The University of Derby

College of Engineering and Technology

Department of Engineering

A Proposed Stereophonic to B-Format Up-Mix Algorithm Using Multilevel Thresholding

by

Haydon Cardew

September 2016

Submitted for in part-fulfilment of the requirements for the MSc in Audio Engineering

Acknowledgements

I would like to thank my supervisors Bruce Wiggins and Adam Hill; their guidance and knowledge has given me a far deeper understanding of the audio world and made it possible for me to conduct the following research.

I would also like to extend my thanks to all members of staff at the University of Derby who have helped me over the past few years I have spent there.

I would like to thank my parents for supporting me through all my years at University and encouraging me to do what I enjoy doing!

Finally, I thank Jade for putting up with me throughout this Masters course. For all the nights I spent working on my laptop and all the days I spent at the library, thank you for understanding!

Abstract

Ambisonics is experiencing a resurgence of interest in the audio industry. Since the most widely used format in audio today is stereophonic, an up-mix algorithm for the conversion of stereophonic to B-format would allow for a far greater amount of Ambisonic material to be available. This thesis proposes an algorithm using Otsu’s thresholding technique to extract sources and azimuth positions from a stereophonic signal and then use the Furse-Malham equations to pan these sources in a B-format signal. The algorithm was tested and proved highly successful at extracting audio sources and their respective azimuth. Informal listening tests also returned positive responses.

List of Contents

List of Figures

List of Equations

List of Tables

1 Introduction

Surround sound and spatial audio has become an area of increasing research and innovation within the audio industry. As Kronlachner (2014a) notes, brands such as *Dolby Atmos* and *Auro 3D* are pushing surround sound with height in cinemas and home systems. Wiggins (2004) describes the experience of surround sound to be the 'you are there' illusion where the listener is 'placed' inside a sound field, this is opposed to the 'they are here' illusion created by stereophonic reproduction.

During the 1970's Michael Gerzon developed Ambisonics, a method for synthesising an entire sound field. Using spherical harmonics, sound sources can be encoded in any direction around a full spherical sound field. Ambisonics can be viewed as advantageous over other surround sound formats due to its flexibility, the method for encoding the captured sound field is independent from the decoding of the audio. This means that Ambisonics is not limited to any predefined loudspeaker arrangement, rather it can be decoded to any arbitrary loudspeaker set up. Due mainly to an increase in the performance of computing power Ambisonics has recently experienced a resurgence of interest in the audio industry, with multiple examples given later.

For many years stereophonics has been the standard format for commercial audio; however, if a listener desires to playback stereophonic audio through a surround sound set up an up-mix algorithm is required. Whilst many up-mix algorithms have been developed (Marston (2011) tests multiple stereophonic to 5.1 algorithms) stereophonic to B-format conversion is still in its infancy (where B-format is a common format for recording an Ambisonic sound field). This thesis will propose a stereophonic to B-format up-mix algorithm. The Algorithm is performed by first conducting blind source separation of the signal, of which three different methods will be discussed. The extracted sources azimuth will then be estimated and the source panned into an Ambisonic sound field. Testing will then be conducted on the algorithm and code for the execution of the algorithm presented.

2 Background

2.1 Demand in Ambisonics

Ambisonics was pioneered by Gerzon (1974a) as a sound reproduction system designed with a psychoacoustic understanding. It can be described as having many advantages over other surround sound formats. Unlike other surround sound systems (Gerzon, 1974b) ambisonics is based on a general metatheory on how humans perceive and localise sound (Gerzon, 1992) and uses this metatheory when recreating a sound field around a listener. Another advantage of Ambisonics is its flexibility, leading to the system being described as “true, future-proof surround sound audio” (Wiggins, 2008).
However, despite its many advantages, in its 40-year history ambisonics hasn’t yet become a commonly used surround sound format in the audio industry. As this project is based on a conversion into an ambisonic format it’s important to consider the immediate relevance of this project. For reasons discussed later, there are numerous examples of recent uptakes in the use of Ambisonics within the audio industry and it is this revived interest in Ambisonics which, once again, makes it an interesting and promising area of research.

2.2 Why Ambisonics Has Had A Resurgence

A strong argument can be made that Ambisonics was ahead of its time. Wiggins (2008) presented the argument that at the time Ambisonics was being conceived by Gerzon, in the 1970’s, the technological capabilities were well below the level required for the software needed. Due to limited processing power, B-format encoder and decoder software wasn’t available and although working/mixing in Ambisonics could still be done through specialist hardware (Daubney, 1982) it was not a straight forward option for mix engineers, nor was it a cheap or easy solution for widespread use.
Modern day technology has caught up with the needed performance for the creation and playback of Ambisonic material. Figure 2.1 shows the extent of growth in processing power since the first works of Ambisonics through to the 2006.

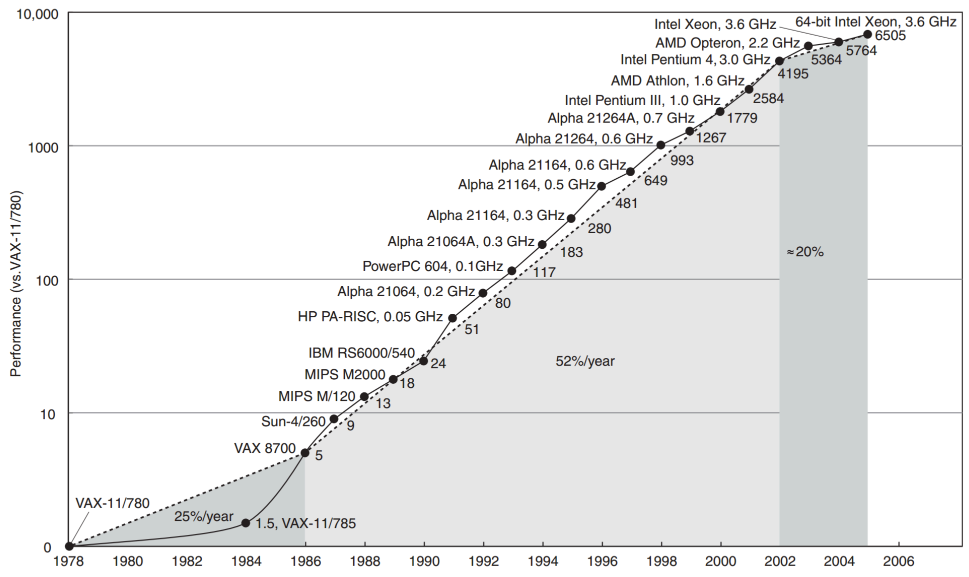


Figure 2.1 - Computer processing power increase 1978-2006

This dramatic increase in the performance of computer processing power has brought about an abundance of cheap or freely available software for working in Ambisonics:
- *Reaper* (2016) is a DAW (digital audio workstation) that shows great flexibility in the number of channels per track (suitable for higher order Ambisonics) and the routing of individual tracks. The software is also extremely affordable compared to other commercial DAWs.
- Wiggins (2010) provides VST plugins for encoders, decoders, panners and reverbs for up to 5th order Ambisonics.
- Kronlachner (2014b) offers an Ambisonic plugin suite for the production and post-production of 3D ambisonic content.
- *Blue Ripple Sound* (2015a) provide "a set of tools that provide all the basics to produce a higher order ambisonic 3D mix" for free.

With processing power no longer being an issue, and software for working within Ambisonics becoming widely available, the benefits of this system are now being utilised by the commercial audio industry.

2.3 Recent Uses of Ambisonics

Presented are a number of cases that demonstrate a shift in the industry towards the use of Ambisonics.

2.3.1 Google

A recently released *Spatial Media* API (application program interface) demonstrates a push by Google to provide a 3D experience for users, both in video and audio (Google, 2016a). The format chosen by Google to deliver spatial audio is Ambisonics, in which they allow a developer to either play back pre-recorded Ambisonic sound fields or create virtual sound objects which can be placed spatially around the listener (Google, 2016b). So far their audio system supports full 3D first order Ambisonics, using SN3D normalisation (different order Ambisonics are discussed in section 4.1).

2.3.2 Virtual and Augmented Reality

It is expected that the technology market for virtual and augmented reality will experience rapid growth within the next decade, with estimates that it's market will be worth $80 billion by 2025 (Goldman, 2016). The audio sector of the virtual and augmented reality market has already seen investment from leading companies in the industry; Google buying Thrive audio (Google Acquisitions, 2015) and Oculus licensing 3D audio technology from VisiSonics (PR Newswire, 2014).
Virtual and augmented reality audio systems use binaural audio to allow a listener to experience externalisation and localisation of sound objects. An issue with binaural audio when used in a virtual reality setting is the interpolation between different head-related transfer functions (HRTF), when a listener's head position moves in a sound field the HRTF applied to an object to place it spatially in the soundscape will need to change accordingly. The interpolation between two HRTFs can cause degrading of the localisation of sound objects. Noisternig (2003) describes a method for using Ambisonics to decode a sound field to a set of virtual speakers where the listener is placed in the centre, binaural signals are then created by convolving the loudspeaker signals with HRTFs related to their virtual position. This method removes the need for interpolation between HRTFs, as head tracking can be

done via simple rotation matrices in the ambisonic domain. It is also described by Noisternig as a computationally efficient method for rendering binaural sound fields.

2.3.3 Live Sound

An ‘Experimental Soundfield’ stage is held at Glastonbury by Funktion-One (Funktion-One, 2016). This stage is a surround sound stage which uses Ambisonics and a set-up of loudspeakers around the audience to allow the listeners to experience a 3D soundscape in a live sound setting.

2.3.4 Research

In July 2011 the BBC launched the Audio Research Partnership, with the goal of bringing together the leading audio researchers from the UK; with examples of research departments being the University of Surrey, University of Salford, Queen Mary University of London, University of Southampton, University of York and BBC R&D (BBC, 2011a). One main project undertaken by the research partnership is the S3A project. This aims to bring a step-change in the quality of audio consumed by the public by unlocking the potential of 3D sound. With this goal in mind, they have shown clear interest in the capabilities of Ambisonics and its ability to provide height information in surround sound (BBC, 2011b).

3 Aim of this Project

This project is based on creating an algorithm to allow for Ambisonic playback of stereophonic signals. With a clear resurgence in Ambisonics it seems very probable that in time there will be a demand for allowing the conversion of widely used formats, with stereophonics being the most widely used, to an Ambisonic format. Currently there is a method allowing for stereophonic files to be converted to Ambisonics using a Universal-HJ (UHJ) decoder (using its ‘super stereo’ feature), however UHJ was originally designed for allowing 2, 3 or 4 channel encoding of spatial sounds, with this conversion an extra feature.

The algorithm developed in this project will need to meet the following objectives:

- Use conventional, commercial stereophonic audio.
- Use an appropriate source separation method to divide the audio into constituent parts.
- Compute a horizontal angle (azimuth) of the extracted parts.
- Encode extracted audio into B-format.

4 Fundamentals

4.1 Ambisonics

This project is based around converting to an Ambisonic format so it is important to understand the background theory involved in Ambisonics.

To understand how ambisonics works it is helpful to first consider a fundamental difference between ambisonics and conventional stereo signals. A stereo signal will be mixed to, and stored as, two channels of audio which directly correspond to left and right speaker feeds. This limits the playback of the signal to the predefined positioning of the left and right speakers (defined as $\pm 30°$ by ITU (1997)); although a listener could move the left and right speakers to any arbitrary angle the originally intended sonic image will be distorted. Ambisonics, however, differs from this approach; as Gerzon (1985) explains, the approach of ambisonics is to capture, store and reproduce a spatial sound field in all its directions. Another way to view this contrast is to consider a stereo signal as reproducing a sonic image in front of a listener using a predefined set up whereas ambisonics aims to *place* the listener inside a sound field, without any prior knowledge of the playback system. Unlike stereo, the encoding (recording) side of ambisonics is independent of the decoding (reproduction) of the audio.

4.1.1 Blumlein's Coincident Microphone Technique

Wiggins (2004) describes ambisonics as an extension of Blumlein's reproduction system. Blumlein (1931) proposed the technique of being able to steer a resultant figure of eight microphone from two perpendicular figure of eight microphones. Using Eq. 4-1 and the two microphone response patterns shown in Figure 4.1 a figure of eight microphone pointing in any direction can be found by varying θ, shown in Figure 4.2

$$sum = \frac{(X+Y)}{\sqrt{2}}$$

$$dif = \frac{(X-Y)}{\sqrt{2}}$$

$$steered\ mic\ pattern = (\cos(\theta)\cdot sum) + (\sin(\theta)\cdot dif)$$

Eq. 4-1

θ = the response angle of the resultant microphone

X= microphone in the X axis

Y= microphone in the Y axis

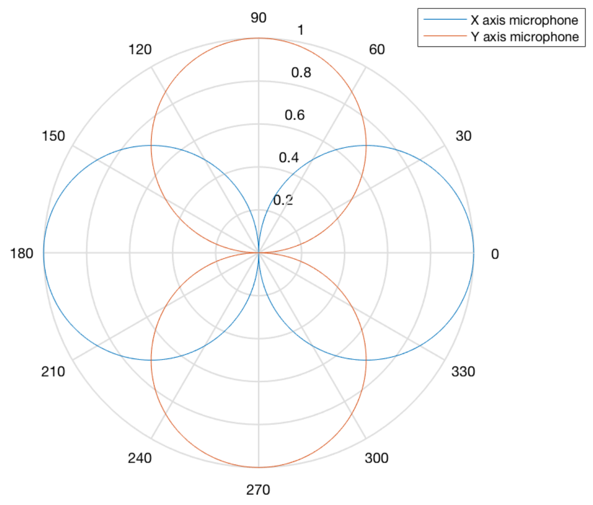


Figure 4.1 - Perpendicular figure of eight patterns

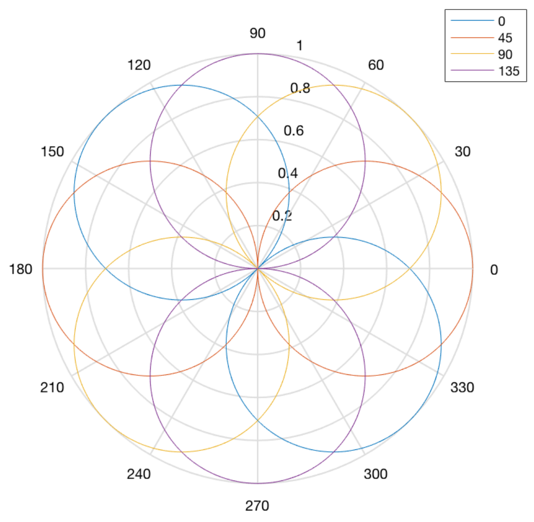


Figure 4.2 – Example resultant polar patterns

By allowing each channel to represent the direction of a sound field, Blumlein's stereo was similar to ambisonics as the encoding of the stereo was independent of the decoding. However, ambisonics furthered this idea by allowing spatial information from all three dimensions (i.e. including height) and allowing the decoded response pattern of the microphones to vary between omnidirectional, any cardioid response and a figure of eight (examples given in section 4.1.4).

4.1.2 Spherical Harmonic Decomposition

Kronlachner (2014a) describes ambisonics as being based on the spherical harmonic decomposition of a sound field, where a surround sound signal can be considered as time-varying data distributed on the surface of a sphere. Williams (1999) explains that any arbitrary function on the surface of a sphere can be expanded in terms of spherical harmonics.

Kronlachner (2014a) considers an audio signal, $f(t)$, arriving from direction (θ, α) where θ represents azimuth and α represents elevation. The surround sound signal, $f(\theta, \alpha, t)$, can be represented using a spherical harmonic expansion of order N.

$$f(\theta,\alpha,t) = \sum_{n=0}^{N}\sum_{m=-n}^{n} Y_n^m(\theta,\alpha)\emptyset_{nm}(t)$$

Eq. 4-2

where Y_n^m is the spherical harmonic of order n, degree m and $\emptyset_{nm}(t)$ is the expansion coefficient.

Spherical harmonics are composed of a normalisation term $N_n^{|m|}$, the associated Legendre polynomial $P_n^{|m|}$ and a trigonometric function.

$$Y_n^m(\theta,\alpha) = N_n^{|m|}P_n^{|m|}(\sin(\,\theta))\begin{cases}\sin(|m|\alpha, \text{for } m < 0.\\ \cos(|m|\alpha, \text{for } m \geq 0.\end{cases}$$

Eq. 4-3

The Legendre polynomial can be found via Eq. 4-4.

$$(n-m)P_n^m(x) = x(2n-1)P_n^m(x) - nP_{n-2}^m(x)$$

$$P_m^m = (-1)^m(2m-1)!!\,(1-x^2)^{m/2}$$

$$P_{m+1}^m = x(2m+1)P_m^m$$

Eq. 4-4

Figure 4.3 shows a graphical representation of the Associated Legendre function up to $n = 2$.

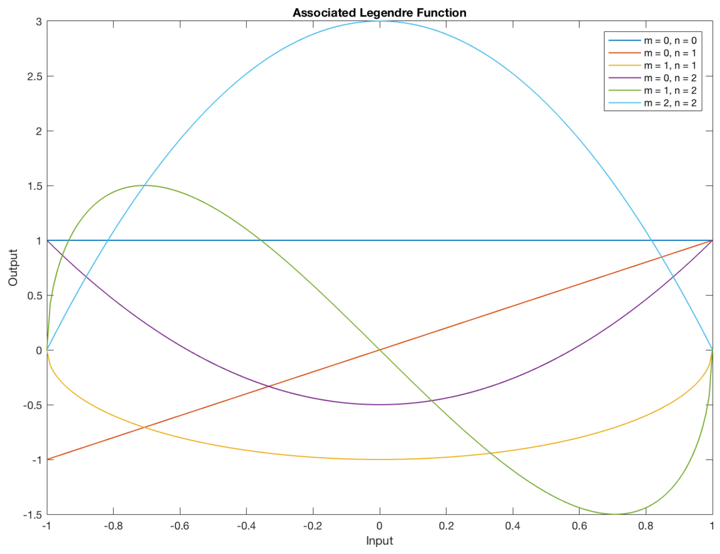


Figure 4.3 - Associated Legendre function

Two main functions can be used to find the normalisation term, $N_n^{|m|}$. Fully normalised (N3D) spherical harmonics are found through Eq. 4-5.

$$N_n^{|m|} = \sqrt{\frac{(2n+1)(2-\delta_m)}{4\pi}\frac{(n-|m|)!}{(n+|m|)!}}$$

Eq. 4-5

Where

$$\delta_m = \begin{cases} 1 \; if \; m = 0 \\ 0 \; if \; m \neq 0 \end{cases}$$

Whilst the Schmidt semi-normalised function is shown below.

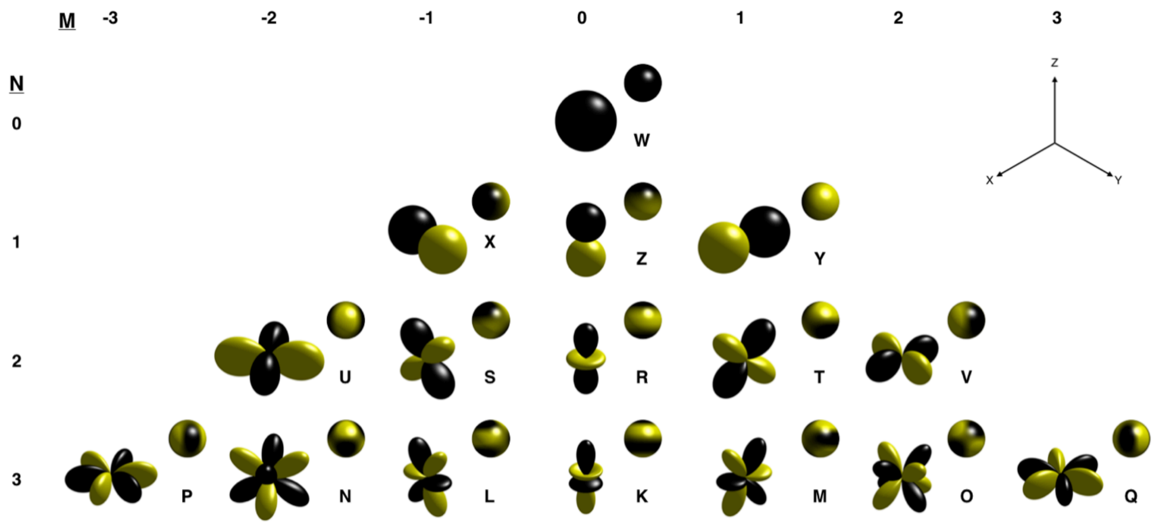
$$N_n^{|m|} = \sqrt{(2-\delta_m)\frac{(n-|m|)!}{(n+|m|)!}}$$

Eq. 4-6

Where

$$\delta_m = \begin{cases} 1 \; if \; m = 0 \\ 0 \; if \; m \neq 0 \end{cases}$$

As the order number of the spherical harmonic increases a higher spatial resolution is possible. Figure 4.4 shows balloon plots alongside a plot on the surface of a sphere for spherical harmonics up to 3rd order. The harmonics are labelled with their conventional channel lettering along with a chart showing the conversion to ambisonic channel number, where $ACN = n^2 + n + m$.



ACN	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Channel Label	W	X	Z	Y	U	S	R	T	V	P	N	L	K	M	O	Q

Figure 4.4 - Balloon and sphere plots for 3rd order spherical harmonics

The spherical harmonics can be shown mathematically using the Furse-Malham set, Table 4-1.

Label	**Order**	**Angle/Elevation Representation**	**Cartesian Representation**
W	0	sqrt(1/2)	sqrt(1/2)
X	1	cos(A)cos(E)	x
Y	1	sin(A)cos(E)	y
Z	1	sin(E)	z
R	2	(1/2)((3sin(E)sin(E)-1)	(1/2)(3zz-1
S	2	cos(A)sin(2E)	2zx
T	2	sin(A)sin(2E)	2yz
U	2	cos(2A)cos(E)cos(E)	xx-yy
V	2	sin(2A)cos(E)cos(E)	2xy
K	3	(1/2)sin(E)(5sin(E)sin(E)-3)	(1/2)z(5zz-3)
L	3	sqrt(135/256)cos(A)cos(E)(5sin(E)sin(E)-1)	sqrt(135/256)x(5zz-1)
M	3	sqrt(135/256)sin(A)cos(E)(5sin(E)sin(E)-1)	sqrt(135/256)y(5zz-1)
N	3	sqrt(27/4)cos(2A)sin(E)cos(E)cos(E)	sqrt(27/4)z(xx-yy)
O	3	sqrt(27/4)sin(2A)sin(E)cos(E)cos(E)	sqrt(27)xyz
P	3	cos(3A)cos(E)cos(E)cos(E)	x(xx-3yy)
Q	3	sin(3A)cos(E)cos(E)cos(E)	y(3xx-yy)

Table 4-1 - Furse-Malham Set (Blue Ripple Sound, 2015b)

4.1.3 B-Format

Figure 4.5 shows only the first four spherical harmonics.

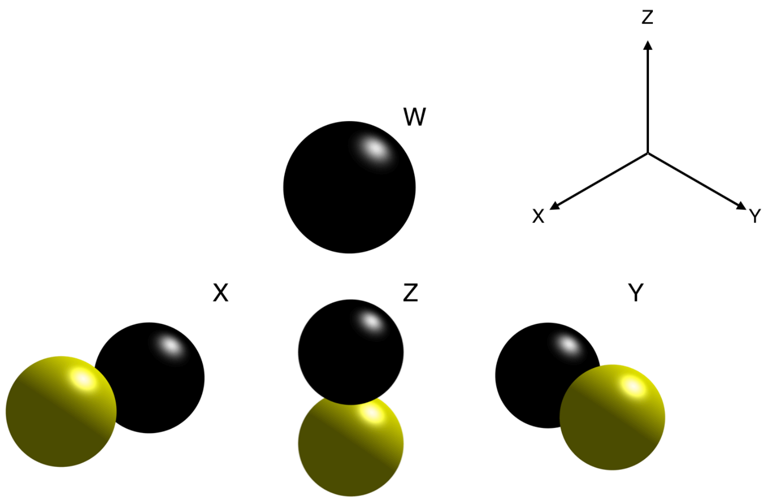


Figure 4.5 - Balloon plots for 1st order spherical harmonics

These four harmonics represent 1st order Ambisonics and when used in equations Eq. 4-7 to store audio signals are they are collectively known as B-format (note that B-format can be extended to 9 channels for 2nd order, 16 channels for 3rd order, etc).

When considering 1st order ambisonics it is clear to see that channels X, Y and Z form figure of eight patterns that are perpendicular to each other. Expanding the previously explained principles of Blumlein stereo, these three channels can be cross-faded and used to steer a figure of eight pattern pointing in any azimuth or elevation. The W channel is the 0th order channel and represents an omnidirectional polar pattern.

An example can be given to show how an audio source is encoded into 1st order B-format. Consider a mono source with an azimuth of 93° and an elevation of 27°. The W, X, Y and Z gains will be as follows (taken from the Furse-Malham set).

$$W_{gain} = \sqrt{1/2} \approx 0.7071$$
$$X_{gain} = \cos(93°) \cdot \cos(27°) \approx -0.0466$$
$$Y_{gain} = \sin(93°) \cdot \cos(27°) \approx 0.8898$$
$$Z_{gain} = \sin(27°) \approx 0.4540$$

Eq. 4-7

Thus the W, X, Y and Z channels will be encoded as follows.

$$W = 0.7071 \times mono$$
$$X = -0.0466 \times mono$$
$$Y = 0.8898 \times mono$$
$$Z = 0.4540 \times mono$$

4.1.4 Decoding

To play a B-format signal through a multichannel system the signal needs to be decoded. As has already been stated, a benefit of ambisonics is that the B-format signal can be played back through a sound system consisting of any number of channels/loudspeakers. To calculate the signal sent to a chosen loudspeaker a virtual microphone needs to be pointed at the position of the loudspeaker. The same equations found in the Furse-Malham set can be used to steer the virtual microphone in the direction of the given loudspeaker. However, the polar pattern of the virtual microphone can also be adjusted, this is done using the equation Eq. 4-8 (Farina et al, 2001).

$$W_{gain} = \sqrt{1/2}$$
$$X_{gain} = \cos(\theta) \cdot \cos(\alpha)$$
$$Y_{gain} = \sin(\theta) \cdot \cos(\alpha)$$
$$Z_{gain} = \sin(\alpha)$$

$$S = 0.5 \times \left[(2-d)W_{gain} + d\left(X_{gain} + Y_{gain} + Z_{gain}\right)\right]$$

Eq. 4-8

S = speaker output

θ = speaker azimuth

α = speaker elevation

d = directivity factor (0 to 2)

An example of how the polar pattern of a virtual microphone created in the X-Y plane changes with different values of d is shown in Figure 4.6.

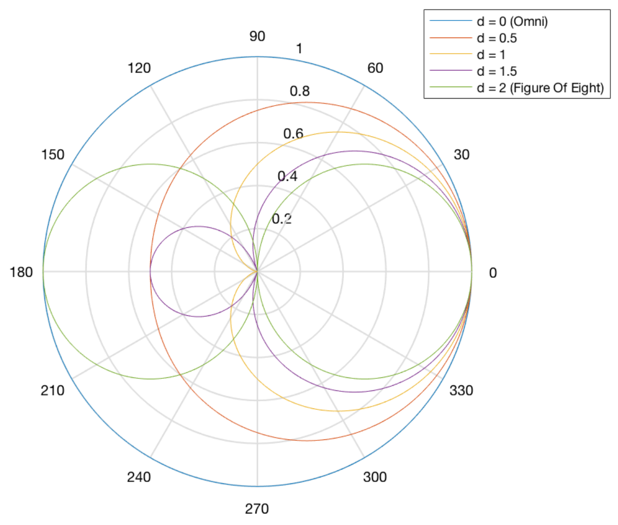


Figure 4.6 - Polar pattern response for different 'd' values

As can be seen, changing the value of d allows for the polar pattern to change from omnidirectional to a figure of eight.

4.1.4.1 Optimising Decoders Using Psychoacoustics

Gerzon (1985) showed that energy and velocity vectors could be used to estimate the perceived quality and localization of a sound source when reproduced through multiple loudspeakers. The velocity vector represents low frequency performance whilst the energy vector represents high frequency performance, they can be calculated by using equations Eq. 4-9.

$$P = \sum_{i=1}^{n} g_i$$

$$V_x = \sum_{i=1}^{n} g_i \cos(\theta_i)\,/P$$

$$V_y = \sum_{i=1}^{n} g_i \sin(\theta_i)\,/P$$

$$E = \sum_{i=1}^{n} g_i^2$$

$$E_x = \sum_{i=1}^{n} g_i^2 \cos(\theta_i)\,/E$$

$$E_y = \sum_{i=1}^{n} g_i^2 \sin(\theta_i)\,/E$$

Eq. 4-9

Where g_i is the gain of the ith loudspeaker, n is the number of speaker and θ_i is the azimuth of the ith loudspeaker.

Wiggins (2004) explains that for regular arrays, if the response patterns of all virtual microphones used to feed the array are the same then both the velocity and energy values would stay constant across all angles being reproduced and the reproduced angle of a source would be the same as it's encoded angle. This meant the only optimization needed for regular arrays was to make the energy and velocity vectors as close to 1 as possible, this is achieved by varying the value d (therefore changing the response pattern of the virtual microphones) for high or low frequencies (with 700Hz being a reasonable threshold).

Wiggins (2004) discusses that a problem arises when an irregular array is used as it needs to be optimised not only for vector lengths but also for decoded source angles and perceived volume. This meant that it wasn't only the polar pattern of the virtual microphones that needed changing, thus the previous technique would not work. Gerzon and Barton (1992) describe a technique to solve this problem by using two different decoder designs, one for the high frequencies and one for the low frequencies, and using a cross-over filter to blend the two.

Figure 4.7 and Figure 4.8 shows velocity and energy vectors for a regular array and an un-optimised 5 speaker array (setup as a 5.1 array).

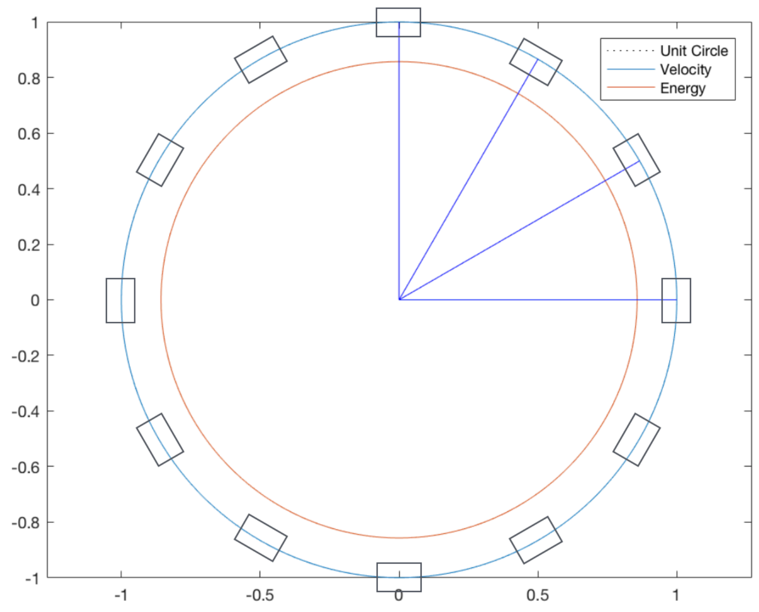


Figure 4.7 - Velocity and energy vectors for a regular array

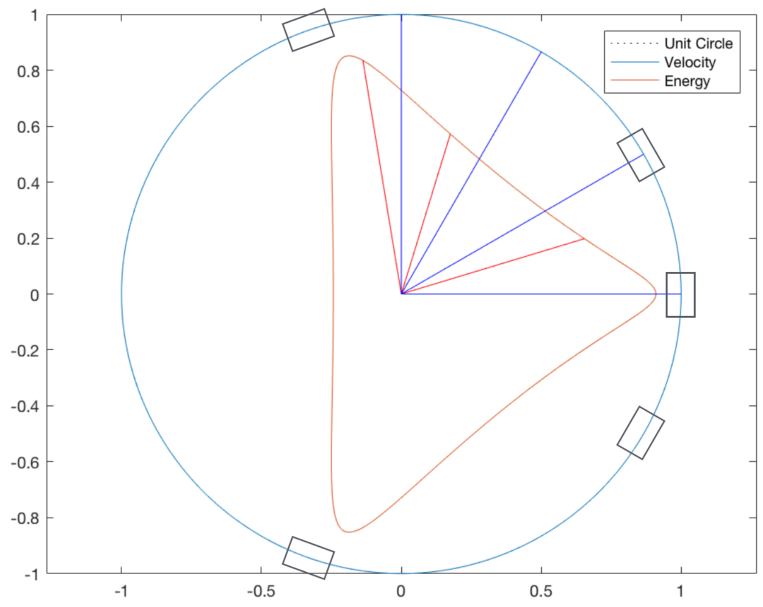


Figure 4.8 - Un-optimised velocity and energy vectors for a 5.1 set up

4.2 Stereo Signal

Kraft and Zolzer (2015) present a model for a stereo signal as follows.

$$X_L(n) = \left[\sum_{j=1}^{J} \alpha_{L_j} \cdot d_j(n)\right] + n_L(n)$$

$$X_R(n) = \left[\sum_{j=1}^{J} \alpha_{R_j} \cdot d_j(n)\right] + n_R(n)$$

Eq. 4-10

This model assumes J amplitude panned sources $d_j(n)$, where $j = 1, \ldots, J$. The direct sources are multiplied, respectively, by the panning coefficients $\alpha_{L_j}$and α_{R_j} and summed to form the left and right channels of the stereo signal. Uncorrelated and unweighted ambient signals introduced into either channel, from various means, are represented by $n_L(n)$ and $n_R(n)$.
Eq. 4-10 can be transformed to the time-frequency domain and presented in Eq. 4-11.

$$X_L(b,k) = \left[\sum_{j=1}^{J} a_{L_j} \cdot D_j(b,k)\right] + N_L(b,k)$$

$$X_R(b,k) = \left[\sum_{j=1}^{J} a_{R_j} \cdot D_j(b,k)\right] + N_R(b,k)$$

Eq. 4-11

Pulkki (2006) discuss that at any one instance in a single time and frequency band there is one dominant source signal heard by the listener (discussed further in section 5.1.1). Using this consideration and assuming that the ambient noise in each channel is of a similar level but different phase, Eq. 4-12 can be presented.

$$X_L(b,k) = a_L(b,k) \cdot D(b,k) + N(b,k)$$

$$X_R(b,k) = a_R(b,k) \cdot D(b,k) + e^{j\phi} \cdot N(b,k)$$

Eq. 4-12

In most music mixes the amplitude of the direct signal will be far greater than the amplitude of the ambient noise. Considering the magnitudes of each side and using the previously mentioned assumption, Eq. 4-13 can be stated.

$$|X_L(b,k)| \approx a_L \cdot |D(b,k)|$$

$$|X_R(b,k)| \approx a_R \cdot |D(b,k)|$$

Eq. 4-13

4.2.1 Panning Laws

As Becker and Bernard (2016) discuss the use of amplitude panning law can be traced back to Blumlein's early work (1931). A monophonic signal can be effectively 'placed' in between two speakers by applying different gains to each speaker feed (as shown in Figure 4.9). Given appropriate gain levels the monophonic source can be positioned between the two speakers and heard as a phantom image. The gain levels for each speaker feed are determined through various Panning Laws. Griesinger (2002) describes two of the most common Panning Laws as the Sine Law and the Tangent Law.

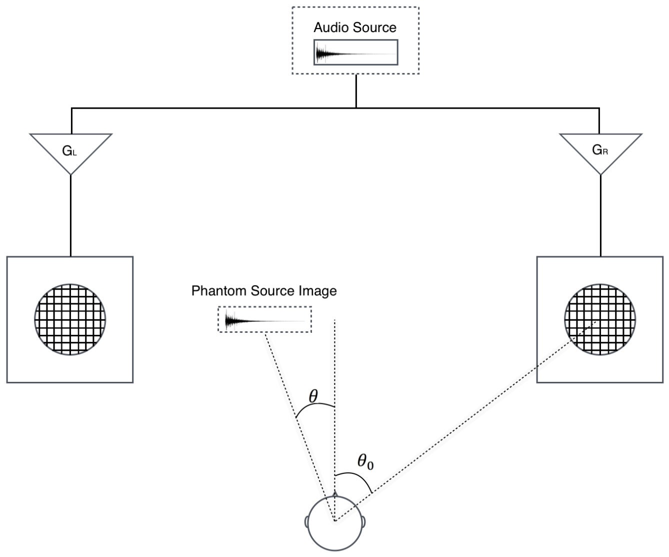


Figure 4.9 - Phantom image of a sound source

4.2.1.1 The Tangent Law

The Tangent Panning Law can be described through equation Eq. 4-14.

$$\frac{\tan(\theta)}{\tan(\theta_0)} = \frac{G_L - G_R}{G_L + G_R}$$

Eq. 4-14

θ = phantom source position

θ_0 = speaker angle

G_L = left channel gain

G_R = right channel gain

As the pan pot position for a source goes from -1 to 1 the gains for either channel can be calculated as follows (shown in degrees).

$$P_{mapped} = (P + 1) \times 45$$

Eq. 4-15

Equation Eq. 4-15 maps the pan pot range of -1 to 1 to a range of 0 to 90. This new range can then be used to calculate the individual gains of the left and right channel. Eq. 4-16 can also be rearranged to show where the phantom source image will be perceived.

$$G_L = \cos(P_{mapped})$$

$$G_R = \sin(P_{mapped})$$

$$\theta = \tan^{-1}(\tan(\theta_0) \cdot \frac{G_L - G_R}{G_L + G_R})$$

Eq. 4-16

Graphing the change of gain for each channel over the pan pot range of -1 to 1 leads to Figure 4.10.

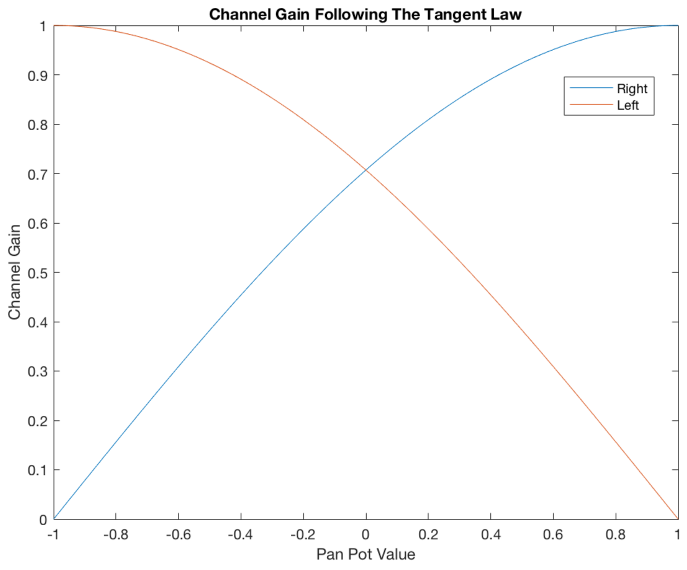


Figure 4.10 - Tangent panning law

Panning obeying the Tangent Law is also known as Constant Power panning, this is due to the power output of the audio source being constant across the whole pan pot range, as Eq. 4-17 shows.

$$G_L^2 + G_R^2 = 1$$

Eq. 4-17

4.2.1.2 The Sine Law

The Sine Panning Law can be described through Eq. 4-18.

$$\frac{\sin(\theta)}{\sin(\theta_0)} = \frac{G_L - G_R}{G_L + G_R}$$

Eq. 4-18

θ = phantom source position

θ_0 = speaker angle

G_L = left channel gain

G_R = right channel gain

As opposed to the Tangent Law the Sine Law only maps the pan pot position to a value of 0 to 1, shown in Eq. 4-19.

$$P_{mapped} = \frac{(P+1)}{2}$$

Eq. 4-19

The left and right channel gains can then be calculated as follows and the perceived phantom image source position is found through Eq. 4-20.

$$G_R = P_{mapped}$$

$$G_L = 1 - P_{mapped}$$

$$\theta = \sin^{-1}(\sin(\theta_0) \cdot \frac{G_L - G_R}{G_L + G_R})$$

Eq. 4-20

Graphing the change of gain for each channel over the pan pot range of -1 to 1 leads to Figure 4.11.

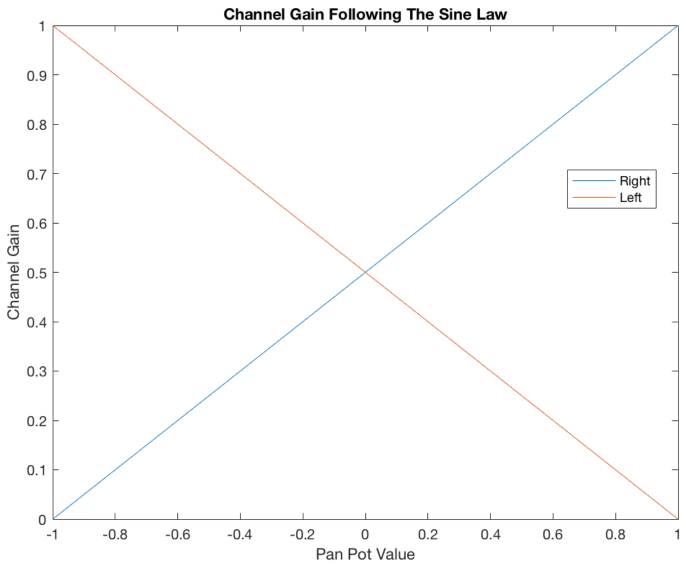


Figure 4.11 - Sine panning law

This style of panning is also known as Constant Voltage panning as the output voltage remains constant across the pan pot range, as shown by Eq. 4-21.

$$G_R + G_L = 1$$

Eq. 4-21

4.2.2 Vector Based Amplitude Panning

Pulkki (1997) developed a vector based amplitude panning (VBAP) law for loudspeaker in a two or three dimensional set up. Considering VBAP used for a stereophonic loudspeaker set up, Figure 4.12 can be found.

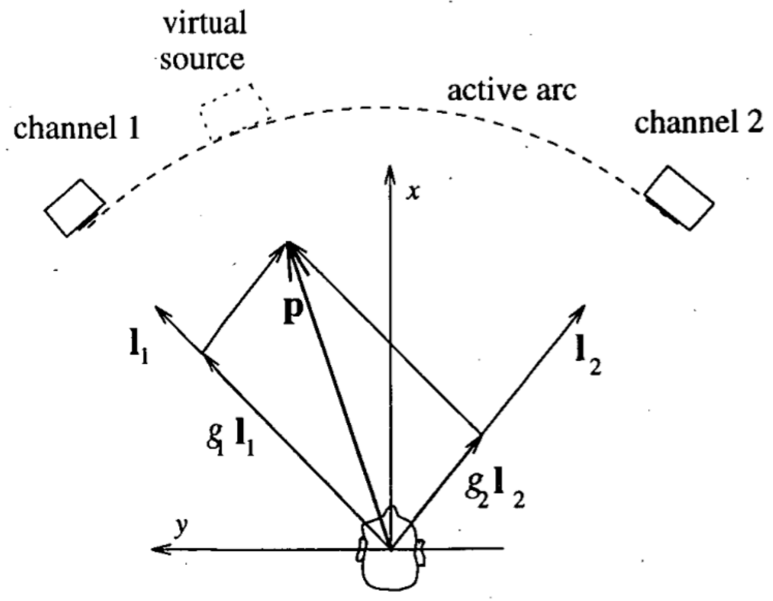


Figure 4.12 - Vector based amplitude panning law in two dimensions (Pulkki, 1997)

The vectors L_1 and L_2 point towards the left and right loudspeakers respectively. The vector p points towards the desired virtual source position and is shown as a linear sum of L_1 and L_2 after gains of g_1 and g_2 have been applied to them, Eq. 4-22.

$$p = g_1 L_1 + g_2 L_2$$

Eq. 4-22

However, in this situation g_1 and g_2 are unknowns whilst p is a vector defined by the user. Figure 4.13 shows how the vector p defined by azimuth θ and distance r can be understood in an $x - y$ plane.

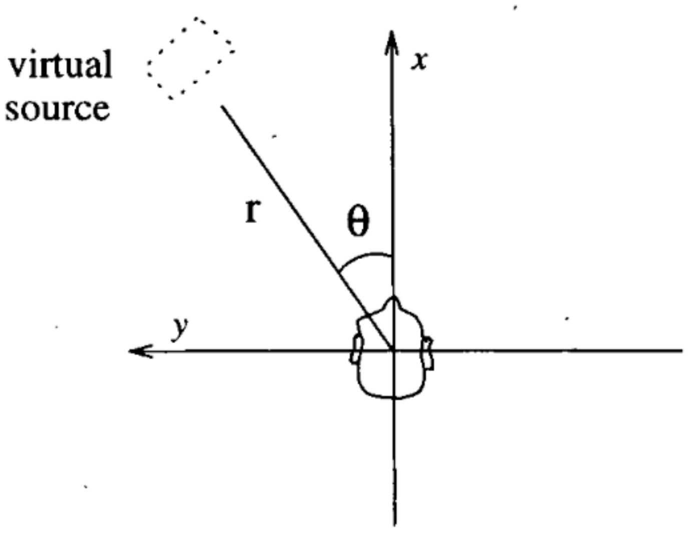


Figure 4.13 - Virtual source in x-y plane (Pulkki, 1997)

Eq. 4-22 can be rewritten by considering all vectors individual x and y coordinates.

$$p = g_1[L_{1x} \quad L_{1y}] + g_2[L_{2x} \quad L_{2y}]$$

Eq. 4-23

By merging L_1 and L_2 into a matrix, L_{12}, and g_1 and g_2 into a vector, $g = [g_1 \ \ g_2]$, Eq. 4-24 can be found.

$$p = g \begin{bmatrix} L_{1x} & L_{1y} \\ L_{2x} & L_{2y} \end{bmatrix}$$

Eq. 4-24

Where $p = [p_1 \quad p_2]$. As p, L_1 and L_2 are known g can be found by finding the inverse of the matrix L_{12}, denoted as L_{12}^{-1}. The matrix L_{12}^{-1} will therefore satisfy the equation $L_{12}L_{12}^{-1} = \boldsymbol{I}$, where $\boldsymbol{I}$ is the identity matrix. Eq. 4-25 shows how g_1 and g_2 can now be found.

$$[g_1 \ \ g_2] = [p_1 \quad p_2] \, L_{12}^{-1}$$

Eq. 4-25

Once the gain factors have been found, if the loudspeakers are not orthogonal to each other (i.e. the angle between them is not 90°) the gain factors will need to be normalised, Eq. 4-26.

$$g_{scaled} = \frac{\sqrt{C}g}{\sqrt{g_1^2 + g_2^2}}$$

Eq. 4-26

Where C is the volume control.

Expanding VBAP into three dimensions leads to Figure 4.14

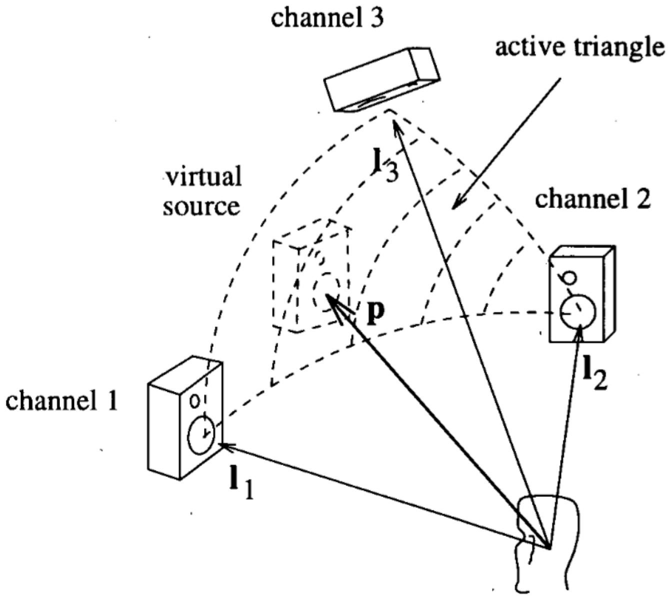


Figure 4.14 - Vector based amplitude panning law in three dimensions (Pulkki, 1997)

The same matrix calculations described previously can be used to determine vector p, however p, g, L_1 , L_2 and L_3 will now all be a three dimensional vectors thus the inverse of a 3x3 matrix will needed to be calculated.

The vector p will always point towards a position that lies on the surface of a segment of a sphere defined by the positions of the loudspeakers, this segment can be called the 'active triangle'.

4.3 Short-Time Fourier Transform

Before reviewing various BSS (blind source separation) techniques, it is first important to understand the Short-Time Fourier Transform (STFT). The STFT splits an input signal into multiple time segments and returns a Discrete Fourier Transform (DFT) for each segment.

4.3.1 Discrete Fourier Transform

The DFT allows for the conversion of a signal from the time domain to the frequency domain and, as (Fuerst and Lynn, 1998) state, it would be hard to exaggerate the importance of the DFT in digital signal processing (DSP); it allows for rapid frequency domain analysis and processing of digital signals.
Eq. 4-27 shows how the complex-valued DFT coefficients for each frequency bin of the DFT are found.

$$X(k) = \sum_{n=0}^{N-1} x_n \cdot e^{\frac{-2\pi jkn}{N}}$$

Eq. 4-27

N = number of time samples

n = current sample

x_n = value of current time sample

k = current frequency bin

$X(k)$ = amount of current frequency in the signal (returned as a complex value)

Eq. 4-27 can be expressed in terms of sine and cosine, Eq. 4-28.

$$X(k) = \sum_{n}^{N-1} x_n \left\{\cos\left(\frac{2\pi kn}{N}\right) - j\sin\left(\frac{2\pi kn}{N}\right)\right\}$$

Eq. 4-28

Assuming that the input of the DFT, x_n, is real then Eq. 4-28 can be split into real and imaginary part.

$$\Re\big(X(k)\big) = \sum_{n}^{N-1} x_n \cdot \cos\left(\frac{2\pi kn}{N}\right)$$

$$\Im\big(X(k)\big) = -\sum_{n}^{N-1} x_n \cdot \sin\left(\frac{2\pi kn}{N}\right)$$

Eq. 4-29

From Eq. 4-29 it can be seen that the DFT can be understood as a simple correlation between sine and cosine functions of frequencies dictated by the given frequency bin and the input, x_n.

The returned real and imaginary parts can then be used to determine the magnitude and phase of the given frequency bin using Eq. 4-30.

$$|X(k)| = \sqrt{(\Re(X(k)))^2 + (\Im(X(k)))^2}$$

$$\phi_k = \tan^{-1}\left(\frac{\Im(X(k))}{\Re(X(k))}\right)$$

Eq. 4-30

$|X(k)|$ = magnitude of complex number

ϕ_k = phase of complex number

$\Im$ = imaginary part of complex number

$\Re$ = real part of complex number

An example of how the DFT can be used for frequency domain analysis is shown as follows. Figure 4.15 shows a 1kHz cosine wave, sampled at 44100Hz.

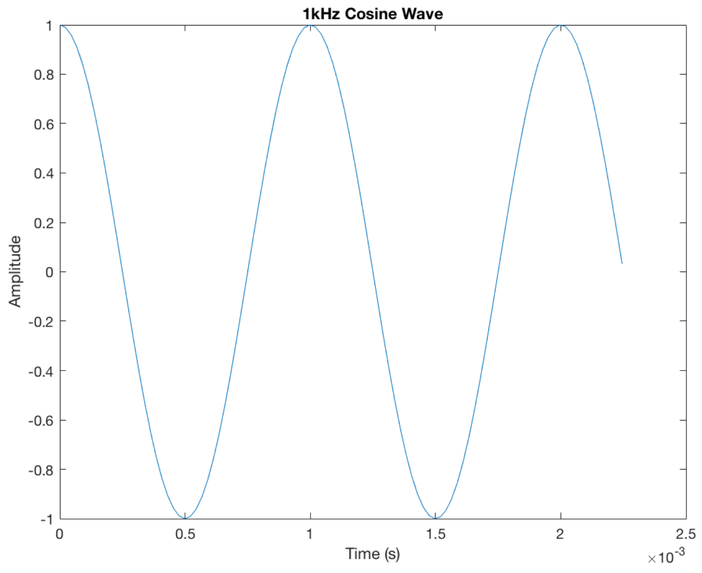


Figure 4.15 - 1kHz Cosine wave

Using a DFT with the same number of frequency bins as the sample rate (i.e. $k \in \mathbb{Z} | \ 1 \ \leq k \leq \ 44100$) a set of 44100 frequency bin coefficients can be found. Eq. 4-30 can then be used to produce Figure 4.16 and Figure 4.17.

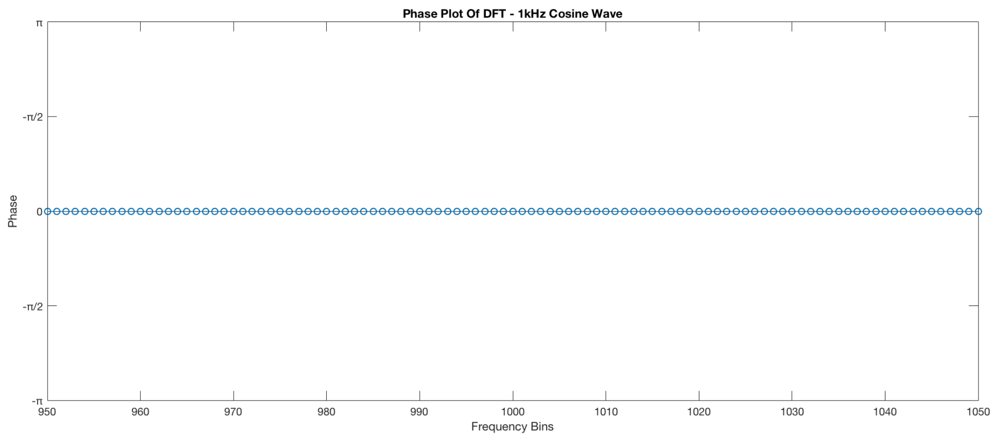


Figure 4.16 - Phase plot of 1kHz cosine wave

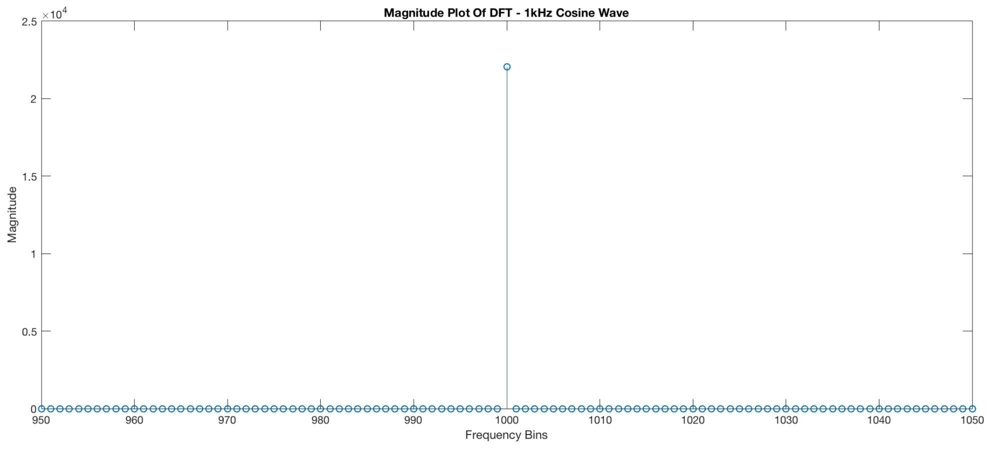


Figure 4.17 - Magnitude plot of 1kHz cosine wave

The frequency bins shown are in the range of 950 to 1050, this is because in this instance (due to the number of frequency bins equalling the sampling rate of the signal) the bin number matches the frequency represented and as the signal is a 1000Hz wave it is clearly seen that the DFT coefficients are for a signal with a frequency of 1000Hz and a no phase shift (compared to a cosine wave).

The effect of a phase shift on the wave can be shown as follows. Figure 4.18 shows a 1000Hz sine wave, sampled at 44100Hz.

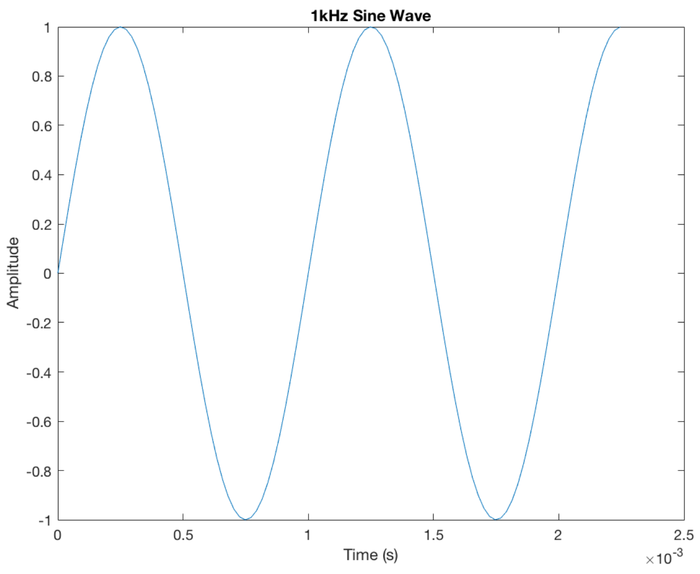


Figure 4.18 - 1kHz Sine wave

The same DFT performed and the outcome is shown in Figure 4.19 and Figure 4.20.

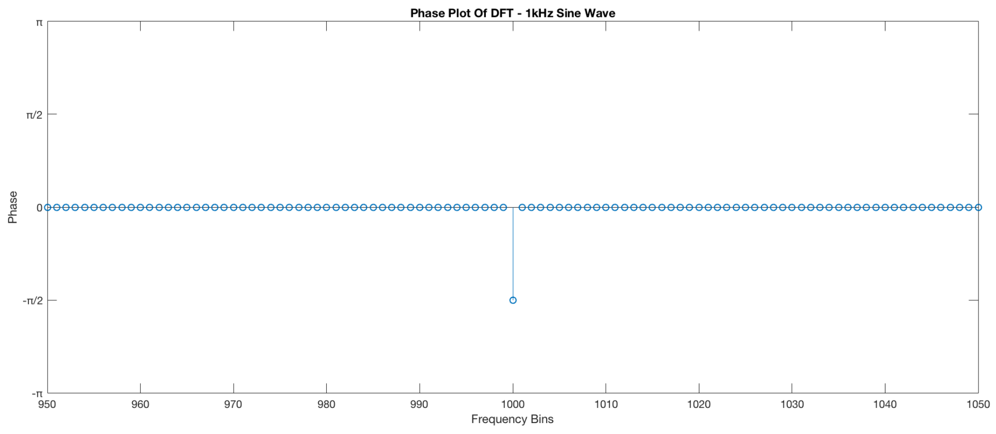


Figure 4.19 - Phase plot of 1kHz sine wave

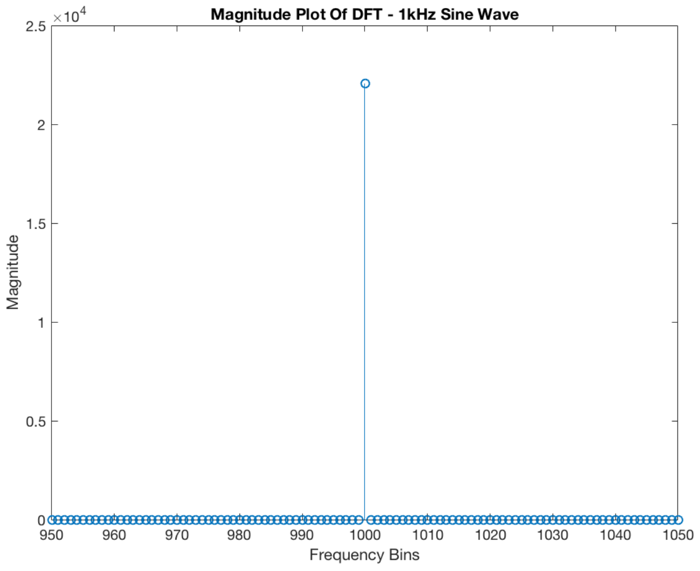


Figure 4.20 - Magnitude plot of 1kHz sine wave

The magnitude of the wave is shown to be the same, as they are both similar 1000Hz waves. However, a phase shift of $-\pi/2$ can now be seen in Figure 4.20.

It should be noted that in practice various Fast Fourier Transform (FFT) algorithms are used over the above described equations to calculate the DFT, due to increased efficiency.

4.3.2 Windowing

As Bateman and Paterson-Stephens (2001) note, an ideal case for a DFT to be carried out is where the frequency of a given wave is an exact integer multiple of the bin spacing (f_s/N). The examples shown below show a magnitude plot of a DFT taken of a 200Hz sine wave and a 215.3Hz sine wave respectively, with 1000 frequency bins used and a sampling rate of 100Hz.

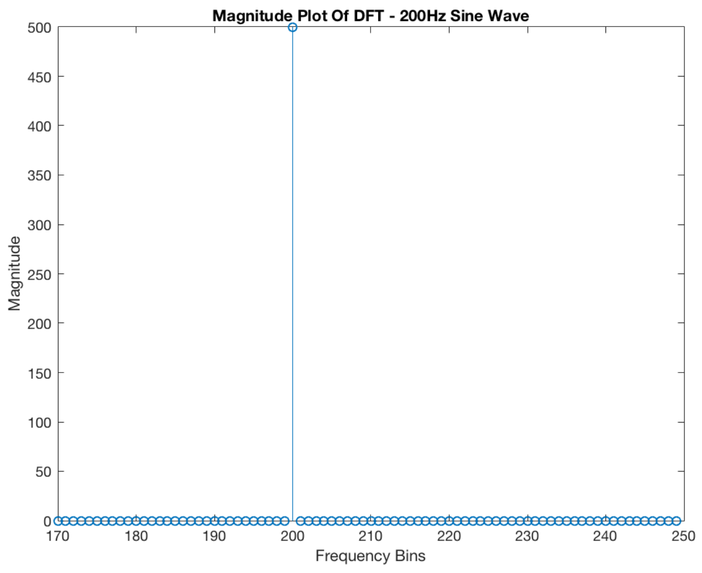


Figure 4.21 - Magnitude plot of 200Hz sine wave

Such a clear picture is not seen when the frequency of the wave is changed to a non-integer multiple.

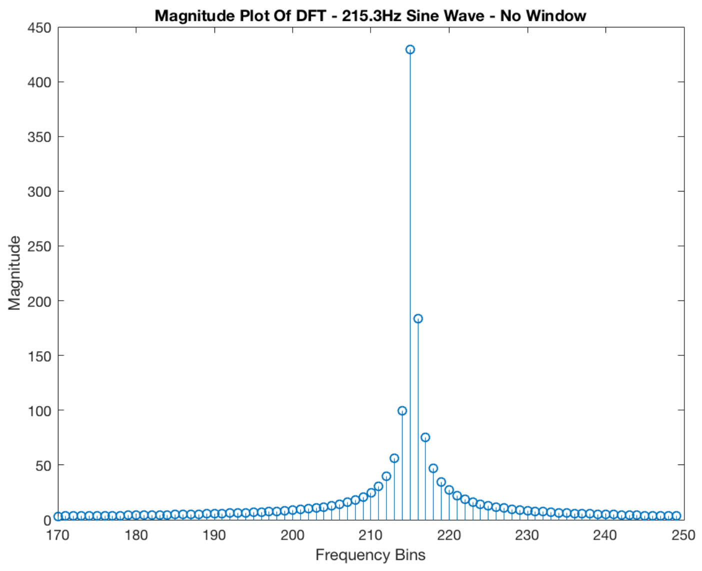


Figure 4.22 - Magnitude plot of 215.3Hz sine wave

As there is no bin which exactly represents 215.3Hz, a wave of that frequency spills into multiple bins surrounding that frequency. Another way to consider this is to understand that the DFT treats the input signal as if it were a periodic signal. As National Instruments (2015) explains, for the DFT both the time and frequency domains are circular topologies and as such are interpreted as though the two end points of the time domain are connected. Figure 4.23 shows two cases; one where the frequency of the input signal is a multiple integer of the bin spacing (thus no bin leakage) and another where the input signal is not (resulting in bin leakage due to the high frequency noise generated by the mismatching of the start and end points of the wave).

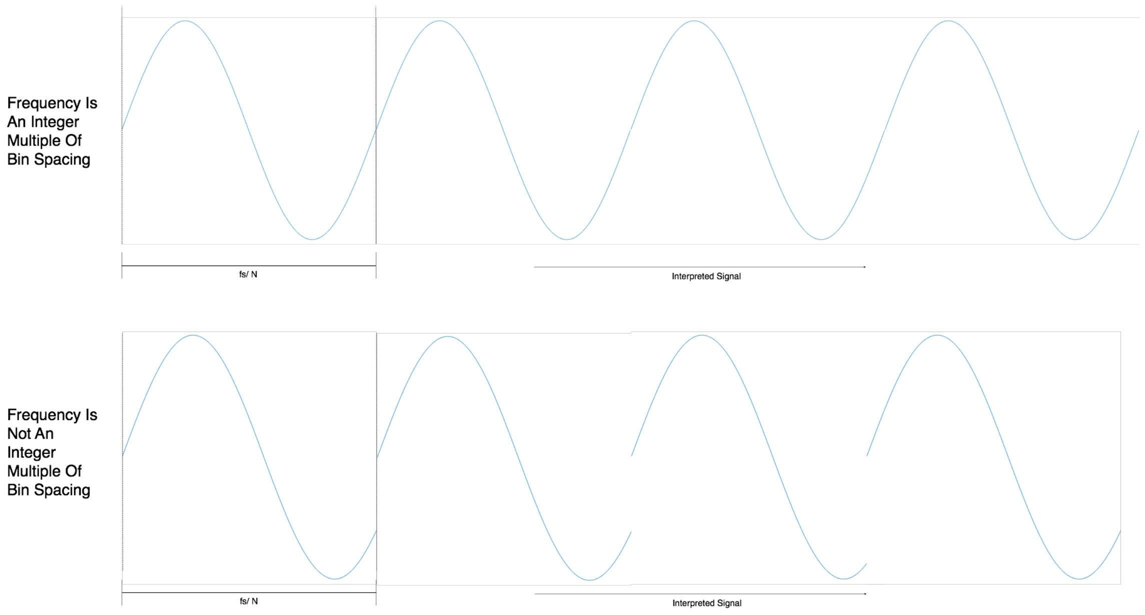


Figure 4.23 - Integer and non-integer multiple of bin spacing

A method for reducing bin leakage is windowing the input signal. To do this the signal is first multiplied by a windowing function (shown in Figure 4.24) and then the DFT is carried out. As shown by Figure 4.25 and Figure 4.26 this causes the start and end points of the signal to match reducing bin leakage. However, there is a trade-off in doing this in that the overall sensitivity and frequency resolution of the DFT will be reduced (Bateman and Paterson-Stephens, 2001).

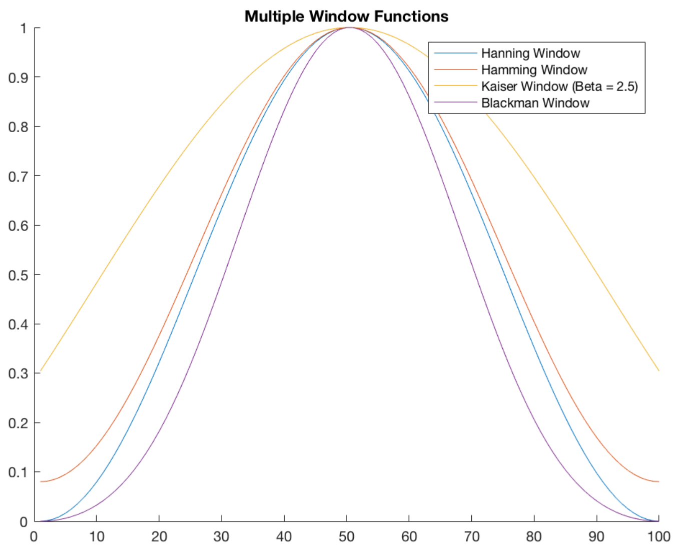


Figure 4.24 - Example window functions

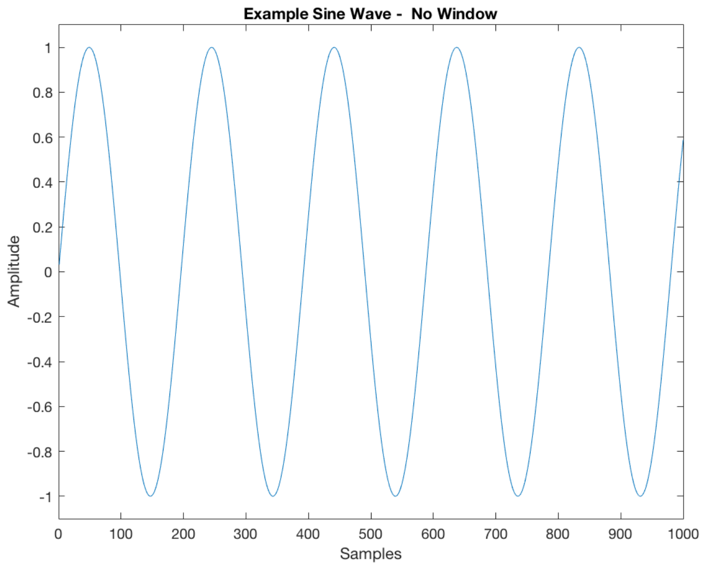


Figure 4.25 - Example sine wave with no applied window

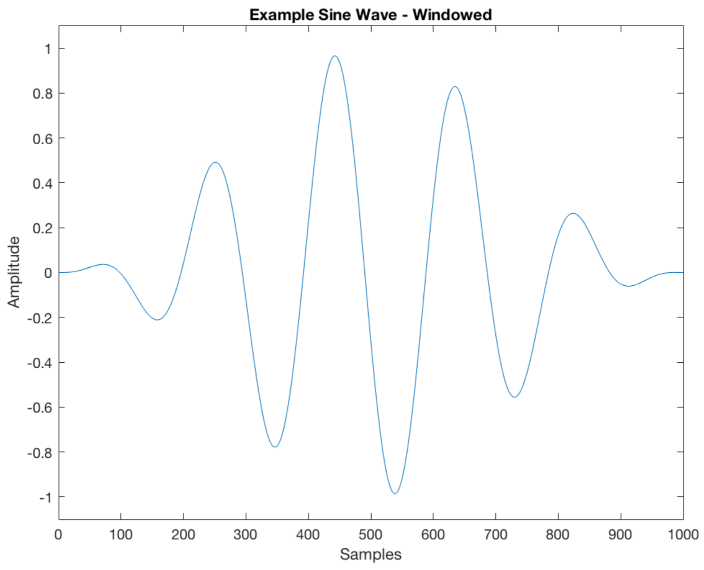


Figure 4.26 - Example sine wave with applied window

Using the windowing technique, a Hanning window can be applied to the 215.3Hz signal analysed in Figure 4.22 and the overall bin spillage can be reduced, Figure 4.27.

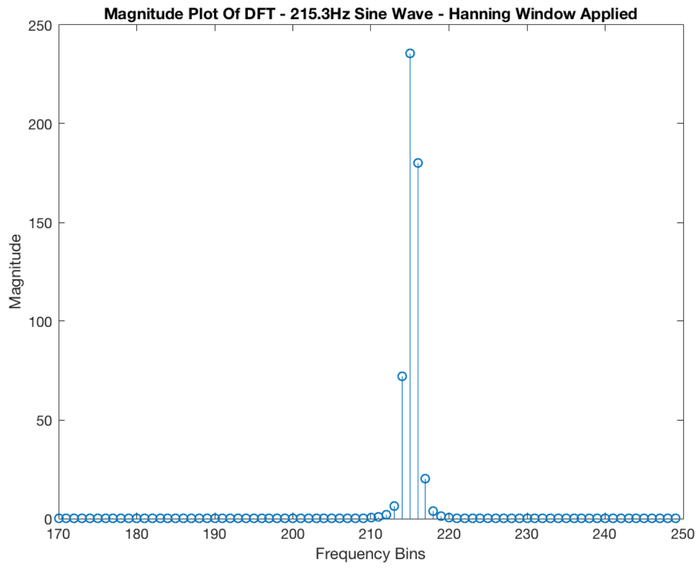


Figure 4.27 - Bin leakage after applying a window function

4.3.3 Benefits of the STFT

The DFT is shown in section 4.3.1 to be a powerful tool in DSP, however what if the input is of a continuous time domain signal? Or if a user desires to see how the frequency content of a signal is changing over time? It is in these instances where the STFT can be used to great effect. By processing small sections of a time domain signal, the frequency content in each section can be analysed, thus allowing a time-frequency analysis.

To split the input signal into smaller sections a window function is used. Using a window function not only allows for reduction of noise from the DFT (as discussed in section 4.3.2) but by overlapping the windows the original audio signal can still be reconstructed. Figure 4.28 shows multiple Hanning windows being overlapped, with the summation of the windows showing a near flat response over a broad range of the output (although normalisation is required).

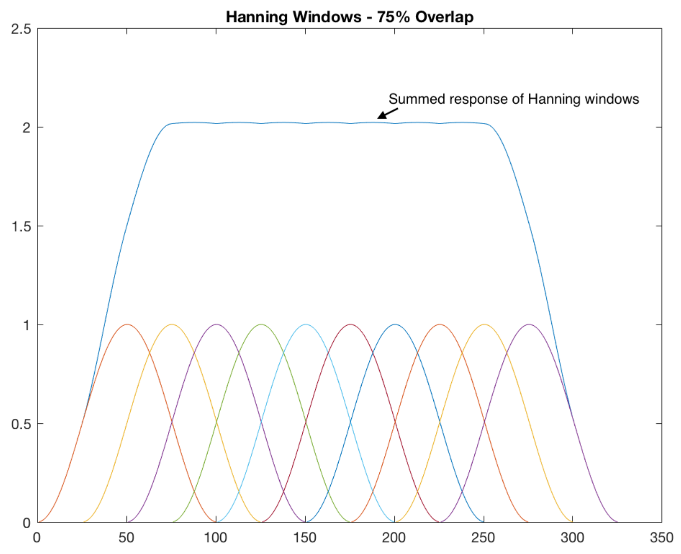


Figure 4.28 - Summation of overlapped Hanning windows

To demonstrate the advantages of the STFT over the DFT a sample signal has been constructed using the code shown below. The signal consists of a 1 second linear chirp from 20Hz to 20kHz, along with a 3kHz sine wave for the 0s to 0.5s and a 6kHz sine wave for 0.5s to 1s.

```
% Sample Signal
t = 0:1/44100:1;
y = chirp(t,20,1,20000);
x = sin(3000 * 2 * pi * t);
x1 = sin(6000 * 2 * pi * t);
y(1:22050) = y(1:22050) + x(1:22050);
y(22051:44101) = y(22051:44101) + x1(22051:44101);
```

Plotting the magnitude of a DFT of this sample signal leads to Figure 4.29. Here it can clearly be seen that there is lot of energy in the signal at 3kHz and 6Khz but there is no indication as to how this energy is spread temporally, nor is there an indication of the linear chirp (although it can be seen that there is an even spread of energy across the frequency band).

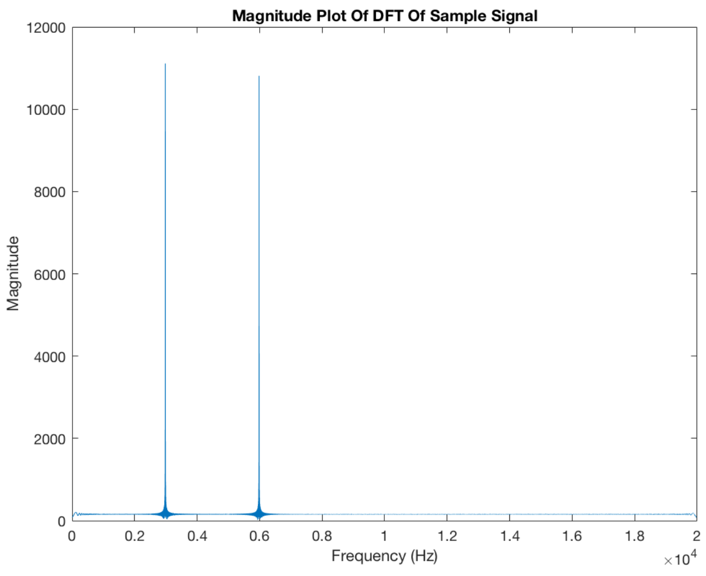


Figure 4.29 - Magnitude plot of sample signal

However, performing an STFT of the same signal allows for a clear analysis of both the time and frequency content of the signal. Figure 4.30 clearly shows a linear chirp stretching the frequency band along with the two sine waves.

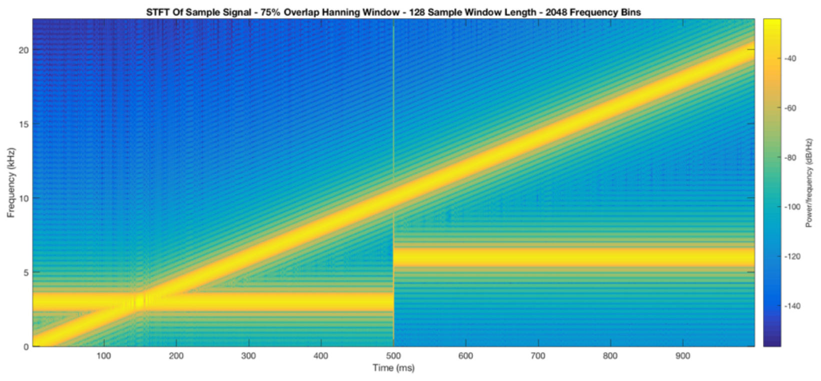


Figure 4.30 - STFT of sample signal

4.3.4 Window Length

When analysing or processing signals in using the STFT the length of the window function used must be considered. A larger window length results in a higher frequency resolution but a lower time resolution, whereas a smaller window increases the time resolution and reduces the frequency resolution (Van Veen, 2013). To demonstrate this property of the STFT a sample signal has been constructed consisting of a 0.5ms 3kHz sine wave followed by a 0.5ms 6kHz sine wave.

```
% Sample Signal
t = 0:1/44100:1;
y = zeros(1,44101);
x = sin(3000 * 2 * pi * t);
x1 = sin(6000 * 2 * pi * t);
y(1:22050) = x(1:22050);
y(22051:44101) = x1(22051:44101);
```

Figure 4.31 and Figure 4.32 show two STFTs with different window length of the sample signal.

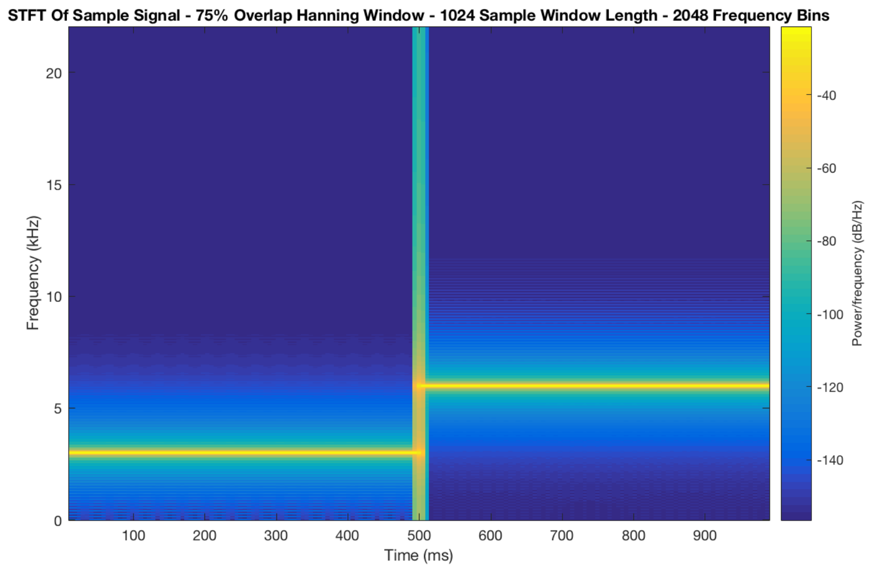


Figure 4.31 - STFT with 1024 sample window length

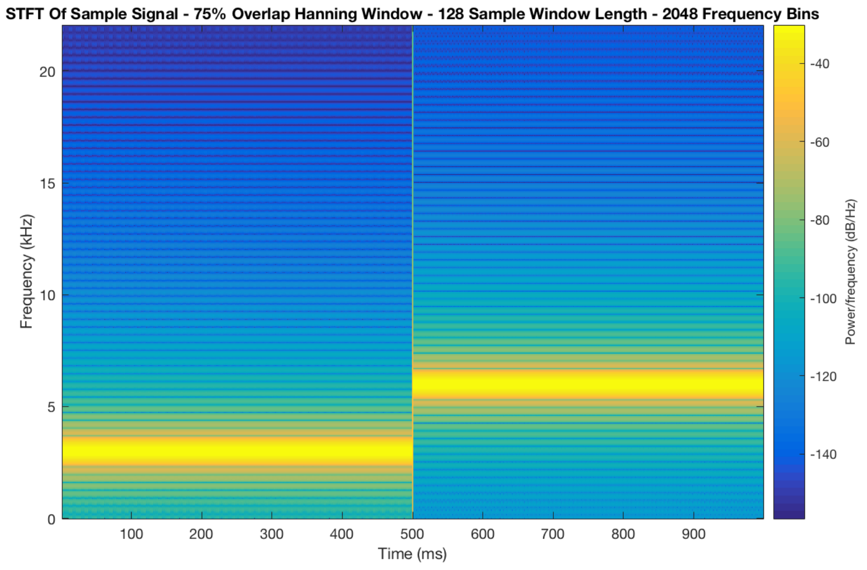


Figure 4.32 - STFT with 128 sample window length

As can be seen in Figure 4.31 a greater window length results in a narrower yellow line (representing energy in the frequency spectrum) thus allowing for a more precise analysis in the frequency domain. However, the transition between the two sine waves in Figure 4.31 shows a, roughly, 20ms blur whereas Figure 4.32 shows a far clearly changeover.

4.3.5 Zero Padding

When a continuous time domain signal is transformed to the frequency domain for processing the technique of zero-padding should be considered. To explain how zero-padding is used it is first helpful to review convolution of fixed length time domain signals. As (Wiggins, 2004) explains, convolution in the time domain can be described through Eq. 4-31.

$$y = c \otimes h$$

$$y(n) = \sum_{i=1}^{N} c(n-i)h(i)$$

Eq. 4-31

c = input signal

h = impulse response

y = convolved signal

n = sample number

N = total number of samples

It can be realised that the total number of samples in the output of the convolution c with h will follow the equation Eq. 4-32.

$$length(y) = length(c) + length(h) - 1$$

Eq. 4-32

Convolution Theorem states that time domain convolution is the equivalent of frequency domain multiplication (Arfken, 1985) thus a fast convolution of two signals can be done in the frequency domain. An example is shown below for the process of convolving a signal with a length of 925 samples with an impulse response with a length of 100 samples.

Fast convolution:

1) Calculate length of final convolved signals, so 925 + 100 – 1 = 1024.
2) Perform FFT with 1024 frequency bins on both signals.
3) Perform point for point multiplication of both FFTs.
4) Apply inverse-FFT to the result.

If the input signal for the convolution is a continuous time signal, then it is impossible to know the total length of the signal and the convolution must therefore be applied to individual slices of the time signal. Using the process of the previously described STFT and then multiplying the individual frequency segments with a given impulse response leads to an issue. As described by Eq. 4-32, after the multiplied frequency segments have had the inverse FFT applied to them they will be longer than their original time slice thus adding sequential slices back together will not lead to the desired output. To overcome this problem the original time slices can be zero-padded, along with the impulse response, and the output time slices can be overlapped and summed (Bateman and Paterson-Stephens, 2001). Figure 4.33 shows how this works.

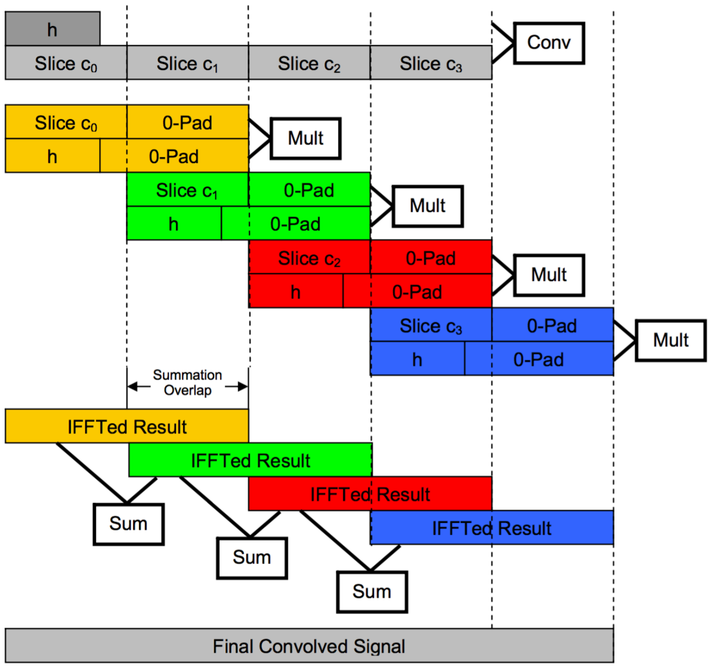


Figure 4.33 - Fast convolution (Wiggins, 2004)

5 Up-mix and Separation Methods

For this project three main methods of source extraction or up-mixing were reviewed and tested and the most suitable was selected to be used in the final algorithm.

5.1 Laitenin

Laitinen (2014) proposes a method for the conversion of stereo signals to B-format by utilizing the principles of DirAC (Directional Audio Coding).

5.1.1 DirAC

Pulkki (2006) discusses the following assumptions that DirAC is based on:

1. Interaural time difference (ITD), interaural level difference (ILD) and monoaural localisation cues are dependent on the direction of sound.
2. Interauaral coherence cues are dependent on the diffuseness of sound.
3. Timbre of a sound is dependent on the monaural spectrum together with the ITD, ILD and interaural coherence values.
4. The spatial image perceived by a listener is dependent on the direction of arrival, diffuseness and spectrum of sound measured in a point with the temporal and spectral resolution of the human hearing system.

These assumptions lead to the notion that a spatial sound field needn't be reproduced perfectly for a listener to perceive it as the original. The direction, diffuseness and spectrum of sound need to match the original at a chosen listening point for the spatial sound field to be perceived as similar.
A further assumption of DirAC is that at a single frequency band within a single time instance there is only one dominant source perceived by a listener. This means that a single time-frequency band is composed of a mix of diffuse sound and directional cues (ITD and ILD) from a single sound source.
DirAC creates two cross-fading streams for a single frequency band, one containing a non-directional diffuse signal and the other containing a non-diffuse directional signal. To create

these two streams, the input for the DirAC analysis is a four channel (W, X, Y and Z) B-format signal in the time-frequency domain (i.e. after a STFT is performed), where the X, Y and Z channels are directional along their respective Cartesian axis and the W channel is omnidirectional. The diffuseness, ψ, and direction of sound, θ_0, can be found by first forming the vector calculated in Eq. 5-2.

$$V = \frac{[X, Y, Z]}{\sqrt{2}}$$

Eq. 5-1

The active intensity, I_a, of the signal at the frequency band k and time bin n can then be found through Eq. 5-2. The direction of sound, θ_0, is defined as the opposite of the direction of the active intensity.

$$I_a(k,n) = -Re\{W^*(k,n)V(k,n)\}$$

Eq. 5-2

Where $Re\{\}$ denotes the real part and * is the complex conjugate.

To calculate the diffuseness of a chosen time-frequency bin Eq. 5-3 can be used.

$$\psi(k,n) = 1 - \frac{\left\| E\left\{Re\{W^*(k,n)V(k,n)\}\right\}\right\|}{E\{|W(k,n)|^2 + \|V(k,n)\|^2/2\}}$$

Eq. 5-3

Where $\|\cdot\|$ is the norm of a vector and $E\{\cdot\}$ is the expectation value, found through temporal averaging. The diffuse value, ψ, will be between 0 and 1, where 0 indicates that the analysed time-frequency bin consists only of a direct sound whilst a value of 1 indicates that the bin consists only of a diffuse signal. A diffuse value in between 0 and 1 shows a mixture of direct and diffuse sound.

After the above analysis stage, virtual microphone feeds need to be created for each loudspeaker location in the playback system. To create these feeds, the direct non-diffuse stream is reproduced using the VBAP technique leading to different gains of the direct signal across the various loudspeakers of the playback system, this causes the direct signal to be heard as a phantom image at the desired point in space. Due to potential abrupt changes in the

direction of the non-diffuse stream, the gain values for the different speaker feeds are smoothed temporally.
The diffuse stream is reproduced via de-correlation of multiple copies of the virtual microphones feeds and sending them to corresponding loudspeakers, creating the impression of a surrounding sound.
Laitinen (2014) visualises this process in Figure 5.1.

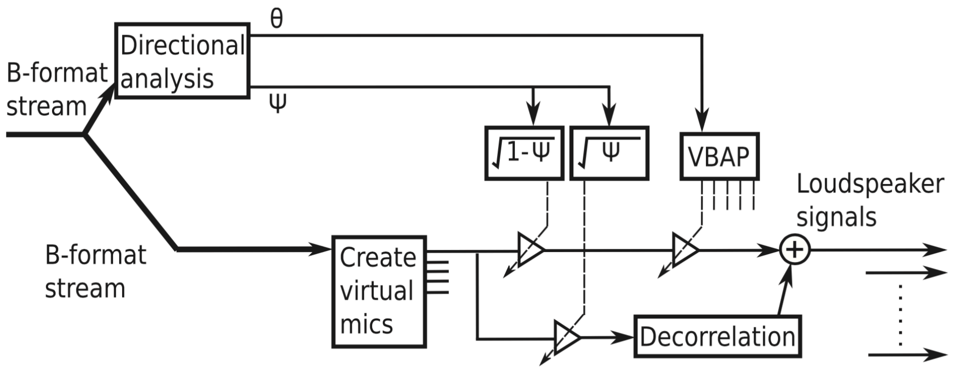


Figure 5.1 - DirAC example (Laitinen, 2014)

5.1.2 Stereo to B-Format Conversion

Laitinen (2014) presents a simple method for creating a B-format signal from a stereo source. A virtual recording is performed by placing virtual loudspeakers at $\pm 30°$ from a virtual microphone, Figure 5.2.

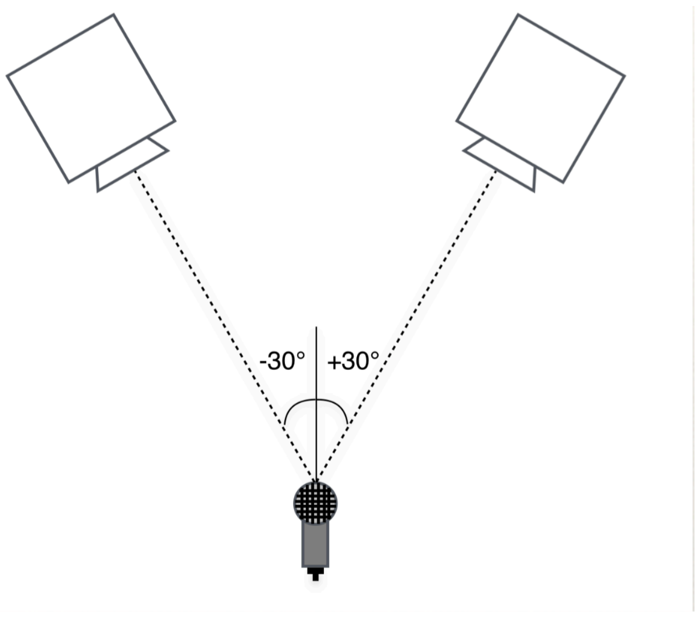


Figure 5.2 - Virtual microphone recording

The X and Y channels are computed through Eq. 5-4.

$$X = \sqrt{2} \cdot \sum_{i=1}^{2} \cos(\theta_i) \cdot S_i$$

$$Y = \sqrt{2} \cdot \sum_{i=1}^{2} \sin(\theta_i) \cdot S_i$$

Eq. 5-4

Where i is the index of the stereo channel, θ_i is the angle of the i_{th} loudspeaker and S_i is the ith channel feed.
The Z channel is set to zero as this channel records height information and a stereo signal is not mixed with height information. The W channel is an omnidirectional channel and is therefore created by summing the left and right channels of the stereo signal.

There are issues with this approach; if an incoherent sound is played in a generic stereo set up the sound will appear to the listener as diffuse and spacious, due to human perception. However, in DirAC the diffuseness of the signal is calculated using Eq. 5-2 and with loudspeakers at $\pm 30°$ in front of a receiving medium (in this case the virtual microphone) there is a greater amount of energy coming from in front than behind, leading to low diffuseness values. This causes the reproduction of diffuse parts to become biased towards the centre of a playback system, rather than spread out.
Laitinen notes that Laitinen and Pulkki (2011) found a solution to a similar issue by using two different loudspeaker set ups for the virtual recording. Laitinen and Pulkki's (2011) two virtual recording set ups were a standard 5.1 system ($0°, \pm 30°$ and $\pm 110°$) and an evenly distributed 5 channel set up ($0°, \pm 72°$ and $\pm 144°$). This meant that the standard set up produced the correct spatial positioning for direct signals whilst the even set up produced correct diffuseness values. The output B-format signal for the whole virtual recording process would be a cross-faded mixture of the standard and evenly distributed set ups, with the amount of each set up used being controlled by the diffuseness value, ψ_{even}, calculated from the evenly distributed set up.
In the case of a stereo recording it is proposed by Laitinen (2014) that the standard $\pm 30°$ should be used with a $\pm 72°$ even set up.
This does lead to one final problem, typically in stereo recordings amplitude panning is used to create a phantom image of a sound source somewhere in between the left and right loudspeakers. Considering the virtual loudspeakers placed at $\pm 72°$, when a source has been centre-panned its Y positioning will be zero and its X positioning will be close to zero, due to the wide positioning of the loudspeakers (shown in Figure 5.3).

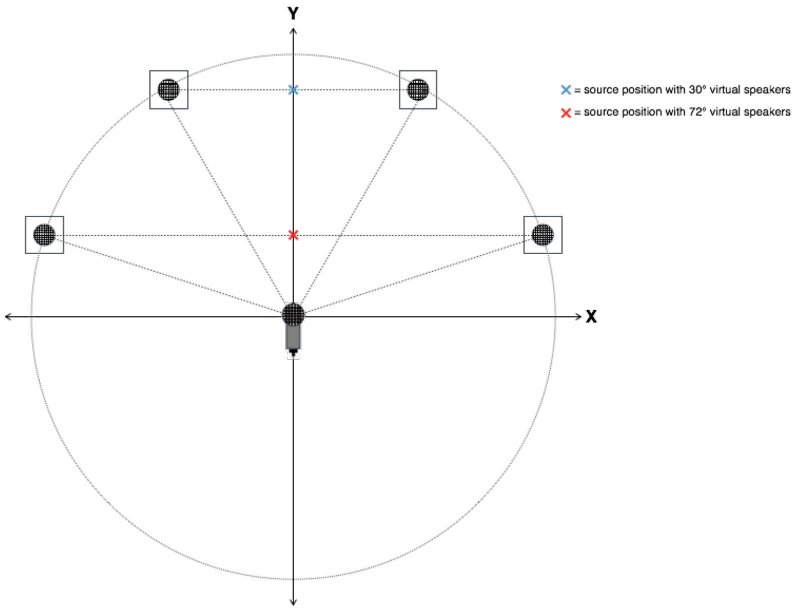

Y
X
× = source position with 30° virtual speakers
× = source position with 72° virtual speakers

Figure 5.3 - Source image position with varying loudspeaker positions

Considering the equations Eq. 5-1 to Eq. 5-3 to calculate diffuseness, V in this instance will be close to a zero vector thus the diffuseness value will be high. The high diffuseness value will cause the centre-panned source to be decorrelated and therefore perceived by the listener as a wide image, not the point-like phantom image originally intended.

The solution proposed by Laitinen (2014) is to steer the diffuseness by calculating the coherence between the left and right channels, using equation Eq. 5-5.

$$\rho(k,n) = \frac{Re\{s_l(k,n) \cdot s_r^*(K,n)\}}{\sqrt{(s_l(k,n) \cdot s_l^*(k,n)) \cdot (s_r(k,n) \cdot s_r^*(k,n))}}$$

Eq. 5-5

Where s_l is the left channel and s_r is the right channel.

From this equation a modified diffuse value can be found, equation Eq. 5-6.

$$\psi'(k,n) = \min\left[\psi_{even}(k,n), 1 - \max\left[0, \rho(k,n)\right]\right]$$

Eq. 5-6

This means for cases where a source is centre-panned the coherence, ρ, will equal close to 1 thus the modified diffuse value, ψ', will be close to 0.

The overall process for the stereo to B-format conversion proposed by Laitinen (2014) is shown in Figure 5.4.

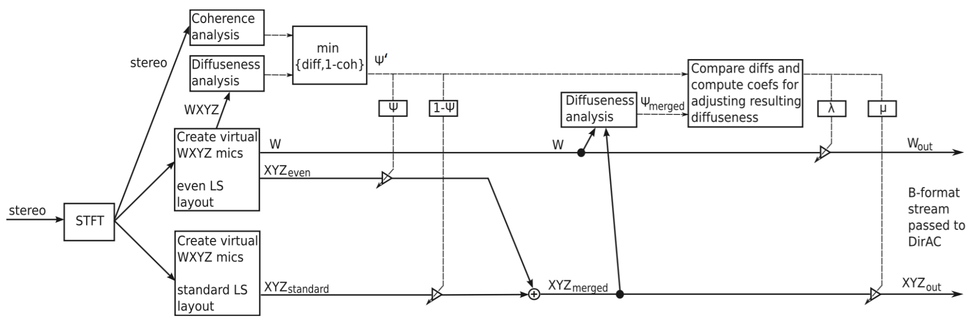

Coherence analysis
Diffuseness analysis
stereo
min {diff,1-coh}
Ψ'
WXYZ
Ψ
1-Ψ
Diffuseness analysis
Ψmerged
Compare diffs and compute coefs for adjusting resulting diffuseness
λ
μ
Create virtual WXYZ mics even LS layout
W
W
Wout
XYZeven
stereo
STFT
B-format stream passed to DirAC
Create virtual WXYZ mics standard LS layout
XYZstandard
XYZmerged
XYZout

Figure 5.4 - Stereo to B-format conversion by Laitinen (2014)

5.2 Menzer and Faller

Before discussing the proposed separation algorithm by Faller and Menzer (2010) it's first important to briefly review the mathematical concept of orthogonality. Anton and Rorres (2010) define two vectors as orthogonal if their inner product, or dot product, is equal to zero. As shown later in this chapter, orthogonal vectors are perpendicular to each other.
The inner product of two vectors can be calculated as equation Eq. 5-7 shows.

$$u = (a_1, a_2, a_3, a_4)$$

$$v = (b_1, b_2, b_3, b_4)$$

$$\langle u, v \rangle = a_1 b_1 + a_2 b_2 + a_3 b_3 + a_4 b_4$$

Eq. 5-7

The following example shows proof that two vectors are orthogonal and proves graphically that this means they are perpendicular (Figure 5.5).

$$u = (4, -4, -6)$$

$$v = (7, 10, -2)$$

$$\langle u, v \rangle = (4 \times 7) + (-4 \times 10) + (-6 \times -2)$$

$$= 28 - 40 + 12$$

$$= 0$$

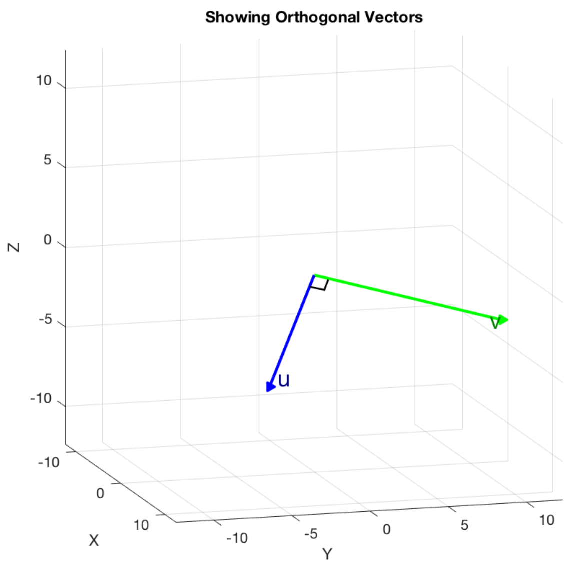


Figure 5.5 - Orthogonal vectors – drawn using Bonarenko (2009)

5.2.1 Signal Model

Faller and Menzer (2010) propose a separation method based on modelling a stereo signal as a vector in an N-dimensional vector space. For the following explanation of their work $x(n)$ is used to represent the samples of signal whilst x is used when representing the signal as a vector. The dot product of two vectors is notated as $\langle u, v \rangle$ and the norm of a vector is shown as $\|x\|$.

The authors follow the same assumption as DirAC, that for a single time instance in a given frequency band there is only one dominant sound source heard by the listener.
The separation algorithm is based around the stereo signal model as shown by Eq. 5-8 and works by separating out the coherent and diffuse parts of the signal.

$$s_l(n) = s_{l,coh}(n) + s_{l,dif}(n)$$

$$s_r(n) = s_{r,coh}(n) + s_{r,dif}(n)$$

Eq. 5-8

These equations are further broken down into Eq. 5-9, where the coherent parts of the signal are shown as a single source with different amplitude panning levels for the left and right channel.

$$s_{l,coh}(n) = \alpha_l c(n)$$

$$s_{r,coh}(n) = \alpha_r c(n)$$

Eq. 5-9

And the diffuse part is shown as being composed as three separate diffuse signals; a diffuse signal only in the left channel, a diffuse signal only in the right channel and a diffuse signal common to both.

$$s_{l,dif}(n) = d_l(n) + d_c(n)$$

$$s_{r,dif}(n) = d_r(n) + d_c(n)$$

Eq. 5-10

Furthermore, it is assumed that d_l, d_r and d_c are orthogonal to each other.

$$d_c \perp d_l \perp d_r \perp d_c$$

It is also assumed that the coherent sound is orthogonal to all diffuse parts.

$$c \perp d_c$$

$$c \perp d_r$$

$$c \perp d_l$$

It is noted that the orthogonality of the above assumptions will only hold true over a long timescale. Also, as the algorithm is processed in the time-frequency domain and STFT must be performed. For this transformation to preserve orthogonality the transform must have non-overlapping windows; although the authors note that when performing this algorithm with an STFT with overlapping windows is used good results are produced.

The algorithm relies on using the sum and difference of each channel.

$$\begin{gathered} s_+(n) = s_l(n) + s_r(n) \\ = (\alpha_l + \alpha_r)c(n) + d_r(n) + d_l(n) + 2d_c(n) \end{gathered}$$

$$\begin{gathered} s_-(n) = s_l(n) - s_r(n) \\ = (\alpha_l - \alpha_r)c(n) + d_l(n) - d_r(n) \end{gathered}$$

Eq. 5-11

The above can be expressed as follows.

$$s_{+}(n) = \alpha_{+}c(n) + \ d_{+}(n)$$

$$s_{-}(n) = \alpha_{-}c(n) + \ d_{-}(n)$$

Eq. 5-12

where $\alpha_{+} = \ \alpha_{l} + \alpha_{r}$

$$\alpha_{-} = \ \alpha_{l} - \alpha_{r}$$

$$d_{+} = \ d_{r} + d_{l} + 2d_{c}$$

$$d_{-} = \ d_{l} - d_{\square}$$

Assuming that the powers of the left and right diffuse signals are equal it can be shown that $d_{+} \perp d_{-}$.

$$\|d_{l}\|^{2} = \|d_{r}\|^{2}$$

$$\langle d_{+}, d_{-} \rangle \ = \ \langle d_{r} + d_{l} + 2d_{c} \ , \ d_{l} - d_{r} \ \rangle$$

As we have already stated $d_{r} \ \perp \ d_{c} \perp \ d_{l}$, meaning the above can be simplified.

$$\langle d_{+}, d_{-} \rangle \ = \ \langle d_{r} + d_{l} \ \ , \ d_{l} - d_{r} \ \rangle$$

$$= \ \langle d_{r} \ , d_{r} \ \rangle - \ \langle d_{l} \ , d_{l} \ \rangle$$

$$= \ \langle d_{r} \ , d_{r} \ \rangle - \ \langle d_{l} \ , d_{l} \ \rangle$$

$$= \|d_{l}\|^{2} - \|d_{r}\|^{2}$$

$$= 0$$

5.2.2 Separation Algorithm

Given equations Eq. 5-12 and that we know the values of $s_+(n)$ and $s_-(n)$ the separation algorithm is based on estimating the values for $\alpha_+(n)$, $\alpha_-(n)$, $d_-(n)$ and $d_+(n)$. As stated previously, the algorithm is based on vector geometry and as such vectors need to be formed, this is done by grouping together and processing each critical band of a single timeframe in the time-frequency domain. These vectors are now represented as upper case letters, meaning the signal model now becomes the following.

$$S_+ = \alpha_+ C + D_+$$

$$S_- = \alpha_- C + D_-$$

Eq. 5-13

It can now be seen that the three orthogonal signals which need to be extracted are D_+, D_- and C. As we are given two vectors but are trying to estimate three this can be viewed as an under-determined problem with infinite solutions, this means some assumptions must be made to solve this problem.

The problem can be simplified by reducing it to finding an orthogonal basis $\{e_C, e_+, e_-\}$ which defines the directions of D_+, D_- and C. By projecting S_+ onto e_C and e_+ respectively $\alpha_+ C$ and D_+ can be found. Similarly, projecting S_- onto e_C and e_- respectively will find $\alpha_- C$ and D_-. Figure 5.6 shows the explained separation method graphically.

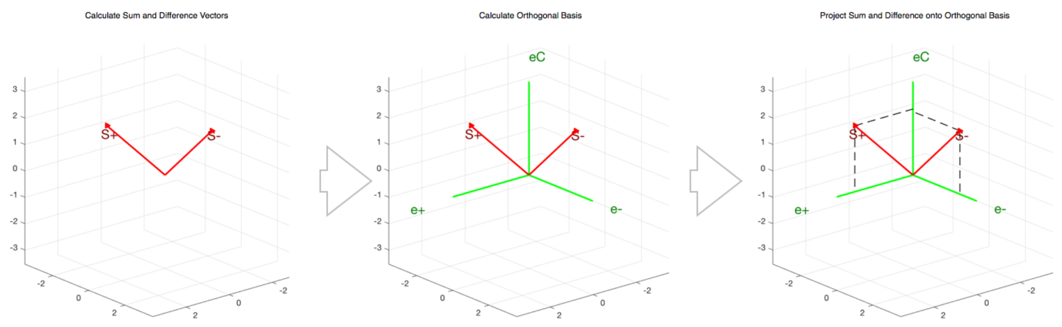

Calculate Sum and Difference Vectors
Calculate Orthogonal Basis
Project Sum and Difference onto Orthogonal Basis
S+
S-
eC
e+
e-

Figure 5.6 - Projection of S+ and S- onto orthogonal basis

It can be seen that S_+must lie in the plane spanned by e_C and e_+ and that S_-must lie in the plane spanned by e_C and e_-, this means that given e_C, S_+ and S_- the vectors e_+ and e_- can be found by performing the first step of the Gram-Schmidt orthonormalisation. Knowing this, it is apparent that the only assumption necessary for this method to work is that of the vector e_C.

5.2.3 The Separation Algorithm

The algorithm proposed by Faller and Menzer (2010) is essentially a recursive random search to find values for □ e_+ and e_- which make the inner product for the vector (e_C, e_+, e_-) as close to zero as possible.

An initial value for e_C is shown below.

$$e_C^0 = \frac{S_+ + \widetilde{S_-}}{\left\| S_+ + \widetilde{S_-} \right\|}$$

Eq. 5-14

Where, $\widetilde{S_-} = \begin{cases} S_- \; if \; \|S_+ + S_-\| \; \geq \|S_+ - S_-\| \\ \quad -S_- \qquad otherwise \end{cases}$

e_+ and e_- can then be calculated using the following equations.

$$e_+^{(u)} = \frac{S_+ - \langle e_C^{(u)}, S_+ \rangle e_C^{(u)}}{\left\| S_+ - \langle e_C^{(u)}, S_+ \rangle e_C^{(u)} \right\|}$$

$$e_-^{(u)} = \frac{\widetilde{S_-} - \langle e_C^{(u)}, \widetilde{S_-} \rangle e_C^{(u)}}{\left\| \widetilde{S_-} - \langle e_C^{(u)}, \widetilde{S_-} \rangle e_C^{(u)} \right\|}$$

Eq. 5-15

where u indicates the step of the iteration.

The recursive part of the method is shown in the following steps, where a near-orthogonal basis is found. Note that on each iteration $\langle e_+, e_- \rangle$ is evaluated as by design $\langle e_C, e_+ \rangle$ and

$\langle e_C, e_- \rangle$ will equal zero, thus the closer the absolute value of $\langle e_+, e_- \rangle$ is to zero the closer the vector (e_C, e_+, e_-) is to being orthogonal.

1) Generate N random unit vectors $\widehat{e_C}_v^{(u+1)}$ $(v \in \{1, \dots, N\})$ that are close to e_C^u.
2) Calculate $\widehat{e_+}_v^{(u+1)}$ and $\widehat{e_-}_v^{(u+1)}$ for all values of v.
3) Calculate $\hat{q}_v^{(u+1)}$, using equation Eq. 5-16, for all values of v.

$$\hat{q}_v^{(u+1)} = \left| \langle \widehat{e_+}_v^{(u+1)}, \widehat{e_-}_v^{(u+1)} \rangle \right|$$

Eq. 5-16

4) Assign the vector $\widehat{e_C}_v^{(u+1)}$ that produces the smallest value $\hat{q}_v^{(u+1)}$ and assign it to become e_C^{u+1}.
5) If $|\langle e_+^{u+1}, e_-^{u+1} \rangle|$ drops below a given threshold then the iteration stops, else restart at step 1 using the value chosen in step 4.

When the threshold has been surpassed at M iterations the basis found can be stated as follows.

$$\{e_C, e_+, e_-\} = \{e_C^{(M)}, e_C^{(M)}, e_C^{(M)}\}$$

From this basis the following vectors can be found.

$$\alpha_+ C = \langle e_C, S_+ \rangle e_C$$

$$\alpha_- C = \langle e_C, S_- \rangle e_C$$

$$D_+ = \langle e_+, S_+ \rangle e_+$$

$$D_- = \langle e_-, S_+ \rangle e_-$$

And from the above the coherent and diffuse vectors can then be found.

$$S_{l,coh} = \frac{1}{2}(\alpha_+ C + \alpha_- C)$$

$$S_{r,coh} = \frac{1}{2}(\alpha_+ C - \alpha_- C)$$

$$S_{l,dif} = \frac{1}{2}(D_+ + D_-)$$

$$S_{r,coh} = \frac{1}{2}(D_+ - D_-)$$

Eq. 5-17

5.3 Multi-Thresholding

Cobos and Lopez (2008) present a method for blind source separation of a stereo signal. This method is heavily based around Otsu's algorithm for maximising inter-class variance (Otsu, 1979) and it is first beneficial to understand this algorithm.

5.3.1 Otsu's Algorithm

Otsu's algorithm was developed to perform image thresholding, whereby a grey-level image can be transformed to a binary (black or white) image. It does this by considering each pixel in the image as being one of two classes (background or foreground). By constructing a histogram using the brightness levels of each pixel, Otsu's algorithm finds an optimum threshold whereby the two classes are separated so that their inter-class variance is maximised. What Otsu showed was that using a method to maximise inter-class variance resulted in minimising within-class variance. Although the algorithm can be extended to find multiple thresholds, below explains the method for finding a single threshold (bi-level thresholding).

Consider a histogram with L number of bins, the probability that a value chosen at random from the total data set falls into bin n is as follows.

$$p_n = \frac{H(n)}{N}$$

Eq. 5-18

$$N = \sum_{n=1}^{L} H(n)$$

The histogram will be separated by a threshold, t, into two classes; the first class, c_1, contains histogram bins $n \in [1, \dots, t]$ whilst the second class, c_2, contains histogram bins $n \in [t + 1, \dots, L]$. Considering values within a given class, the probably that a value falls within a given bin inside that class is shown below.

$$c_1: \frac{p_1}{\omega_1(t)}, \dots, \frac{p_t}{\omega_1(t)}$$

$$c_2: \frac{p_{t+1}}{\omega_2(t)}, \dots, \frac{p_L}{\omega_2(t)}$$

where $\omega_1(t) = \sum_{n=1}^{t} p_n$ and $\omega_2(t) = \sum_{n=t+1}^{L} p_n$.

The means for each class can be calculated as follows.

$$\mu_1 = \sum_{n=1}^{t} n \frac{p_n}{\omega_1(t)}$$

$$\mu_2 = \sum_{n=t+1}^{L} n \frac{p_n}{\omega_2(t)}$$

Eq. 5-19

Let μ_T be the mean for the whole data set.

$$\omega_1\mu_1 + \omega_2\mu_2 = \mu_T$$

$$\omega_1 + \omega_2 = 1$$

Eq. 5-20

Otsu proves that to find the optimum threshold, t^*, the between-class variance, σ_B^2, must be maximised. Where σ_B^2 can be found through equation Eq. 5-21.

$$\sigma_B^2 = \omega_1(\mu_1 - \mu_T)^2 + \omega_2(\mu_2 - \mu_T)^2$$

Eq. 5-21

Thus equation Eq. 5-22 can be stated.

$$t^* = \operatorname{Arg}\max\{\sigma_B^2(t)\}, \qquad 1 \leq t \leq L$$

Eq. 5-22

In the case of multilevel thresholding the histogram is divided into M classes, therefore $M - 1$ thresholds are to be found. The classes can be explained as followed.

$$c_1\colon [1, \dots, t_1]$$
$$c_2\square[t_1 + 1, \dots, t_2]$$
$$\cdots$$
$$c_M\colon [t_{M-1} + 1, \dots, L]$$

Equation Eq. 5-21 can be expanded to allow for multiple thresholds.

$$\sigma_B^2 = \sum_{k=1}^{M} \omega_k (\mu_k - \mu_T)^2$$

Eq. 5-23

$$\omega_k = \sum_{n \in C_k} p_n$$

$$\mu_k = \sum_{n \in C_k} n \frac{p_n}{\omega(k)}$$

And the optimum thresholds are found when the following is satisfied.

$$\{t_1^*, t_2^*, \dots, t_{M-1}^*\} = \operatorname{Arg\,max}\{\sigma_B^2(t_1, t_2, \dots, t_{M-1})\}, \qquad 1 \le t \le L$$

Cobos and Lopez (2008) go on to show that the search for optimum thresholds can be reduced to maximising a simplified form of the between-class variance, σ'^2_B.

$$\sigma'^2_B = \sum_{k=1}^{M} \omega_k \mu_k^2$$

Eq. 5-24

5.3.2 Faster Algorithm

As the aim of this project is to create an algorithm with the potential to be created into software, it is important to consider the efficiency and, where possible, reduce the number of calculations needed to achieve the desire outcome. Liao, Chen and Chung (2001) present a faster algorithm for finding multiple thresholds.

The idea behind the faster algorithm is to compute the bulk of the calculations, and create look-up tables, before the iterate process to find the optimum thresholds begins.

Firstly, two look-up tables are created for a $u - v$ interval.

$$P(u,v) = \sum_{n=u}^{v} p_n$$

$$S(u,v) = \sum_{n=u}^{v} np_n$$

Eq. 5-25

To efficiently compute the values in these tables the first row (where $u = 1$) can be filled as follows.

$$P(1,0) = 0$$

$$P(1, v+1) = P(1,v) + p_{v+1}$$

$$S(1,0) = 0$$

$$S(1, v+1) = S(1,v) + (v+1)p_{v+1}$$

The rest of the tables can then be filled using the following,

$$P(u,v) = P(1,v) - P(1,u-1)$$

$$P(u,v) = P(1,v) - P(1,u-1)$$

A final look-up table can then be created.

$$G(t_{i-1}+1, t_i) = \frac{S(t_{i-1}+1, t_i)^2}{P(t_{i-1}+1, t_i)}$$

Eq. 5-26

And, using this table, the simplified between-class variance, σ'^2_B, of a chosen group of thresholds can be found.

$$\sigma'^2_B = G(1, t_1) + G(t_1 + 1, t_2) + \cdots + G(t_{M-1} + 1, L)$$

Eq. 5-27

Where the search range the thresholds is $1 \leq t_1 \leq L - M + 1,\ t_1 + 1 \leq t_2 \leq L - M + 2, \ldots, t_{M-2} \leq t_{M-1} \leq L - 1$

A comparison of the two methods calculating 1 to 4 optimum thresholds for the same data set is shown in Figure 5.7.

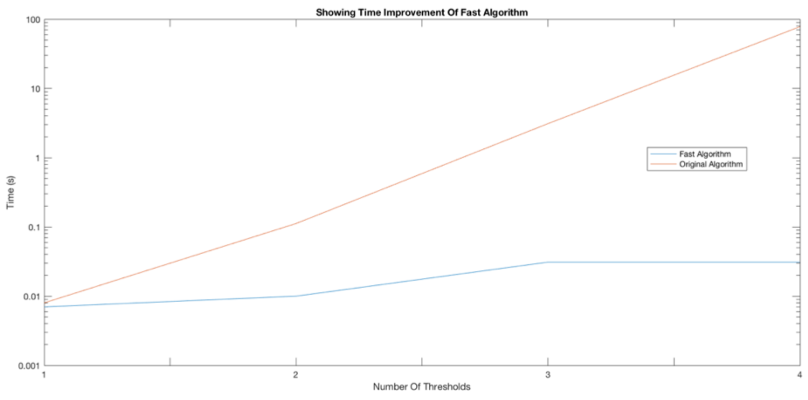


Figure 5.7 - Comparison of calculation times for original and fast threshold algorithms

Note the logarithmic scale on the time axis, this was necessary as the times for the algorithms to calculate 4 thresholds came to be 0.529s compared to 78.041s. These times will clearly be dependent on the platform with which the calculations are carried out however they show an obvious, and very significant, improvement.

5.3.3 Separation Algorithm

5.3.3.1 Pan Maps

The first step for the source separation algorithm proposed by Cobos and Lopez (2008) is to convert the left and right channels of the stereo signal using the STFT and create a pan map. Figure 5.8 and Figure 5.9 show an example of a resultant STFT, carried out using a 180ms Hanning window with 75% overlap. Note the number of frequency bins used in the STFT was calculated as follows.

nfft = wlen + zeroPadSize - 1;c
nfft = 2 ^ nextpow2(nfft);

Where the zero pad size equalled the size of the window length.

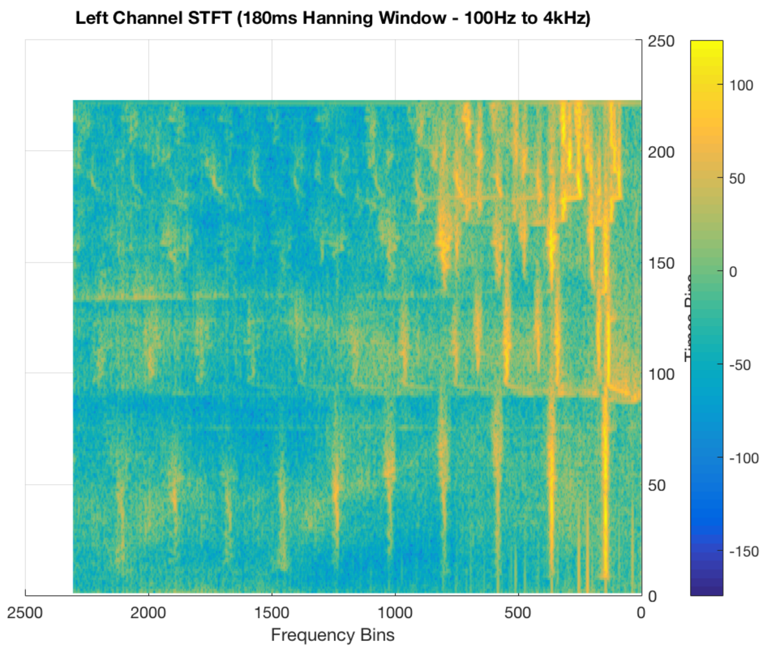


Figure 5.8 - Left channel STFT

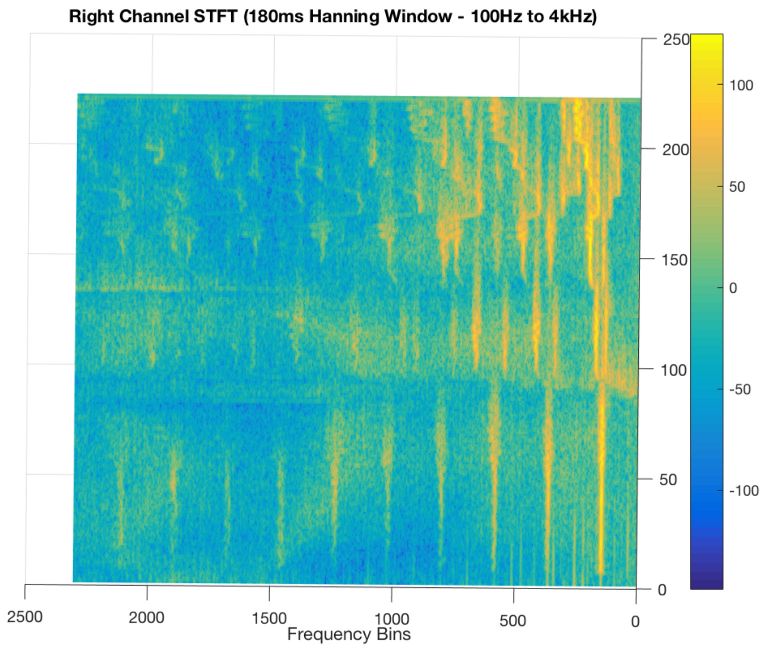


Figure 5.9 - Right channel STFT

Note that although the separation process is carried out for the whole frequency spectrum of the stereo signal, when detecting individual sources only the frequency bins representing 100Hz to 4kHz are used. This is due to the assumption that musical sources will hold most of their energy within this frequency band whilst energy over 4kHz is mainly due to higher harmonics.

Using equation Eq. 5-28 a pan map of the stereo signal can be produced, Figure 5.10.

$$PanMap(k,m) = 20\,log\left(\frac{|Left(k,m)|}{|Right(k,m)|}\right)$$

Eq. 5-28

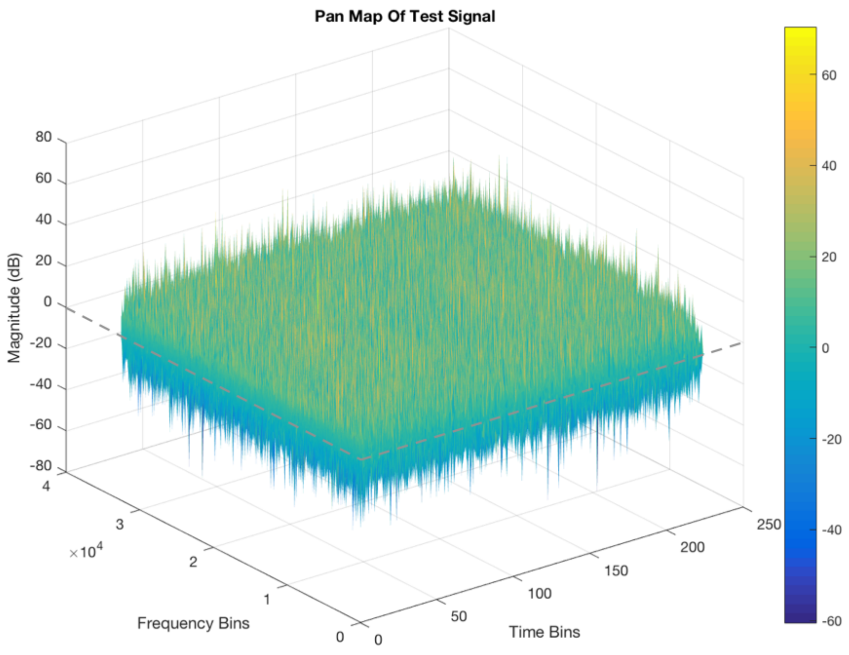


Figure 5.10 - Pan map

By plotting the magnitude of the signal in the left channel divided by the signal in the right channel this pan map effectively shows how far left or right each time-frequency bin is panned, where a louder level is the left channel represents a signal panned further to left and vice versa.

The pan map can be broken into two maps showing time-frequency bins panned left or right, respectively. Using equations Eq. 5-29, Figure 5.11 and Figure 5.12 show the absolute level of each bin in the left and right channel.

$$LeftPanMap(k,m) = \begin{cases} PanMap(k,m) \; if \; PanMap(k,m) \geq 0 \\ 0 \; if \; PanMap(k,m) < 0 \end{cases}$$

$$RightPanMap(k,m) = \begin{cases} PanMap(k,m) \; if \; PanMap(k,m) < 0 \\ 0 \; if \; PanMap(k,m) > 0 \end{cases}$$

Eq. 5-29

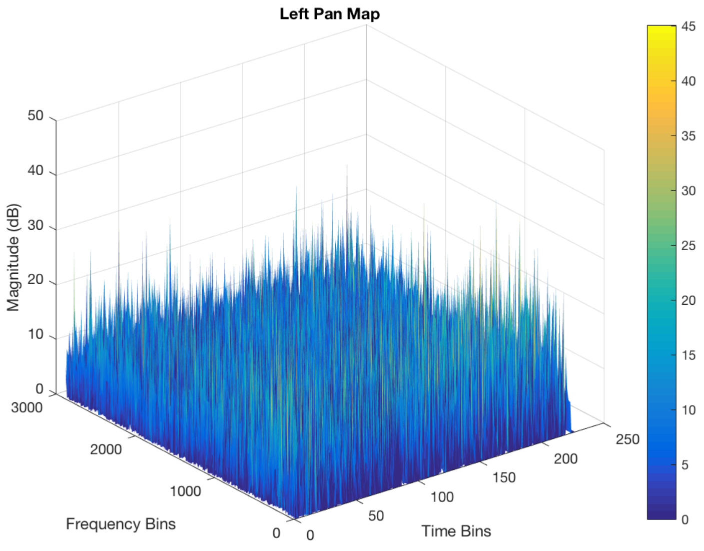


Figure 5.11 - Left pan map

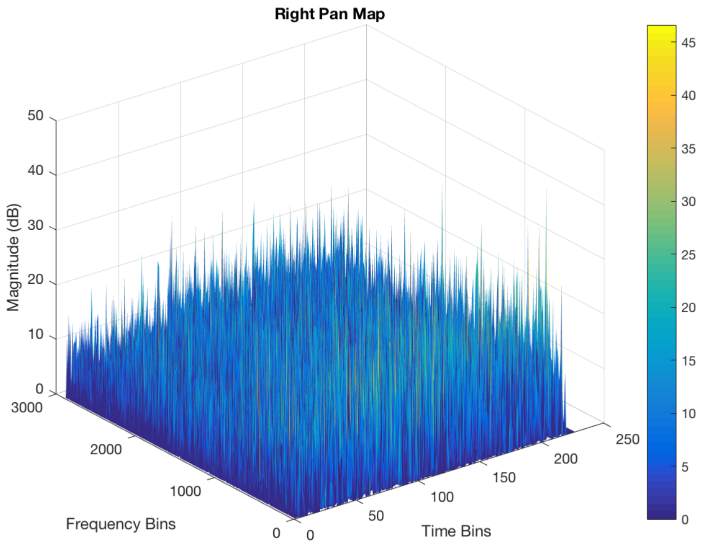


Figure 5.12 - Right pan map

5.3.3.2 Convert to Histogram

The next step in the separation process is to create histograms from each of the left and right pan maps. To convert the pan maps to histograms they are first normalised to a range of 0 to 1.

Cobos and Lopez (2008) describe a method for weighting the histogram using equation Eq. 5-30.

$$g(k) = \frac{\log(100)}{\log(100 + k - k_{min})}$$

Eq. 5-30

Where $g(k)$ is the new time-frequency bin value, k is the current value and $k_{min}$is the frequency bin number representing, or closest to, 100Hz.
They argue that using this weighting function will help match the linear spacing of the STFT bins to human hearing, which follows a logarithmic scale. However, it was decided that this weighting function would not be used on the grounds that, although it is true humans hear in a logarithmic scale, the audio won't have been panned with any such frequency weighting thus applying weighting factors to the time-frequency bins seems unnecessary. Instead two histograms were created using linear spaced edge values ranging from 0 to 1 in increments of 0.01. The resulting histograms are shown below.

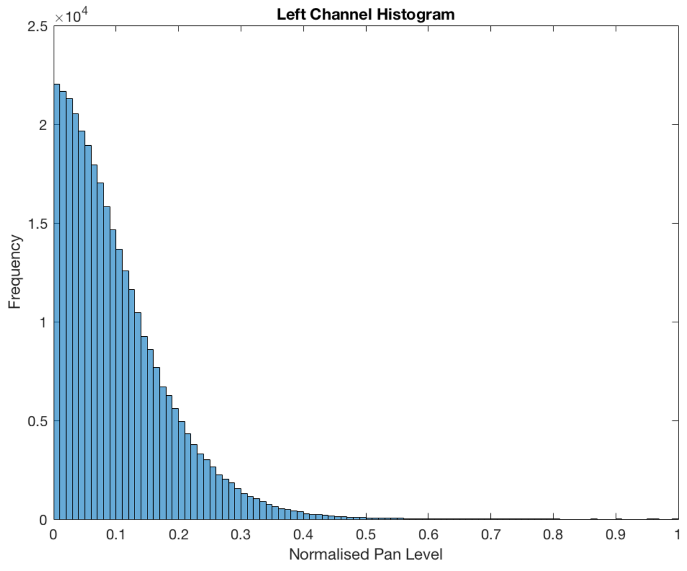


Figure 5.13 - Left channel histogram

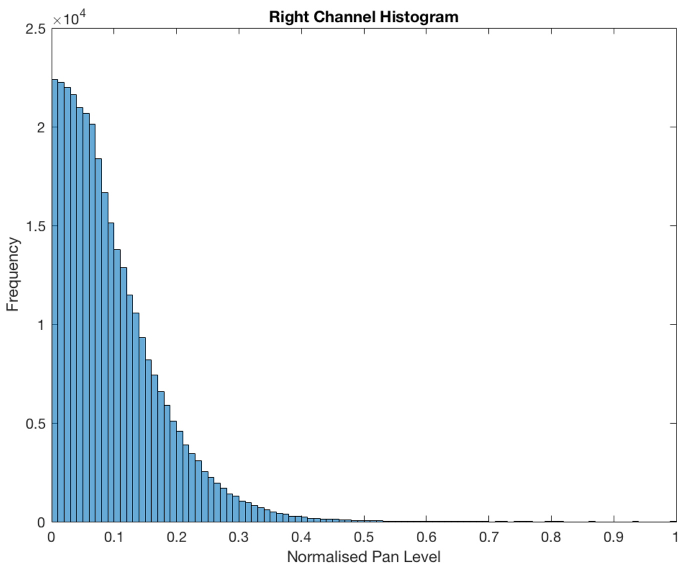


Figure 5.14 - Right channel histogram

Note that the individual left and right pan maps will contain numerous bins with a value of zero (these indicate where the bin has been panned in the opposite channel), to create the above histograms the zero values must be ignored else the first bin will wrongly show as containing a significantly higher amount of time-frequency bins than it actually does.

5.3.3.3 Thresholding

The next step is to separate the bins in the histogram into individual sources, this is where Otsu's algorithm (described in section 5.3.1) is used. In this example the histograms are split into three different sources, meaning two thresholds are found.

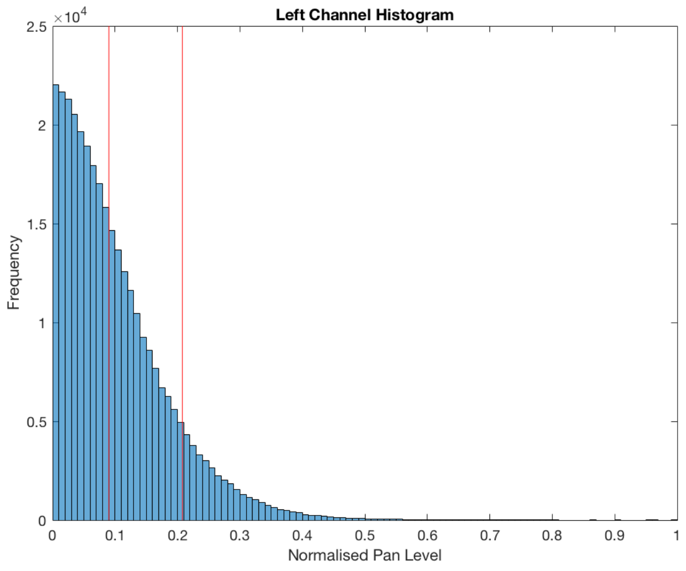


Figure 5.15 - Left channel histogram with thresholds

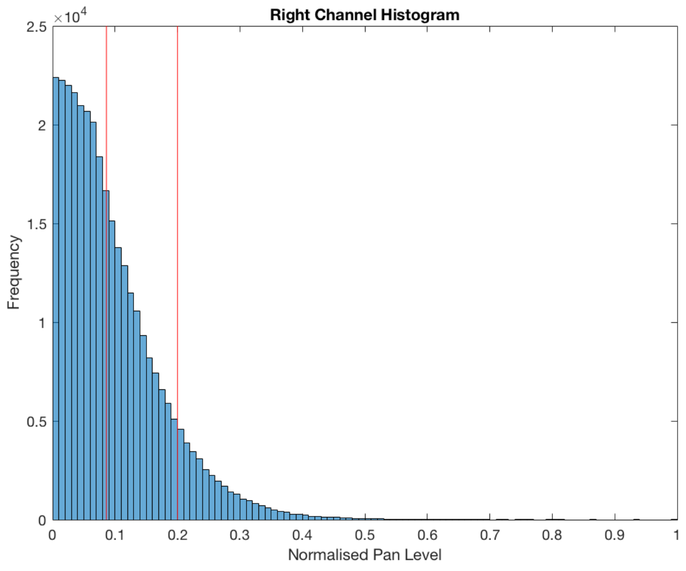


Figure 5.16 - Right channel histogram with thresholds

5.3.3.4 Create Binary Source Masks

Using the thresholds found and Eq. 5-31, binary masks of each source can be created.

$$Left\ Source\ Binary\ Map_i = \begin{cases} 1\ if\ Th_{i-1}^{Left} < LeftPanMap \leq Th_i^{Left} \\ 0\ elsewhere \end{cases}$$

$$Right\ Source\ Binary\ Map_i = \begin{cases} 1\ if\ Th_{i-1}^{Right} < RightPanMap \leq Th_i^{Right} \\ 0\ elsewhere \end{cases}$$

Eq. 5-31

Note that the source maps created are now contain the whole frequency range of the original signal.

The first three of these masks are shown in Figure 5.17 to Figure 5.19 (given a left and right channel where three sources have been found in each, six masks will be created overall).

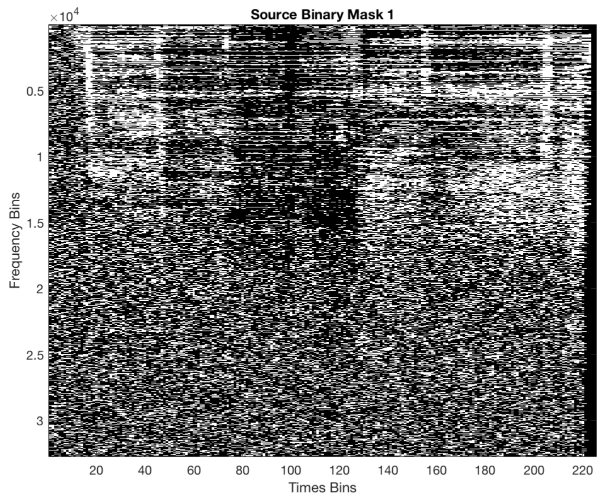


Figure 5.17 - Binary mask 1

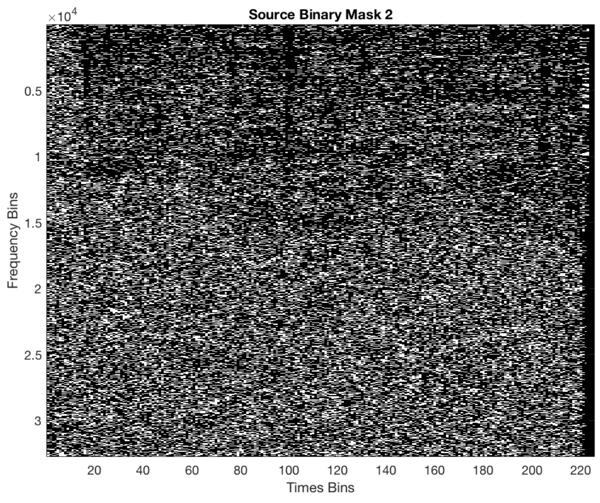


Figure 5.18 - Binary mask 2

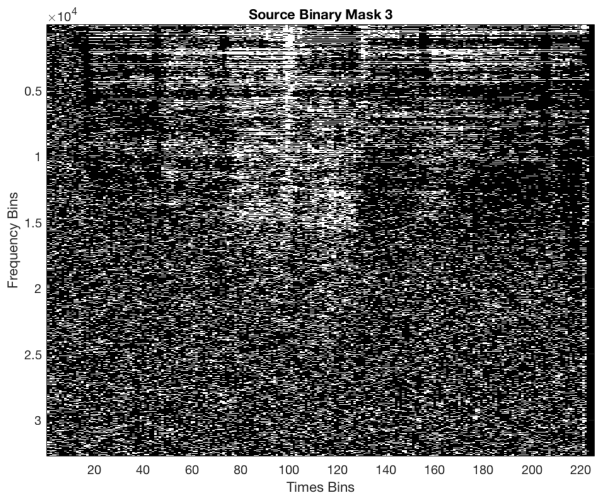


Figure 5.19 - Binary mask 3

Studying the above figures the dark regions in the time axis clearly show transients within the audio, whilst parallel dark lines in the frequency axis can be understood as harmonics.

5.3.3.5 Grouping Similar Source Maps

A problem arises due to the uncertainty of how many sources are actually present within the audio. If the number of thresholds found is equal to or greater than the number of sources panned in the original signal then this method will create binary source masks which don't contain a whole source, only fragments of other, closely panned, sources. For this reason, a method is shown to compare and merge similar binary source masks.

This method works by first ordering the masks so that they are azimuth adjacent; going from hard left to hard right this means the loudest source from the left channel would be the 1st source mask, then the masks would follow order through to the lowest level left source mask, then the lowest level right mask through to the loudest source mask in the right channel. They are ordered this way as due to the way this extraction method works any time a source is split over two masks those masks will always be azimuth adjacent.

Once they're ordered correctly an equation is used to calculate the average *distance* between two adjacent masks. This is done by dividing each mask into a grid Ndivisions wide in the frequency dimension and M divisions wide in the time dimension. Corresponding cells between adjacent binary masks are then subtracted from each other and the absolute value of all cells between each mask is averaged, giving us the average *distance* between masks. This can be shown in the equation below.

$$d_{i,i+1} = \frac{1}{N \times M} \sum_{n=1}^{N \times M} \left| m_i(n) - m_{i+1}(n) \right|$$

Eq. 5-32

d = distance

N, M = grid size

m_i = sum of current cell for i_{th} mask

A grid being applied to the previous three masks is shown in Figure 5.20.

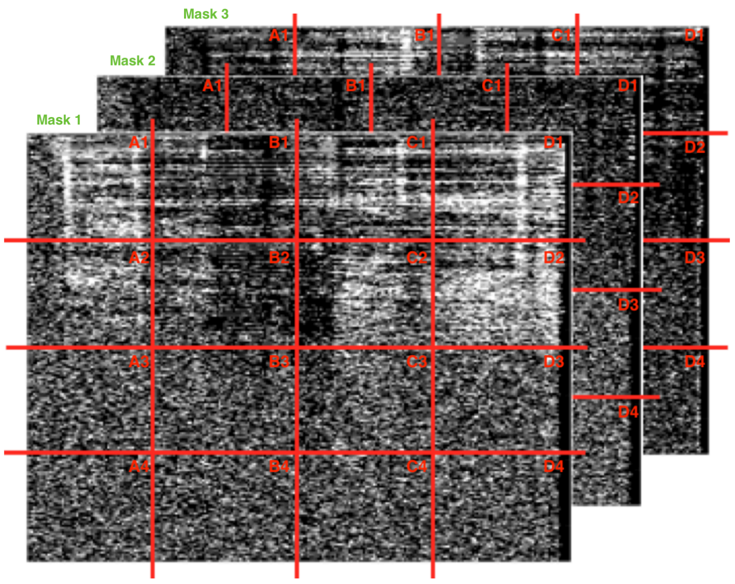


Figure 5.20 - Grid applied for distance calculation

Cobos and Lopez (2008) state that where the distance value between corresponding masks is an absolute or local minimum then the two masks can be merged.

In practice this method posed two main problems; what is an appropriate grid sizing to use? And what classes as a local minimum?

The first question was solved by running Eq. 5-32 for a set of 8 masks whilst increasing the grid sizing, going from 1 division to 25 (note that means $25^2 = 625$ separate cells to be compared per mask). The results from this test are shown in Figure 5.21.

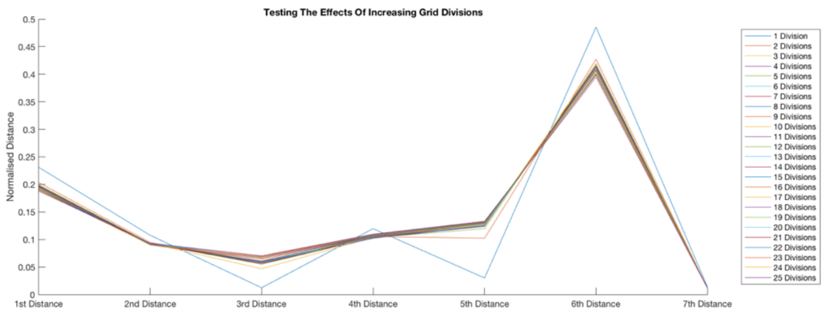


Figure 5.21 - Change in distance values due to increased grid divisions

It can clearly be shown that using around 4 divisions should be enough to get an accurate result for the binary source masks distances. It is also relevant to note that the computing power necessary for higher divisions goes up significantly as a higher number of divisions are used.

The second question proved difficult to answer. A method is proposed based on the assumption that if each mask represents a whole, individual source then the distance between each mask would be equal (it can also be stated that, using this assumption, that the normalised distance between two sources should be equal to $\frac{1}{number\ \ of\ masks}$). It was then proposed that a maximum deviation from this benchmark is allowed and any distance between masks which falls below this will result in the masks being summed. Figure 5.22 shows the first step of this method, where the distance values of 8 masks has been calculated and plotted alongside the mean and the threshold (the threshold here is set to 80% of the mean).

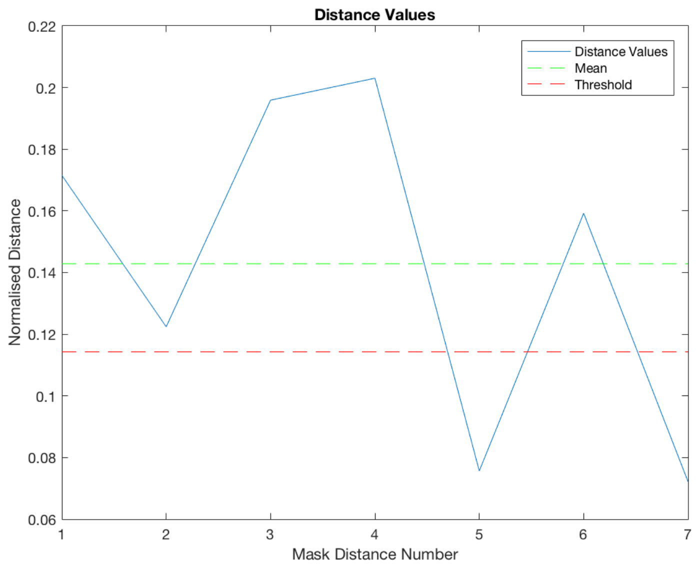


Figure 5.22 - Grouping method using mean distance

Distances 5 and 7 are below the threshold, since distance 7 is the minimum and represents the distance between mask 7 and 8 those masks are then summer and an updated distance values can be calculated, Figure 5.23.

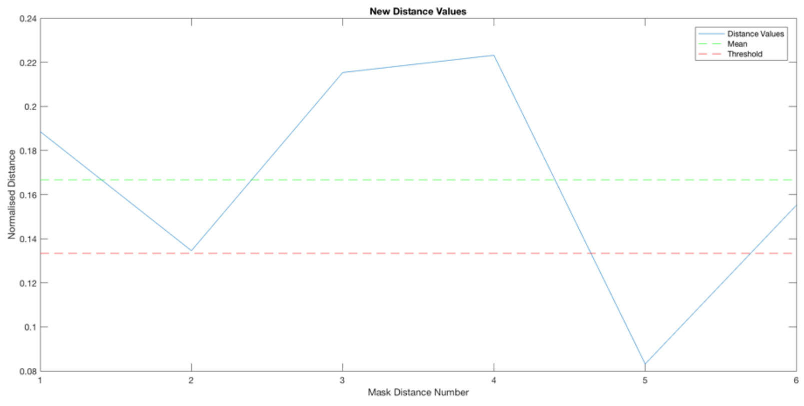


Figure 5.23 - Updated distance values

This process would be repeated until all distances lie above the threshold. This method was tested with multiple audio files which possessed varying numbers of mixed sources and underwhelming results were seen. Through the testing, different threshold levels were tried and an optimum threshold level proved difficult to find.

5.3.3.6 Retrieving Audio Sources

Once the final group of binary source masks has been found then the individual audio sources can be obtained through equation Eq. 5-33.

$$\hat{s}_j = STFT^{-1}\{M_j[S_L + S_R]\}$$

Eq. 5-33

$\hat{s}_j$ = extracted source

M_j= binary source mask

S_L = left channel STFT

S_R = right channel STFT

j = source number

$STFT^{-1}$ = inverse STFT

6 Proposed Algorithm

6.1 Why Choose Multi-Level Thresholding?

Multi-level thresholding was chosen to be used in the proposed up-mix algorithm for the following reasons:

- Individual audio sources are extracted rather than diffuse/direct streams. This made it easier to test the separation section of the algorithm and avoided issues Laitinen (2014) mentions regarding how best to playback diffuse material. In this method reverberant material will be distributed across all thresholds therefore still be spread horizontally around the listener.
- The technique used by multi-level thresholding to extract the individual audio sources allows for the azimuth of each source to be easily and accurately calculated (discussed in section 6.2).

6.2 Azimuth Extraction for Multi-Level Thresholding

To use this method in the overall proposed up-mix algorithm one further parameter is needed to be calculated, the azimuth of the extracted sources. This parameter is not covered by Cobos and Lopez (2008) in this blind source separation technique so a proposed method for this will be described, as follows, and tested in section 7.
From section 4.2.1.1, Eq. 4-17 can be restated.

$$G_L^2 + G_R^2 = 1$$

The above equation will hold true assuming constant power panning is used in the majority of commercial stereo music.
Using the multi-thresholding technique two STFT's are calculated, for the left and right channels respectively. A binary mask is then found for a source and applied to the sum of the left and right STFT's to retrieve the audio. However, if this mask is instead applied to the left and right STFT's respectively the level of the source in the left and in the right channel is found independently. By finding the binary mask M_x, which represents source x, and applying it to either STFT we can find X_{left} and X_{right}, whereby X_{left} and X_{right} are the

frequency domain complex-values for source x in the left and right channels. By finding the average magnitude of X_{left} and X_{right} we find an approximation for xG_L and xG_R, where x is the source signal and G_L and G_R are the left and right panning co-effecients. From knowing $xG_\square$and xG_R, the values G_L and G_R can be derived without knowing the value of x by using Eq. 6-1.

$$G_L = \frac{xG_L}{\sqrt{(xG_L)^2 + (xG_R)^2}}$$

$$G_R = \frac{xG_R}{\sqrt{(xG_L)^2 + (xG_R)^2}}$$

Eq. 6-1

The above equations can be proved as follows.

$$(xG_L)^2 = x^2 {G_L}^2$$

$$(xG_R)^2 = x^2 {G_R}^2$$

Eq. 6-2

Thus equations Eq. 6-1 can be updated as.

$$G_L = \frac{xG_L}{\sqrt{x^2 {G_L}^2 + x^2 {G_R}^2}}$$

$$G_R = \frac{xG_R}{\sqrt{x^2 {G_L}^2 + x^2 {G_R}^2}}$$

Eq. 6-3

Rearranging the denominator and substituting in Eq. 4-17 allows for the following simplification.

$$x^2{G_L}^2 + x^2{G_R}^2 = x^2(G_L^2 + G_R^2)$$

$$x^2{G_L}^2 + x^2{G_R}^2 = x^2$$

Eq. 6-4

Eq. 6-3 then become the following.

$$G_L = \frac{xG_L}{\sqrt{x^2}}$$

$$G_R = \frac{xG_R}{\sqrt{x^2}}$$

Eq. 6-5

Which can easily be rearranged to prove the validity of equation Eq. 6-1.

Once $G_L$and G_R have been found, it is simply a matter of using Eq. 4-16, restated below, to calculate the azimuth for each source.

$$\theta = \tan^{-1}(\tan(\theta_0) \cdot \frac{G_L - G_R}{G_L + G_R})$$

Note that in the up-mix algorithm the speaker separation, θ_0, will be exaggerated (likely left as a variable for the user to control). Where the typical $\pm 30°$ speaker separation is suitable for stereo playback, in ambisonics a full 360° horizontal plane is available. Figure 6.1 shows four audio sources panned in a stereo image. The sources are then remapped in the horizontal plane by changing θ_0. Note that sources 1 and 4, originally panned furthest left and right, show the greatest change in azimuth when θ_0 is changed.

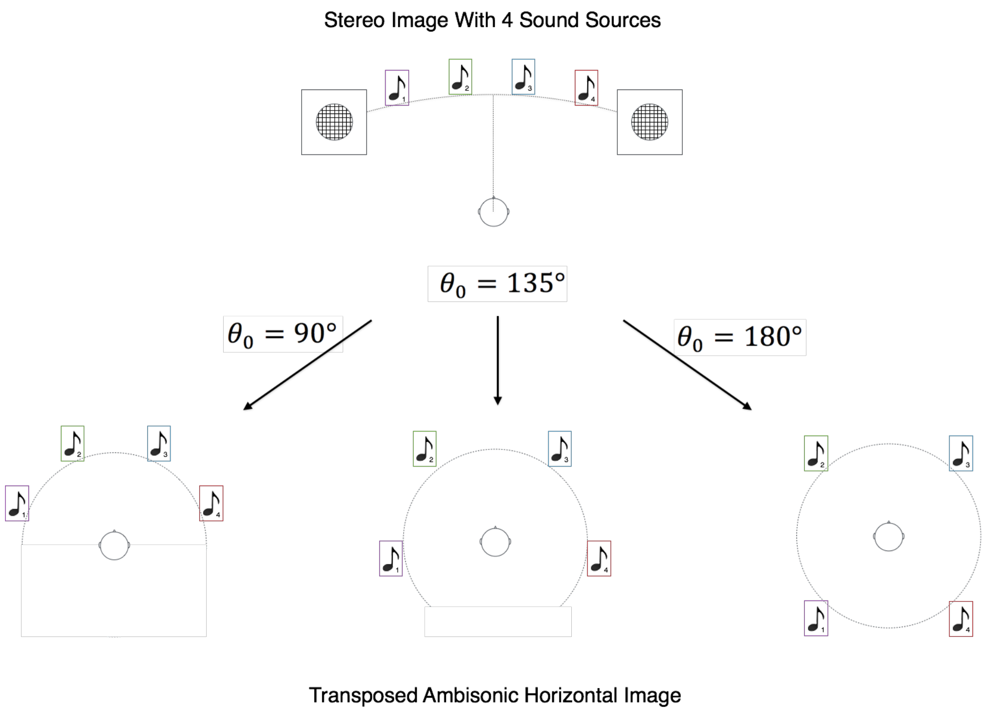

Stereo Image With 4 Sound Sources
$\theta_0 = 135°$
$\theta_0 = 90°$
$\theta_0 = 180°$
Transposed Ambisonic Horizontal Image

Figure 6.1 - Varying position of sound sources around the listener

6.3 Modified Furse-Malham Equations

Once the sound sources and their azimuth have been extracted the next step is to create a B-format signal with the sources positioned correctly. Table 4-1 shows the Furse-Malham equations for positioning a source in three dimensions using, up to, 3rd order spherical harmonics. The extracted sources need only be positioned horizontally therefore Table 4-1 can be rewritten with the elevation already known to be 0°, Table 6-1.

Label	**Order**	**Angle/Elevation Representation**
W	0	sqrt(1/2)
X	1	cos(A)
Y	1	sin(A)
Z	1	0
R	2	(1/2)(-1)
S	2	0
T	2	0
U	2	cos(2A)
V	2	sin(2A)
K	3	0
L	3	sqrt(135/256)cos(A)(-1)
M	3	sqrt(135/256)sin(A)(-1)
N	3	0
O	3	0
P	3	cos(3A)
Q	3	sin(3A)

Table 6-1 - Furse-Malham Equations For Horizontal Sources Only

6.4 Algorithm Overview

An overview of the final stereo to ambisonic up-mix algorithm is presented as a block diagram (Figure 6.2) followed by a brief explanation of each stage. The code for the algorithm can be found at 11.1.1.

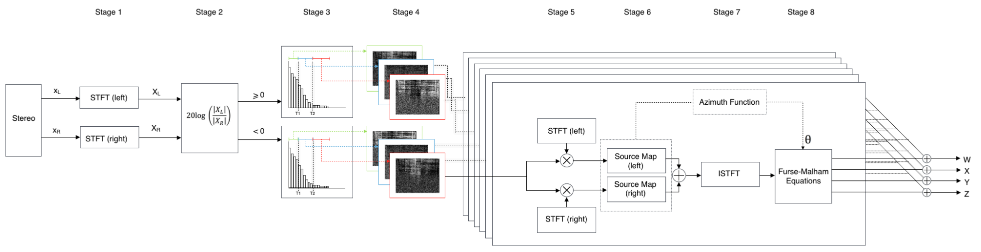


Figure 6.2 – Block diagram of proposed algorithm

The above diagram represents a stereo signal being converted to a 1st order B-format signal using a multi-thresholding technique finding two optimum thresholds. Stages 1 to 4 represent the work of (Cobos and Lopez, 2008) with some modifications (as mentioned in section 5.3.3) with stages 5 to 8 being the additional steps proposed in this work to allow ambisonic playback of the extracted sources.

Stage 1: The left and right channels of the stereo signal are transformed into the time-frequency domain though a STFT. Using a Hanning window of 180ms and an overlap of 75%. See code in 11.1.2.

Stage 2: A pan map is created of the stereo signal. Positive values represent that the given time-frequency bin is panned towards the left channel, whilst negative values represent the time-frequency bin is panned towards the right channel.

Stage 3: Normalised histograms are then made of the left and right panned material respectively. Optimum thresholds are then found using Otsu's algorithm (Otsu, 1979), in the above example two are found. See code in **Error! Reference source not found.**.

Step 4: Binary masks are created which represent each found source. The binary masks are matrices the size of the STFT where all values are either 1 or 0; they are created by using the left or right pan maps and their respective threshold values, if a time-frequency bin value falls within a range specified by the thresholds then a value of 1 is assigned to the binary mask, else a 0 is assigned.

Step 5: Multiplying a sources binary mask by the left and right STFTs will create what is described as a left and right source map. These maps contain only the STFT bins related to a single audio source and zeros out the other bins.

Step 6: The summation of the left and right source maps leads to a source map representing, in an ideal situation, all time-frequency information from one source. This summed source map can then be transformed back into the time domain via the ISTFT giving the user a separated audio source.

Step 7: Using an azimuth extraction function (explained in section 6.2) the azimuth of a chosen source can be approximated by comparing the average magnitudes of the left and right source maps. See code in 11.1.2.

Step 8: The Furse-Malham equations can be used to create a B-format from the extracted audio source and azimuth position. All B-format signals from the extracted sources can then be summed giving an overall B-format signal from the original stereo. See code in 0.

7 Testing

7.1 Source Extraction

As blind source separation by multilevel thresholding has been the chosen method for this project it is necessary to test the code written for this method. A test signal will be created and this method applied to it to allow for a direct comparison of the audio sources before mixing and the extracted audio sources. The test signal was created by mixing three separate audio sources (Open Air, 2016), these sources were a three-piece viola part (bass, tenor and treble). They were panned in separate locations of the stereo signal using constant power panning. The code for the creation of this signal is shown as follows.

```
[x, fs] = audioread('bass 3 part.wav');
y = audioread('tenor 3 part.wav');
z = audioread('treble 3 part.wav');
length = 10;
clipLength = length * fs;
x = x(1:clipLength);
y = y(1:clipLength);
z = z(1:clipLength);
t = 0:(1/fs):length-(1/fs);
t = t';
panL1 = 0.6;
panR1 = sqrt(1 - (panL1.^2));
panL2 = 0.7;
panR2 = sqrt(1 - (panL2^2));
panL3 = 0.9;
panR3 = sqrt(1 - (panL3^2));
left = (panL1 * x) + (panL2 * y) + (panL3 * z);
right = (panR1 * x) + (panR2 * y) + (panR3 * z);
stereo = [left, right];
audiowrite('testsignal.wav',stereo, fs);
```

Figure 7.1 shows the three original tracks along with their separate panning coefficients and the resultant stereo track (the test signal).

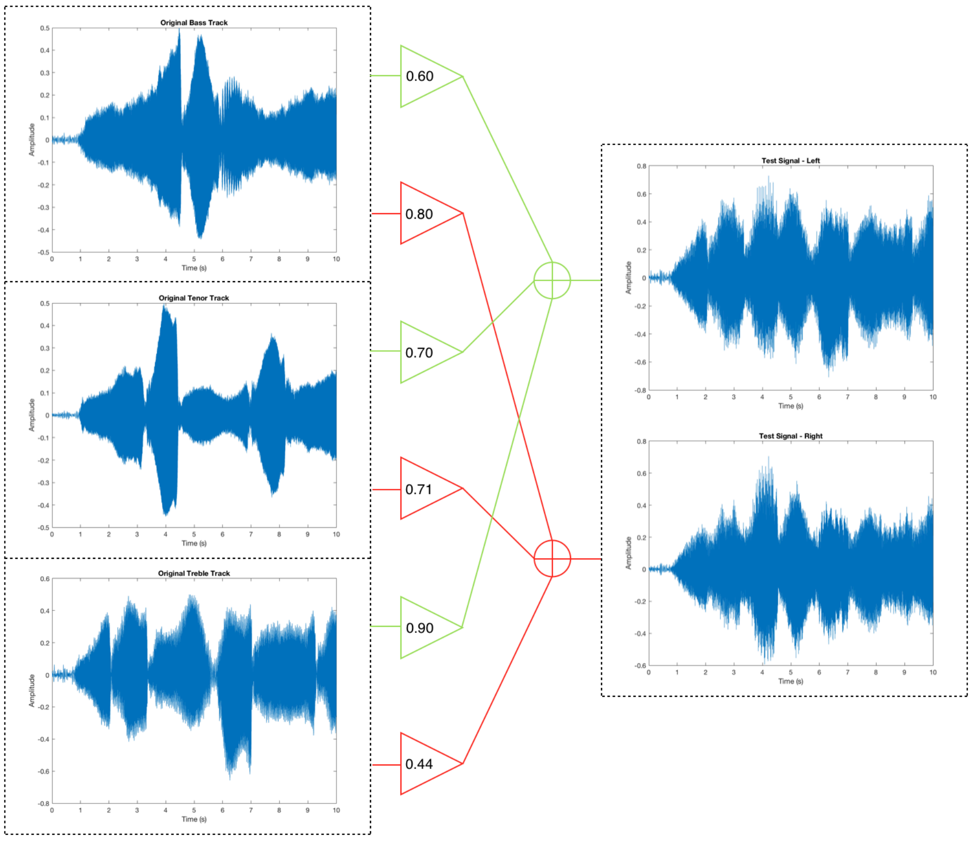


Figure 7.1 - Test signal composition

The multilevel thresholding separation technique was tested three times; with 2, 3 and 4 thresholds dividing the individual pan map histograms. The STFT were carried out using a 180ms Hanning window and 75% overlap (as Cobos and Lopez (2008) used).

As the test signal contains three unique sound sources it is expected that using two thresholds (thus dividing the audio into three sources) will produce optimum results, with three and four thresholds showing progressively worse results. However, it is important to test the effect the number of thresholds has on the extracted sources as when this algorithm is used in the overall up-mix process the number of sources in the up-mixed stereo tracks will be unknown. The left and right pan map histograms, along with their respective threshold values, are shown in Figure 7.2 to Figure 7.7.

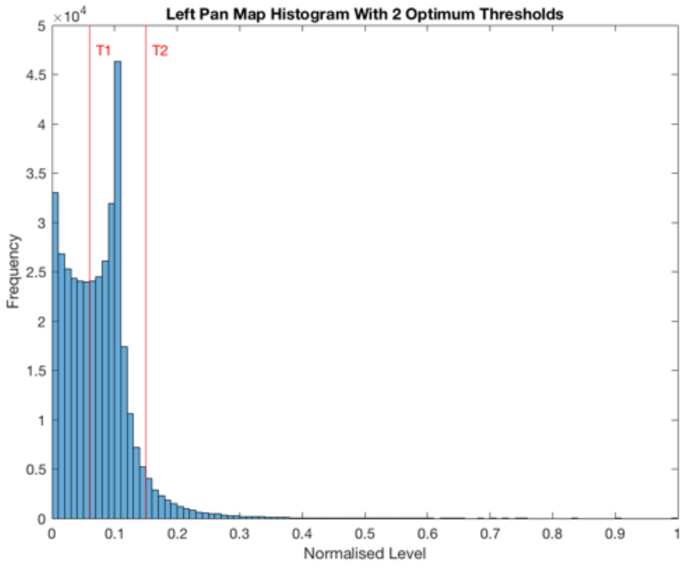


Figure 7.2 - Left histogram with 2 thresholds

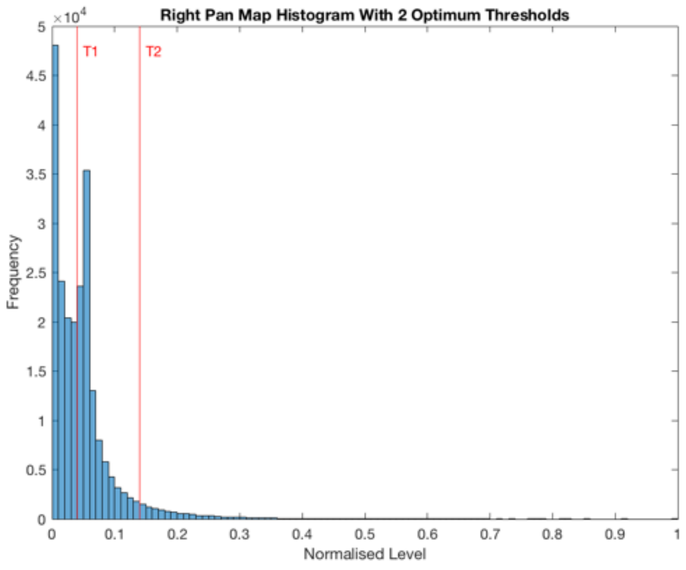


Figure 7.3 - Right histogram with 2 thresholds

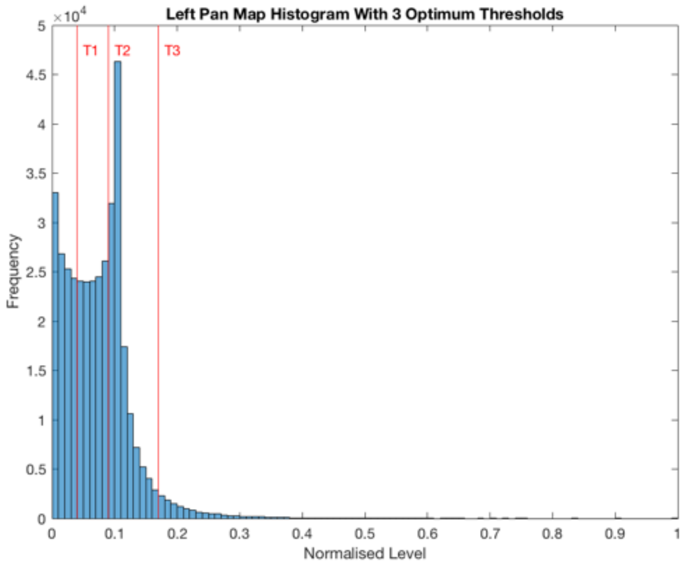


Figure 7.4 - Left histogram with 3 thresholds

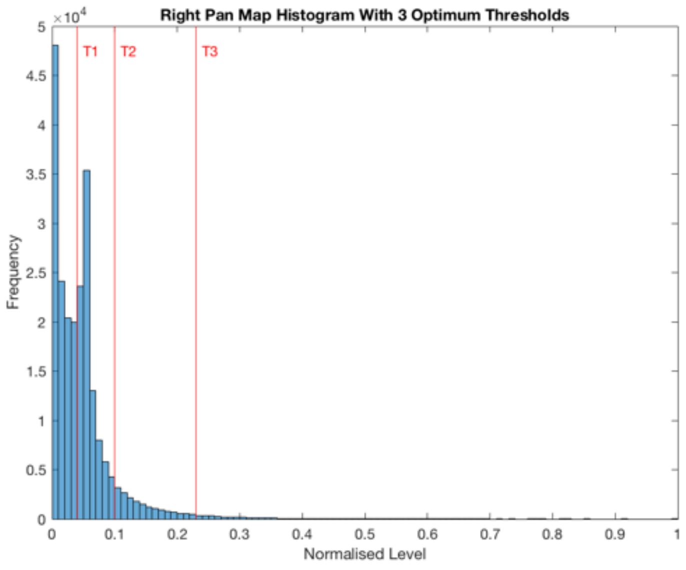


Figure 7.5 - Right histogram with 3 thresholds

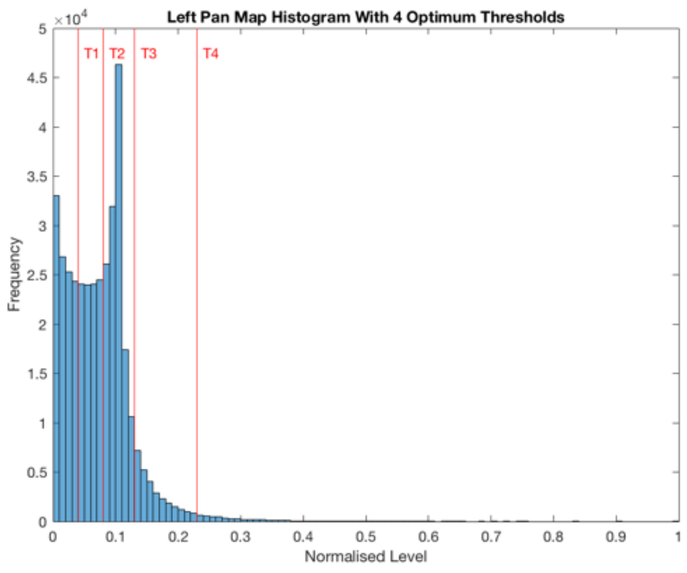


Figure 7.6 - Left histogram with 4 thresholds

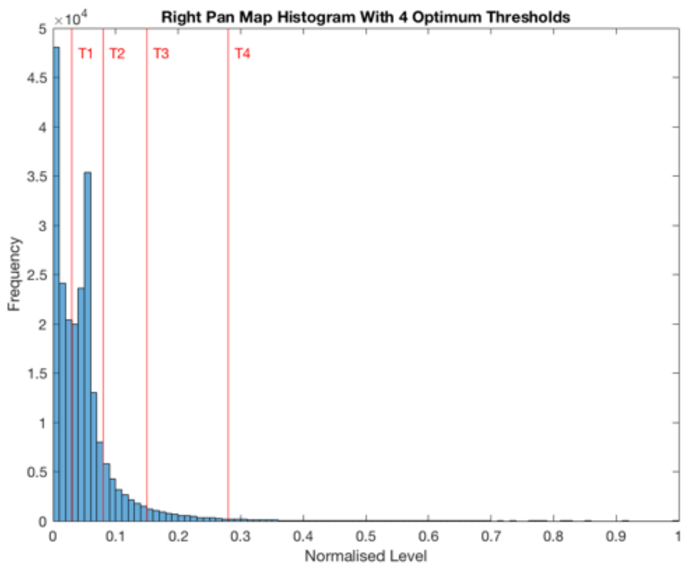


Figure 7.7 - Right histogram with 4 thresholds

Using the above graphs, multiple sources can be extracted from the test signal. The number of sources extracted from the stereo signal can be found as: $number\ of\ extracted\ source = (number\ of\ thresholds + 1) \times 2$. Therefore, for the three different tests 6, 8 and 10 sources will be extracted respectively. This is an example of a previously mentioned issue where a greater number of sources are extracted than are in the signal. This is not necessarily an issue, as everything extracted will still be panned correctly thus any incomplete sources extracted will be panned near the source they were incorrectly pulled from meaning this issue will likely go unnoticed by a listener. However, a method for identifying *true* sources has been found. It is useful to find $(xG_L)^2 + (xG_R)^2$ as this value is used in determining the azimuth of each source. This value is the sum of the average magnitude of an extracted source's left and right STFT values and is referred to, in this paper, as the sources total power. Graphs for all extracted sources total power are shown below.

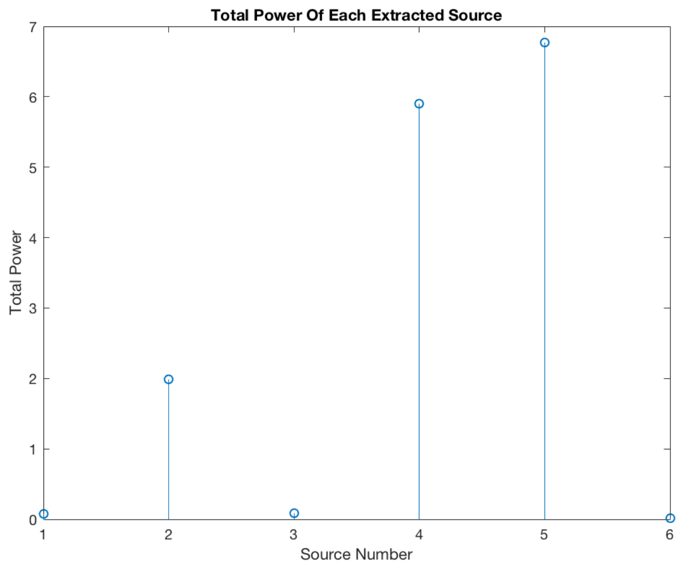


Figure 7.8 - Total power of extracted sources using 2 thresholds

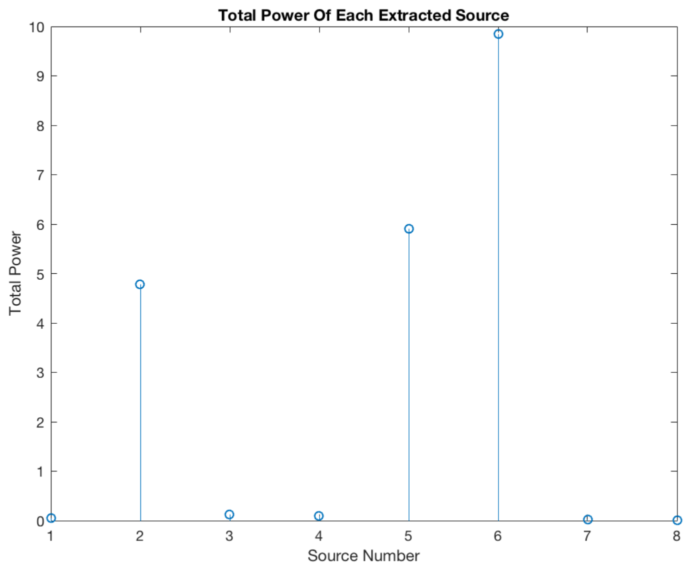


Figure 7.9 - Total power of extracted sources using 3 thresholds

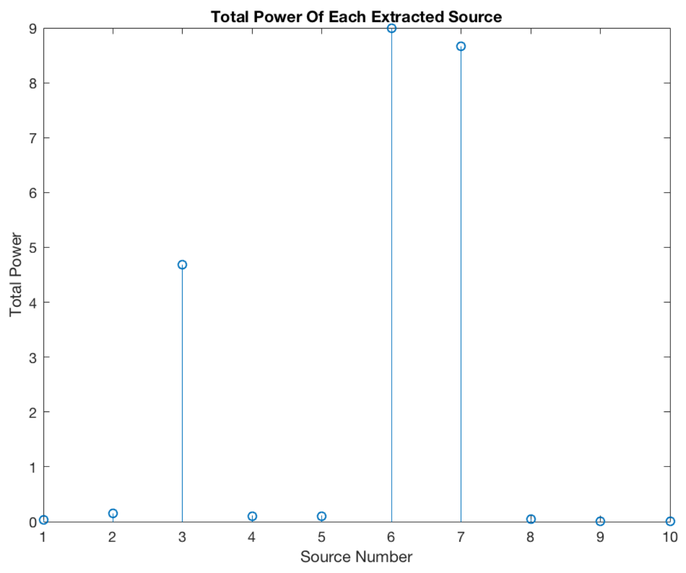


Figure 7.10 - Total power of extracted sources using 4 thresholds

Figure 7.8 to Figure 7.10 show that regardless of the number of sources extracted, the three original sources were always independently extracted as their own unique source (i.e. a larger number of thresholds didn't result in *true* sources being spread across extracted sources) and that *true* sources can be easily identified using this value.

Figure 7.11 to Figure 7.13 show the three original sources and the extracted sources for the three different tests. Although it is hard to tell how similar the signals are to the original, it is clear that the separation algorithm has managed to identify and extract the three different sources.

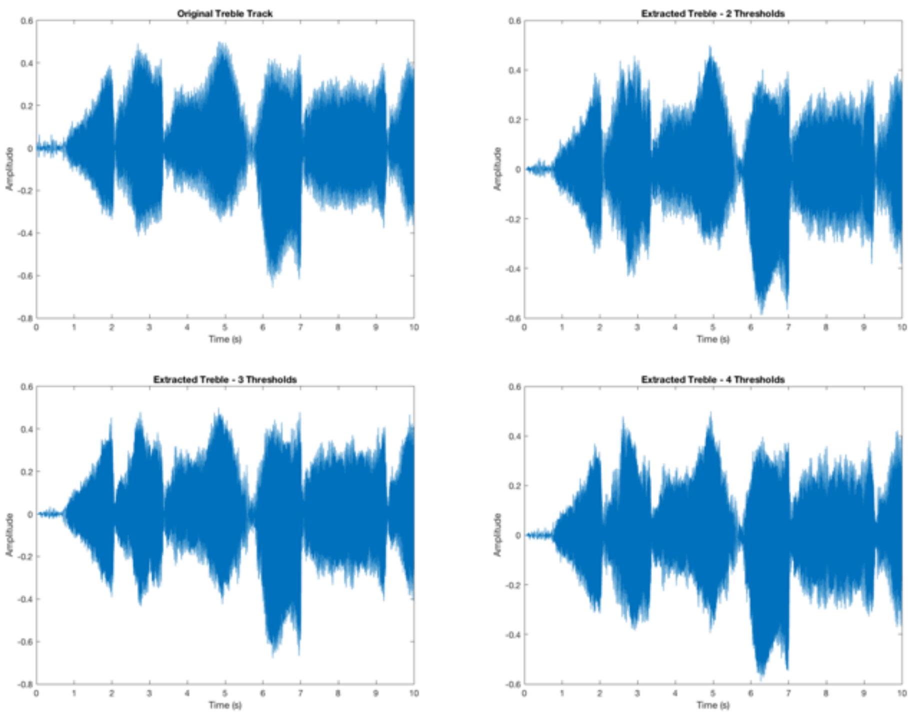


Figure 7.11 - Original and extracted treble signals

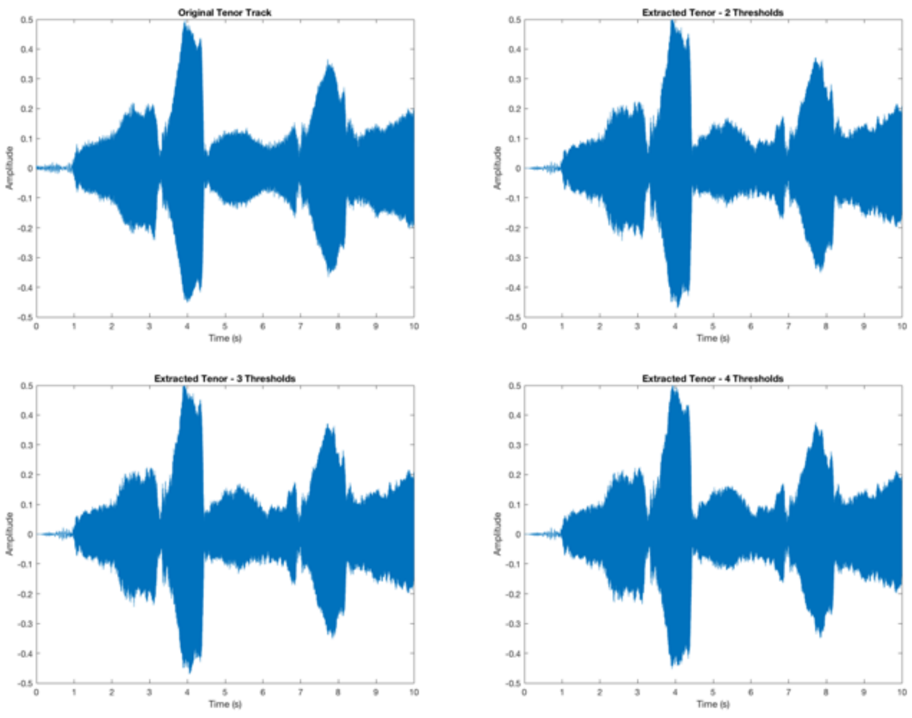


Figure 7.12 - Original and extracted tenor signals

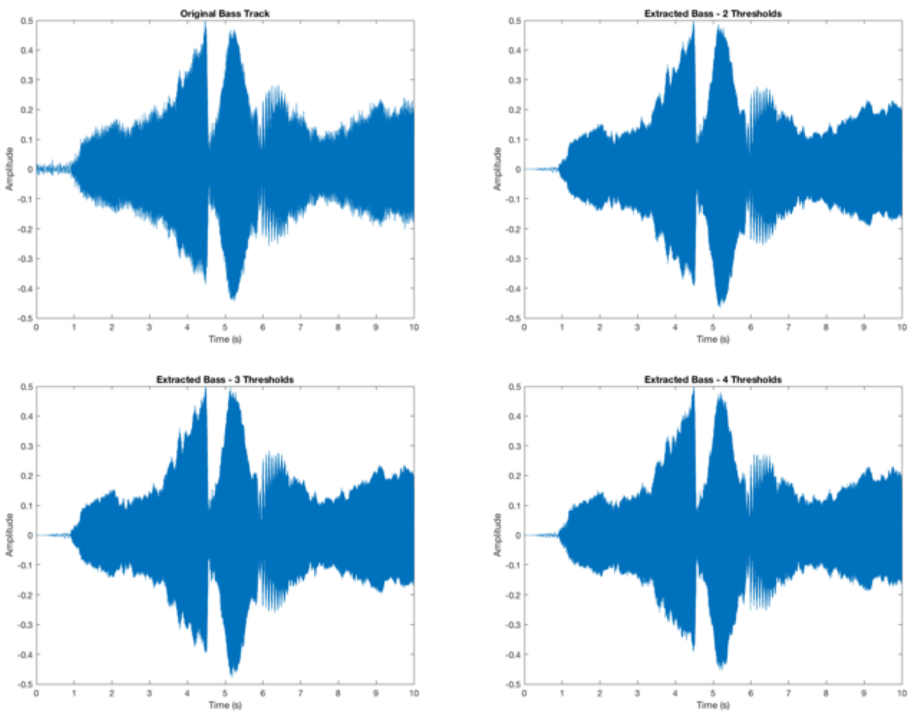


Figure 7.13 - Original and extracted bass signals

Note that the extracted sources have been normalised to the level of the original.

The original and extracted sources are compared objectively using two methods, a pairwise correlation function and a signal-to-noise calculation. The correlation was calculated using the MatLab (MathWorks, 2016) correlation function and returns a value between 0 and 1, where 1 represents two identical signals and 0 represents two completely different signals. The signal-to-noise calculation was calculated using the following equation.

$$SNR = 10\ log10\left(\frac{noise^2}{signal^2}\right)$$

Eq. 7-1

Where $signal$ is the original signal and $noise$ is the average absolute value of the extracted signal minus the original signal (note this is after the extracted signal has been normalised to the correct level but no time alignment is needed due to the separation method not introducing any latency). The signal-to-noise ratio is returned as a decibel value with a higher value meaning less noise in the extracted signal.

Along with the correlation and signal-to-noise values, the left and right gain values were calculated for all signals and have been compared in Table 4-1.

			Number Of Thresholds Used			
			2	3	4	Original Source
	Time Taken For Calculation (s)		7.55	9.98	11.03	
Extracted Source	Treble	Gain (L)	0.8945	0.9014	0.8977	0.9000
		Gain (R)	0.4471	0.4329	0.4406	0.4359
		Correlation	0.9780	0.9621	0.9613	
		SNR (dB)	20.76	21.07	20.97	
	Tenor	Gain (L)	0.6956	0.6956	0.6975	0.7000
		Gain (R)	0.7185	0.7185	0.7166	0.7141
		Correlation	0.9848	0.9848	0.9855	
		SNR (dB)	20.36	20.36	20.37	
	Bass	Gain (L)	0.5982	0.5995	0.6023	0.6000
		Gain (R)	0.8014	0.8004	0.7982	0.8000
		Correlation	0.9884	0.9882	0.9876	
		SNR (dB)	20.36	20.38	20.39	

Table 7-1 - Extracted Source Gains, Correlation and SNR for 2, 3 and 4 Threshold

All audio (both the original and extracted samples) are available on the DVD provided with this thesis (see 11.2).

7.2 Source Movement

A 13 second stereo signal was created by panning a mono audio source from right to left (using constant power panning). The test signal was created as shown below.

```
 [y, fs] = audioread('tenor 3 part.wav');
length = 13;
clipLength = length * fs;
y = y(1:clipLength);

pan = 90/clipLength:(90/clipLength):90;

panL1 = cosd(pan);
panR1 = sind(pan);

left = (panL1' .* y);
right = (panR1' .* y);

stereo = [left, right];

audiowrite('movingTestSignal.wav',stereo, fs);
```

Initially this test signal led to an error whilst being converted through the proposed algorithm. Through debugging this was found to be due to the method used for determining thresholds in the left and right pan map histograms. Figure 7.14 shows the right pan map histogram of the first time instance of the test signals STFT.

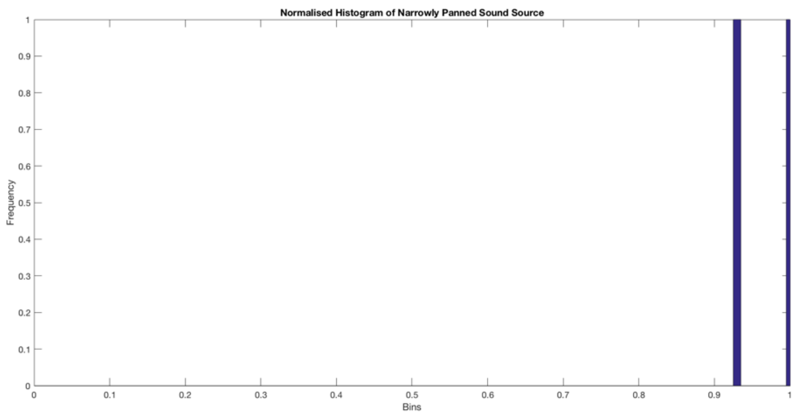


Figure 7.14 - Pan map histogram for narrowly panned single source

Here it can be seen that the right channel only possesses energy in a very narrow region. It is impossible to find more than two optimum thresholds in this instance due to there being too few bins possessing non-zero values. This problem will always occur when the number of thresholds to be found is larger than the number of non-zero valued histogram bins. A fix to this problem is to pre-allocate thresholds evenly across a histogram and then overwrite a threshold if an optimum threshold has been found, shown in Figure 7.15.

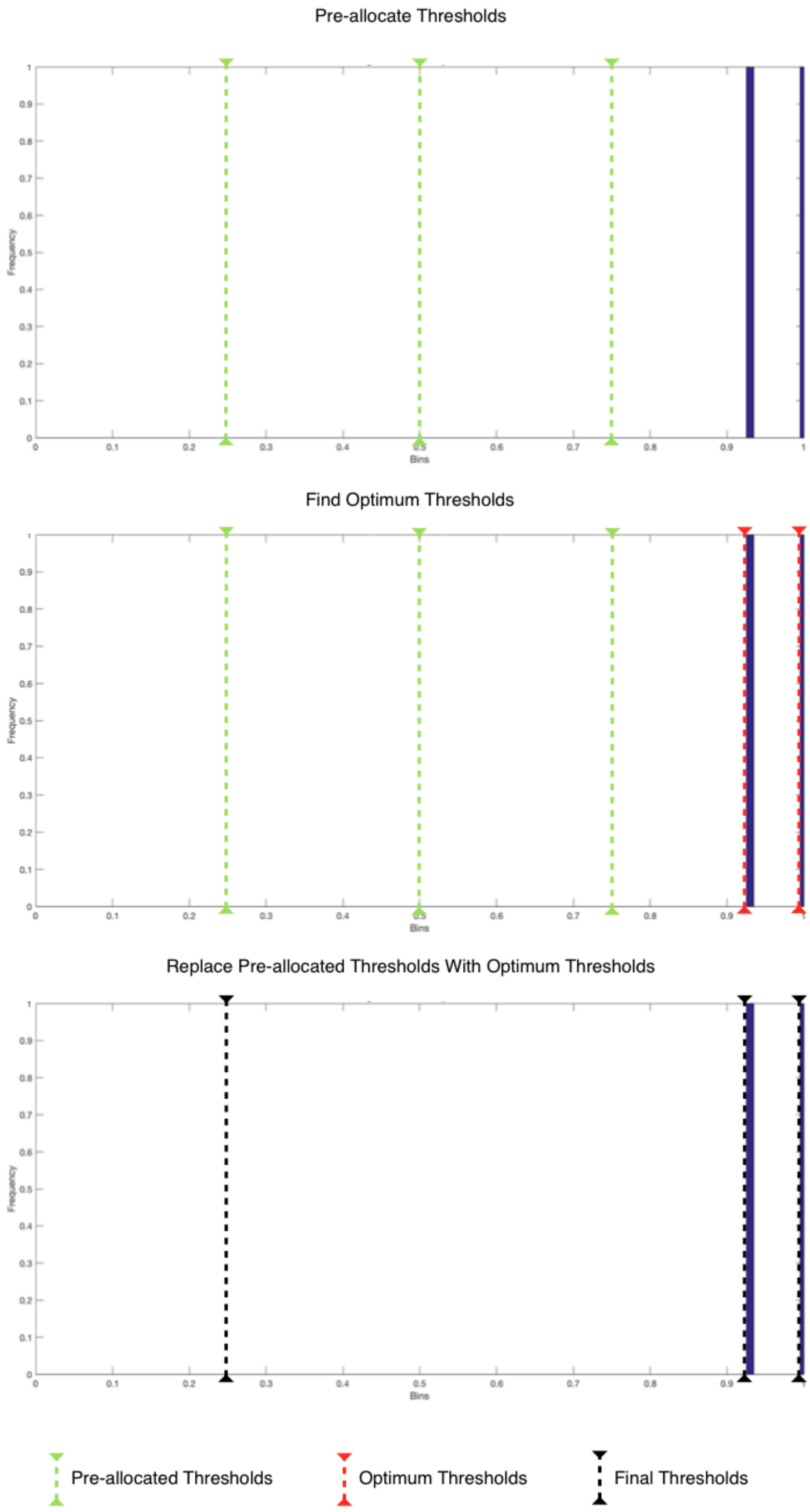


Figure 7.15 - Pre-allocating and overwriting thresholds

This solution stops the above mentioned error and also allows for silence in a track (in either one or both channels) to be processed. When used on commercial stereophonic tracks this method works well however when used on the test signal notable 'jumps' in artefacts in the audio are noticed upon playback, although the main signal can be clearly heard to move from right to left. By mapping the azimuth of the extracted sources a clearer picture of why this is happening can be seen, Figure 7.16.

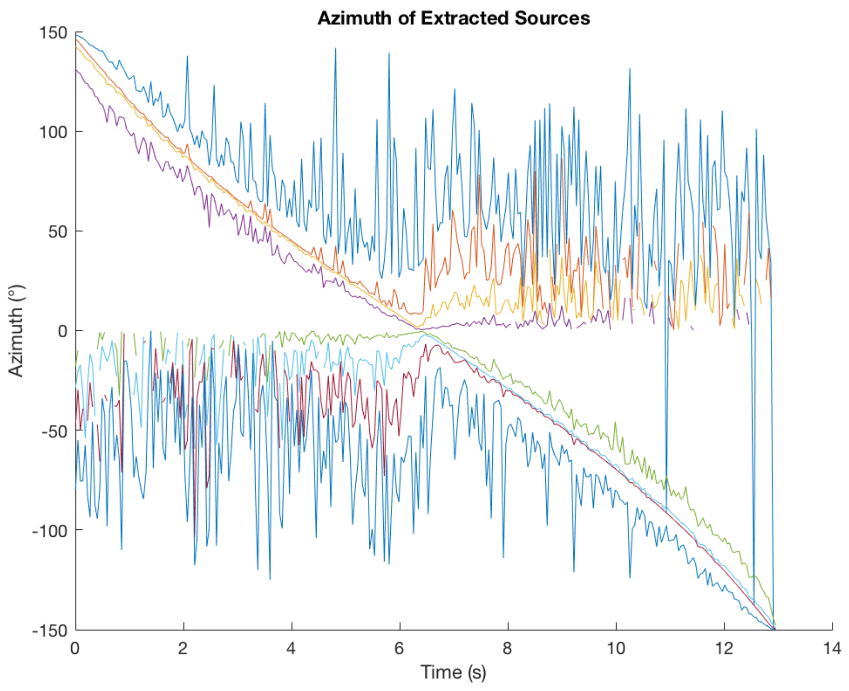


Figure 7.16 - Azimuth extraction of test signal

The above clearly shows a sound source being tracked from +150° to -150° (as the algorithm was set to a width of ±150°). However, an issue is clearly causing the others sources to show a random and sporadic azimuth. This issue is caused by a previously mentioned problem (section 5.3.3.5), there a more sources being extracted by the algorithm than there are in the signal thus artefacts from the signal are being extracted as sources. The differences in level between artefacts in the left and right channels will be random and not constant, thus show as a constantly and dramatically changing azimuth. A proposed fix to this issue is to use the property found in section 6.2, that *real* sources have a far greater total power (sum of energy in the left and the right STFT). The mean total power of sources extracted was calculated and any source whose total power fell below this mean was given the same azimuth value as the closest (azimuthally) *true* source. The result of this fix is shown in Figure 7.17.

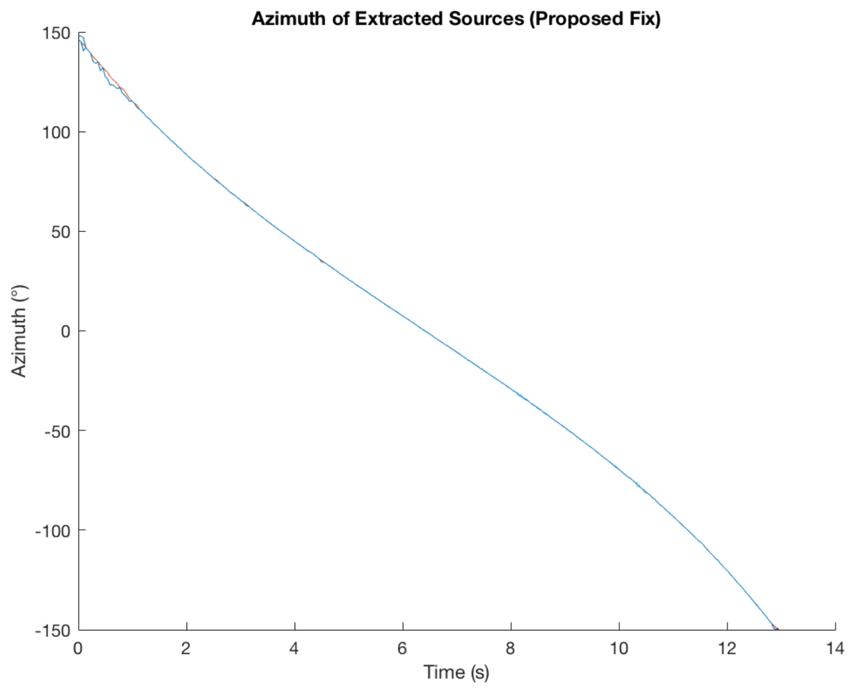


Figure 7.17 - Azimuth extraction of test signal using proposed fix

Not only is the improved method shown to work but it also highlights how accurately the overall algorithm can track a sources azimuth.

When used on commercial stereophonic audio the following figure can be made.

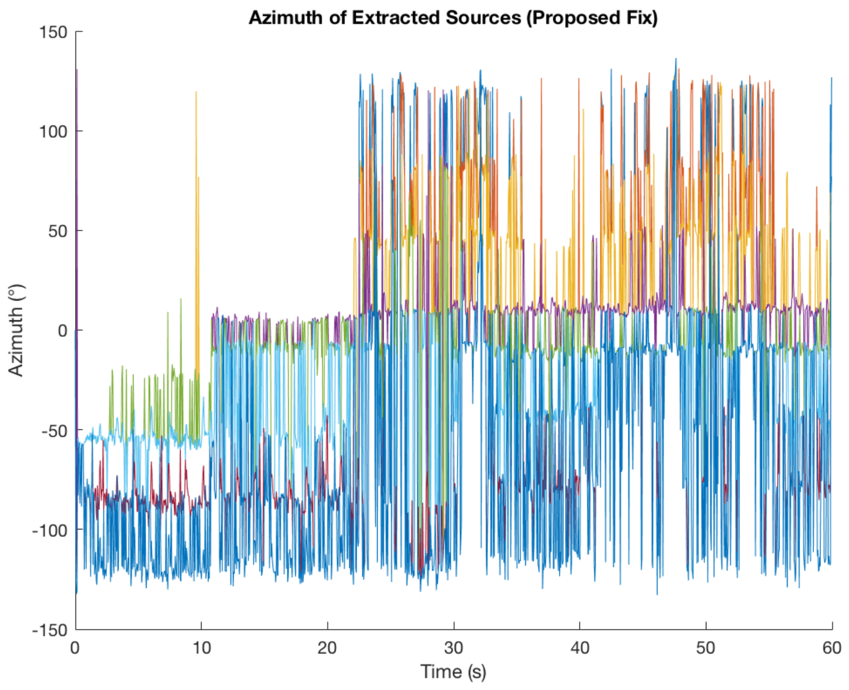


Figure 7.18 - Azimuth extraction of a commercial stereophonic signal

Although Figure 7.18 shows heavy fluctuation in source azimuth, dark regions (highlighted in Figure 7.19) show where sources have been extracted and placed.

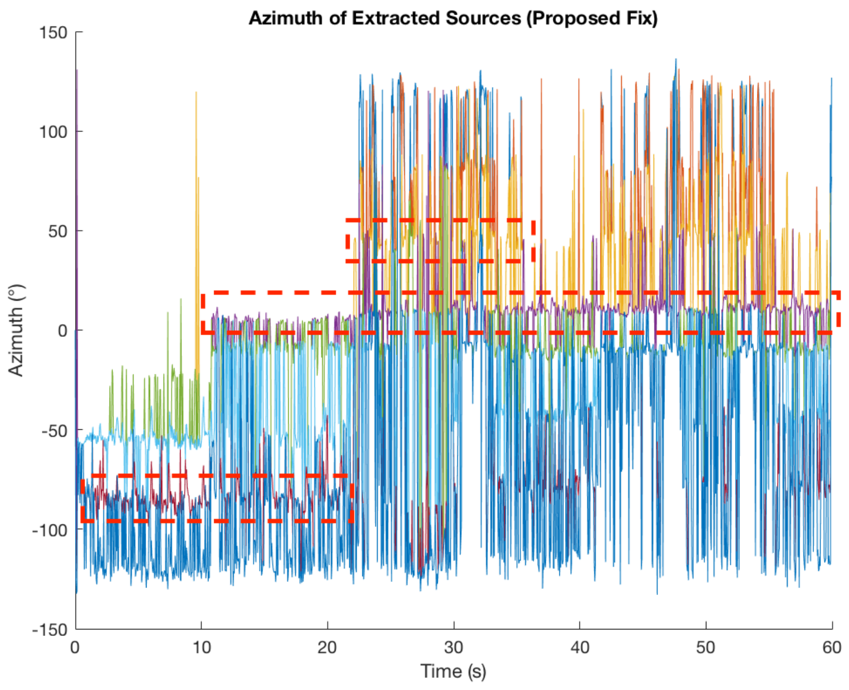


Figure 7.19 - Source identification through extracted azimuth

When viewing this graph and listening to the audio it is noticed that these dark regions match up both in time and position to the contents of the audio (i.e. at ~11 seconds a centre panned bass is introduced).
Although the proposed fix works well for the test signal and further proves that the algorithm is extracting sources and their azimuth correctly (by the dark regions in Figure 7.19), when a B-format signal of a commercial stereophonic song is produced using this method the sources appear to 'jump' horizontally therefore leading to the previous method being preferred.
Code for both methods are shown in appendix 11.1.2 (for current) and 11.1.4 (proposed fix) and the audio for the test signal pre and post fix is made available on the files provided with this thesis (see 11.2).

8 Discussion

The advantages and disadvantages of the final proposed algorithm will be discussed in this section.

8.1 Advantages

- The algorithm is capable of converting any standard commercial stereophonic signal to up to 3rd order B-format.
- It is shown that sound sources within the audio are extracted with a high degree of accuracy and, for constant power panned sources, their azimuth will be also be extracted accurately.
- The user is able to define the horizontal width of the outputted signal.

8.2 Disadvantages

- When used with a signal with few sources noise can be picked up as a source and panned irregularly.
- The panning extraction is designed specifically for constant power panning; other panning laws will still work but not as accurately.
- No height information is outputted.

9 Conclusion

This thesis aimed at creating an algorithm for the up-mix of stereophonic audio to B-format, for Ambisonic playback. The background theory of Ambisonics was reviewed and the need for this up-mix algorithm justified through the recent resurgence of interest in Ambisonics. Background fundamentals of stereophonic audio were presented with an assumption made that the majority of commercial audio is produced using a Constant Power panning law. The benefits and limitations of working in the time-frequency domain, through the use of the short-time Fourier transform, were also analysed with the proposed algorithm doing all signal processing in this domain.

Three methods for source extraction of stereophonic signals were considered and a technique using Otsu's multilevel thresholding was chosen. Testing this technique, along with a newly proposed azimuth extraction method, showed the ability to extract sources and their panned positions with a high degree of accuracy. When combined with the use of an adapted Furse-Malham set (where elevation is predefined as zero), the overall algorithm was tested and improved using a specific test signal designed to pan from hard right to hard left. Performing informal listening tests of commercial audio processed through this algorithm showed positive results.

9.1 Future Work

The clear next step for this project would be to conduct formal listening tests. Audio created specifically in B-format could be tested against its equivalent stereophonic mix being processed through the up-mix algorithm (i.e. three sources panned in B-format tested against an up-mixed B-format signal from a stereophonic signal which *should* produce the same source azimuth positions). It would also be interesting to test the algorithm in an audio-visual context; up-mixing the stereophonic format of a film and comparing that to a 5.1 release (although there is a danger of comparing listener preference of 5.1 compared to a larger Ambisonic array).

Implementation of the algorithm in real-time would allow for a much wider usage. Considering a live sound context, having to process all audio before an event would likely cost too much time for a venue. However, allowing real-time processing of any chosen audio for the venue would make the algorithm, and Ambisonics in general, more accessible for live sound events.

Height has not been considered in this proposed algorithm due to stereophonic mixes only possessing horizontal information. However, height information could be added to the audio. A possibility would be to add a filter to effectively 'spread' the frequency spectrum vertically. A combination of the tested separation methods could yield improved results; by extracted direct and diffuse material and then performing Otsu's thresholding technique to only the direct signals could lead to improved accuracy in the source and azimuth extraction whilst the diffuse material could be de-correlated and spread across a multichannel array.

10 References

Anton, H. and Rorres, C (2010) **Elementary Linear Algebra**. 10th Edition. New York: John Wiley.

Arfken, G. (1985) **Mathematical Methods for Physicists**. Boston: Academic Press

Bateman, A. and Paterson-Stephens, I. (2001) **The DSP Handbook, Algorithms, Applications and Design Techniques.** Prentice Hall. Harlow

BBC (2011a) **Audio Research Partnership.** BBC [online] Available at: http://www.bbc.co.uk/rd/projects/audio-research-partnership

BBC (2011b) **Surround Sound with Height**. BBC [online] Available at: http://www.bbc.co.uk/rd/projects/surround-sound-with-height

Becker, F. and Bernard, B. (2016) **Stereo Panning Law Remastering Algorithm based on Spatial Analysis.** Audio Engineering Society 140th Convention. Paper 9523

Blue Ripple Sound (2015) **TOA Core VST** [online] Available at: http://www.blueripplesound.com/products/toa-core-vst (accessed: 2 Sept. 2016)

Blue Ripple Sound (2015b) **HOA Technical Notes – B-Format** [online] Available at: http://www.blueripplesound.com/b-format (accessed: 2 Sept. 2016)

Blumlein, A. (1931) **Improvements in and relating to Sound-transmission, Sound-Recording and Sound-Reproducing Systems**. British Patent Application 394325.

Bonarenko, V (2009) **drawLA - Draw Toolbox for Linear Algebra**. MathWorks, File Exchange [Computer Program] Available at: http://uk.mathworks.com/matlabcentral/fileexchange/23608-drawla-draw-toolbox-for-linear-algebra

Cobos, M. and Lopez, J. (2008) **Stereo Audio Source Separation Based on Time-Frequency Masking and Multilevel Thresholding**. Digital Signal Processing 18 (960-976)

Columbia (2012) **History of Processor Performance.** Columbia University [online] Available at: http://www.cs.columbia.edu/~sedwards/classes/2012/3827-spring/advanced-arch-2011.pdf

Daubney, Chris (1982) **Ambisonics – An Operational Insight.** Studio Sound.

Faller, C. and Menzer, F. (2010) **Stereo-to-Binaural Conversion Using Interaural Coherence Matching.** Audio Engineering Society 128th Convention. Paper 7986

Farina, A. et al. (2001) **Ambiophonic Principles for the Recording and Reproduction of Surround Sound for Music**. Proceedings of the 19th AES International Conference of Surround Sound, Schloss Elmau, Germany, p. 26- 46.

Fuerst, W. and Lynn, P. (1998) **Introductory Digital Signal Processing with Computer Applications.** 2nd Edition. Chichester: John Wiler

Funktion-One (2016) **Funktion One's John Newsham Chats to I Like Music.** Funktion-One [online] Available at: http://www.funktion-one.com/dl/files/2054.pdf (accessed: 2 Sept. 2016)

Gerzon, M. A. (1974a) **Sound Reproduction Systems**. Patent No. 1494751.

Gerzon, M. A. (1974b) **What's Wrong with Quadrophonics**. Studio Sound.

Gerzon, M. A. (1985) **Ambisonics in Multichannel Broadcasting and Video**. Journal of the Audio Engineering Society. Vol. 33, No. 11

Gerzon, M. A. (1992) **General Metatheory of Auditory Localisation**. Audio Engineering Society, 92nd Convention.

Gerzon, M. A. and Barton, G. J. (1992) **Ambisonic Decoders for HDTV**. Proceedings of the 92nd International AES Convention, Vienna. 24 – 27 March. Preprint 3345.

Goldman Sachs (2016) **The Real Deal with Virtual and Augmented Reality** [online] Available at: http://www.goldmansachs.com/our-thinking/pages/virtual-and-augmented-reality.html?cid=PS_01_04_07_00_01_16_01&mkwid= (accessed: 2 Sept. 2016)

Google Acquisitions (2015) **Thrive Audio** [online] Available at: http://google-acquisitions.silk.co/page/Thrive-Audio (accessed: 8 Mar. 2016)

Google (2016a) **Spatial Media**. GitHub. [online] Available at: https://github.com/google/spatial-media/issues (accessed: 2 Sept. 2016)

Google (2016b) **Spatial Audio API.** Google VR. [online] Available at: https://developers.google.com/vr/android/ndk/reference/group/audio (accessed: 2 Sept. 2016)

Griesinger, D (2002) **Stereo and Surround Panning in Practice.** Audio Engineering Society 112th Convention.

ITU (1997) **Methods for the Subjective Assessment of Small Impairments in Audio Systems Including Multichannel Sound Systems.** International Telecommunications Union, Geneva, Switzerland. ITU-R BS.1116.

Kraft, S. and Zolzer, U. (2015) **Stereo Signal Separation and Up Mixing By Mid-Side Decomposition in the Frequency-Domain.** Proceedings of the international conference on digital audio effects.

Kronlachner, M (2014a) **Spatial Transformations for the Alteration of Ambisonic Recordings.** Masters Thesis. Graz University of Technology

Kronlachner, M (2014b) **Ambisonic Plugin Suite** [online] Available at: http://www.matthiaskronlachner.com/?p=2015 (accessed: 2 Sept. 2016)

Laitinen, M. (2014) **Converting Two-Channel Stereo Signals to B-format for Directional Audio Coding Reproduction.** Audio Engineering Society. 137th Convention. 9127

Laitinen, M. and Pulkki, V. (2011) **Converting 5.1 Audio Recording to B-format for Directional Audio Coding Reproduction**. IEEE International Conference on Acoustics, Speech and Signal Processing. Prague, Czech Republic.

Liao, P. Chen, T. and Chung, P. (2001) **A Fast Algorithm For Multilevel Thresholding**. J. Inform. Sci. Eng. 17. P. 713–717.

MathWorks (2016) **MatLab R2016a**. [Computer Program] Available at: http://uk.mathworks.com/products/matlab/ (accessed: 2 Sept. 2016)

Marston, D. (2011) **Assessment of Stereo to Surround Sound Upmixers for Broadcasting**. Audio Engineering Society 130th Convention. Paper 8448

National Instruments (2015) **Understanding FFTs and Windowing.** White Papers. [online] Available at: http://www.ni.com/white-paper/4844/en/#toc2 (accessed: 2 Sept. 2016)

Noisternig, M. et al. (2003) **A 3D Ambisonic Based Binaural Sound Reproduction System**. Proceedings of the 24th International Conference on Multichannel Audio, Banff, Canada.

Open Air (2016) **Piece for Three Viols Presented as Three Solo Tracks for Multitracking (back mic).** Anechoic Data. [online] Accessed at: http://www.openairlib.net/anechoicdb/content/piece-three-viols-presented-three-solo-tracks-mulitracking-back-mic (accessed: 2 Sept. 2016)

Otsu, N (1979) **A Threshold Selection Method from Grey-Level Histogram.** IEEE Trans. System Man Cybernet. SMC-9 (1) 62–66.

PR Newswire (2014) **VisiSonics' RealSpace 3D Audio Software Licensed by Oculus for Virtual Reality.** [online] Available at: http://www.prnewswire.com/news-releases/visisonics-realspace-3d-audio-software-licensed-by-oculus-for-virtual-reality-278413231.html (accessed: 2 Sept. 2016)

Pulkki, V (1997) **Virtual Sound Source Positioning using Vector Based Amplitude Panning**. Journal of the Audio Engineering Society, Vol. 45, No.6.

Pulkki, V (2006) **Directional Audio Coding in Spatial Sound Reproduction and Stereo Upmixing**. Audio Engineering Society 28th International Conference.

Reaper (2016) **Reaper: Digital Audio Workstation** [online] Available at: http://www.reaper.fm/) (accessed: 2 Sept. 2016)

Van Veen, B (2013) **Short-time Fourier Transform and the Spectrogram**. YouTube [online] Available at: https://www.youtube.com/watch?v=NA0TwPsECUQ (accessed: 2 Sept. 2016)

Wiggins, B. (2004) **An Investigation into the Real-Time Manipulation and Control of Three-Dimensional Sound Fields**. Doctoral Thesis. University of Derby

Wiggins, B. (2008) **Has Ambisonics Come of Age?** Proceedings of the Institute of Acoustics.Vol.30 Pt.6

Wiggins (2010) **WigWare** [online] Available at: http://www.brucewiggins.co.uk/?page_id=78 (accessed: 2 Sept. 2016)

Williams, E. (1999) **Fourier Acoustics**. Academic Press.

Zhivomirov, H (2015) **Short-Time Fourier Transformation (STFT) with Matlab Implementation**. MathWorks, File Exchange [Computer Program] Available at: https://uk.mathworks.com/matlabcentral/fileexchange/45197-short-time-fourier-transformation--stft--with-matlab-implementation (accessed: 2 Sept. 2016)

11 Appendix

11.1 MatLab Code

11.1.1 Stereo to Ambisonic Algorithm

```
% ********************* Stereo To Ambisonic Converter *******************

% *********************      User Variables       *******************
[x, fs] = audioread('Crash Into Me.m4a');

clipLength = 2;       % in seconds - LEAVE 0 FOR WHOLE SONG
separation = 300;     % output horizontal width (i.e 300∞ = +/- 150∞)
order = 3;            % ambi order

outputFileName = 'movingTestSignalNoDebug-3Ord-BFormat.wav';

% *********************     Algorithm Variables     *******************

stftWindow = 0.180;    % in seconds
overlap = 75;          % in percent
zeroPadSize = 1;       % multiples of stftWindow
lowBandFreq = 100;
highBandFreq = 4000;
thresNum = 3;          % max 4
edges = 0:0.01:1;      % histogram edges

% ********************* Stereo To Ambisonic Converter *******************



wb = waitbar(0,'Please wait...');

% convert seconds to samples and limit
clipLength = clipLength * fs;
if clipLength <= 0 || clipLength > length(x)
   clipLength = length(x);
```

```
end

% set up and perform STFT's
wlen = ceil(stftWindow * fs);
zeroPadSize = ceil(zeroPadSize * wlen);
nfft = wlen + zeroPadSize - 1;
nfft = 2 ^ nextpow2(nfft);
h = ceil((1 - (overlap/100)) * wlen);
x = x(1:clipLength,:);
leftX = x(:,1);
rightX = x(:,2);
[left, fbins, tbins] = stft2(leftX, wlen, h, nfft, fs, zeroPadSize);
right = stft2(rightX, wlen, h, nfft, fs, zeroPadSize);
tbins = size(tbins,2);

% set up a zero'd output matrix
outputLength = ceil(( h .* (tbins-1) ) + nfft);
ambiChannelNos = (order + 1)^2;
output = zeros(ambiChannelNos, outputLength);

% calculate and band pan maps
[leftPanMap, rightPanMap, panMap] = panMapFunc(left, right);
freqStep = fs/nfft;
Kmin = ceil(lowBandFreq/freqStep);
Kmax = ceil(highBandFreq/freqStep);
leftPanMapBand = leftPanMap(Kmin:Kmax,:);
rightPanMapBand = rightPanMap(Kmin:Kmax,:);

sourceNum = thresNum + 1;
b = 0;

[freqBins, timeBins] = size(leftPanMap(:,1));
both = left + right;

% initialise arrays to be refilled in loop for speed
sourceMap = zeros(freqBins, timeBins, (2*sourceNum));
audioMap = zeros(size(sourceMap));
wbStep = round(tbins/50);
preOutput = zeros(sourceNum*2, nfft);
```

```
leftThres = zeros(1,thresNum);
rightThres = leftThres;
leftThresPre = leftThres;
rightThresPre = leftThres;
leftHist = zeros(1,(length(edges)-1));
rightHist = leftHist;
leftPanMapCurrent = zeros(size(leftPanMap(:,1)));
rightPanMapCurrent = leftPanMapCurrent;

for j = 1:tbins

    leftPanMapBandCurrent = leftPanMap(:,j);
    rightPanMapBandCurrent = rightPanMap(:,j);

    % normalise pan maps
    weight = max(leftPanMapBandCurrent(:));
    if weight ~= 0
        leftPanMapBandNorm = leftPanMapBandCurrent ./ weight;
    else
        leftPanMapBandNorm = leftPanMapBandCurrent;
    end
    weight = max(rightPanMapBandCurrent(:));
    if weight ~= 0
        rightPanMapBandNorm = rightPanMapBandCurrent ./ weight;
    else
        rightPanMapBandNorm = rightPanMapBandCurrent;
    end

    % calculate histograms and thresholds
    leftPanMapBandNorm = leftPanMapBandNorm(leftPanMapBandNorm~=0);
    rightPanMapBandNorm = rightPanMapBandNorm(rightPanMapBandNorm~=0);
    leftHist = histcounts(leftPanMapBandNorm,edges);
    rightHist = histcounts(rightPanMapBandNorm,edges);
    leftThresPre = fastMultiLevelThresholdFunc(leftHist,thresNum);
    rightThresPre = fastMultiLevelThresholdFunc(rightHist,thresNum);

    for i = 1:thresNum
        leftThres(1,i) = ((edges(leftThresPre(i)+1) + edges(leftThresPre(i))) /2) .* max(leftPanMapBand2(:));
        rightThres(1,i) = ((edges(rightThresPre(i)+1) + edges(rightThresPre(i))) /2) .* max(rightPanMapBand2(:));
```

```
  end

  % divide source maps into binary source masks using found thresholds

  for i = 1:sourceNum
    if i == 1
      sourceMap(:,:,sourceNum) = (leftPanMap(:,j) < leftThres(i) & leftPanMap(:,j) ~= 0);
      sourceMap(:,:,(sourceNum+1)) = (rightPanMap(:,j) < rightThres(i) & rightPanMap(:,j) ~= 0);
    elseif i == sourceNum
      sourceMap(:,:,1) = (leftPanMap(:,j) >= leftThres(i-1));
      sourceMap(:,:,(2*sourceNum)) = (rightPanMap(:,j) >= rightThres(i-1));
    else
      sourceMap(:,:,(sourceNum+1-i)) = (leftPanMap(:,j) >= leftThres(i-1) & leftPanMap(:,j) < leftThres(i));
      sourceMap(:,:,(sourceNum+i)) = (rightPanMap(:,j) >= rightThres(i-1) & rightPanMap(:,j) <
rightThres(i));
    end
  end

  % convert binary masks back to audio

  for i = 1:(sourceNum*2)
    audioMap(:,:,i) = sourceMap(:,:,i) .* both(:,j);
  end

  % calculate azimuth, create B-Format signal and overlap-add

  azimuth = aziFunc(sourceMap, left(:,j), right(:,j), separation);

  for i = 1:(sourceNum*2)
    preOutput(i,:) = istft(audioMap(:,:,i), h, nfft, fs);
    output(:,((b+1):(b+nfft))) = output(:, ((b+1):(b+nfft))) + sourceToAmbiFunc(preOutput(i,:), azimuth(i),
order);
  end

  b = b + h;

  % update waitbar
```

```
    if rem(j,wbStep) == 0
        waitbar(j / tbins)
    end

end

output = output';

waitbar(j / tbins, 'Creating Audio File');
audiowrite(outputFileName,output,fs);

close(wb)
```

11.1.2 STFT Function

The following is the work of Zhivomirov (2015), adapted to use a Hanning window and allow for zero-padding.

```
function [stft, f, t] = stft2(x, wlen, h, nfft, fs, zeroPadSize)


% function: [stft, f, t] = stft(x, wlen, h, nfft, fs)
% x - signal in the time domain
% wlen - length of the hanning window
% h - hop size
% nfft - number of FFT points
% fs - sampling frequency, Hz
% f - frequency vector, Hz
% t - time vector, s
% stft - STFT matrix (only unique points, time across columns, freq across rows)


% represent x as column-vector if it is not
if size(x, 2) > 1
    x = x';
end


% length of the signal
% xlen = length(x);


% ***************************************************************************


x = padarray(x, [zeroPadSize, 0], 0, 'post');


xlen = length(x);


% ***************************************************************************


% form a periodic hamming window
win = hanning(wlen, 'periodic');


% form the stft matrix
rown = ceil((1+nfft)/2);            % calculate the total number of rows
```

```
coln = 1+fix((xlen-wlen)/h);        % calculate the total number of columns
stft = zeros(rown, coln);           % form the stft matrix

% initialize the indexes
indx = 0;
col = 1;

% perform STFT
while indx + wlen <= xlen
   % windowing
   xw = x(indx+1:indx+wlen).*win;

   % FFT
   X = fft(xw, nfft);

   % update the stft matrix
   stft(:, col) = X(1:rown);

   % update the indexes
   indx = indx + h;
   col = col + 1;
end

% calculate the time and frequency vectors
t = (wlen/2:h:wlen/2+(coln-1)*h)/fs;
f = (0:rown-1)*fs/nfft;

end
```

11.1.3 Pan Map Function

```
function [leftPanMap, rightPanMap, panMap] = panMapFunc(left, right)


% stop divide by zero error
right(right==0) = 1e-7;
left(left==0)   = 1e-7;


% calculate pan map
panMap = 20 * log10(abs(left)./abs(right));


% split pan map into left and right (.*-1 makes right abs)
leftBinary = panMap >= 0;
rightBinary = panMap < 0;


leftPanMap = leftBinary .* panMap;
rightPanMap = rightBinary .* panMap .* -1;
```

11.1.1 Multi-Level Thresholding Function

```
function [threshold] = fastMultiLevelThresholdFunc(input, thresNum)

% check to see if no. of threshold >= no. of bins
countNonZero = nnz(input);

% check
if countNonZero == 0;
  % create evenly distributed thresholds
  divid = thresNum + 1;
  incre = (length(input) + 1) / divid;

  switch thresNum
    case 1
      threshold = round(incre);
    case 2
      threshold = [round(incre), round(2*incre)];
    case 3
      threshold = [round(incre), round(2*incre), round(3*incre)];
    case 4
      threshold = [round(incre), round(2*incre), round(3*incre), round(4*incre)];
  end

elseif countNonZero < thresNum
  % create evenly distributed thresholds and replace with thresholds at
  % non-zero bins
  divid = thresNum + 1;
  incre = length(input) / divid;

  switch thresNum
    case 1
      threshold = round(incre);
    case 2
      threshold = [round(incre), round(2*incre)];
    case 3
      threshold = [round(incre), round(2*incre), round(3*incre)];
    case 4
      threshold = [round(incre), round(2*incre), round(3*incre), round(4*incre)];
  end

  knownThres = find(input);

  replace = round(knownThres ./ incre);

  L2 = length(replace);

  for i = 1:(L2 - 1)
    if replace(i) == replace(i+1)
      replace(i+1) = replace(i+1) +1;
    end
  end

  if replace(L2) > thresNum
    replace(L2) = thresNum;
  end

  if replace(1) == 0
    replace(1) = 1;
  end

  for i = 1:L2
    threshold(floor(replace(i))) = knownThres(i);
  end

elseif countNonZero == thresNum
  % use non-zero bins as thresholds
  threshold = find(input);

else
  %     perform otsu's (fast) algorithm

  L = length(input);
  P = zeros(L);
  S = P;

  P(1,:) = cumsum(input);
```

```
count = 1:1:L;
S(1,:) = cumsum(count .* input);


for u = 2:L
   for v = u:L

      P(u,v) = P(1,v) - P(1,(u-1));
      S(u,v) = S(1,v) - S(1,(u-1));

   end
end

G = (S.^2)./ P;

varSq = 0;

switch thresNum

   case 1

      for t = 1:(L-1)
         tmp = G(1,t) + G((t+1),L);
         if tmp > varSq
            varSq = tmp;
            threshold = t;
         end
      end

   case 2

      for t = 1:(L-1)
         for t2 = (t+1):(L-2)
            tmp = G(1,t) + G((t+1),t2) + G((t2+1),L);
            if tmp > varSq
               varSq = tmp;
               threshold = [t, t2];
            end
            end
          end

        case 3

          for t = 1:(L-1)
            for t2 = (t+1):(L-2)
              for t3 = (t2+1):(L-3)
                tmp = G(1,t) + G((t+1),t2) + G((t2+1),t3) + G((t3+1),L);
                if tmp > varSq
                  varSq = tmp;
                  threshold = [t, t2, t3];
                end
              end
            end
          end

        case 4

          for t = 1:(L-1)
            for t2 = (t+1):(L-2)
              for t3 = (t2+1):(L-3)
                for t4 = (t3+1):(L-4)
                  tmp = G(1,t) + G((t+1),t2) + G((t2+1),t3) + G((t3+1),t4) + G((t4+1),L);
                  if tmp > varSq
                    varSq = tmp;
                    threshold = [t, t2, t3, t4];
                  end
                end
              end
            end
          end

      end

    end
```

11.1.2 Azimuth Function

```
function [azimuth, aL, aR, power] = aziFunc(sourceMap, leftSTFT, rightSTFT, separation)


numberOfSources = size(sourceMap,3);
sourceLevelLeft = zeros(1,numberOfSources);
sourceLevelRight = zeros(1,numberOfSources);


% for all sources calculate power in left and right STFTs
for i = 1:numberOfSources
    tmpL = sourceMap(:,:,i) .* leftSTFT;
    tmpL = tmpL(tmpL~=0);
    sourceLevelLeft(i) = mean(abs(tmpL(:)));
    tmpR = sourceMap(:,:,i) .* rightSTFT;
    tmpR = tmpR(tmpR~=0);
    sourceLevelRight(i) = mean(abs(tmpR(:)));
end


power = ((sourceLevelLeft).^2) + ((sourceLevelRight).^2);
aL = sourceLevelLeft ./ sqrt(power);
aR = sourceLevelRight ./ sqrt(power);


azimuth = zeros(1,numberOfSources);
sep = separation/2;


% calculate azimuth using total power and power in left/right STFTs
for i = 1:numberOfSources
    azimuth(i) = ((aL(i)-aR(i)) / (aL(i)+aR(i))) * sep ;
end
```

11.1.3 Source to Ambisonic Function

```
function [output] = sourceToAmbiFunc(input, azimuth, order)
% input - mono audio
% azimuth - horizontal position in degrees
% order - nth order ambisonic format to be returned
% NOTE - no elevation

% check/make audio input a horizontal vector
if size(input, 2) > 1
    input = input';
end

% initialise output

audioLength = length(input);

switch order
    case 1
        output = zeros(audioLength,4);
    case 2
        output = zeros(audioLength,9);
    case 3
        output = zeros(audioLength,16);
end

% calculate audio channels
% 1st order
% W
output(:,1) = input .* 0.7071;
% X
output(:,2) = input .* cosd(azimuth);
% Y
output(:,3) = input .* sind(azimuth);
% Z
%  channel for elevation only, so stay as zero

% 2nd order
if order > 1
```

```
    % R
    output(:,5) = input .* -0.5;

    % S
    % channel zero'd by sin(2E)
    % T
    % channel zero'd by sin(2E)
    % U
    output(:,8) = input .* cosd(2 * azimuth);
    % V
    output(:,9) = input .* sind(2 * azimuth);
end

% 3rd order
if order > 2
    % K
    %  channel for elevation only, so stay as zero
    % L
    output(:,11) = input .* -0.7262 * cosd(azimuth);
    % M
    output(:,12) = input .* -0.7262 * sind(azimuth);
    % N
    % channel zero'd by sin(2E)
    % O
    % channel zero'd by sin(2E)
    % P
    output(:,15) = input .* cosd(3 * azimuth);
    % Q
    output(:,16) = input .* sind(3 * azimuth);
end

output = output';
```

11.1.4 Proposed Fix to Azimuth Function

```
function [azimuth, aL, aR, power] = aziFunc(sourceMap, leftSTFT, rightSTFT, separation)

numberOfSources = size(sourceMap,3);
sourceLevelLeft = zeros(1,numberOfSources);
sourceLevelRight = zeros(1,numberOfSources);

% for all sources calculate power in left and right STFTs
for i = 1:numberOfSources
   tmpL = sourceMap(:,:,i) .* leftSTFT;
   tmpL = tmpL(tmpL~=0);
   sourceLevelLeft(i) = mean(abs(tmpL(:)));
   tmpR = sourceMap(:,:,i) .* rightSTFT;
   tmpR = tmpR(tmpR~=0);
   sourceLevelRight(i) = mean(abs(tmpR(:)));
end

power = ((sourceLevelLeft).^2) + ((sourceLevelRight).^2);
aL = sourceLevelLeft ./ sqrt(power);
aR = sourceLevelRight ./ sqrt(power);

azimuth = zeros(1,numberOfSources);
sep = separation/2;

% calculate azimuth using total power and power in left/right STFTs
for i = 1:numberOfSources
   azimuth(i) = ((aL(i)-aR(i)) / (aL(i)+aR(i))) * sep ;
end
azimuth(isnan(azimuth)) = 0;



% find 'true' sources where power is > mean
% move 'fake' sources, where power is < mean, to nearest 'true' source
power(isnan(power)) = 0;
thres = mean(power);
real = power > thres;
```

```
if sum(real) ~= 0
    y = find(real);
    for i = 1:numberOfSources
        if real(i) == 0
            [m, index] = min(abs(y - i));
            azimuth(i) = azimuth(y(index));
        end
    end
end
```

11.2 Content of Provided DVD

The provided DVD holds the following folders/files:

↳ IES Final Project – Haydon Cardew

↳ MatLab Code

Here all the code required to carry out the up-mix algorithm is provided

↳ Example Converted Audio

↳ Original Stereo Audio

The three example songs in stereophonic format.

The songs are as follows:

1) *Tycho (2104)* ***Awake****. Awake. Ghostly International*
2) *Avicii (2012)* ***Superlove****. Superlove Remix. T.Y.S Inc*
3) *Muse (2015)* ***Drones****. Drones. Warner Music*

↳ Converted B-Format Signals

The three stereophonic tracks after up-mixing.

↳ Example Tests

↳ Single Moving Source

The stereophonic file of a single source being panned across the stereo field is provided, along with the B-format up-mixed versions (one with and one without the fix mentioned in section 7.2).

↳ Source Extraction

↳ Original Audio Files

A test stereophonic single along with its three constituent parts (see Figure 7.1).

↳ Extraction Using 2 Thresholds

Three parts extracted using 2 thresholds.

↳ Extraction Using 3 Thresholds

Three parts extracted using 3 thresholds.

↳ Extraction Using 4 Thresholds

Three parts extracted using 4 thresholds.