DT0117
Design tip

Microphone array beamforming in the PCM and PDM domain

By Andrea Vitali

Purpose and benefits

This design tip explains how to combine signals from multiple omnidirectional microphones to synthetize a virtual microphone that captures sound from a specific direction in space while rejecting sounds from other directions. This is also known as beamforming (BF).

Benefits:

- Learn how beamforming works, in order to optimize the layout and the frequency response of the microphone array.
- Learn how to exploit the DFSDM digital filters in order to tune PDM delays and perform beamforming on STM32L4, L4+, H7, F412/413/423 and F76/77 (AN4957, AN4990).
- Learn how to use the 2x2 micro-array of microphones present in the BlueCoin starter kit (STEVAL-BCNKT01V1).
- Understand the beamforming library AcousticBF.
- Use reference Matlab© code to simulate the performance of any configuration of microphone arrays.

Scope

Devices and product series	
IMP34DT05	MEMS audio sensor omnidirectional digital microphone for industrial applications
MP34DT05-A	MEMS audio sensor omnidirectional digital microphone
MP34DT06J	MEMS audio sensor omnidirectional digital microphone
STM32L4, L4+, H7	Complete series of microcontrollers with DFSDM peripheral
STM32F412/413/423	Complete line of microcontrollers with DFSDM peripheral
STM32F75/76/77	Microcontrollers with DFSDM peripheral
STEVAL-BCNKT01V1	BlueCoin starter kit

Introduction

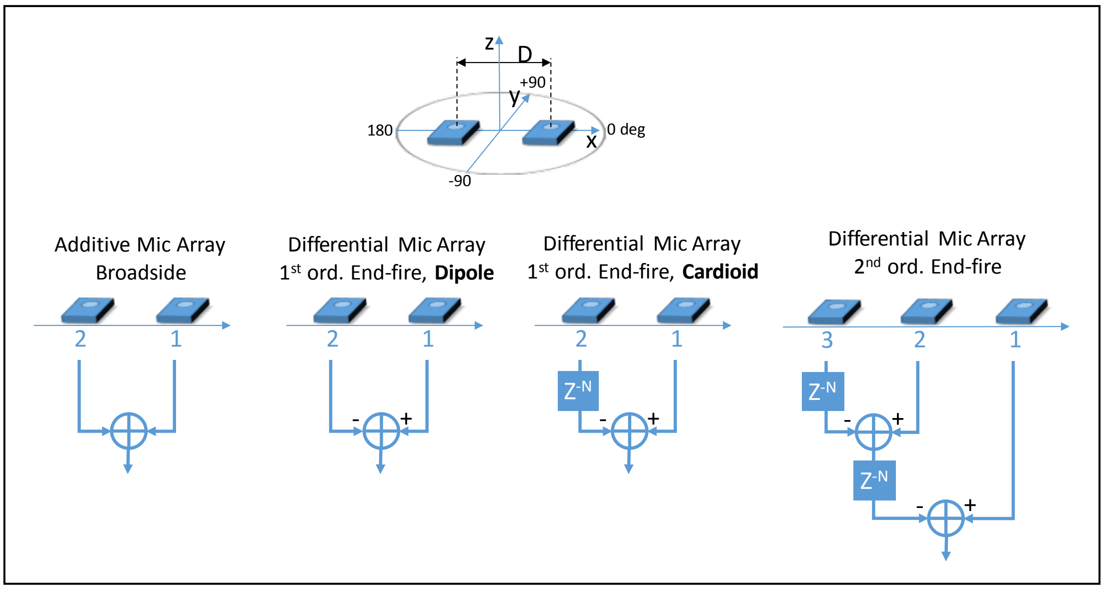
MEMS microphones have an omnidirectional response, which means they capture sound coming from all directions in space. However, it may be convenient in typical audio applications to use a directional microphone to capture the signal of interest from a specific direction while rejecting noise coming from other directions. This is especially useful when the signal of interest and the noise overlap in frequency.

By combining the signal of multiple omnidirectional microphones in a specific arrangement, one can achieve directionality. There are two main types of microphone arrays:

- Additive Microphone Arrays (AMA), also known as “Broadside” configuration because the array captures broadband sound coming from the side of the line along which the microphones are arranged.
- End-fire configuration for Differential Microphone Arrays (DMA), also known as “End-fire” configuration because the array captures the sound coming from one or both ends of the line along which the microphones are arranged.

This design tip explores Additive Microphone Arrays (AMA broadside), 1st order Differential Microphone Arrays (DMA end-fire configuration, for Dipole and Cardioid) and the 2nd order DMA. See Figure 1.

Figure 1. Linear Microphone Array configurations explored in this design tip



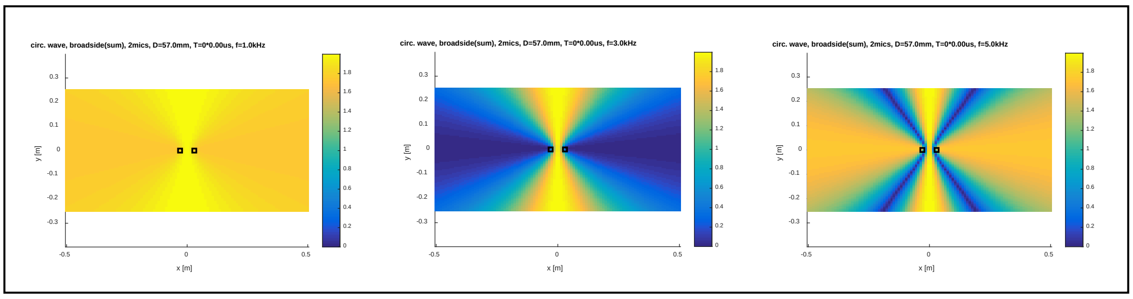
Additive Microphone Arrays (AMA), Broadside configuration

In the simplest case there are two microphones spaced by D millimeters. The output is simply summed together. This arrangement captures the sound coming from ±90 degrees. At 0 and 180 degrees the sound is rejected, but only if it has the right frequency, that is, it is rejected if it is near the frequency for which the system is tuned.

At 0 and 180 degrees, the first frequency null is f = v/(2D), where v is the speed of the sound in the air. If the system is tuned to reject 3 kHz, then the spacing must be D = v/(2f) = 57.2 mm, where f = 3 kHz and **v = 343 m/s at 20 degC**.

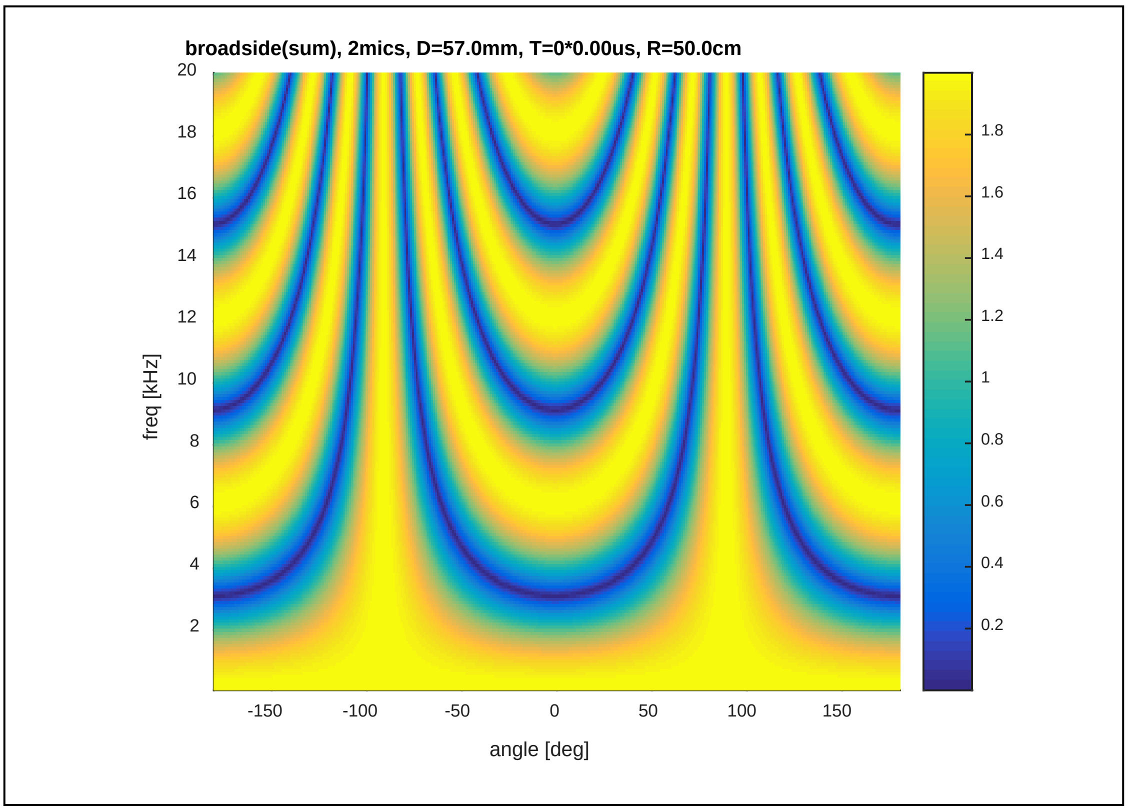
To verity the performance of the system, run the corresponding part of BFtest.m to configure the variables, set the variable f to 1, 3 and 5 kHz, then run BFsim2D.m. One can see that the system has the expected behavior only at 3 kHz. Below that frequency, the system has a weak directivity, not rejecting the sound from 0 and 180 degrees as expected. Above that frequency, the system has a different directivity: there are two frequency nulls at about ±45 and ±135 degrees. See figure 2.

Figure 2. Broadside configuration: intensity of the sound with specific frequency for every point in the plane of the board: f = 1 kHz (left), 3 kHz (center), 5 kHz (right). Output of BFsim2D.m.



Run BFsimAF.m to see what happens at every frequency and for every angle of arrival in the plane of the board. One can see that the system captures all frequencies at ±90 degrees. On the opposite, at 0 and 180 degrees, the system rejects only the frequencies near multiples of 3 kHz, the frequency for which the system was tuned. See figure 3.

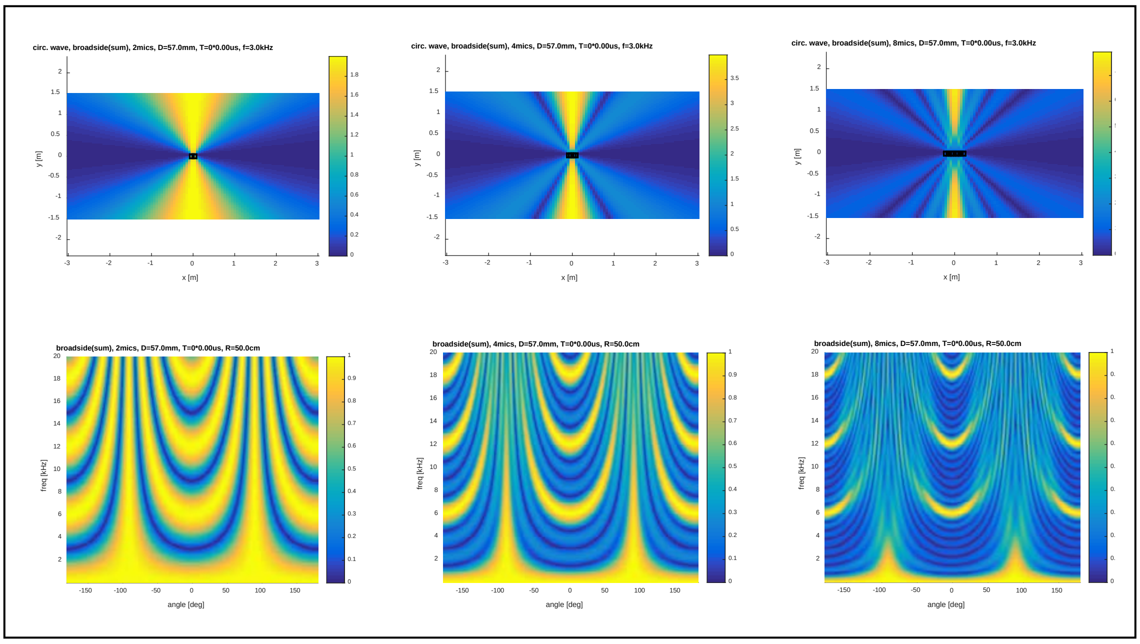
Figure 3. Broadside configuration: intensity of sound for every frequency and every angle of arrival in the plane of the board at a reference distance of 50 cm. Output of BFsimAF.m.



Broadside configuration with 2, 4 or 8 microphones

By adding more microphones, the capture cone becomes narrower and the frequency null becomes wider. Note that in the near field, the system rejects sound from all directions. One must enlarge the simulation space in BFsim2D.m to see the correct behavior (or one must look at the simulation done with planar wavefronts). See figure 4.

Figure 4. Broadside configuration with 2 microphones (left), 4 microphones (center) and 8 microphones (right). Output of BFsim2D.m (top row), normalized output of BFsimAF.m (bottom row).



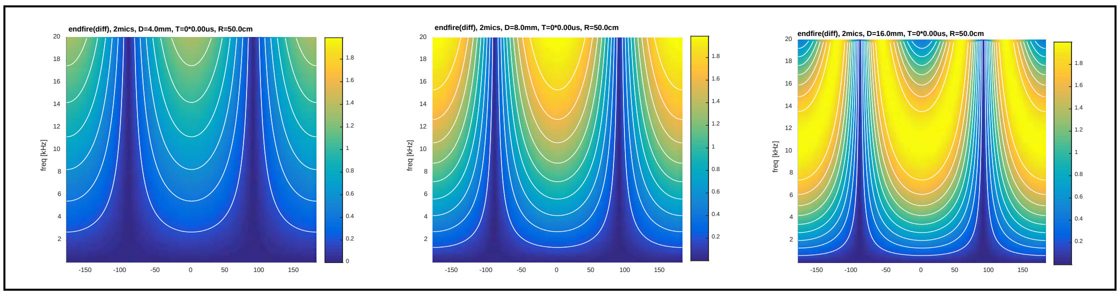
Differential Microphone Arrays (DMA), 1st order end-fire Dipole

In this case there are two microphones spaced by D millimeters. The output of the second microphone is subtracted from the output of the first microphone. This arrangement rejects the sound coming from ±90 degrees. At 0 and 180 degrees the sound is captured but only if it has the right frequency, that is, it is captured if it is near the frequency for which the system is tuned.

If the system is tuned to capture 3 kHz, the distance is D = v/(2f) = 57.2 mm. To capture an audio signal with bandwidth >= 20 kHz, the distance should be D <= 8.6 mm.

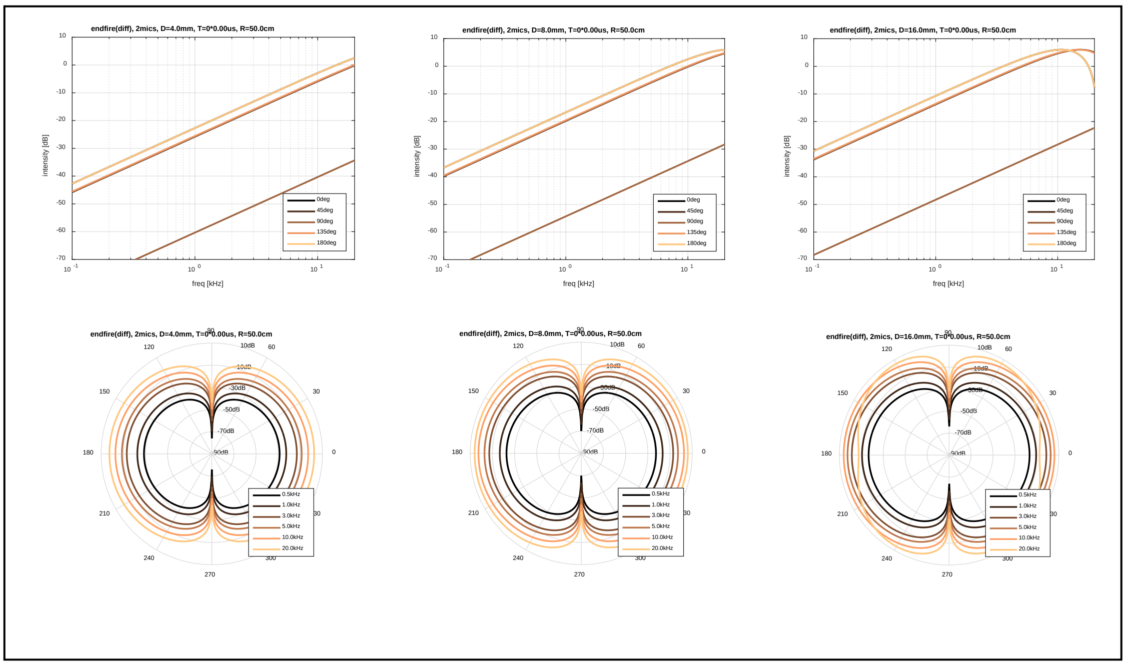
To verity the performance of the system, run the corresponding part of BFtest.m to configure the variables, set the variable D to 4, 8 and 16 mm then run BFsimAF.m. One can see that the system rejects all frequencies at ±90 degrees. On the opposite, at 0 and 180 degrees, the system captures frequencies that are near the frequency for which the system is tuned (f = 42.8, 21.4 and 10.7 kHz respectively). See figure 5.

Figure 5. First order end-fire Dipole configuration: intensity of sound for every frequency and every angle of arrival in the plane of the board at a reference distance of 50 cm; D = 4 mm (left), 8 mm (center), 16 mm (right). Output of BFsimAF.m.



The same script also creates the logarithmic plots of the intensity vs. frequency at a given angle (frequency response, vertical slice of the 2D plot), and the polar plot the intensity vs. angle at a given frequency (horizontal slice of the 2D plot wrapped in a circle). See figure 6.

Figure 6. First order end-fire Dipole configuration: logarithmic plot of intensity vs. frequency at a given angle (top row), polar plot of intensity vs. angle at a given frequency (bottom row): D = 4 mm (left), 8 mm (center), 16 mm (right). Output of BFsimAF.m.



Note that when the system is tuned for f = 42.8 kHz, the attenuation at 1 kHz is very large (-23 dB); when f = 21.4 kHz, the attenuation at 1 kHz is lower (-17 dB); when f = 10 kHz, the attenuation at 1 kHz is even lower (-11 dB). The latter tuning should be preferred if one is interested in preserving low-frequency sounds. In fact, the signal-to-noise ratio (SNR) is proportional to the attenuation. Digital amplification of attenuated frequencies (equalization) will not improve the degraded SNR. The obvious disadvantage is the larger spacing required by the microphones (D=16 mm instead of D=4 mm).

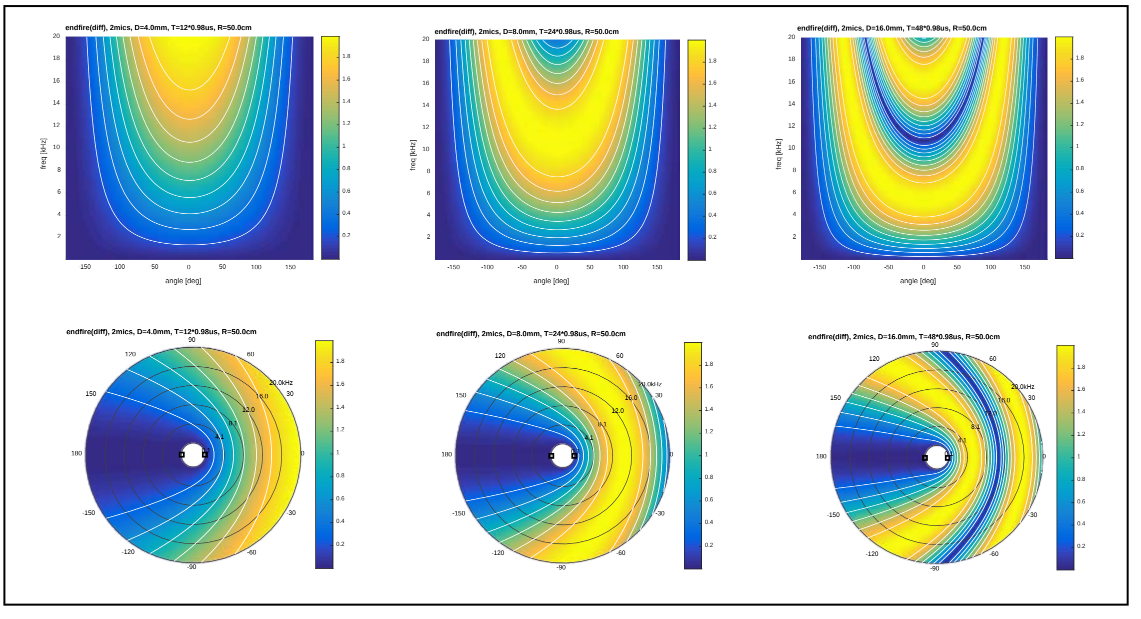
Also note that for frequencies near f, the SNR will be +3 dB higher. In fact, the noise coming from the 2 microphones in the system is not coherent and the summing makes the power double (+3 dB); when the signal is coherent, the summing makes the amplitude double and the power quadruple (+6 dB). See figure 6.

Differential Microphone Arrays (DMA), 1st order end-fire Cardioid

There are two microphones spaced by D millimeters. The output of the second microphone is delayed and then subtracted from the output of the first microphone. It is critical to match the delay T to the distance D. This arrangement rejects the sound coming from 180 degrees (side of the second microphone). At 0 degrees (side of the first microphone) the sound is captured but only if it has the right frequency, that is, it must be far from the frequency for which the system is tuned (frequency null).

At 0 degrees, the frequency null is at f = v/(2D), therefore D = v/(2f). The frequency f must be at least twice the highest frequency of interest. The corresponding delay is T = D/v. If the system is digital, it is relatively easy to delay signals by N samples. In this case T = N*Ts, where Ts = 1/Fs is the sampling interval, and Fs is the sampling frequency of the digital system.

Figure 7. First order end-fire Cardioid configuration: intensity of sound for every frequency and every angle of arrival in the plane of the board at a reference distance of 50 cm; D = 4 mm (left), 8 mm (center), 16 mm (right). Output of BFsimAF.m (top row), output of BFsimAFc.m (bottom row).

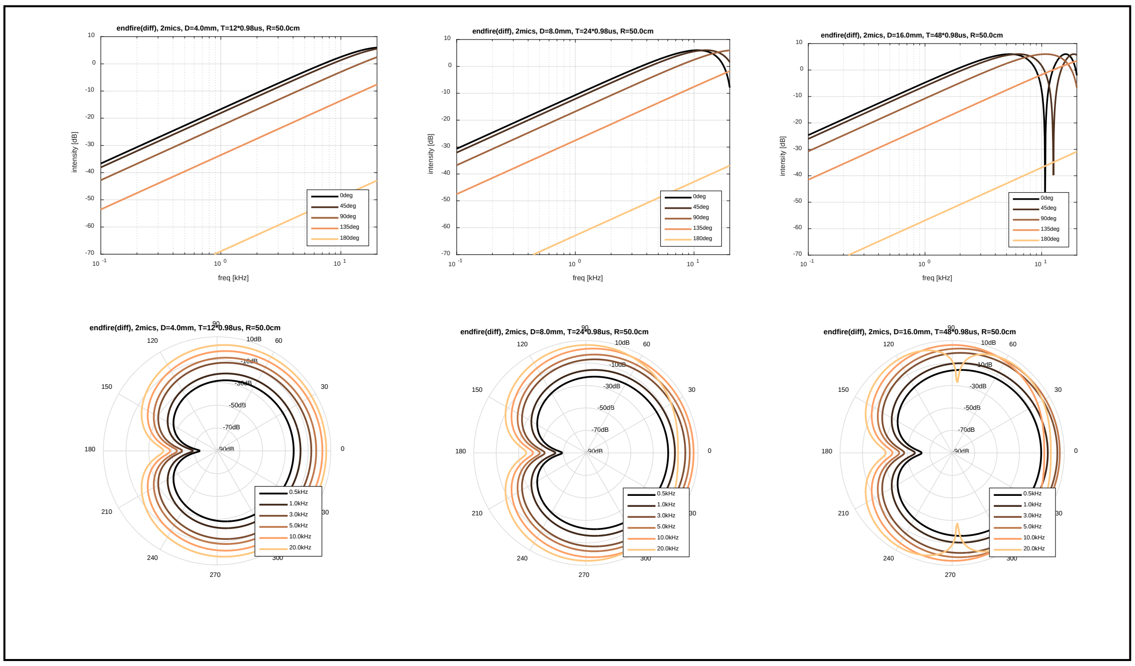


PCM audio has a typical sampling frequency in the range of 16 to 48 kHz. If Fs = 48 kHz, Ts = 20.83 µsec; with N = 1, T = 20.83 µsec, therefore D = T*v = 7.1 mm and the frequency null is at f = v/(2D) = 24 kHz; with N = 3, T = 62.5 µsec, D = 21.4 mm and f = 8 kHz. At a lower sampling frequency, a lower delay is needed to get the same distance and tuning frequency: if Fs = 16 kHz, Ts = 62.5 µsec; with N = 1, T = 62.5 µsec, D = 21.4 mm and f = 8 kHz (bandwidth of interest centered around 4 kHz).

PDM audio (Pulse Density Modulated) has a typical sampling frequency of 1 to 3 MHz. If Fs = 1.024 MHz, Ts = 0.98 µsec; with N = 12, T = 11.71 µsec, D = 4 mm and f = 42.8 kHz, high enough to enable the capture of a full bandwidth audio signal (20 kHz); with N = 24, T = 23.5 µsec, D = 8 mm and f = 21.4 kHz (bandwidth of interest centered around 10.7 kHz).

To verity the performance of the PDM system with Fs = 1.024 MHz, run the corresponding part of BFtest.m to configure the variables, set the variable D to 4, 8 and 16 mm, and set the corresponding delay T to 12*Ts, 24*Ts and 48*Ts, then run BFsimAF.m. The script BFsimAFc.m will generate a similar plot but wrapped in a circle to give a better sense of the directivity of the system. See figure 7.

Figure 8. First order end-fire Cardioid configuration: logarithmic plot of intensity vs. frequency at a given angle (top row), polar plot of intensity vs angle at a given frequency (bottom row): D = 4 mm (left), 8mm (center), 16 mm (right). Output of BFsimAF.m.



Note that when the system is tuned for f = 42.8 kHz, the attenuation at 1 kHz is very large (-17 dB); when f = 21.4 kHz, the attenuation at 1 kHz is lower (-11 dB); when f = 10 kHz, the attenuation at 1 kHz is even lower (-5 dB). The latter tuning should be preferred if one is interested in preserving low-frequency sounds. See figure 8.

Also note that for frequencies near f/2, the SNR will be +3 dB higher.

Beamforming for PDM systems using the DFSDM hardware peripheral

The advantage of PDM systems is that the high sampling frequency enables a small time delay (integer multiple of the small sampling interval), a small distance between microphones (4-8mm), and a high frequency null (42-21kHz) to capture large bandwidth audio signals. To achieve the same goals, PCM systems would require a fractional delay filter, which is very complex and expensive to implement.

PDM is especially convenient on STM32 microcontrollers because there is a dedicated hardware peripheral known as DFSDM. DFSDM stands for Digital Filter for Sigma-Delta-Modulated signals. DFSDM is the filter dedicated to PDM-to-PCM conversion. The delay of the PDM processed by DFSDM can be controlled dynamically at run-time.

Matching the distance D and the time delay T: Sub-Cardioid and Hyper-Cardioid

To get the expected performance for the Cardioid configuration, the time delay T must match the distance D between microphones. If the time delay T is larger, or the distance D is smaller, the system will not have the expected rejection at 180 degrees (Sub-Cardioid). On the contrary, if the time delay T is smaller, or the distance D is larger, the system will reject sound from two different angles evenly spaced from the 180 degree angle (Hyper-Cardioid). See figures 9, 10 and 11.

Figure 9. First order end-fire Hyper-Cardioid (left), Cardioid (center), Sub-Cardioid (right) configuration; PDM system with Fs = 1.024 MHz (top row), PCM system with Fs = 48 kHz (bottom row). Output of BFsimAF.m.

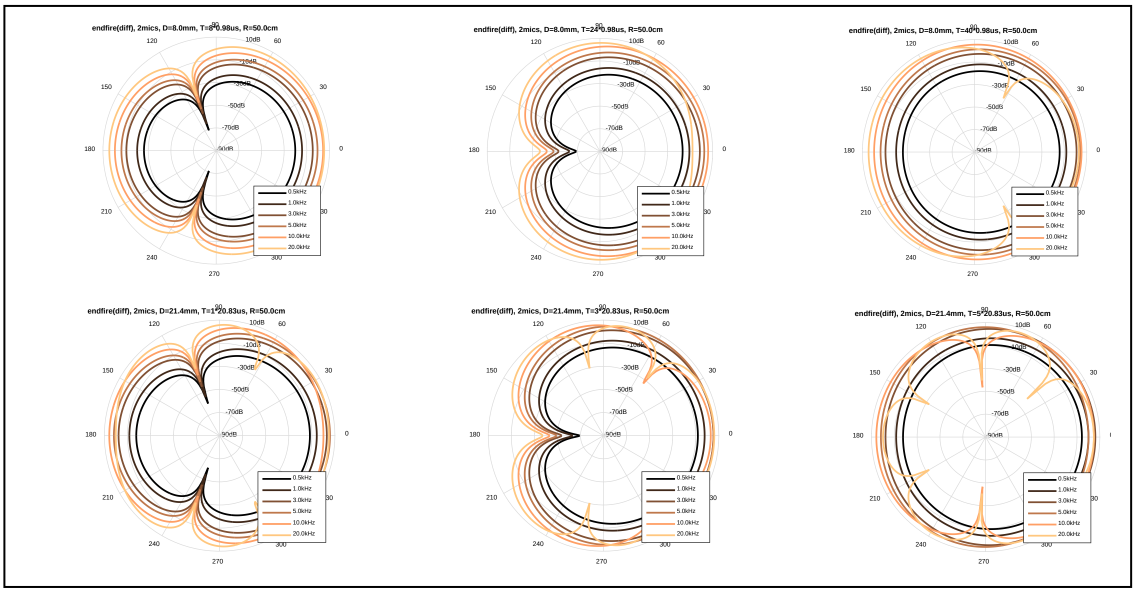


Figure 10. First order end-fire Hyper-Cardioid (left), Cardioid (center), Sub-Cardioid (right) configuration; PDM system with Fs = 1.024 MHz. Output of BFsimAFc.m.

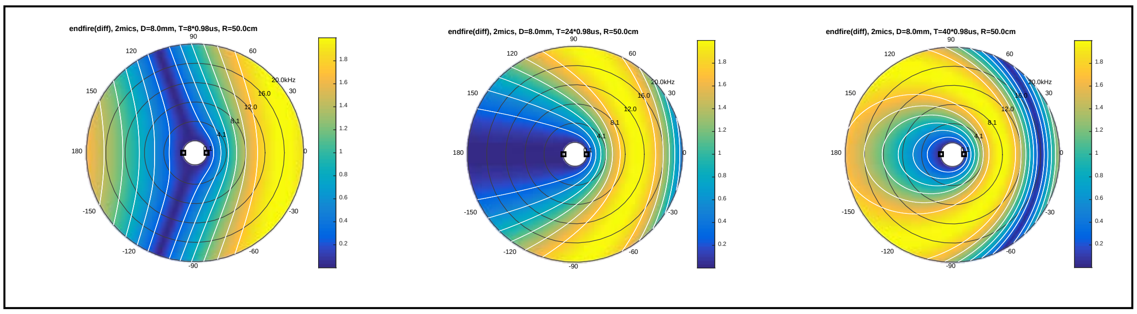
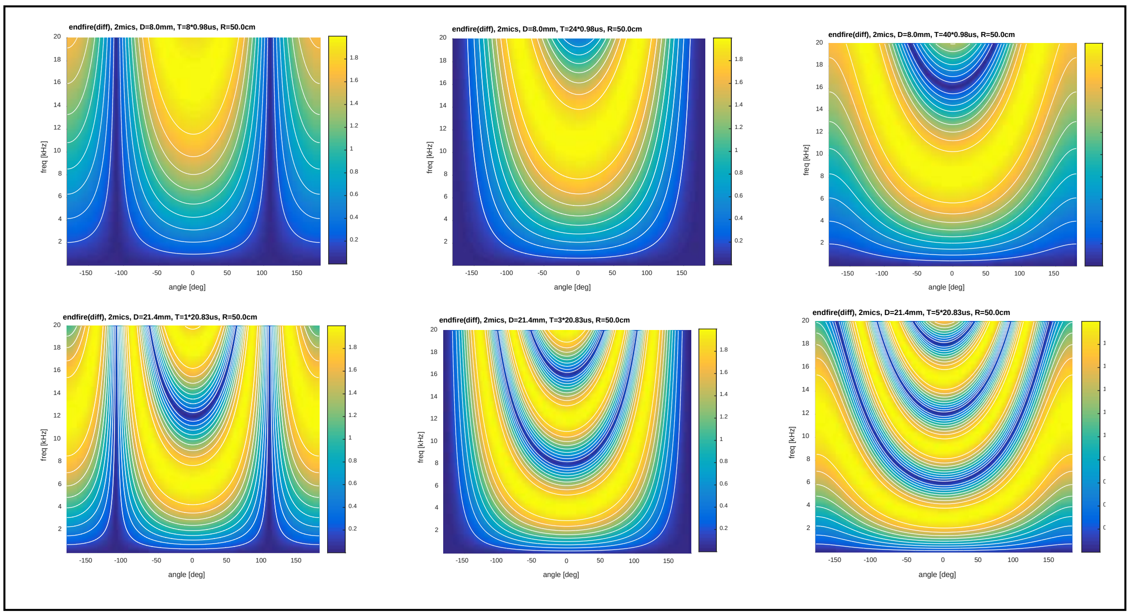


Figure 11. First order end-fire Hyper-Cardioid (left), Cardioid (center), Sub-Cardioid (right) configuration; PDM system with Fs = 1.024 MHz (top row), PCM system with Fs = 48 kHz (bottom row). Output of BFsimAF.m.



Beamforming with Acoustic BF software library: basic Cardioid and strong Cardioid

The library implements a Cardioid based on two omnidirectional MEMS microphones. Both PCM and PDM domains are supported.

- PCM domain is supported with Fs = 16 kHz and D = 21 mm.

PDM domain is supported with Fs = 1.024 MHz and D ≥ 4 mm.

The library implements a front-facing Cardioid (**basic Cardioid**) by delaying and subtracting the signal of the second microphone from the signal of the first microphone. There is the option to activate a de-noise filter (**de-noise Cardioid**), see below for details.

The library has the option to implement an enhanced Cardioid (**strong Cardioid**): in this case, both the front and the back-facing Cardioid are computed, by swapping the roles of microphones. The signal coming from the back-facing Cardioid is the noise reference. A real-time adaptive filter properly scales the noise reference so that, by subtraction, it can cancel the residual noise captured from the front-facing Cardioid. If the real-time adaptation were not present, the subtraction could result in an addition of the noise. For best results, the strong Cardioid always exploits the de-noise filter, see below for details.

The de-noise filter can be activated after the equalization. Equalization is needed to compensate for the unwanted attenuation of low frequencies, which is especially strong when the distance D between microphones is small. Attenuated frequencies will have a worse signal-to-noise ratio (SNR). The de-noise filter is designed to make this less noticeable by the human auditory system. If Automatic Speech Recognition (ASR) is used, the de-noise filter should be de-activated by selecting the correct option (basic Cardioid or **ASR-ready strong Cardioid**).

Beamforming with the 2x2 microphone array in the BlueCoin starter kit: steerable Cardioid

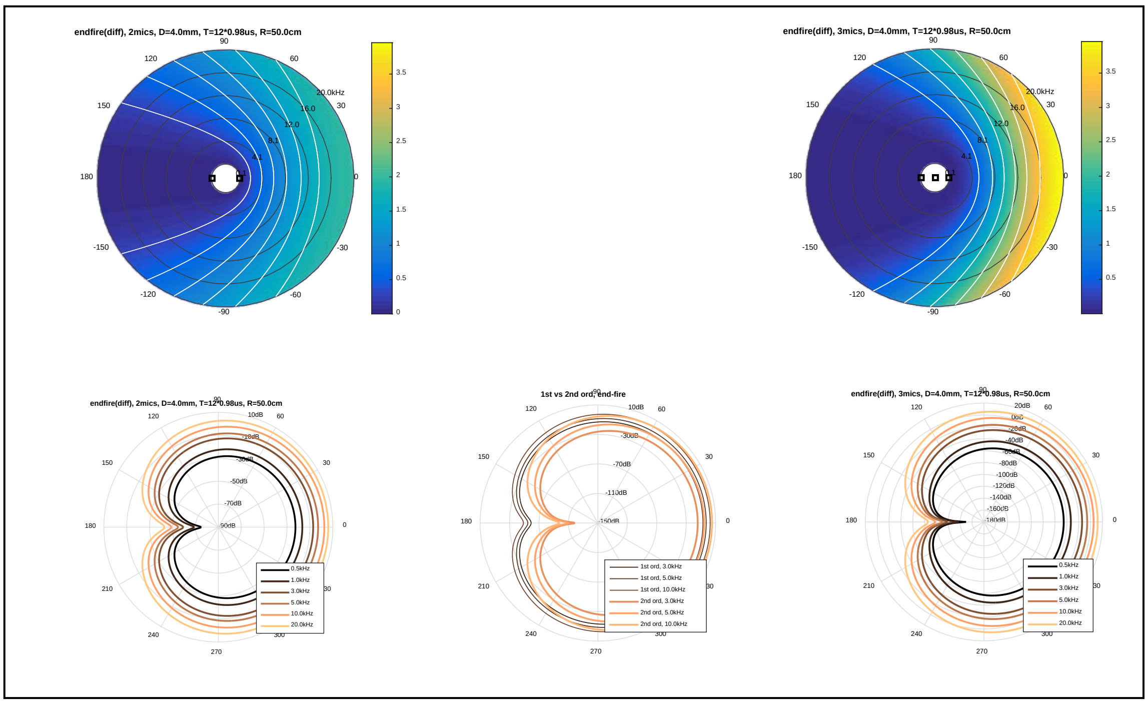
The BlueCoin starter kit (STEVAL-BCNKT01V1) has an array of 2x2 microphone with D = 4 mm spacing. The firmware can steer the Cardioid in real-time by selecting two microphones out of the four that are available. As a result, the Cardioid orientation can be changed in 45 degree steps to capture sound from 0, ±45, ±90, ±135, and 180 degrees.

Differential Microphone Arrays (DMA), 2nd order end-fire Cardioid

This is the most complex case from the processing point of view. There are three microphones arranged along a line and spaced by D millimeters. The output of the third microphone is delayed and subtracted from the output of the second, the result is again delayed and subtracted from the first microphone to compute the final output. It is critical to match the delay T to the distance D.

As for the Cardioid, this configuration rejects the sound coming from 180 degrees (side of the third microphone) and captures the sound coming from 0 degrees (side of the first microphone). With respect to the first order Cardioid, the attenuation at ±90 degrees and at 180 degrees is higher. See figure 12.

Figure 12. First order (left) and second order (right) end-fire Cardioid configuration. Output of BFsimAFc.m (top row), output of BFsimAF.m (bottom row). Comparison polar plot in the middle of the bottom row.



Comparison of Broadside, Dipole and Cardioid configuration

See figures 13, 14 and 15 for a comparison of the Broadside, Dipole and Cardioid configuration and how the system performs in the 2D and 3D space.

Figure 13. Broadside (left), Dipole (center), Cardioid (right). Output of BFsim2D.m (top row), output of BFsim3D.m (bottom row). Reference frequency f = 3 kHz.

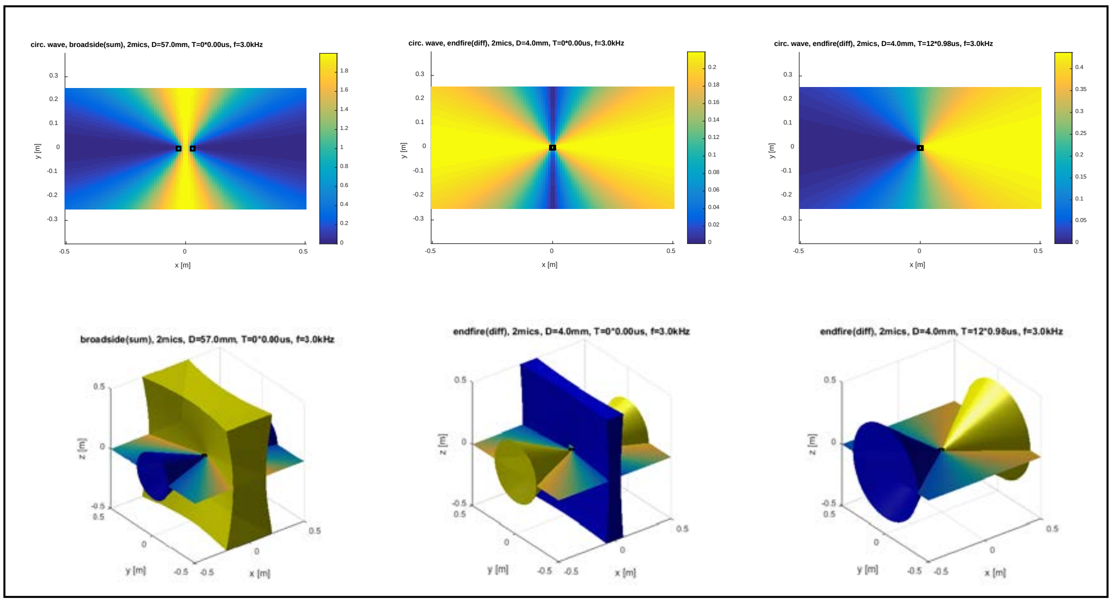


Figure 14. Broadside (left), Dipole (center), Cardioid (right). Output of BFsimAF.m (top row), output of BFsimA3.m (bottom row). Reference frequency f = 3 kHz.

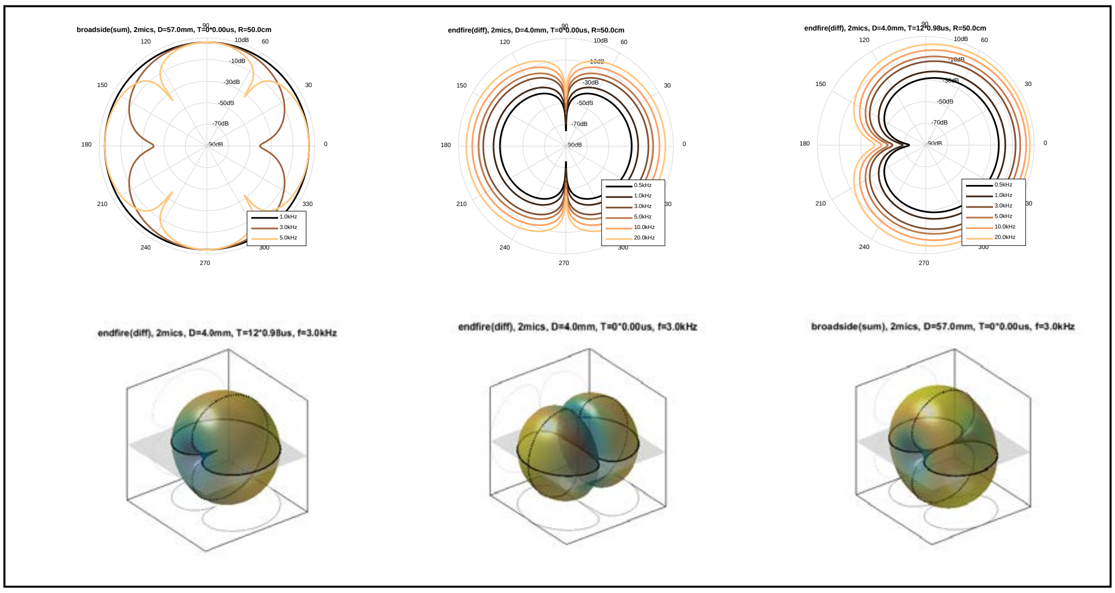
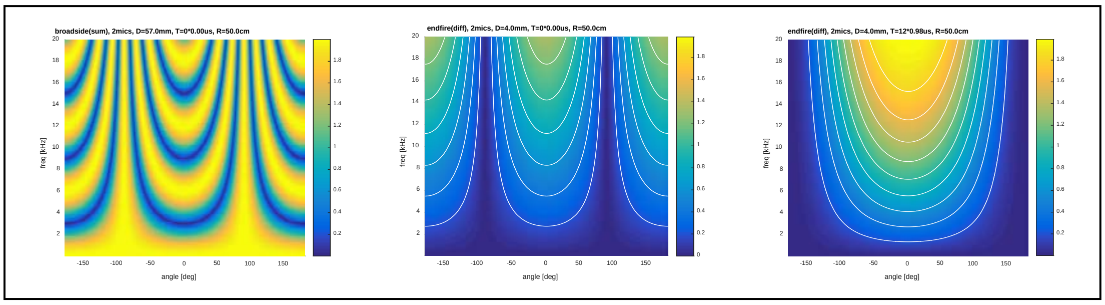


Figure 15. Broadside (left), Dipole (center), Cardioid (right). Output of BFsimAF.m.



Beamforming performance

Microphone matching

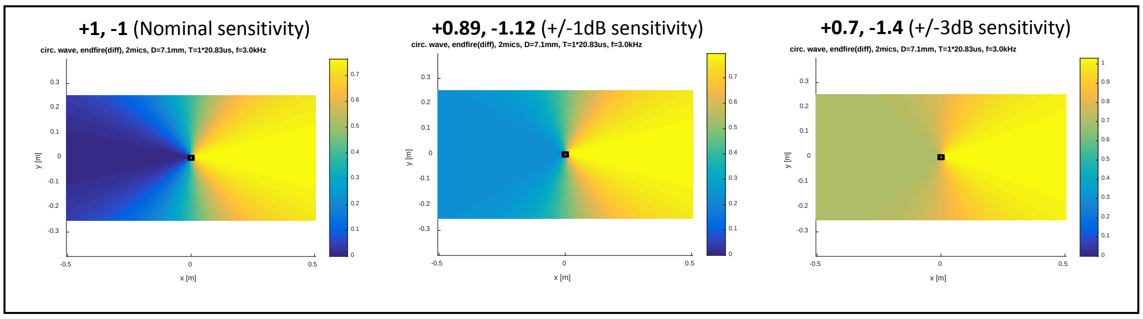
Sensitivity and frequency response of all microphones in the array should be as close as possible. If the sensitivity is not the same, weights (mw vector in the scripts) can be adjusted to correctly simulate the effect of the mismatch. To account for a specific frequency response, weights can also be a function of the frequency.

The sensitivity of a digital microphone is the percentage in dB of the full-scale output generated by a reference 94 dB SPL (1 Pa) input at 1 kHz: sensitivity [dBFS] = 20 log10(output/FullScale), where the output corresponds to the peak value of the waveform, not its RMS value. The typical sensitivity is -26 dBFS, 5% of the full scale. If the sensitivity is specified ±3 dB, a weight with nominal value W actually is in the range W * $10^{(+/-3/20)}$ = 0.7 W to 1.4 W. One can run Monte-Carlo simulations by setting weights to random values in that range.

If microphones are calibrated, by measuring their effective sensitivity at the frequency of interest, weights can be adjusted to compensate for the mismatch. As an example, if the measured sensitivity is -28 dBFS, instead of -26 dBFS as it should be, then one can correct the weight W to W * $10^{((28-26)/20)}$ = 1.2589 W to compensate for the smaller sensitivity.

If microphones cannot be calibrated, it is advised to select matched microphones where the sensitivity is as close as possible (e.g. ±1 dB instead of ±3 dB). See figure 16 to see how a mismatch in sensitivity can compromise the directivity of the system.

Figure 16. End-fire configuration for Cardioid: matched sensitivity (left), ±1 dB mismatch (center), ±3 dB mismatch (right). Output of BFsim2D.m.



Calibration can be done in real-time by assuming that the audio power captured from each microphone is the same, which is true if microphones are very close together as happens in PDM systems. The beamforming system needs to compute the audio power for each microphone and normalize the delayed outputs before the final weighted sum.

Signal-to-Noise ratio

Beamforming is essentially a weighted sum of signals, where the signals can also be delayed by a specific time delay.

If the noise of any microphone is white (uncorrelated in time) and uncorrelated with noise from other microphones (uncorrelated in space), the total output power of noise is the sum N ($\sum w_n^2$), where w_n is the weight for the n-th microphone and N is the noise power assumed equal for all microphones. The sum ($\sum w_n^2$) is the noise gain.

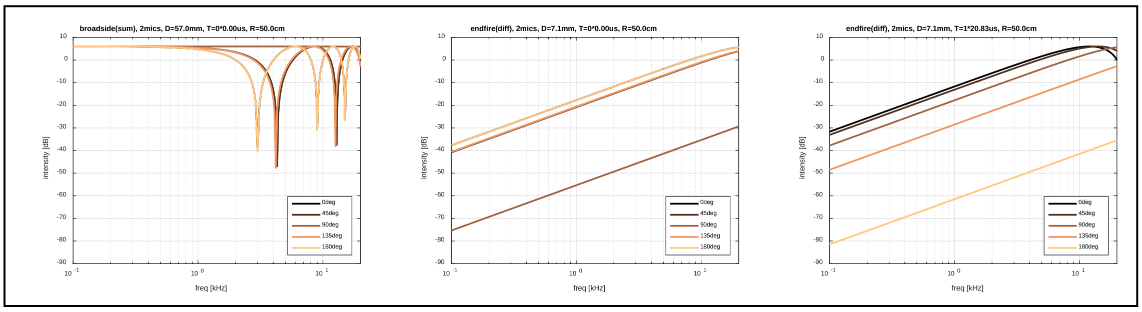
If the signal of interest is delayed so that it is summed in phase, the total output power of the signal is S $(\sum w_n)^2$, where S is the signal power. The sum $(\sum w_n)^2$ is the signal gain.

With respect to the reference SNR = S/N, the SNR of the beamforming system is multiplied by the ratio between signal and noise gain: $(\sum w_n)^2 / (\sum w_n^2)$.

As an example, for the broadside configuration with 2 microphones, the weight vector is [+1, +1] and the SNR of the system is enhanced by a factor $(1+1)^2 / (1^2+1^2) = 4/2 = 2$, that is 20 log10(2) = +3 dB. This enhancement holds for all frequencies, as the frequency response is flat at ±90 degrees. See figure 17.

For other beamforming configurations, one must look at the frequency response H(f) to find the actual signal gain at a specific frequency. See figure 17. $H(f)^2$ is the numerator to be used in the ratio mentioned above, it substitutes the sum $(\sum w_n)^2$.

Figure 17. Frequency response H(f) for Broadside (left) configuration, Dipole (center), and Cardioid (right). Output of BFsimAF.m.



As an example, for the end-fire configuration with 2 microphones, the weight vector is [+1, -1] but the SNR of the system is enhanced by +3 dB only at the frequency for which the system is tuned, in fact, at that frequency H(f) = 2. At other frequencies, the signal gain is lower. When H(f) < 1, the SNR is degraded. Boosting the output to compensate for the lower gain (as the equalization filter does) does not change the SNR because the noise is amplified together with the signal.

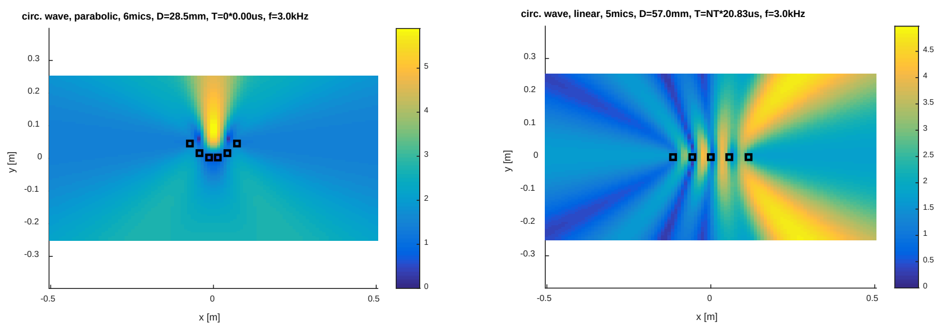
The best way to enhance the system SNR is to filter the output and remove out-of-band noise (see DT0091 and DT0092 for an efficient implementation of Lattice Wave Digital

Filters: LWDF are IIR filters, hence very short, but stable as FIR filters). This effectively mitigates the effect of the aforementioned noise gain.

Beamforming simulation accuracy

The simulation scripts presented in the following paragraphs are reasonably accurate for typical configurations where microphones are arranged in a line. However, scripts do not account for audio “shadows” or for diffraction effects. Simulations may be inaccurate for complex arrangements (e.g. with concave or convex surfaces).

Figure 18. Parabolic array (left) and steered linear array (right). Output of BFsim2D.m.



Reference Matlab code for Beamforming simulation

BFtest.m

This is the main script. In the example below it calls BFconfig.m to compute the actual parameters used in the simulation.

```
% run ONE of the following lines
BFtype=0; Nmic=2; D=0.057;  Ts=0;         NT=0;  % broadside configuration
BFtype=1; Nmic=2; D=0.004;  Ts=0;         NT=0;  % 1st ord end-fire, Dipole
BFtype=1; Nmic=2; D=0.0214; Ts=1/16e3;    NT=1;  % 1st ord end-fire, Cardioid 16kHz PCM
BFtype=1; Nmic=2; D=0.0071; Ts=1/48e3;    NT=1;  % 1st ord end-fire, Cardioid 48kHz PCM
BFtype=1; Nmic=2; D=0.004;  Ts=1/1.024e6; NT=12; % 1st ord end-fire, Cardioid 1.024MHz PDM
BFtype=1; Nmic=3; D=0.004;  Ts=1/1.024e6; NT=12; % 2nd ord end-fire, Cardioid 1.024MHz PDM

% then run the configuration utility (config=0 makes it interactive)
config=1; % 1 if vars are set here: BFtype(0-1), Nmic(1-8), D[m], Ts[s] and NT
BFconfig; % set position, weight and time delay of each microphone

% run ANY of the desired simulation scripts
BFsim2D; % 2D sim with planar/circular wavefronts at given frequency
BFsim3D; % 3D sim with circular wavefronts at given frequency
BFsimAF;  % 2D sim, plot of intensity vs angle of arrival vs frequency
BFsimAFc; % 2D sim, same as BFsimAF but plots a ring instead of a plane
BFsimA3;  % 3D sim, intensity vs angle of arrival at given frequency
```

On the opposite in the example below, all parameters are computed manually.
See figure 18.

```
% not using BFconfig.m, try one of the following blocks

% parabolic array, capture sound at focal length
Nmic=6; D=0.057/2; Ts=0; NT=0; focalLen=4;
x=([0:Nmic-1]'-(Nmic-1)/2)*D; y=x.^2/(focalLen*D); m=[x,y,zeros(Nmic,1)];
mw=ones(Nmic,1); maxgain=sum(abs(mw)); mt=zeros(Nmic,1);
f=3e3; f1=0.1e3; f2=20e3; config=1; v=343;
BFstr = sprintf('parabolic, %dmics, D=%.1fmm, T=%d*%.2fus',Nmic,D*1000,NT,Ts*1e6);

% linear array, capture sound from specific angle
Nmic=5; D=0.057; Ts=1/48e3; myAngle=pi/4; % any angle from pi/4 to 3*pi/4
```

```
x=([0:Nmic-1]'-(Nmic-1)/2)*D; z=zeros(Nmic,1); m=[x,z,z];
mw=ones(Nmic,1); maxgain=sum(abs(mw));
mt=[0:Nmic-1]'*D/v*cos(myAngle); mt=round(mt/Ts)*Ts; % delay is NT*Ts
f=3e3; f1=0.1e3; f2=20e3; config=1; v=343;
BFstr = sprintf('linear, %dmics, D=%.1fmm, T=NT*%.2fus',Nmic,D*1000,Ts*1e6);
```

BFconfig.m

This script is used to compute the position, the weight and the time delay for each microphone in the array. If the “config” variable does not exist, or it is set to zero, the script becomes interactive and prompts the user.

```
v = 343; % sound speed in air [m/s], 343 m/s at 20degC, 331 m/s at 0degC
% reference equation: D/v = 1/(2*f) = NT*Ts
if ~exist('config','var') || config==0,
  BFtype = input('BeamForming type: 0=broadside(summing), 1=endfire(differential)? ');
  Nmic = input('Number of microphones (1-8, but for endfire: 2-3)? ');
  if BFtype>0, % endfire, D = v*NT*Ts, Ts=1/Fs
    if (Nmic<2) || (Nmic>3), fprintf('Wrong number of mics!\n'); return; end;
    Fs = input('Sampling frequency (typ. 48e3 for PCM, 1e6 for PDM)? '); Ts = 1/Fs;
    f1 = 3e3; N1 = ceil(1/(2*f1*Ts)); % null is at 180deg, null freq range 3-48kHz
    f2 = 48e3; N2 = max(floor(1/(2*f2*Ts)),ceil(0.004/v/Ts)); % mic distance >= 4mm
    for NT = round(linspace(max(N2,1),N1,min(8,N1-N2))),
      fprintf('Delay NT = %d (D = %.1fmm) for null at %.1fkHz\n', ...
        NT,v*NT*Ts*1000,1/(2*NT*Ts)/1e3);
    end;
    NT = input('Delay (N samples, set null freq, typ. 2x freq of interest) ? ');
    D = v*NT*Ts; f = v/(2*D);
  else % broadside
    Ts = 0; NT = 0; % default time delay NT*Ts=0
    if (Nmic<1) || (Nmic>8), fprintf('Wrong number of mics!\n'); return; end;
    f = input('Frequency (Hz, set null freq, typ. 1x freq of interest)? ');
    D = v/(2*f);
  end
  D = input(sprintf('Microphone spacing (%.1fmm for null at %.1fkHz)? ',D*1000,f/1e3));
  D = D/1000; f = v/(2*D); fprintf('Actual null freq: %.1fkHz\n',f/1e3);
  if BFtype>0, f = f/2; end; % endfire: show peak at quarter-wavelength
  config = 1;
else % config exists
  fprintf('BFtype=%d; Nmic=%d; D=%.3f; NT=%d; Ts=1/%.4g;\n',BFtype,Nmic,D,NT,1/Ts);
end;

%    y ^
%      |
%    z .--> x
% set position, weight and time delay of each microphone
if BFtype>0, % endfire (differential),
  BFstr = 'endfire(diff)';
  if Nmic==2, % 1st order
    m = [+D/2, 0; -D/2, 0]; % XY position [m]
    mw = [+1; -1]; % weights (no need to normalize)
    mt = [ 0; NT*Ts]; % time delays [s]
  else %Nmic==3 % 2nd order
    m = [+D, 0; 0, 0; -D, 0]; % XY position [m]
    mw = [+1; -2; +1]; % weights (no need to normalize)
    mt = [ 0; NT*Ts; 2*NT*Ts]; % time delays [s]
  end
else % broadside (summing)
  BFstr = 'broadside(sum)';
  m = zeros(Nmic,2); % XY position [m]
  m(1:Nmic,1) = [0:Nmic-1]*D; % X multiple of D
  m(:,1) = m(:,1) - max(m(:,1))/2; % center around X=0
  mw = ones(Nmic,1); % weights (no need to normalize)
  mt = zeros(Nmic,1);  % time delays [s]
end
m = [m(:,1),m(:,2),zeros(size(m,1),1)]; % add 3rd coord z=0
%mw = sum(mw); % no need to normalize weights
maxgain = sum(abs(mw)); % max gain may be used to normalize plots

% other simulation parameters
f = 3e3; % ref freq, 300-3400Hz voice band, 3-4kHz min hearing threshold
f1 = 0.1e3; f2 = 20e3; % lower and upper freq bounds [Hz]
BFstr = sprintf('%s, %dmics, D=%.1fmm, T=%d*%.2fus',BFstr,Nmic,D*1000,NT,Ts*1e6);
```

BFsim2D.m

This script performs a 2D simulation: for every point in the 2D plane of the board, it simulates a circular and planar wavefront and computes the intensity of the beamforming output. Set the frequency f as desired before calling the script.

```
% BeamForming 2D simulation at fixed frequency f
% . circular wavefront from near source in 2D space
% . planar wavefront from infinitely far source in 2D space

if ~exist('config','var'), BFconfig; end;

N = 100; % subdivisions in the range
S = 0.5; % reference to define x,y range [m]
xrange = [-S,+S]; % [m]
yrange = [-S/2,+S/2]; % [m]
xv = linspace(min(xrange),max(xrange),N);
yv = linspace(min(yrange),max(yrange),N);

intMc = zeros(N); % NxN intensity matrix for circular wavefront
intMp = zeros(N); % NxN intensity matrix for planar wavefront
wT = 1/f; wl = wT*v; % wavecycle time interval [s] and wavelength [m]
mv = zeros(Nmic,1); % aux vect
mb = mean(m,1); % barycenter of mic position
for ix = 1:N,
  for iy = 1:N,
    x = xv(ix); y = yv(iy);

    % circular wavefront (accurate everywhere, including near field)
    for im = 1:Nmic,
      xd = m(im,1)-x; yd = m(im,2)-y;
      ph1 = (rem(mt(im),wT)/wT*2*pi); % phase delay from time delay
      ph2 = (rem(sqrt(xd*xd+yd*yd),wl)/wl*2*pi); % phase from distance
      mv(im) = ph1+ph2; % here +pi is equivalent to weight*(-1)
    end
    intMc(iy,ix) = abs(sum(mw.*exp(1i.*mv))); % max intensity

    % planar wavefront (accurate only in far field, NOT in near field)
    x0 = m(1,1); y0 = m(1,2);
    mv(1) = (rem(mt(1),wT)/wT*2*pi); % 1st mic is reference
    for im = 2:Nmic,
      % line through x1,y1=barycenter and x2,y2=x,y has given orientation
      % line through x1,y1=n-th mic   and x2,y2=x+xd,y+yd has same orientation
      xd = m(im,1)-mb(1); yd = m(im,2)-mb(2);
      x1 = m(im,1); y1 = m(im,2);
      x2 = x+xd; y2 = y+yd;
      A = y2-y1; B = x1-x2; %C = x2*y1-y2*x1; % coefficients for line equation, Ax+By+C=0
      Ap = -B; Bp = A; % coefficients for perpendicular line, dot=A*Ap+B*Bp=0
      C = -Ap*m(im,1)-Bp*m(im,2); % must pass through 2nd mic
      d = (Ap*x0 + Bp*y0 + C) / sqrt(Ap*Ap+Bp*Bp); % perpendicular distance, sign here matters!
      ph1 = (rem(mt(im),wT)/wT*2*pi); % phase from time delay
      ph2 = (rem(d,wl)/wl*2*pi); % phase from distance
      mv(im) = ph1+ph2; % here +pi is equivalent to weight*(-1)
    end
    intMp(iy,ix) = abs(sum(mw.*exp(1i.*mv))); % max intensity
  end
end

sc = max(max(intMc(:)),max(intMp(:))); % true max value
%sc = maxgain; % largest possible value

intMcp=intMc-intMp;
dsc=max(abs(intMcp(:))); % [-dsc +dsc] range so that 0 (no diff) is in the middle

figure; hold on;
imagesc(xv,yv,intMcp,[-dsc +dsc]); axis xy; axis equal; colorbar;
xlabel('x [m]'); ylabel('y [m]');
for im = 1:Nmic, plot(m(im,1),m(im,2),'ks','LineWidth',2); end;
plot(mb(1),mb(2),'kx','LineWidth',2);
title(sprintf('circ.-planar, %s, f=%.1fkHz',BFstr,f/1e3));

figure; hold on;
imagesc(xv,yv,intMp,[0 sc]); axis xy; axis equal; colorbar;
xlabel('x [m]'); ylabel('y [m]');
for im = 1:Nmic, plot(m(im,1),m(im,2),'ks','LineWidth',2); end;
plot(mb(1),mb(2),'kx','LineWidth',2);
title(sprintf('planar wave, %s, f=%.1fkHz',BFstr,f/1e3));

figure; hold on;
imagesc(xv,yv,intMc,[0 sc]); axis xy; axis equal; colorbar;
xlabel('x [m]'); ylabel('y [m]');
for im = 1:Nmic, plot(m(im,1),m(im,2),'ks','LineWidth',2); end;
title(sprintf('circ. wave, %s, f=%.1fkHz',BFstr,f/1e3));
```

BFsim3D.m

This script performs a 3D simulation: for every point in the 3D space, it simulates a circular wavefront and computes the intensity of the beamforming output. Set the frequency f as desired before calling the script.

```
% BeamForming 3D simulation at fixed frequency f
% circular wavefront from point source in 3D space

if ~exist('config','var'), BFconfig; end;

N = 100; % subdivisions in the range
S = 0.5; % reference to define x,y range [m]
xrange = [-S,+S]; % [m]
yrange = [-S,+S]; % [m]
zrange = [-S,+S]; % [m]
xv = linspace(min(xrange),max(xrange),N);
yv = linspace(min(yrange),max(yrange),N);
zv = linspace(min(zrange),max(zrange),N);

intM = zeros(N,N,N); % NxNxN intensity matrix for circular wavefront
wT = 1/f; wl = wT*v; % wavecycle time interval [s] and wavelength [m]
mv = zeros(Nmic,1); % aux vect
for ix = 1:N,
  for iy = 1:N,
    for iz = 1:N,
      x = xv(ix); y = yv(iy); z = zv(iz);
      % circular wavefront (accurate everywhere, including near field)
      for im = 1:Nmic,
        xd = m(im,1)-x; yd = m(im,2)-y; zd = m(im,3)-z;
        ph1 = (rem(mt(im),wT)/wT*2*pi); % phase delay from time delay
        ph2 = (rem(sqrt(xd*xd+yd*yd+zd*zd),wl)/wl*2*pi); % phase from distance
        mv(im) = ph1+ph2;
      end
      intM(iy,ix,iz) = abs(sum(mw.*exp(1i.*mv))); % max intensity
    end
  end
end

alpha=0.9;

[XV,YV,ZV]=meshgrid(xv,yv,zv);
cmax = max(intM(:)); cmin = min(intM(:)); % true range
%cmax = maxgain; cmin = 0; % largest possible range
c1 = cmin+0.9*(cmax-cmin); % isosurface to show captured directions
c2 = cmin+0.1*(cmax-cmin); % isosurface to show rejected directions

figure; hold on;
%hy = slice(XV,YV,ZV,intM,[],0,[]);
%hy.FaceAlpha = 0.5; hy.EdgeColor = 'none';
hz = slice(XV,YV,ZV,intM,[],[],0);
hz.EdgeColor = 'none';
set(gca,'CLim',[cmin cmax]);

p1 = patch(isosurface(XV,YV,ZV,intM,c1));
isonormals(XV,YV,ZV,intM,p1)
p1.FaceAlpha = alpha; p1.FaceColor = 'yellow'; p1.EdgeColor = 'none';
p1c = patch(isocaps(XV,YV,ZV,intM,c1,'above'));
p1c.FaceAlpha = alpha; p1c.FaceColor = 'yellow'; p1c.EdgeColor = 'none';

if abs(c2-c1)>0.01,
  p2 = patch(isosurface(XV,YV,ZV,intM,c2));
  isonormals(XV,YV,ZV,intM,p2)
  p2.FaceAlpha = alpha; p2.FaceColor = 'blue'; p2.EdgeColor = 'none';
  p2c = patch(isocaps(XV,YV,ZV,intM,c2,'below'));
  p2c.FaceAlpha = alpha; p2c.FaceColor = 'blue'; p2c.EdgeColor = 'none';
end

camlight left
lighting gouraud

view(3); axis equal; axis([-S +S -S +S -S +S]); grid on;
xlabel('x [m]'); ylabel('y [m]'); zlabel('z [m]');

for im = 1:Nmic, plot3(m(im,1),m(im,2),m(im,3),'ks','LineWidth',2); end;
title(sprintf('%s, f=%.1fkHz',BFstr,f/1e3));
```

BFsimAF.m

This script performs a 2D simulation: for every angle of arrival in the 2D plane of the board, and for all frequencies in the specified range, it computes the intensity at a reference distance R. This script uses two auxiliary scripts: mypolarsetup.m and mypolar.m. Set f1 and f2 as desired before calling the script.

```
% BeamForming 2D simulation
% circular wavefront from point source on circumference with radius R
% plot surface to show intensity at given angle of arrival and frequency
% . extract frequency response H(f) at given angle of arrival
% . extract response as function of angle H(a) at given frequency

if ~exist('config','var'), BFconfig; end;

R = 0.5; % reference radius [m], intensity computed at this distance
avect = linspace(-pi,+pi,360); % angle vector, direction of arrival [rad]
fvect = linspace(f1,f2,200); % frequency vector [Hz]
NA = length(avect);
NF = length(fvect);

intM = zeros(NF,NA); % intensity matrix
mv = zeros(Nmic,1); % aux vect
for i = 1:NF,
  for ia = 1:NA,
    ft = fvect(i); wTt = 1/ft; wlt = wTt*v; % wavelength [m]
    a = avect(ia); x = R*cos(a); y = R*sin(a);

    % circular wavefront (accurate everywhere, including near field)
    for im = 1:Nmic,
      xd = m(im,1)-x; yd = m(im,2)-y;
      ph1 = (rem(mt(im),wTt)/wTt*2*pi); % phase delay from time delay
      ph2 = (rem(sqrt(xd*xd+yd*yd),wlt)/wlt*2*pi); % phase from distance
      mv(im) = ph1+ph2; % here +pi is equivalent to weight*(-1)
    end
    intM(i,ia) = abs(sum(mw.*exp(1i.*mv))); % max intensity
  end
end

%sc = max(intM(:)); % true max value
sc = maxgain; % largest possible value
%intM=intM/sc; % normalize
intMdB = 20*log10(intM);
dBmax = ceil(max(intMdB(:))/10)*10;
dBmin = floor(min(intMdB(:))/10)*10;
dBnr = round((dBmax-dBmin)/20)+1;

% 3D plot (copy figure bitmap)
figure;  hold on;
h = surf(avect*180/pi,fvect/1e3,intM);
h.FaceAlpha = 0.9; h.FaceColor = 'interp'; h.EdgeColor = 'none';
camlight left; lighting gouraud
contour3(avect*180/pi,fvect/1e3,intM);
grid on; zoom on; view(3);
xlabel('angle [deg]'); ylabel('freq [kHz]'); zlabel('intensity');
title(sprintf('%s, R=%.1fcm',BFstr,R*100));

% 2D plot
figure; hold on;
imagesc(avect*180/pi,fvect/1e3,intM); axis xy; axis tight; colorbar;
contour(avect*180/pi,fvect/1e3,intM,'Color',[1 1 1]);
xlabel('angle [deg]'); ylabel('freq [kHz]'); zlabel('intensity');
title(sprintf('%s, R=%.1fcm',BFstr,R*100));

% log plot of frequency response at given angle
a2vect = [0, 45, 90, 135, 180]*pi/180;
L = length(a2vect); cmap = copper(L); labels = {};
figure;
for i = 1:L,
  j = find(avect>=a2vect(i));
  if ~isempty(j),
    j=j(1);
    semilogx(fvect/1e3,20*log10(intM(:,j)),'Color',cmap(i,:),'LineWidth',2); hold on;
    labels{length(labels)+1} = sprintf('%.0fdeg',a2vect(i)*180/pi);
  end
end
legend(labels,'Location','southeast');
xlabel('freq [kHz]'); ylabel('intensity [dB]');
title(sprintf('%s, R=%.1fcm',BFstr,R*100));
axis([f1/1e3 f2/1e3 dBmin dBmax]); grid on; zoom on;

% polar plot of angle response at given frequency
% dBmin = -90; dBmax = +10; dBnr = 6; % override dB range
f2vect = [0.5 1 3 5 10 20]*1e3;
L = length(f2vect); cmap = copper(L); labels = {}; handles = [];
mypolarsetup([],linspace(dBmin,dBmax,dBnr),'%.0fdB');
for i = 1:L,
  j = find(fvect>=f2vect(i));
  if ~isempty(j),
    j=j(1);
    h = mypolar(avect,intMdB(j,:),[dBmin dBmax]);
    h.Color = cmap(i,:); h.LineWidth = 2;
    handles(length(handles)+1) = h;
    labels{length(labels)+1} = sprintf('%.1fkHz',f2vect(i)/1e3);
  end
end
legend(handles,labels,'Location','southeast');
title(sprintf('%s, R=%.1fcm',BFstr,R*100));
```

BFsimAFc.m

This scripts is the same as BFsimAF.m except that the plot is a ring instead of a plane. The radius corresponds to a given frequency, the angle corresponds to the angle of arrival.

```
% BeamForming 2D simulation
% circular wavefront from point source on circumference with radius R
% plot ring to show intensity at given angle of arrival and frequency

if ~exist('config','var'), BFconfig; end;

N = 5*100; % subdivisions in the range
S = max(abs(m(:,1)))*10; % reference to define x,y range [m]
xrange = [-S,+S]; % [m]
yrange = [-S,+S]; % [m]
xv = linspace(min(xrange),max(xrange),N);
yv = linspace(min(yrange),max(yrange),N);

mb=mean(m,1); % barycenter of mic position
d = sqrt(m(:,1).^2+m(:,2).^2); % distance origin-mic
XR = [xrange; 0 0]; YR = [0 0; yrange];
DR = sqrt(XR.^2+YR.^2); % distance origin-middlepoints
R1 = 1.05*max(d); % min radius correspond to f1
R2 = 0.95*min(DR(:)); % max radius correspond to f2
R = 0.5; % reference radius [m], intensity computed at this distance

intM = NaN(N); intA = zeros(N); % intensity and alpha matrix
mv = zeros(Nmic,1); % aux vect
for ix = 1:N,
  for iy = 1:N,
    x = xv(ix); y = yv(iy); % x,y -> freq,angle
    d = sqrt(x*x+y*y);
    if d<R1, continue; end;
    if d>R2, continue; end;
    ft = f1 + (f2-f1)*(d-R1)/(R2-R1); wTt = 1/ft; wlt = wTt*v; % wavelength [m]
    a = atan2(y,x); % angle -> x,y
    x = R*cos(a); y = R*sin(a);

    % circular wavefront (accurate everywhere, including near field)
    for im = 1:Nmic,
      xd = m(im,1)-x; yd = m(im,2)-y;
      ph1 = (rem(mt(im),wTt)/wTt*2*pi); % phase delay from time delay
      ph2 = (rem(sqrt(xd*xd+yd*yd),wlt)/wlt*2*pi); % phase from distance
      mv(im) = ph1+ph2; % here +pi is equivalent to weight*(-1)
    end
    intM(iy,ix) = abs(sum(mw.*exp(1i.*mv))); % max intensity
    intA(iy,ix) = 1;
  end
end

%sc = max(intM(:)); % true max value
sc = maxgain; % largest possible value
%intM=intM/sc; % normalize

figure; hold on;
h = imagesc(xv,yv,intM); set(h,'AlphaData',intA); axis xy; axis equal; colorbar;
for r = linspace(R1,R2,6),
  h = polar(linspace(0,2*pi,50),r*ones(1,50),'b-'); set(h,'Color',[0.25 0.25 0.25]);
end
contour(xv,yv,intM,6,'Color',[1 1 1]);
for r = linspace(R1,R2,6),
  rf = f1 + (f2-f1)*(r-R1)/(R2-R1); if r<R2, sunit=''; else sunit='kHz'; end
  text(r*cos(pi/4),r*sin(pi/4),sprintf('\n%.1f%s',rf/1e3,sunit));
end
h = gca; set(h,'XTick',[]); set(h,'YTick',[]); set(h,'YColor',[1 1 1]); set(h,'XColor',[1 1 1])
for im = 1:Nmic, plot(m(im,1),m(im,2),'ks','LineWidth',2); end;
for a = +pi : -pi/6 : -pi+0.001,
  if cos(a)>0, hstr='left'; else hstr='right'; end; if abs(cos(a))<0.01, hstr='center'; end;
  if sin(a)>0, vstr='bottom'; else vstr='top'; end; if abs(sin(a))<0.01, vstr='middle'; end;
  h=text('Position',[R2*cos(a),R2*sin(a),0],'String',sprintf('%.0f',a*180/pi), ...
         'HorizontalAlignment',hstr,'VerticalAlignment',vstr);
end
title(sprintf('%s, R=%.1fcm',BFstr,R*100));
```

BFsimA3.m

This script performs a 3D simulation: for every angle of arrival in the 3D space, for the specified frequency, it computes the intensity at a reference distance R. Set f as desired before calling the script.

```
% BeamForming 3D simulation at fixed frequency f
% circular wavefront from point source on circumference with radius R
% plot surface to show intensity at given angle of arrival

if ~exist('config','var'), BFconfig; end;
```

```
N = 100; % subdivisions in the range, should be even
R = 0.5; % reference radius [m], intensity computed at this distance

[Xm,Ym,Zm]=sphere(N); Xm=Xm*R; Ym=Ym*R; Zm=Zm*R;
intM=zeros(N+1);
wT = 1/f; wl = wT*v; % wavecycle time interval [s] and wavelength [m]
mv = zeros(Nmic,1); % aux vect
for ix = 1:N+1,
  for iy = 1:N+1,
    x = Xm(ix,iy); y = Ym(ix,iy); z = Zm(ix,iy);
      % circular wavefront (accurate everywhere, including near field)
      for im = 1:Nmic,
        xd = m(im,1)-x; yd = m(im,2)-y; zd = m(im,3)-z;
        ph1 = (rem(mt(im),wT)/wT*2*pi); % phase delay from time delay
        ph2 = (rem(sqrt(xd*xd+yd*yd+zd*zd),wl)/wl*2*pi); % phase from distance
        mv(im) = ph1+ph2;
      end
      intM(ix,iy) = 20*log10(abs(sum(mw.*exp(1i.*mv)))); % max intensity in dB
  end
end
i = isfinite(intM(:)); intM=intM-min(intM(i)); intM(~i)=0;
dBmax=ceil(max(intM(:))/10)*10;
intMX = Xm.*intM/R; intMY = Ym.*intM/R; intMZ = Zm.*intM/R;
axv=linspace(-dBmax,+dBmax,2); [axM,~]=meshgrid(axv,axv); axM1=ones(length(axv));

i=round(linspace(0,1,256).^2*255)+1; % increase visibility of bluish tones (attenuation)
figure; hold on; cmap=parula(256); cmap=cmap(i,:); colormap(cmap);
h = surf(intMX,intMY,intMZ,intM+dBmax); axis equal; axis off
h.FaceAlpha = 0.6; h.FaceColor = 'interp'; h.EdgeColor = 'none';
h = patch([-dBmax -dBmax +dBmax +dBmax],[-dBmax +dBmax +dBmax -dBmax],[0 0 0 0],0);
h.FaceAlpha = 0.4; h.FaceColor = [0.85 0.85 0.85]; h.EdgeColor = 'none';

mygray = [0 0 0];
surf(axM,axM',-dBmax*axM1,0,'EdgeColor',mygray,'FaceColor','w');
surf(axM,+dBmax*axM1,axM',0,'EdgeColor',mygray,'FaceColor','w');
surf(+dBmax*axM1,axM,axM',0,'EdgeColor',mygray,'FaceColor','w');

N2 = N/2+1; mygray = [0.5 0.5 0.5];
plot3(intMX(N2,:),intMY(N2,:),intMZ(N2,:),'k-','LineWidth',2); % XY plane
plot3(intMX(N2,:),intMY(N2,:),intMZ(N2,:)-dBmax,'Color',mygray); % XY plane, projection
plot3(intMX(:,1),intMY(:,1),intMZ(:,1),'k:','LineWidth',2); % XZ plane, X<0
plot3(intMX(:,N2),intMY(:,N2),intMZ(:,N2),'k:','LineWidth',2); % XZ plane, X>0
plot3(intMX(:,1),intMY(:,1)+dBmax,intMZ(:,1),':','Color',mygray); % XZ plane, X<0, projection
plot3(intMX(:,N2),intMY(:,N2)+dBmax,intMZ(:,N2),':','Color',mygray); % XZ plane, X>0, projection

camlight left; lighting gouraud; view(3);
title(sprintf('%s, f=%.1fkHz',BFstr,f/1e3));
```

mypolarsetup.m

```
function mypolarsetup(at,rt,fmt) % angle and radius ticks
  if isempty(at), at = linspace(0,330,12)*pi/180; end
  if isempty(rt), rt = linspace(0,1,6); end
  N = 100; a = linspace(0,2*pi,N); ax = pi*75/180;
  figure; axis off; axis equal; axis([-1 1 -1 1]); hold on;
  rtmax = max(rt); rtmin = min(rt); rtn = (rt-rtmin)/(rtmax-rtmin);
  p = patch(cos(a),sin(a),[1 1 1]); p.EdgeColor = 'none';
  for i = 1:length(rt),
    line(cos(a)*rtn(i),sin(a)*rtn(i),'Color',[0.85 0.85 0.85]);
    text(cos(ax)*rtn(i),sin(ax)*rtn(i),sprintf(fmt,rt(i)));
  end
  for i = 1:length(at),
    line([0 cos(at(i))],[0 sin(at(i))],'Color',[0.85 0.85 0.85]);
    t = text(1.1*cos(at(i)),1.1*sin(at(i)),sprintf('%3.0f',at(i)*180/pi));
    t.HorizontalAlignment = 'center';
    t.VerticalAlignment = 'middle';
  end
end
```

mypolar.m

```
function h = mypolar(a,r,sc) % angle vect, radius vect, scaling [rmin rmax]
  rmin = min(sc); rmax = max(sc); rn = (r-rmin)/(rmax-rmin);
  rn = max(0,rn); % avoid mirroring around center
  h = plot(cos(a).*rn,sin(a).*rn); hold on;
end
```

Support material

Related design support material
Databrief DB3258, STEVAL-BCNKT01V1, BlueCoin starter kit
Function pack, FP-AUD-SMARTMIC1, STM32 ODE function pack for MEMS microphone acquisition, advanced audio processing and audio output
Documentation
User manual, UM2240, Getting started with STEVAL-BCNKT01V1 BlueCoin kit: augmented acoustic and motion sensing development platform
User manual, UM2219, Getting started with STM32 ODE function pack for MEMS microphones acquisition, advanced audio processing and audio output
User manual, UM2214, Getting started with AcousticBF real-time beamforming middleware
Application note, AN3998, PDM audio software decoding on STM32 microcontrollers
Application note, AN4957, How to synchronize the DFSDM filters and how to program the pulse skipper on the STM32F413/423 line device
Application note, AN4990, Getting started with sigma-delta digital interface on applicable STM32 microcontrollers
Application note, AN5027, Interfacing PDM digital microphones using STM32-bit ARM© Cortex© MCUs
Design Tip, DT0091, Lattice wave digital filter design and automatic C code generation
Design Tip, DT0092, Lattice wave digital filter test and performance verification

Revision history

Date	Version	Changes
18-Sep-2018	1	Initial release