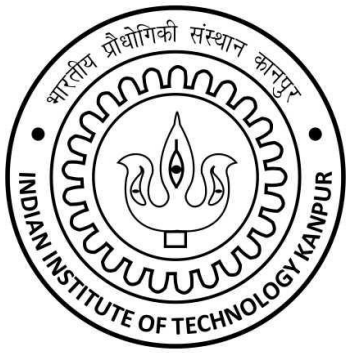


Indian Institute of Technology Kanpur

EE627: Speech Signal Processing
Final Report for Term Paper Implementation

Group 8

Encoding Higher Order Ambisonics with AAC

GROUP MEMBERS:

Akash Singh (14048)

Sunil Kumar Chawla (14721)

Vikas Verma (14801)

Vimal Kumar Meena (14804)

SUPERVISOR:

Prof. Rajesh Hegde

MENTOR:

Gyanajyoti Routray

Objective

In this project our primary aim is to reduce the bit rate needed for transmitting and storing Higher Order Ambisonics (HOA). The HOA B-format signals are simply encoded using Advanced Audio Coding (AAC) as if they were individual mono signals. The compressed B-format signal is converted to D-format (loudspeaker) signal. Wave Field Simulations are used to compare the errors as a function of distance for different no. of channels and different frequencies.

Experimental Setup

We have assumed the source is at 30° in clockwise direction with respect to listener and assumed 4 speaker layout first at 45° with respect to listener and other at 90° to the adjacent speakers in a circular array.

A-Format

A-Format, is produced by the Soundfield microphone itself and consists of the four signals from the microphone capsules - left-front, left-back, right-front and right-back.

B-Format

Ambisonics B-Format is the standardized format which uses 4 channels to store signal information. The first channel carries the amplitude information of the signal, while the other channels determine the directionality through phase relationships between each other. B-Format consists of four signals called W, X, Y, Z. Signal W is relative to the pressure component of the sound field in all directions, while X, Y, Z refer to the horizontal components of velocity on the horizontal plane (X, Y) and the vertical component (Z) of velocity. A simple Ambisonic encoder takes a source signal S and two parameters, the horizontal angle Θ and the elevation angle ϕ. It positions the source at the desired angle by distributing the signal over the Ambisonic components with different gains:

$W = S/\sqrt{2}$
$X = S.\cos\Theta.\cos\phi$
$Y = S.\sin\Theta.\cos\phi$
$Z = S.\sin\phi$

For 2-D case, we choose ϕ=0.

D-format

D-Format is the format that made Ambisonics compatible with common surround speaker systems, such as 5.1, 7.1, but also with arrays of different sizes and geometries (either regular or irregular geometries). Signals in D-Format can be derived from either B-Format or C-Format with the use of a decoder. The number of speakers is not limited in theory. The minimum requirements are, however, 4 speakers for adequate surround playback, 6 is better and full periphony (and therefore the information relative to the height) can be obtained through 8 speakers.

If we choose our environment where four loudspeakers are placed with 90° separation between the adjacent pairs of speakers and $\phi 1 = 45°$

$\phi i+1 = \phi I + \pi/2$ where $1 \le i \le 3$

$$Speaker\ Weights\ [SW] = \begin{bmatrix} 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} & 1/\sqrt{2} \\ \cos\phi 1 & \cos\phi 2 & \cos\phi 3 & \cos\phi 4 \\ \sin\phi 1 & \sin\phi 2 & \sin\phi 3 & \sin\phi 4 \end{bmatrix}$$

$B\ to\ D\ converter = [SW]\ T \times [W\ X\ Y]$

Advanced Audio Coding (AAC)

Advanced audio coding (AAC) is a technique used for compressing and encoding scheme digital audio files. An Advanced Audio Coding (AAC) encoder splits the signal into frames of 2048 samples (or 256 samples if transients are detected) overlapping with 50%, and transforms each frame into the frequency domain using the Modified Discrete Cosine Transform (MDCT). From a psychoacoustic analysis the quantization threshold for each sub-band is selected and finally, the resulting coefficients are Huffman coded.

Step:1

- The B-format signal is converted from time-domain to frequency-domain using forward modified discrete cosine transform (MDCT).
- Preprocessing of the signal for MDCT:

The signal is windowed into frames of 2048 samples with 50% overlapping.

$$y[n] = x[n].w[n]$$

- Now, we convert each frame of the signal to the frequency domain using MDCT.

MDCT:

$$X_k = \sum_{n=0}^{2N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2} + \frac{N}{2}\right)\left(k + \frac{1}{2}\right)\right]$$

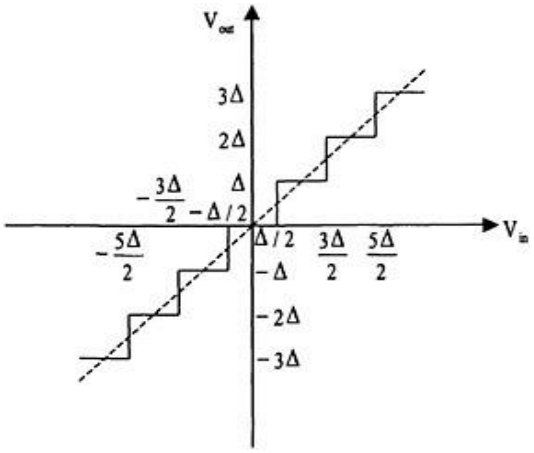
$k=0,1,\ldots,\ N-1$

$x_n = h_n a_n$ is the windowed input signal; a_n is the input signal with 2N samples and h_n is the window function.

Step:2

- Signal X (n, ω) is in a finite range (fmin , fmax) then it is divided into L equal intervals of length Q (quantization step size)

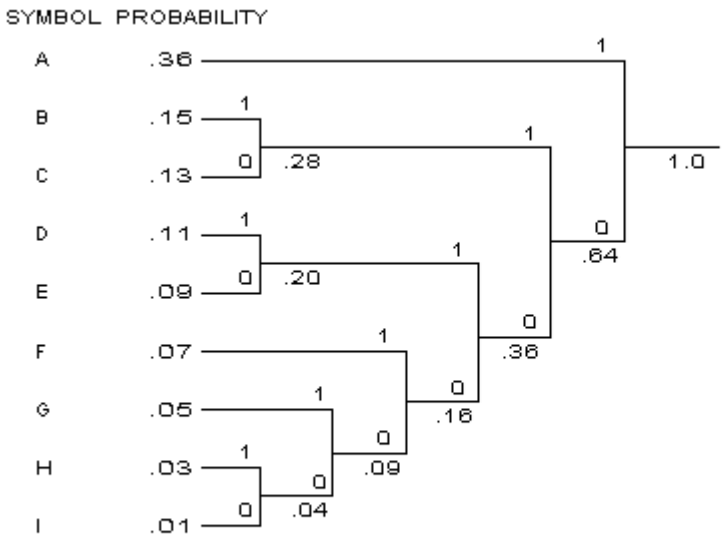
$$Q = (fmax - fmin)/L$$



Mid Tread Quantizer

Step:3

- For the Noiseless coder we are going to use the Huffman coding algorithm.
- Huffman coding is a lossless compression.
- In this method, more common symbols are generally represented using fewer bits than less common symbols.
- We get the quantized signal from quantizer then we apply Huffman coding as stated below:



Wave field Analysis

Wave field analysis is performed by comparing an original sound field with a processed sound field over a reproduction area (a circle of radius 10 cm). The original soundfield consists of a single virtual source positioned at infinite distance, and with no room reflections or reverberation included. A final wave field snapshot was then generated by plotting the instantaneous wave field across the reproduction area.

Execution Steps

A2B.m

This code takes an audio file as input and this audio file with the given environmental condition is treated as A-format signal. This code convert the A-format signal to the corresponding B-format signal in 2-D i.e. break it down into W, X, Y component.

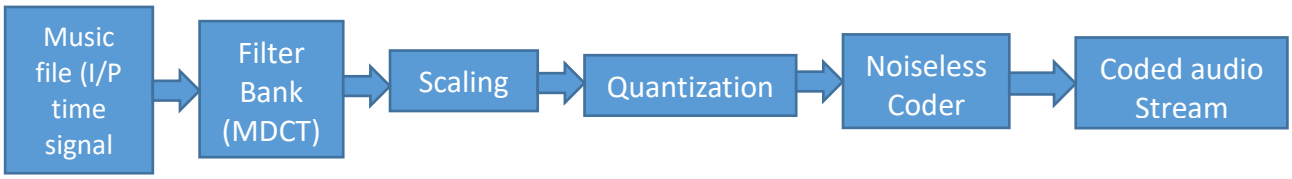
B2D.m

This code takes the B-format signal (obtained above) as input and converts it into D-format signal according to the given number of speakers and their layout.

Encoded_Df.m / Encoded_Bf.m

This code takes the D-format/B-format signal (.wav file) as input and compress it, then this converted file is saved as output file (.aac file) in the directory as mentioned in the code.

- ***main_aacencoderD:***



  - Input Signal is framed using `longwindow.mat` and MDCT coefficients are calculated using `mdct4.m` function.
  - Scaling is done through pre-defined scale factors in `sfb_offset.mat`
  - Noiseless Coding (Huffman Coding) is done using `huffmancode_specdata.m.` This function uses pre-defined Huffman Code-book (`Huffman_tables.mat`) and give the compressed file as output.

AAC Decoding

MATLAB doesn't support the .aac file format, so we used an online tool (faad.exe) to decode .aac files to loudspeaker signal.

wavefieldsynthe.m:

The loudspeaker signals were transformed using a DFT and eq. 4 was applied one frequency at a time, and an inverse DFT gave the time-domain signal.

$$p(r,k,\theta) = \sum_{m=1}^{M} e^{jkr\cos(\theta_m-\theta)} l_m(0,k,\theta_m), \qquad (4)$$

where l_m is the signal from the m'th loudspeaker, which is placed in the angle θ_m.

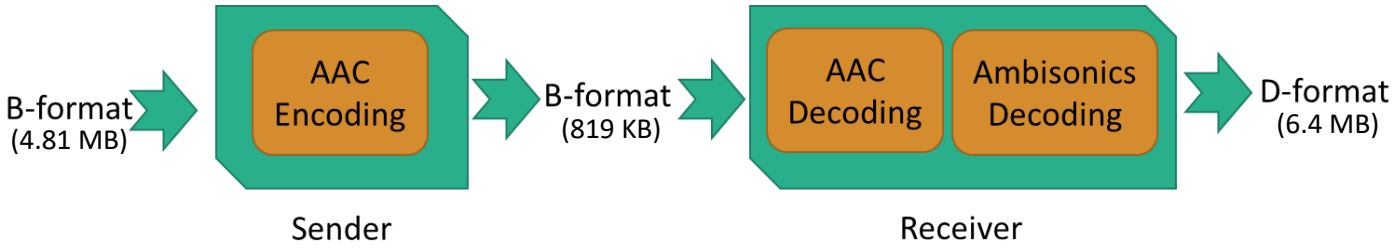
Error as function of distance is calculated using eq. 5

$$\epsilon(r,\theta,t) = 10 * log_{10} \frac{(p_c(r,\theta,t) - p_r(r,\theta,t))^2}{p_r(r,\theta,t)^2}, \quad (5)$$

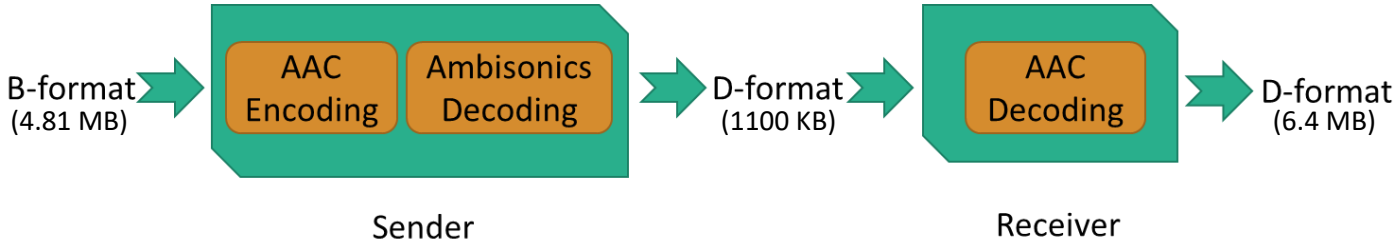
Finally, plots are obtained for Angle-averaged error vs. Distance from center for different no. of channels and different frequencies.

Results

- Comparing two-system model process as follows:



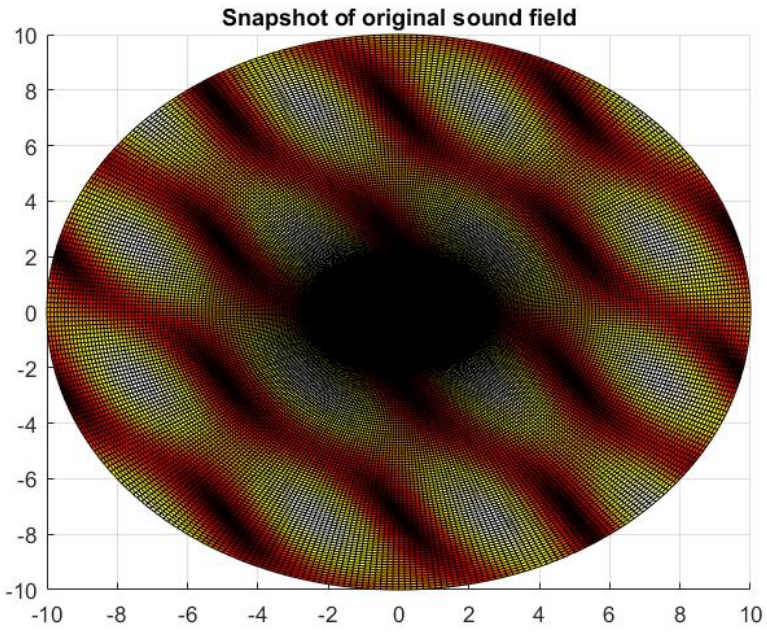
PROCESS-1 (4 CHANNELS)

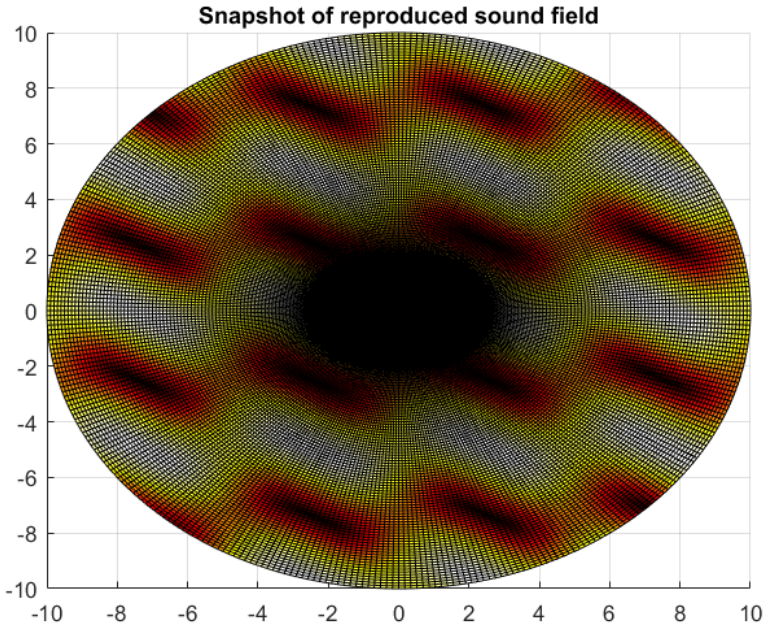


PROCESS-2 (4 CHANNELS)

As can be seen from above flow diagram, the size of the file needed during transmission is less in Process-I than Process-II. And as the no. of channels increases, the size of transmitting file may increases further in Process-II. Thus, for transmission Process-I is efficient.

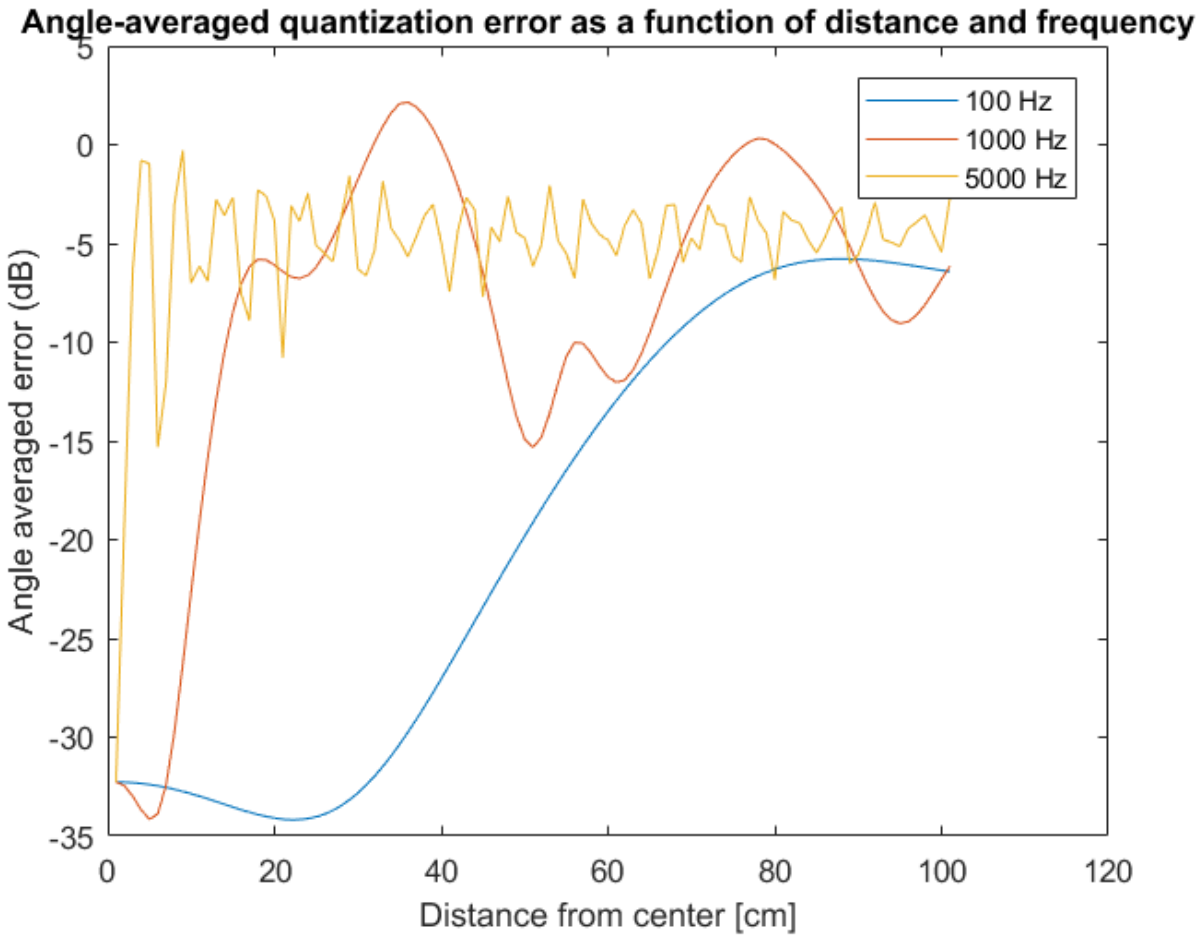
- Sound Field Comparison



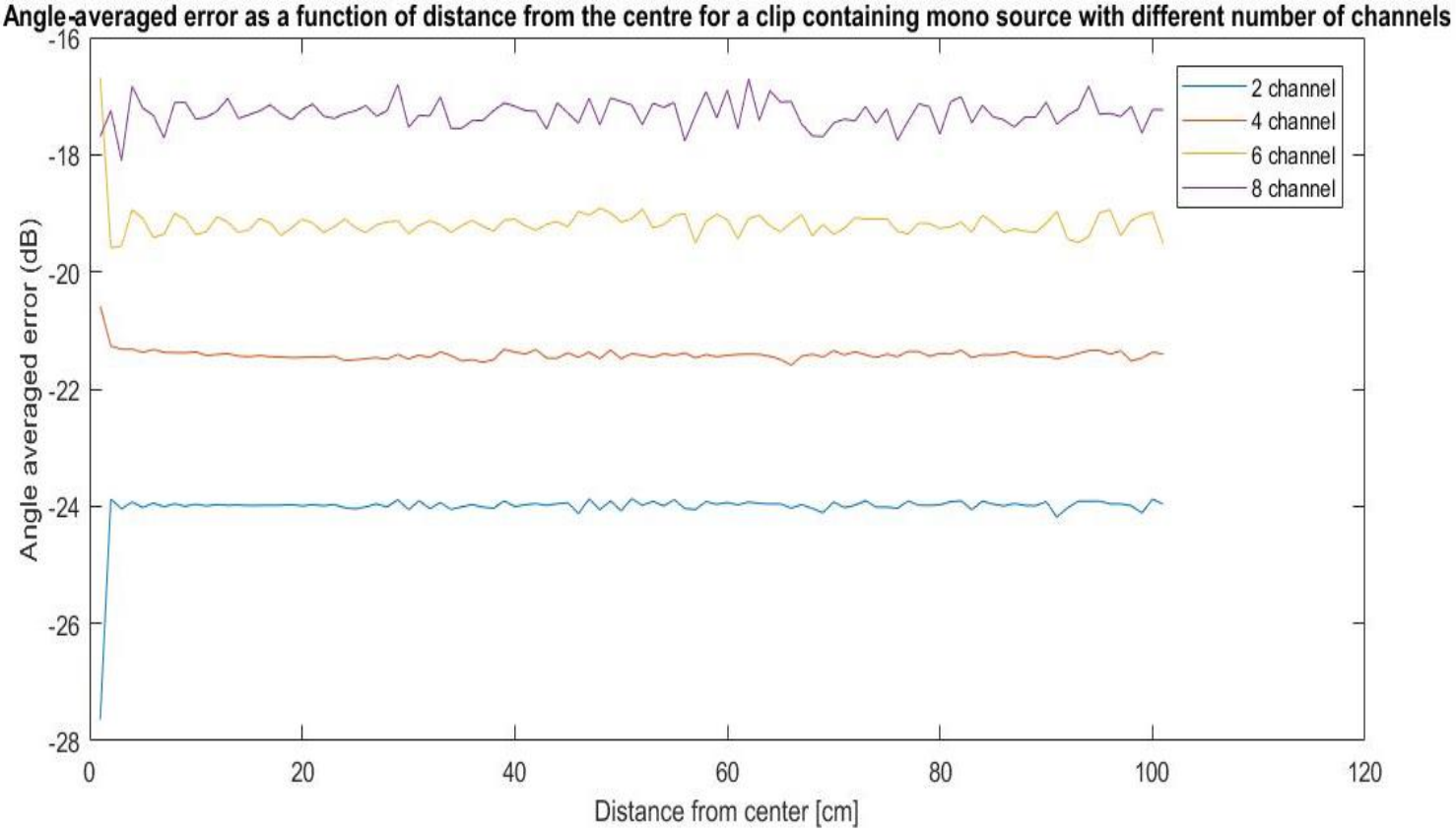


As can be seen from above wavefield snapshot (for 500 Hz), there is a slight change in the reproduced sound field.

- Error as function of distance



The radial extent of the area with the lowest error depends on the frequency. This frequency dependence is illustrated by evaluating the angle-averaged error for single-frequency wave field of frequency 100, 1000 or 5000 Hz. In this case, the signal was transformed using the MDCT and quantized. As can be seen in figure, the higher the frequency is, the smaller is the area with a lower error. The perfect reconstruction radius for a frequency of 1000 Hz is 30 cm.



We have calculated the angle averaged error for different no. of channels and found the more the no. of channels the less is the error.

MATLAB Published Report

A2B.m

```
clear all;
clc;

% A-format to B-format
% Assuming 4-speaker layout first at 45degree w.r.t. listener and other at 90 degree to the adjacent
speakers in a circle
% Assuming the source is at 30degree in clockwise direction w.r.t. listener

[y,Fs]=audioread('original_input.wav');

sorangle = 30;
w = pi()/180;

WXY_mat = [1/sqrt(2)   cos(w*sorangle)   sin(w*sorangle)];
B_mat = (WXY_mat')*(y)';

W = B_mat(1,:);
X = B_mat(2,:);
Y = B_mat(3,:);

filename = '.\Bformat\W.wav';
audiowrite(filename,W,Fs);
filename = '.\Bformat\X.wav';
audiowrite(filename,X,Fs);
filename = '.\Bformat\Y.wav';
audiowrite(filename,Y,Fs);
```

B2D.m

```
clear all;
clc;
% Converting B-format signal to D-format (4 channels)
[W,~]=audioread('.\Bformat\W.wav');
[X,~]=audioread('.\Bformat\X.wav');
[Y,Fs]=audioread('.\Bformat\Y.wav');
w = pi()/180;
```

```
B_mat = [W,X,Y]';
D4_mat = [ 1/sqrt(2)     1/sqrt(2)         1/sqrt(2)         1/sqrt(2) ;
            cos(w*0)     cos(w*45)         cos(w*90)         cos(w*135);
            sin(w*0)     sin(w*45)         sin(w*90)         sin(w*135)];


D_format = (D4_mat')* B_mat;

filename = '.\Dformat\4\D1.wav';
audiowrite(filename,D_format(1,:),Fs);
filename = '.\Dformat\4\D2.wav';
audiowrite(filename,D_format(2,:),Fs);
filename = '.\Dformat\4\D3.wav';
audiowrite(filename,D_format(3,:),Fs);
filename = '.\Dformat\4\D4.wav';
audiowrite(filename,D_format(4,:),Fs);
```

Encoded_Bf.m

```
clear all
clc
[W,~]=audioread('.\Bformat\W.wav');
[X,~]=audioread('.\Bformat\X.wav');
[Y,fs]=audioread('.\Bformat\Y.wav');

main_aacencoder(W,fs,'W')
main_aacencoder(X,fs,'X')
main_aacencoder(Y,fs,'Y')
```

Encoded_Df.m

```
clear all
clc
[D1,~]=audioread('.\Dformat\4\D1.wav');
[D2,~]=audioread('.\Dformat\4\D2.wav');
[D3,~]=audioread('.\Dformat\4\D3.wav');
[D4,fs]=audioread('.\Dformat\4\D4.wav');

% All wav files converted to aac format
main_aacencoderD(D1,fs,'D1')
main_aacencoderD(D2,fs,'D2')
main_aacencoderD(D3,fs,'D3')
main_aacencoderD(D4,fs,'D4')
```

main_aacencoderD.m

```
function  main_aacencoder(signal,fs,string)
format long
% Hamming Window
load longwindow.
% scale factor offset for scaling
load sfb_offset.mat


MAX_SFB = 43;
y = signal;
nbits = 16;
if(fs ~= 44100 || nbits ~= 16)
    disp('Not supported - Only 44100 Mono files');
    return;
end

fp = fopen(strcat('.\Encoded_Dformat\encoded_',string,'.aac'),'w');
y = y*2^15;
y=[zeros(1,1024) y']';
len = fix(length(y)/1024);
y = reshape(y(1:len*1024),1024,len)';
long = long_window;
for frame_index = 1:len - 1
    % Windowing the signal
    wf = [y(frame_index,:) y(frame_index+1,:)];
    wf = wf.*[long flipud(long')'];
```

```
    % Calculating MDCT coefficient
    wf = mdct4(wf)'.*sqrt(512);
    wf = wf.*4;
    wf(744:end) = 0;
    mdct_coeff = wf;
    % No MS Encode, since we are working on Mono stream
    % Quantizing the MDCT_Coefficients
    xr_pow = abs(mdct_coeff).^(3/4);
    % Compute the ScaleFactor and Quantized MDCT-Coefficients
    % Minimal logic
    mdct_coeff_quant = [];
    max_sfb_encoded = MAX_SFB;
    scale_factors = zeros(1,max_sfb_encoded);
    % You can call below as stupid Inner loop !! or Even Pathetic Rate
    % control, even though there is no rate to control :)

    for sfb_offind=1:max_sfb_encoded

        max_coeff = max(xr_pow([sfb_offset(sfb_offind)+1 : sfb_offset(sfb_offind + 1)]));% 0.0001 to
avoid 1/0 -> Inf
        avg_coeff = sum(xr_pow([sfb_offset(sfb_offind)+1 : sfb_offset(sfb_offind + 1)]))/4;
        maxsf_start = fix((log10(1/max_coeff)/log10(2^(3/16))) - 0.5);

        % SF_OFFSET = 100;
        % MAGIC_NUMBER = 0.4054
        % x_quant = int (( abs( mdct_line ) * (2^((- 1/4) * (sf_decoder - 100))) )^(3/4) + 0.405)
        arr = xr_pow([sfb_offset(sfb_offind)+1 : sfb_offset(sfb_offind + 1)]);
        x_unquant = abs(mdct_coeff([sfb_offset(sfb_offind)+1 : sfb_offset(sfb_offind + 1)]));
        x_quant = zeros(1,length(arr));
        while maxsf_start < -39

            x_quant = fix(arr.*((2^(maxsf_start/4))^(3/4)));
            % Inverse quantize and check average
            x_iquant = (x_quant.^(4/3))./(2^(maxsf_start/4));
            if( mean(x_iquant) > (mean(x_unquant)/1.3))
                maxsf_start = maxsf_start + 1;
                break;
            end
            maxsf_start = maxsf_start + 1;
        end
        scale_factors(sfb_offind) = maxsf_start;
        if(max_coeff <=0)
            scale_factors(sfb_offind) = 0;
        end
        mdct_coeff_quant = [mdct_coeff_quant x_quant];
    end

    global_gain = 0; % No Inner and Outer loops, so no gg :)
    % Add sign's
    mdct_coeff_quant(find(mdct_coeff <0)) = mdct_coeff_quant(find(mdct_coeff <0)).*-1;

    % What we have at this point
```

```
    % 1. We have mdct_coeff_quant, which will be used for Noiseless coding
    % 2. We also have scale_factors, which will be differentially coded

    % Lets assign codebooks directly, with out calculating the number of
    % bits, later we will do that too, explained below. (Because we do not
    % have any kind of rate control, so why to calculate bits, assume (May
    % not be best, but what the F$%^)
    % Read Read_Me.m, given with this folder

    codebook_used_sfb = zeros(1,43);

    for temp_len = 1: length(mdct_coeff_quant)
        if((mdct_coeff_quant(temp_len)) > 12)
            mdct_coeff_quant(temp_len) = 12;
        elseif ((mdct_coeff_quant(temp_len)) < -12)
            mdct_coeff_quant(temp_len) = -12;
        end
    end

     for sfb_offind=1:max_sfb_encoded
         max_abs_quant_coeff = max(abs(mdct_coeff_quant([sfb_offset(sfb_offind)+1 :
sfb_offset(sfb_offind + 1)])));

         if(max_abs_quant_coeff == 0)
             codebook_used_sfb(sfb_offind) = 0;
         elseif(max_abs_quant_coeff < 2)
             codebook_used_sfb(sfb_offind) = 2; % Signed
         elseif(max_abs_quant_coeff < 3)
             codebook_used_sfb(sfb_offind) = 4; % Not signed
         elseif(max_abs_quant_coeff < 5)
             codebook_used_sfb(sfb_offind) = 6; % Signed
         elseif(max_abs_quant_coeff < 8)
             codebook_used_sfb(sfb_offind) = 8; % or use 7 Not Signed
         elseif(max_abs_quant_coeff < 13)
             codebook_used_sfb(sfb_offind) = 9; % Not Signed
         else
             codebook_used_sfb(sfb_offind) = 11;% Not Signed
         end
     end


     % offset the difference of common_scalefac and scalefactors by
     % SF_OFFSET (100)

     scale_factors = global_gain - scale_factors + 100; % 100 is SF_OFFSET
     global_gain = scale_factors(1);

     % Noiseless Coding -> Huffman Coding
     % Noiseless Coding of spectral data
     data_code = [];
     data_length = [];
     [data_code, data_length] = huffmancode_specdata(max_sfb_encoded, mdct_coeff_quant,
codebook_used_sfb);
```

```
     bitstream = [];
     bitstream = bitstream_pack_data(global_gain, max_sfb_encoded, codebook_used_sfb, scale_factors,
data_code, data_length);

     temp = bin2dec(reshape(bitstream,8,fix(length(bitstream)/8))')';
     fwrite(fp,temp,'uint8');

     disp([ '...' int2str(frame_index) '...']);

 end  % Main Encoder Frame loop
 disp('....End....');
 fclose(fp);
end
```

mdct4.m

```
function y = mdct4(x)
% MDCT4 Calculates the Modified Discrete Cosine Transform
%   y = mdct4(x)
%
%   Use either a Sine or a Kaiser-Bessel Derived window (KBDWin)with
%   50% overlap for perfect TDAC reconstruction.
%   Remember that MDCT coefs are symmetric: y(k)=-y(N-k-1) so the full
%   matrix (N) of coefs is: yf = [y;-flipud(y)];
%
%   x: input signal (can be either a column or frame per column)
%      length of x must be a integer multiple of 4 (each frame)
%   y: MDCT of x (coefs are divided by sqrt(N))
%
%   Vectorize ! ! !

% ------- mdct4.m ----------------------------------------
% Marios Athineos, marios@ee.columbia.edu
% http://www.ee.columbia.edu/~marios/
% Copyright (c) 2002 by Columbia University.
% All rights reserved.
% --------------------------------------------------------

[flen,fnum] = size(x);
% Make column if it's a single row
if (flen==1)
    x = x(:);
    flen = fnum;
    fnum = 1;
```

```
end
% Make sure length is multiple of 4
if (rem(flen,4)~=0)
    error('MDCT4 defined for lengths multiple of four.');
end

% We need these for furmulas below
N     = flen; % Length of window
M     = N/2;  % Number of coefficients
N4    = N/4;  % Simplify the way eqs look
sqrtN = sqrt(N);

% Preallocate rotation matrix
% It would be nice to be able to do it in-place but we cannot
% cause of the prerotation.
rot = zeros(flen,fnum);

% Shift
t = (0:(N4-1)).';
rot(t+1,:) = -x(t+3*N4+1,:);
t = (N4:(N-1)).';
rot(t+1,:) =  x(t-N4+1,:);
clear x;

% We need this twice so keep it around
t = (0:(N4-1)).';
w = diag(sparse(exp(-j*2*pi*(t+1/8)/N)));

% Pre-twiddle
t = (0:(N4-1)).';
c =   (rot(2*t+1,:)-rot(N-1-2*t+1,:))...
   -j*(rot(M+2*t+1,:)-rot(M-1-2*t+1,:));
% This is a really cool Matlab trick ;)
c = 0.5*w*c;
clear rot;

% FFT for N/4 points only !!!
c = fft(c,N4);

% Post-twiddle
c = (2/sqrtN)*w*c;

% Sort
t = (0:(N4-1)).';
y(2*t+1,:)     =  real(c(t+1,:));
y(M-1-2*t+1,:) = -imag(c(t+1,:));
```

huffmancode_specdata.m

```
function [data_code, data_length] = huffmancode_specdata(max_sfb_encoded, mdct_coeff_quant,
codebook_used_sfb);
% ------- AAC Encoder ----------------------------------------
% Ravi Lakkundi
% ------------------------------------------------------------

load huffman_tables.mat
load sfb_offset.mat

counter = 1;
     data_code = [];
     data_length = [];

     for sfb_offind=1:max_sfb_encoded
         arr = mdct_coeff_quant([sfb_offset(sfb_offind)+1 : sfb_offset(sfb_offind + 1)]);
         len = length([sfb_offset(sfb_offind)+1 : sfb_offset(sfb_offind + 1)]);
         codebook_number = codebook_used_sfb(sfb_offind);

         switch codebook_number
             case 0,
                 data_code(counter) = 0;
                 data_length(counter) = 0;
                 counter = counter + 1;
             case 1, % Signed codebook
                 for k = 1:4:len
                     index = 27*(arr(k)) + 9*(arr(k+1)) + 3*(arr(k+2)) + (arr(k+3)) + 40;
                     data_code(counter) = huff1(index+1,2);
                     data_length(counter) = huff1(index+1,1);
                     counter = counter + 1;
                 end
             case 2, % Signed codebook
                 for k = 1:4:len
                     index = 27*(arr(k)) + 9*(arr(k+1)) + 3*(arr(k+2)) + (arr(k+3)) + 40;
                     data_code(counter) = huff2(index+1,2);
                     data_length(counter) = huff2(index+1,1);
                     counter = counter + 1;
                 end
             case 3, % Unsigned Codebook
                 for k = 1:4:len
                     index = 27*abs(arr(k)) + 9*abs(arr(k+1)) + 3*abs(arr(k+2)) + abs(arr(k+3));
                     data_code(counter) = huff3(index+1,2);
                     data_length(counter) = huff3(index+1,1);
                     counter = counter + 1;
                     % Since codebook is unsigned, append signs in
                     % data_code
                     for j=1:4
                         if(arr(k+j-1) >0)
                             data_code(counter) = 0;
                             data_length(counter) = 1;
                         elseif(arr(k+j-1) < 0)
                             data_code(counter) = 1;
```

```
                    data_length(counter) = 1;
                end
                counter = counter + 1;
            end
        end
    case 4, % Unsigned Codebook
        for k = 1:4:len
            index = 27*abs(arr(k)) + 9*abs(arr(k+1)) + 3*abs(arr(k+2)) + abs(arr(k+3));
            data_code(counter) = huff4(index+1,2);
            data_length(counter) = huff4(index+1,1);
            counter = counter + 1;
            % Since codebook is unsigned, append signs in
            % data_code
            for j=1:4
                if(arr(k+j-1) >0)
                    data_code(counter) = 0;
                    data_length(counter) = 1;
                elseif(arr(k+j-1) < 0)
                    data_code(counter) = 1;
                    data_length(counter) = 1;
                end
                counter = counter + 1;
            end
        end
    case 5, % Signed codebook
        for k = 1:2:len
            index = 9*arr(k) + arr(k+1) + 40;
            data_code(counter) = huff5(index+1,2);
            data_length(counter) = huff5(index+1,1);
            counter = counter + 1;
        end
    case 6, % Signed codebook
        for k = 1:2:len
            index = 9*arr(k) + arr(k+1) + 40;
            data_code(counter) = huff6(index+1,2);
            data_length(counter) = huff6(index+1,1);
            counter = counter + 1;
        end
    case 7, % Unsigned Codebook
        for k = 1:2:len
            index = 8*abs(arr(k)) + abs(arr(k+1));
            data_code(counter) = huff7(index+1,2);
            data_length(counter) = huff7(index+1,1);
            counter = counter + 1;
            % Since codebook is unsigned, append signs in
            % data_code
            for j=1:2
                if(arr(k+j-1) >0)
                    data_code(counter) = 0;
                    data_length(counter) = 1;
                elseif(arr(k+j-1) < 0)
                    data_code(counter) = 1;
                    data_length(counter) = 1;
                end
```

```
                counter = counter + 1;
            end
        end
    case 8, % Unsigned Codebook
        for k = 1:2:len
            index = 8*abs(arr(k)) + abs(arr(k+1));
            data_code(counter) = huff8(index+1,2);
            data_length(counter) = huff8(index+1,1);
            counter = counter + 1;
            % Since codebook is unsigned, append signs in
            % data_code
            for j=1:2
                if(arr(k+j-1) >0)
                    data_code(counter) = 0;
                    data_length(counter) = 1;
                elseif(arr(k+j-1) < 0)
                    data_code(counter) = 1;
                    data_length(counter) = 1;
                end
                counter = counter + 1;
            end
        end
    case 9, % Unsigned Codebook
        for k = 1:2:len
            index = 13*abs(arr(k)) + abs(arr(k+1));
            data_code(counter) = huff9(index+1,2);
            data_length(counter) = huff9(index+1,1);
            counter = counter + 1;
            % Since codebook is unsigned, append signs in
            % data_code
            for j=1:2
                if(arr(k+j-1) >0)
                    data_code(counter) = 0;
                    data_length(counter) = 1;
                elseif(arr(k+j-1) < 0)
                    data_code(counter) = 1;
                    data_length(counter) = 1;
                end
                counter = counter + 1;
            end
        end
    case 10, % Unsigned Codebook
        for k = 1:2:len
            index = 13*abs(arr(k)) + abs(arr(k+1));
            data_code(counter) = huff10(index+1,2);
            data_length(counter) = huff10(index+1,1);
            counter = counter + 1;
            % Since codebook is unsigned, append signs in
            % data_code
            for j=1:2
                if(arr(k+j-1) >0)
                    data_code(counter) = 0;
                    data_length(counter) = 1;
                elseif(arr(k+j-1) < 0)
```

```
                            data_code(counter) = 1;
                            data_length(counter) = 1;
                        end
                        counter = counter + 1;
                    end
                end
            case 11,
                disp('Unfortunately iam here for this stream');% Write code now
            otherwise,
                disp('I cannot come, check quantization and codebook assignment');% Write code now
        end % Switch end
    end  % Noiseless Coding End
```

wavefieldsynthe.m

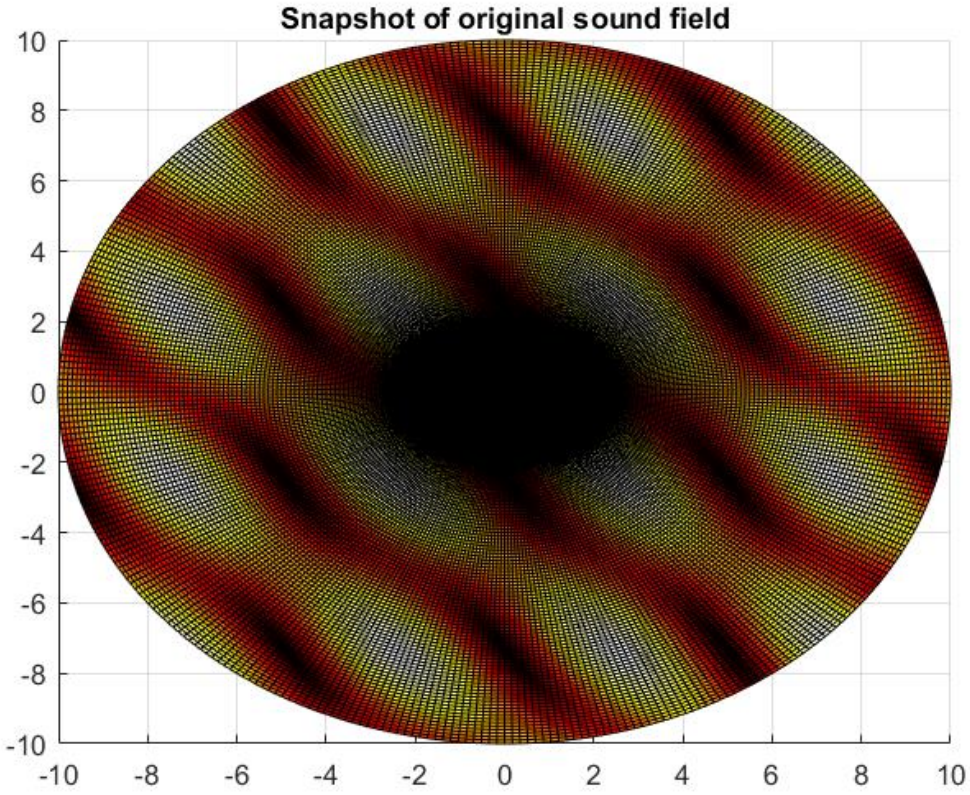
```
close all;
clc;
sorangle = 30;
w = pi()/180;
B_mat = [1/sqrt(2)   cos(w*sorangle)   sin(w*sorangle) 0];

phi1 = 45;
phi2 = 135;
phi3 = 225;
phi4 = 315;

D_mat = [ 1/sqrt(2)     1/sqrt(2)         1/sqrt(2)         1/sqrt(2);
         cos(w*phi1)    cos(w*phi2)      cos(w*phi3)       cos(w*phi4);
         sin(w*phi1)    sin(w*phi2)      sin(w*phi3)       sin(w*phi4);
              0            0                 0                  0];
frnum = [10 50 500];
for yy=1:3

FinalD = D_mat*B_mat';
f = frnum(yy);
c = 346;
alpha = 2*pi()*(1/c);
 [X,Y] = meshgrid(0:0.1:10,0:360);
    p = abs((exp(1i*f*alpha*X.*cosd(45-Y)).*FinalD(1)+exp(1i*f*alpha*X.*cosd(135-
Y)).*FinalD(2)+exp(1i*f*alpha*X.*cosd(225-Y)).*FinalD(3)+exp(1i*f*alpha*X.*cosd(315-Y)).*FinalD(4)));
psurf(:,:,yy) = p;

filename = '.\Encoded_Dformat\4\encoded_D1.wav';
D1=audioread(filename);
```

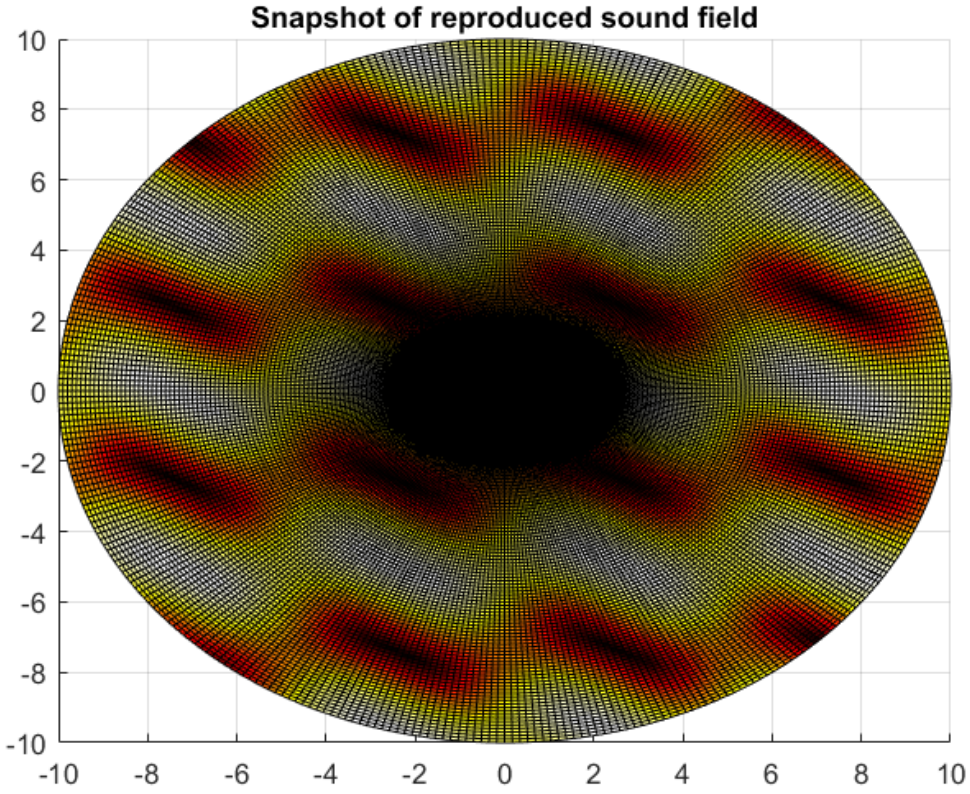
```
filename = '.\Encoded_Dformat\4\encoded_D2.wav';
D2=audioread(filename);
filename = '.\Encoded_Dformat\4\encoded_D3.wav';
D3=audioread(filename);
filename = '.\Encoded_Dformat\4\encoded_D4.wav';
D4=audioread(filename);
Ds=[D1,D2,D3,D4]';
[yf,Fs]=audioread('original_input.wav');
y = yf(1:length(D1));
Z=Ds*pinv(y)';

f = frnum(yy);
c = 346;
x=1;
y=1;
%figure;
[X,Y] = meshgrid(0:0.1:10,0:360);
    pt = abs((exp(1i*f*alpha*X.*cosd(45-Y)).*Z(1)+exp(1i*f*alpha*X.*cosd(135-
Y)).*Z(2)+exp(1i*f*2*pi()*(1/c)*X.*cosd(225-Y)).*Z(3)+exp(1i*f*2*pi()*(1/c)*X.*cosd(315-Y)).*Z(4)));
ptsurf(:,:,yy) = pt;

error = 10*log(sum((pt-p).^2)./sum(p));
plot(error)

hold on
end
legend('100 Hz','1000 Hz' ,'5000 Hz')
xlabel('Distance from center [cm]')
ylabel('Angle averaged error (dB)')
title('Angle-averaged quantization error as a function of distance and frequency')
figure;
surf(X.*cosd(Y),X.*sind(Y),ptsurf(:,:,2))
colormap(hot)
title('Snapshot of reproduced sound field')
view([-0.30 90.00])

figure;
surf(X.*cosd(Y),X.*sind(Y),psurf(:,:,2))
colormap(hot)
title('Snapshot of original sound field')
view([0.10 90.00])

sorangle = 30;
w = pi()/180;
% B format Coefficients
B_mat = [1/sqrt(2)   cos(w*sorangle)   sin(w*sorangle) 0]';
%D format Coefficient matrix for 2,4,6 and 8 channels
D2_mat = [ 1/sqrt(2)     1/sqrt(2);
         cos(w*0)    cos(w*180);
         sin(w*0)    sin(w*180);
              0           0      ];
D4_mat = [ 1/sqrt(2)     1/sqrt(2)         1/sqrt(2)         1/sqrt(2);
         cos(w*45)    cos(w*135)      cos(w*225)       cos(w*315);
         sin(w*45)    sin(w*135)      sin(w*225)       sin(w*315);
```

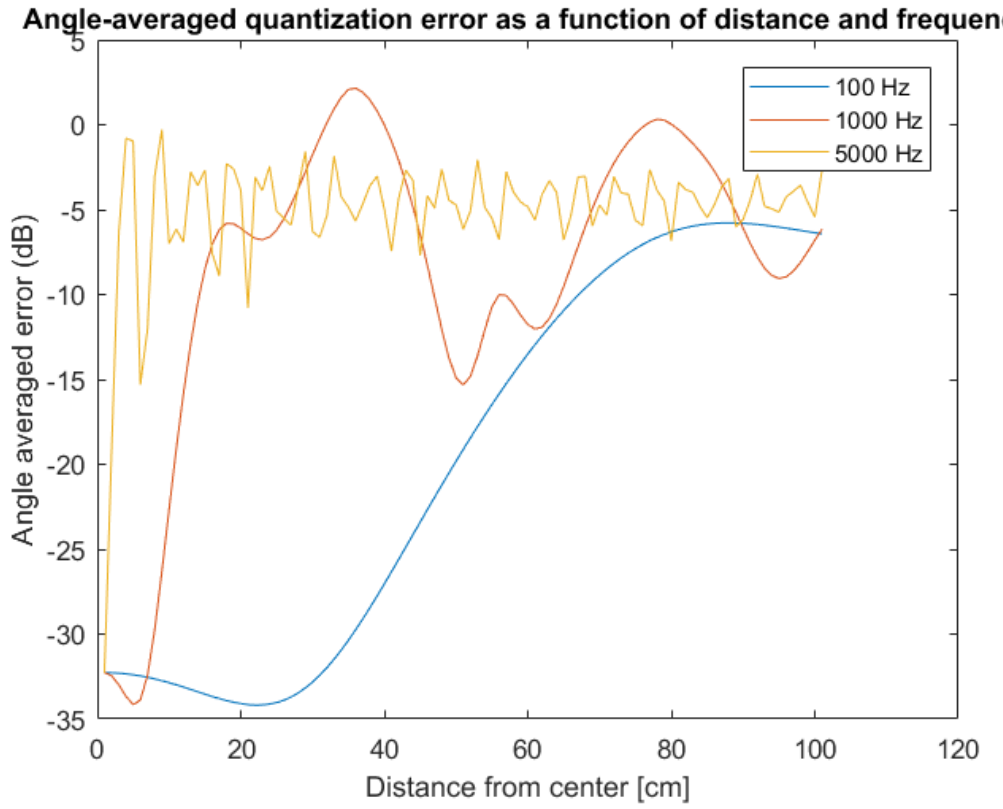
```
             0            0                 0                  0];
D6_mat = [ 1/sqrt(2)     1/sqrt(2)         1/sqrt(2)         1/sqrt(2)   1/sqrt(2)   1/sqrt(2);
         cos(w*0)    cos(w*60)      cos(w*120)       cos(w*180)      cos(w*240)     cos(w*300);
         sin(w*0)    sin(w*60)      sin(w*120)       sin(w*180)      sin(w*240)         sin(w*300);
              0            0                 0                  0               0           0];

D8_mat = [ 1/sqrt(2)     1/sqrt(2)         1/sqrt(2)         1/sqrt(2)   1/sqrt(2)   1/sqrt(2)
1/sqrt(2)       1/sqrt(2);
         cos(w*0)    cos(w*45)      cos(w*90)       cos(w*135)      cos(w*180)     cos(w*225)
cos(w*270)      cos(w*315);
         sin(w*0)    sin(w*45)      sin(w*90)       sin(w*135)      sin(w*180)         sin(w*225)
sin(w*270)      sin(w*315);
              0            0                 0                  0               0           0      0
0];

FinalD2 = D2_mat'*B_mat;
FinalD4 = D4_mat'*B_mat;
FinalD6 = D6_mat'*B_mat;
FinalD8 = D8_mat'*B_mat;
[yf,Fs]=audioread('original_input.wav');  %Mono signal
f=Fs;
y = yf(1:840704);
c = 346;
alpha = 2*pi()*(1/c);
[X,Y] = meshgrid(0:0.1:10,0:0.1:360);

%Pressure calculated for 2,4,6 and 8 channels at particular r,theta
%Reference Sound Field
p2 = abs((exp(1i*f*alpha*X.*cosd(0-Y)).*FinalD2(1)+exp(1i*f*alpha*X.*cosd(180-Y)).*FinalD2(2)));
p4 = abs((exp(1i*f*alpha*X.*cosd(45-Y)).*FinalD4(1)+exp(1i*f*alpha*X.*cosd(135-
Y)).*FinalD4(2)+exp(1i*f*alpha*X.*cosd(225-Y)).*FinalD4(3)+exp(1i*f*alpha*X.*cosd(315-Y)).*FinalD4(4)));
p6 = abs((exp(1i*f*alpha*X.*cosd(0-Y)).*FinalD6(1)+exp(1i*f*alpha*X.*cosd(60-
Y)).*FinalD6(2)+exp(1i*f*alpha*X.*cosd(120-Y)).*FinalD6(3)+exp(1i*f*alpha*X.*cosd(180-
Y)).*FinalD6(4)+exp(1i*f*alpha*X.*cosd(240-Y)).*FinalD6(5)+exp(1i*f*alpha*X.*cosd(300-Y)).*FinalD6(6)));
p8 = abs((exp(1i*f*alpha*X.*cosd(0-Y)).*FinalD8(1)+exp(1i*f*alpha*X.*cosd(45-
Y)).*FinalD8(2)+exp(1i*f*alpha*X.*cosd(90-Y)).*FinalD8(3)+exp(1i*f*alpha*X.*cosd(135-
Y)).*FinalD8(4)+exp(1i*f*alpha*X.*cosd(180-Y)).*FinalD8(5)+exp(1i*f*alpha*X.*cosd(225-
Y)).*FinalD8(6)+exp(1i*f*alpha*X.*cosd(270-Y)).*FinalD8(7)+exp(1i*f*alpha*X.*cosd(315-Y)).*FinalD8(8)));


%2 channel
filename = '.\Encoded_Dformat\2\encoded_D1.wav';
D1=audioread(filename);
filename = '.\Encoded_Dformat\2\encoded_D2.wav';
D2=audioread(filename);
Ds2 = [D1,D2]';
clear D1;
clear D2;
%4 channel
filename = '.\Encoded_Dformat\4\encoded_D1.wav';
D1=audioread(filename);
filename = '.\Encoded_Dformat\4\encoded_D2.wav';
D2=audioread(filename);
filename = '.\Encoded_Dformat\4\encoded_D3.wav';
```

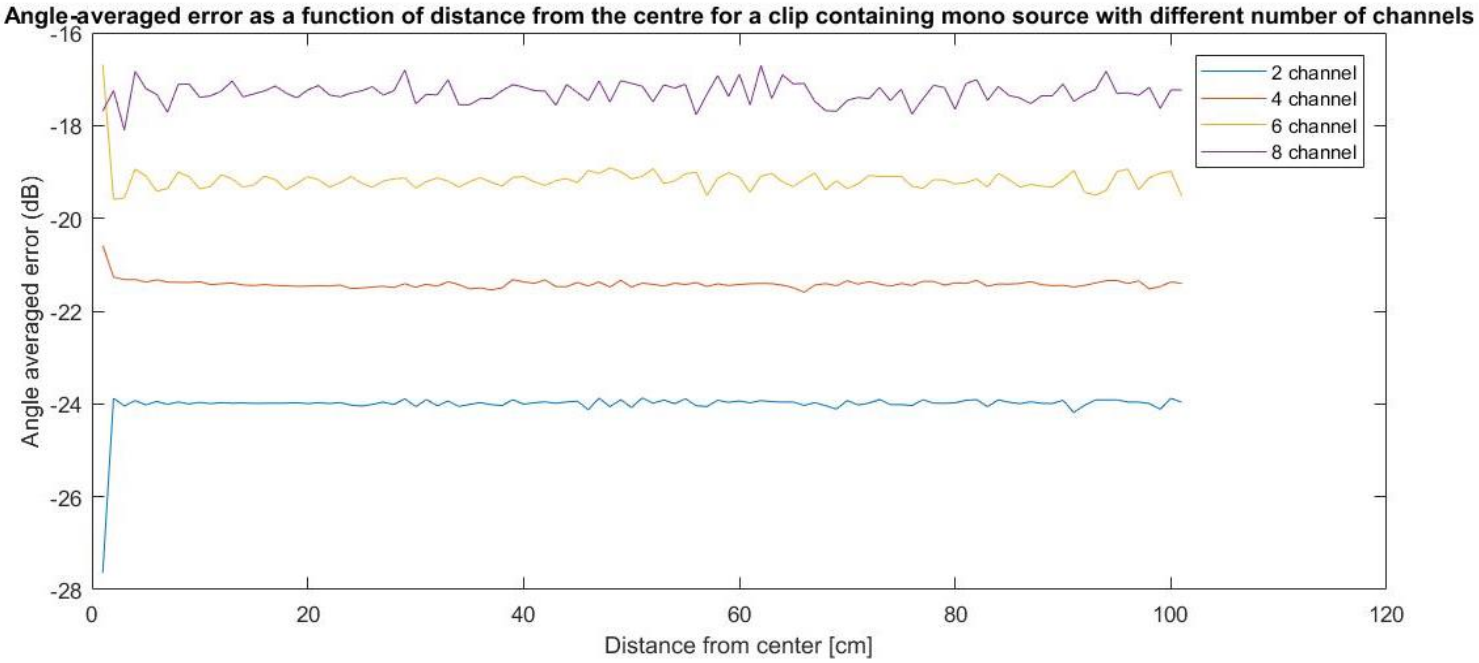
```
D3=audioread(filename);
filename = '.\Encoded_Dformat\4\encoded_D4.wav';
D4=audioread(filename);
Ds4=[D1,D2,D3,D4]';
clear D1;
clear D2;
clear D3;
clear D4;
%6 channel
filename = '.\Encoded_Dformat\6\encoded_D1.wav';
D1=audioread(filename);
filename = '.\Encoded_Dformat\6\encoded_D2.wav';
D2=audioread(filename);
filename = '.\Encoded_Dformat\6\encoded_D3.wav';
D3=audioread(filename);
filename = '.\Encoded_Dformat\6\encoded_D4.wav';
D4=audioread(filename);
filename = '.\Encoded_Dformat\6\encoded_D5.wav';
D5=audioread(filename);
filename = '.\Encoded_Dformat\6\encoded_D6.wav';
D6=audioread(filename);
Ds6=[D1,D2,D3,D4,D5,D6]';
clear D1;
clear D2;
clear D3;
clear D4;
clear D5;
clear D6;
%8 channel
filename = '.\Encoded_Dformat\8\encoded_D1.wav';
D1=audioread(filename);
filename = '.\Encoded_Dformat\8\encoded_D2.wav';
D2=audioread(filename);
filename = '.\Encoded_Dformat\8\encoded_D3.wav';
D3=audioread(filename);
filename = '.\Encoded_Dformat\8\encoded_D4.wav';
D4=audioread(filename);
filename = '.\Encoded_Dformat\8\encoded_D5.wav';
D5=audioread(filename);
filename = '.\Encoded_Dformat\8\encoded_D6.wav';
D6=audioread(filename);
filename = '.\Encoded_Dformat\8\encoded_D7.wav';
D7=audioread(filename);
filename = '.\Encoded_Dformat\8\encoded_D8.wav';
D8=audioread(filename);
Ds8=[D1,D2,D3,D4,D5,D6,D7,D8]';

Z2=Ds2*pinv(y)';
Z4=Ds4*pinv(y)';
Z6=Ds6*pinv(y)';
Z8=Ds8*pinv(y)';
f = Fs;
c = 346;
[X,Y] = meshgrid(0:0.1:10,0:0.1:360);
```

```
%Sound Field resulting from the encoded signal
pt2 = abs((exp(1i*f*alpha*X.*cosd(0-Y)).*Z2(1)+exp(1i*f*alpha*X.*cosd(180-Y)).*Z2(2)));
pt4 = abs((exp(1i*f*alpha*X.*cosd(45-Y)).*Z4(1)+exp(1i*f*alpha*X.*cosd(135-
Y)).*Z4(2)+exp(1i*f*alpha*X.*cosd(225-Y)).*Z4(3)+exp(1i*f*alpha*X.*cosd(315-Y)).*Z4(4)));
pt6 = abs((exp(1i*f*alpha*X.*cosd(0-Y)).*Z6(1)+exp(1i*f*alpha*X.*cosd(60-
Y)).*Z6(2)+exp(1i*f*alpha*X.*cosd(120-Y)).*Z6(3)+exp(1i*f*alpha*X.*cosd(180-
Y)).*Z6(4)+exp(1i*f*alpha*X.*cosd(240-Y)).*Z6(5)+exp(1i*f*alpha*X.*cosd(300-Y)).*Z6(6)));
pt8 = abs((exp(1i*f*alpha*X.*cosd(0-Y)).*Z8(1)+exp(1i*f*alpha*X.*cosd(45-
Y)).*Z8(2)+exp(1i*f*alpha*X.*cosd(90-Y)).*Z8(3)+exp(1i*f*alpha*X.*cosd(135-
Y)).*Z8(4)+exp(1i*f*alpha*X.*cosd(180-Y)).*Z8(5)+exp(1i*f*alpha*X.*cosd(225-
Y)).*Z8(6)+exp(1i*f*alpha*X.*cosd(270-Y)).*Z8(7)+exp(1i*f*alpha*X.*cosd(315-Y)).*Z8(8)));

%Angle Averaged Error for different number of channels
error2 = 10*log(sum((pt2-p2).^2)./sum(p2));
error4 = 10*log(sum((pt4-p4).^2)./sum(p4));
error6 = 10*log(sum((pt6-p6).^2)./sum(p6));
error8 = 10*log(sum((pt8-p8).^2)./sum(p8));
figure;
plot(error2)
hold on
plot(error4)
hold on
plot(error6)
hold on
plot(error8)
legend('2 channel','4 channel' ,'6 channel','8 channel')
xlabel('Distance from center [cm]')
ylabel('Angle averaged error (dB)')
title('Angle-averaged error as a function of distance from the centre for a clip containing mono source
with different number of channels')
```


Snapshot of original sound field
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-8
-6
-4
-2
0
2
4
6
8
10


Snapshot of reproduced sound field
10
8
6
4
2
0
-2
-4
-6
-8
-10
-10
-8
-6
-4
-2
0
2
4
6
8
10


Angle-averaged quantization error as a function of distance and frequency
100 Hz
1000 Hz
5000 Hz
Angle averaged error (dB)
Distance from center [cm]
5
0
-5
-10
-15
-20
-25
-30
-35
0
20
40
60
80
100
120


Angle-averaged error as a function of distance from the centre for a clip containing mono source with different number of channels
2 channel
4 channel
6 channel
8 channel
Angle averaged error (dB)
Distance from center [cm]
-16
-18
-20
-22
-24
-26
-28
0
20
40
60
80
100
120

References

- https://in.mathworks.com/matlabcentral/fileexchange/26137-aac-encoder
- https://www.researchgate.net/publication/266096161_Encoding_Higher_Order_Ambisonics_with_AAC
- https://www.geeksforgeeks.org/greedy-algorithms-set-3-huffman-coding/
- http://flo.mur.at/writings/HOA-intro.pdf