last point. B. After deleting the last point

The track can be played back by opening the *Track Envelopes/Automation* control mentioned above and changing the command from *Write* back to *Read.*

## Resetting the co-ordinates

If the user wishes to reset the recorded parameter data, they may do so by selecting the *Hide/Clear* option for each parameter and then selecting the *Clear Envelope* command, followed by the dialog option *Yes*. The user may open the SAD and move the dot indicating *Track 1* of the chosen audio track back to the original position. Shown below is how to clear the envelope data of a track.

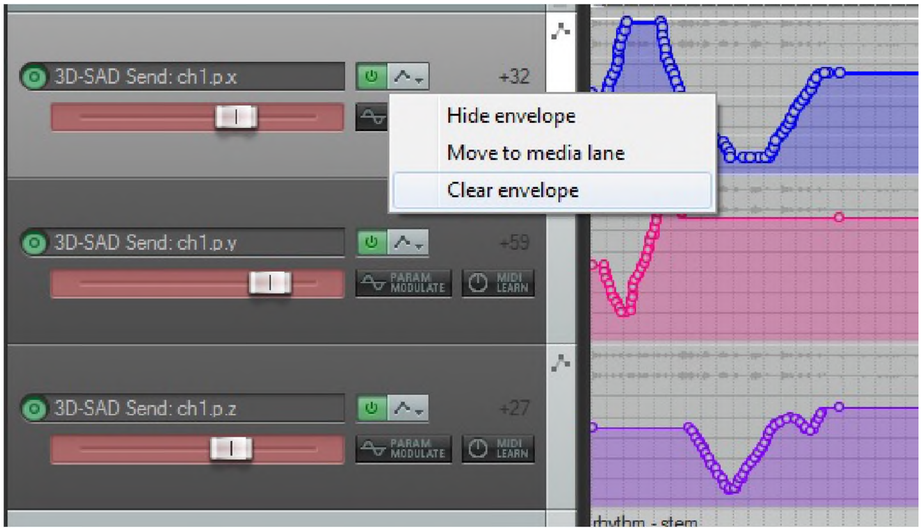


Figure 10: Clearing the data for a single parameter, this needs to be done for each one to remove the automated movement

# Appendix 8.5

# The KinectSound system user manual

Rhodes University

The Kinect Sound project

# User Manual

*Author:*
Mitchell Hedges

*Supervisor:*
Prof. Richard Foss

November 24, 2014

# Device requirements and configuration

The devices required for using the Kinect Sound system are as follows:

1. **Windows Kinect** - The kinect needs to be windows compatible with the latest Kinect SDK installed.

2. **Echo streamware NIC-1** - This card fits into the PCI slot of a workstation and enables the transmission of digital audio packets onto an Audio Video Bridged (AVB) network.

3. **AVB Ethernet Bridge** - An AVB capable Ethernet bridge is required to distribute the audio packets and control packets over the AVB network.

4. **XMOS AVB audio endpoint** - These devices are required for the processing of multiple audio channels and the Digital to Analog (DAC) conversion of audio received from the network. These devices need to be flashed with the required firmware in order to function correctly. This can be done using an *XTAG-2 programmer card* with the *xTIMEcomposer studio*[1] development software.

5. **Powered speakers** - One speaker is required for each endpoint.

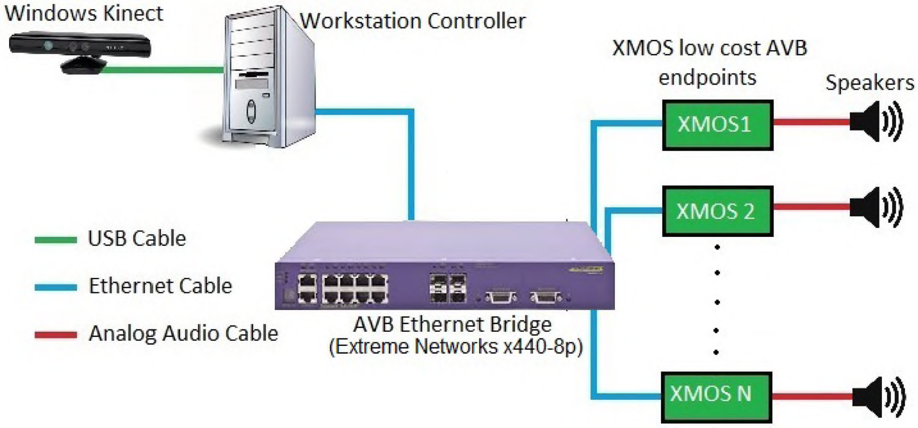


Figure 1: Layout of how the devices are to be configured

# Configuring the digital audio workstation

The Digital Audio Workstation (DAW) is responsible for streaming multiple audio channels to the network. The DAW software shown in this manual is Reaper version 3.72.

[1] Version 13.0 or later is required for proper functionality

## Importing tracks

In order to create a surround sound piece, the user must provide the separate tracks which make up an audio piece. The tracks must be imported into a DAW project and remain as separate tracks in order to have their 3D co-ordinates modified separately. Figure 2 shows the layout of the tracks in a DAW

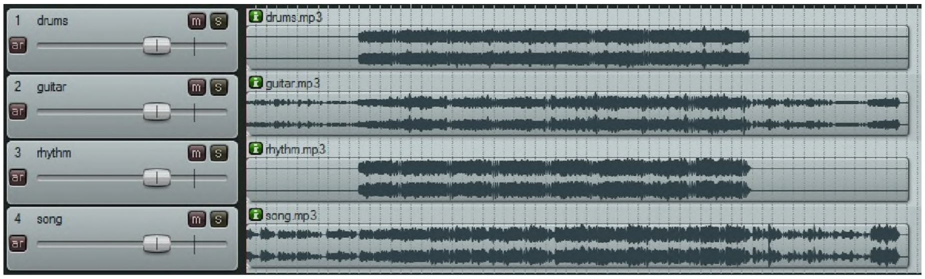


Figure 2: The separate tracks as shown in the DAW, which make up an audio piece

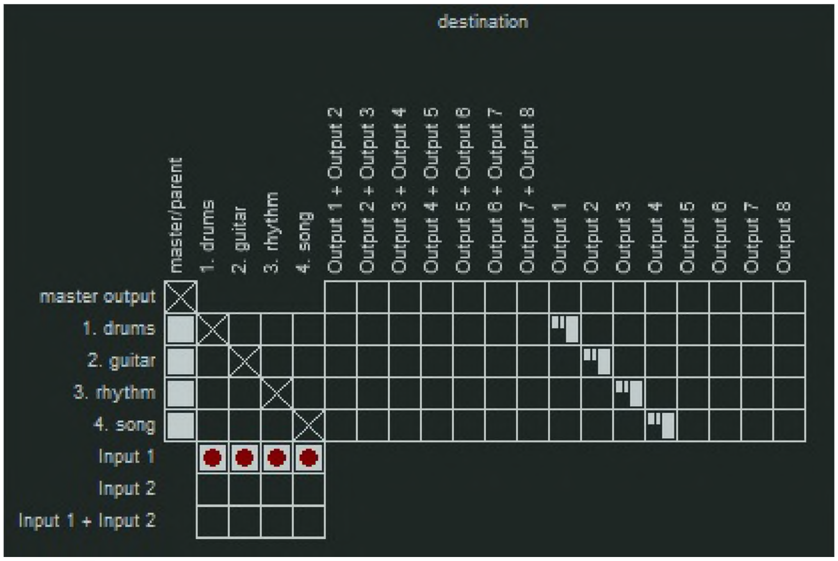
The digital audio workstation has to be configured to allow internal midi commands to control the transport functions. These commands can be found under the *Action List* menu option and are to be configured as follows:

- Transport: Play - MIDI Channel 1 Control/Change (0xB0) control function 0x1A
- Transport: Stop - MIDI Channel 1 Control/Change (0xB0) control function 0x1B

This is required to enable the main system to control the playback of the DAW.

## Routing the tracks tracks

Each track in the DAW is to be routed to its own output channel so they are able to have their locations recorded independently. If the user requires that two single tracks are to have their sound locations the same, then they may route two tracks to a single output. The output tracks correspond to the track numbers seen in the Track Selection user interface. These are best routed as a diagonal to match the shown tracks with the outputs as shown in Figure 3.



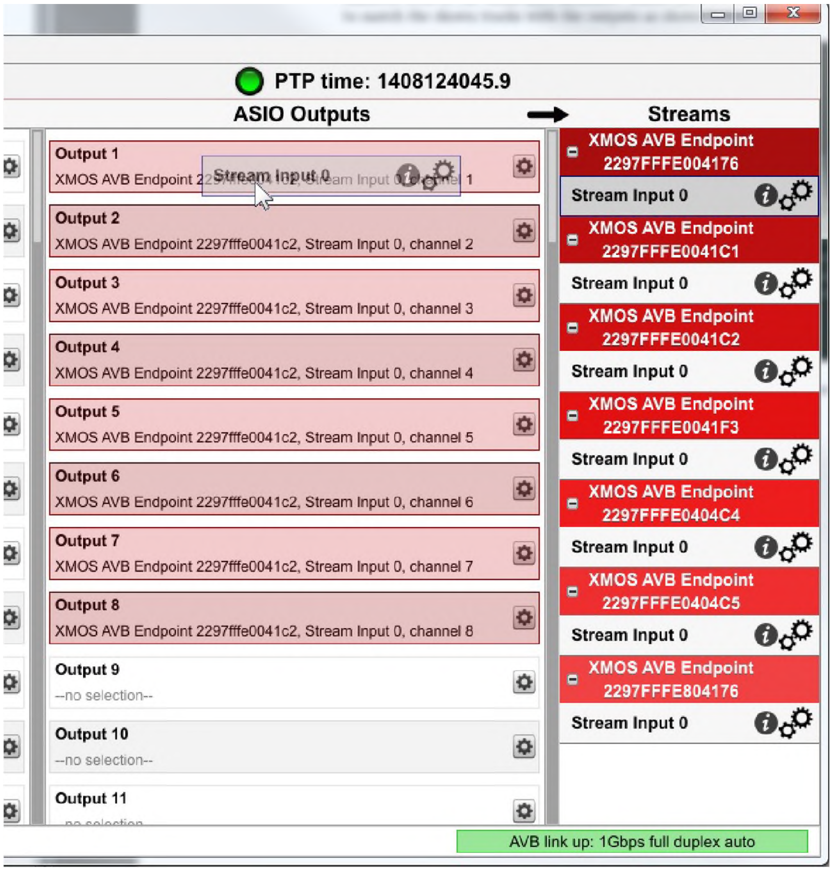
destination

| | master/parent | 1. drums | 2. guitar | 3. rhythm | 4. song | Output 1 + Output 2 | Output 2 + Output 3 | Output 3 + Output 4 | Output 4 + Output 5 | Output 5 + Output 6 | Output 6 + Output 7 | Output 7 + Output 8 | Output 1 | Output 2 | Output 3 | Output 4 | Output 5 | Output 6 | Output 7 | Output 8 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| master output | ☒ | | | | | | | | | | | | | | | | | | | |
| 1. drums | ■ | ☒ | | | | | | | | | | | ▮ | | | | | | | |
| 2. guitar | ■ | | ☒ | | | | | | | | | | | ▮ | | | | | | |
| 3. rhythm | ■ | | | ☒ | | | | | | | | | | | ▮ | | | | | |
| 4. song | ■ | | | | ☒ | | | | | | | | | | | ▮ | | | | |
| Input 1 | | ● | ● | ● | ● | | | | | | | | | | | | | | | |
| Input 2 | | | | | | | | | | | | | | | | | | | | |
| Input 1 + Input 2 | | | | | | | | | | | | | | | | | | | | |

Figure 3: The Reaper routing matrix showing how the tracks are routed to the outputs

## Configuring streams

The user must manually set up the audio streams by using the Echo Streamware Controller provided. The ASIO outputs have to correspond to the stream channels before the stream is connected. This can be done by clicking on a listed device's stream input and dragging it onto the first channel of the ASIO outputs as shown below. The user then needs to click on the gears, and click on *Connect this stream*. This creates an 8 channel stream between the workstation and a single endpoint. Once a stream is connected, the user is then able to multicast connect the stream to the rest of the endpoints by clicking on the gears for each of the other devices and selecting the only option specified under *Multicast connect*. Figure 4 shows how to drag the channels of a device into the ASIO output channel list.



PTP time: 1408124045.9

ASIO Outputs → Streams

Output 1
XMOS AVB Endpoint 2297fffe0041c2, Stream Input 0, channel 1
Stream Input 0

Output 2
XMOS AVB Endpoint 2297fffe0041c2, Stream Input 0, channel 2

Output 3
XMOS AVB Endpoint 2297fffe0041c2, Stream Input 0, channel 3

Output 4
XMOS AVB Endpoint 2297fffe0041c2, Stream Input 0, channel 4

Output 5
XMOS AVB Endpoint 2297fffe0041c2, Stream Input 0, channel 5

Output 6
XMOS AVB Endpoint 2297fffe0041c2, Stream Input 0, channel 6

Output 7
XMOS AVB Endpoint 2297fffe0041c2, Stream Input 0, channel 7

Output 8
XMOS AVB Endpoint 2297fffe0041c2, Stream Input 0, channel 8

Output 9
--no selection--

Output 10
--no selection--

Output 11
--no selection--

XMOS AVB Endpoint 2297FFFE004176
Stream Input 0

XMOS AVB Endpoint 2297FFFE0041C1
Stream Input 0

XMOS AVB Endpoint 2297FFFE0041C2
Stream Input 0

XMOS AVB Endpoint 2297FFFE0041F3
Stream Input 0

XMOS AVB Endpoint 2297FFFE0404C4
Stream Input 0

XMOS AVB Endpoint 2297FFFE0404C5
Stream Input 0

XMOS AVB Endpoint 2297FFFE804176
Stream Input 0

AVB link up: 1Gbps full duplex auto

Figure 4: Specifying the channels with their current outputs

## The User Interfaces

The Human/Computer interaction component is composed of various *interfaces*. Each interface fulfills a specific purpose and provides navigation to relevant interfaces in the system. The interface is made up of a live streaming camera image from the Kinect with the buttons superimposed over it. The interfaces shown in this manual exclude the camera image for ease of explanation. A user may select a control by moving their left hand into the control boundaries and then moving their left hand forward as if they are pressing a button. The camera image indicates the position of the user's left arm. The controls are colour coded as follows.

- **Dark Green** controls represent navigation controls. These controls are responsible for directing the user between interfaces.

- **Yellow** controls represent commands. These controls will perform a specific action or task when selected by the user.

- **Light Blue\Light Green** controls indicate tracks. A track is light blue by default and then changes to light green when a user has selected it. A selected track may have its muting state changed as well as its recording state changed. A track control will have its muting state shown within the control as well as showing a red circle if it is selected for recording. Only one track may be selected for recording at a time.

- **Grey** controls represent any control button that is currently disabled. Disabled controls cannot be interacted with by the user until they become enabled by the system.

- A **Purple** control indicates a command button that has been selected and has changed the state of the system. There are only three controls in the system that may have this colour. These are the *Play, Record,* and *Stop* controls on the recording interface.

### Home Interface

The *Home Interface* is the interface the user is presented with upon starting the system. Shown in Figure 5 is a layout of the controls on the *Home Interface*.

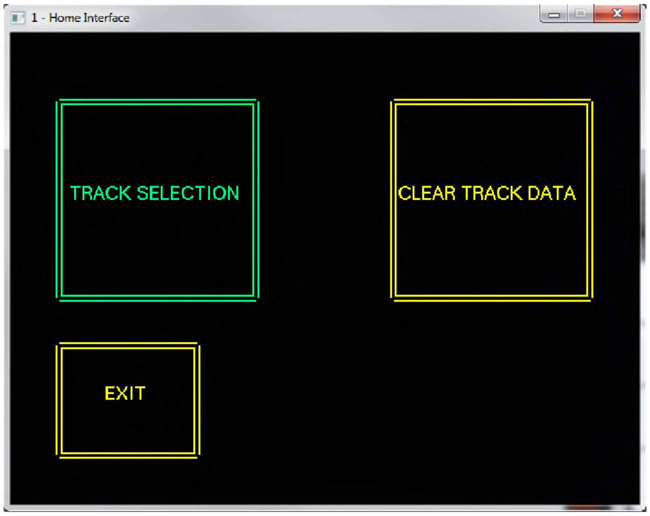


Figure 5: The controls shown on the Home Interface

The home interface contains two command controls and a single navigation control. These controls are used as follows:

- **Track Selection** will navigate the user to the *Track Selection Interface*.
- **Clear Track Data** will clear all stored co-ordinate data from the tracks. This resets the default locations of each track to the center of the room.
- **Exit** will shut the system down

### Track Selection Interface

The *Track Selection Interface* is the interface where the user is able to select which tracks they wish to mute or unmute, as well as which track they wish to record. A layout of the track selection interface is shown in Figure 6.

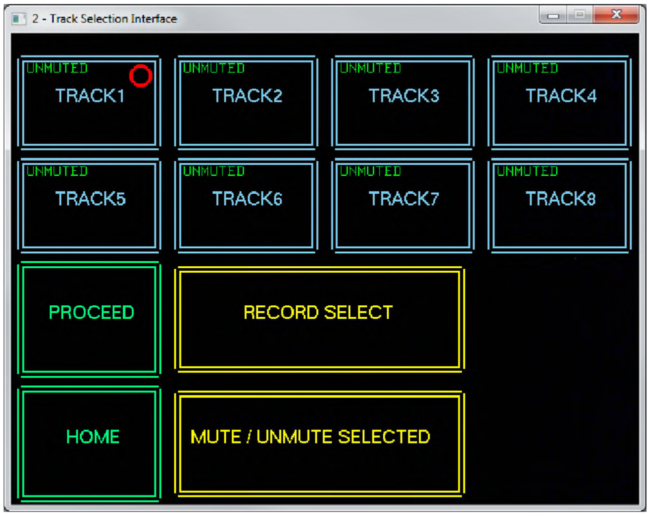


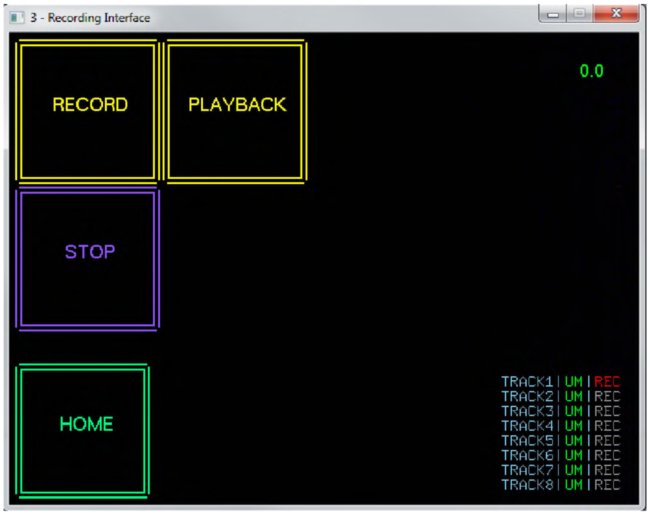
Figure 6: The controls shown on the Track Selection Interface

The *Track Selection Interface* contains eight track buttons which correspond to the tracks in the DAW. If the audio piece the user is recording contains less than eight tracks then the remainder of the tracks can be ignored. The track buttons are *selectable* which allows a user to perform a single action on one track at a time using the provided command controls. This interface contains two command controls as well as two navigation controls, which are as follows.

- **Record Select** allows a user to set the recording state for a selected track. Only one track may be recorded by the user at a time. The track that is currently selected for recording will have a red circle at the top right of its control. Figure 6 shows *TRACK1* as the track that has been selected for recording.
- **Mute / Unmute Selected** allows a user to change the muting state of the currently selected track. The muting state of a track is shown at the top left of the control, this will either be *UNMUTED* shown in green, or *MUTED* shown in red. Muted tracks will not be played when the user is recording or playing the audio piece back. However, if a muted track gets selected for recording, it will have its muting state changed to *unmuted*.
- **Proceed** will navigate the user to the *Record\Playback Interface*.
- **Home** will return the user to the *Home Interface*.

## Record\Playback Interface

The *Recording Interface* allows the user to record the sound location of a track, as well as play the track back. A layout of the recording interface is shown in Figure 7.



TRACK1 | UM | REC
TRACK2 | UM | REC
TRACK3 | UM | REC
TRACK4 | UM | REC
TRACK5 | UM | REC
TRACK6 | UM | REC
TRACK7 | UM | REC
TRACK8 | UM | REC

Figure 7: The controls shown on the Recording Interface

The *Recording Interface* contains 3 command controls for controlling the playback of the audio piece. This interface also contains a listing of the current tracks as well as a time. The track listing shows the user which tracks are muted/unmuted as well as which track is being recorded. The muting state is shown by a green *UM* or a red *MU* for unmuted and muted respectively. The track that has *REC* shown in red next to its name is the track which is in the recording state. The *REC* shown next to the rest of the tracks will appear in grey.
The interface controls do the following:

- **Record** will put the system into a recording state, where the user is able to record the sound locations for a single track. All the tracks that have had their muting state set to *unmuted* will be played back whilst the user is recording.
- **Playback** will put the system into a playback state. This will play any tracks that have had their muting state set to *unmuted*
- **Stop** will put the system back into an idle state. The user may select this option to indicate that they are done recording or playing the audio piece.
- **Home** returns the user back to the home interface.

Any tracks that are unmuted will be played using their recorded co-ordinates when the system is recording or playing. If a track has not been recorded yet

and is not muted, it will be played as if its sound location is in the center of the speaker configuration. When the user selects either the *Record* or *Playback* option, the system will give a 5 second countdown before the system starts recording or playing.
The user uses their right hand to move the sound location of a track around. The position of the users right hand is shown in 3 different places

1. The Kinect Interface - the interface shows a red cross hair over the user's right hand.
2. A Windows Dialog Box - this shows the 3D co-ordinates of the users right hand as X, Y, and Z co-ordinates.
3. Sketchup - Sketchup shows a crosshair of where the system has determined the position of the sound source in the 3D environment.

## Sketchup

Sketchup is a 3D modelling program which is used in this system to display the 3D position of the sound source. This enables the user to have a much better understanding of the location of the sound source in relation to the speakers during recording. Shown in Figure 8 is the layout of the Sketchup interface, showing the crosshair, speakers, Kinect, and user's head.

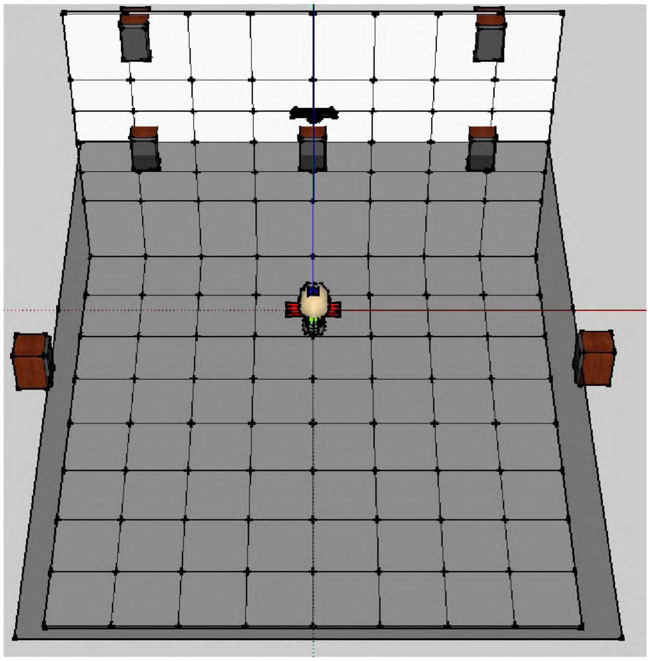
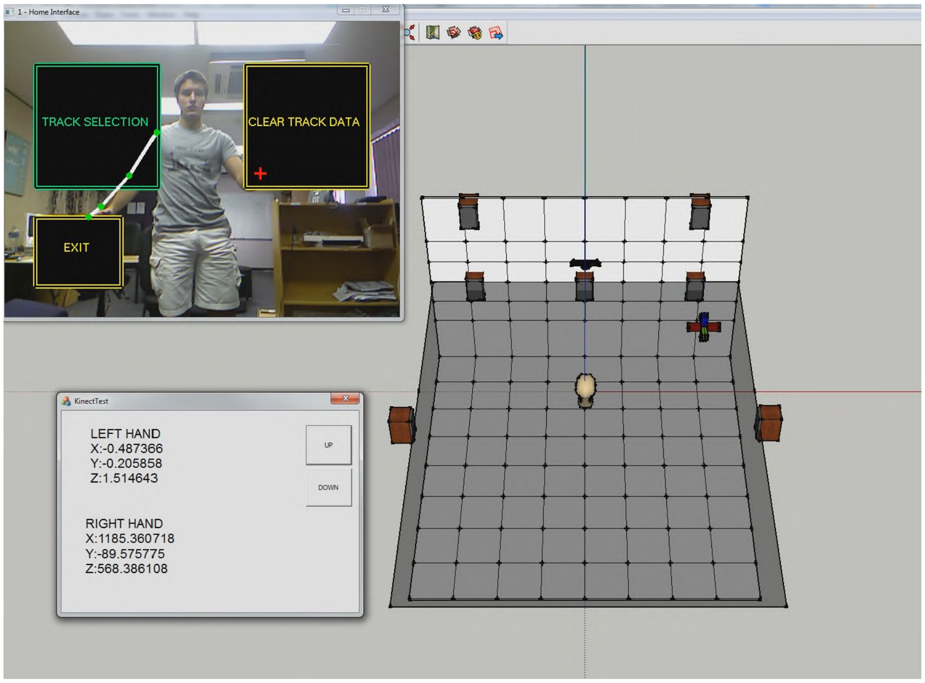


Figure 8: The Sketchup interface showing the crosshair at the origin

The position of the sound source is described using the 3D axes with X showing horizontal displacement from the center, Y showing vertical displacement, and Z as the perpendicular displacement from the eye of the Kinect. A grey translucent plane has been added where Y is equal to the ideal listening height. Figure 9 displays the Kinect interface, the dialog, and Sketchup while the system is in the recording state.



```
LEFT HAND
X:-0.487366
Y:-0.205858
Z:1.514643

RIGHT HAND
X:1185.360718
Y:-89.575775
Z:568.386108
```

Figure 9: The system interfaces - Kinect interface, dialog with 3D co-ordinates, and Sketchup with the crosshair position