UMR IRISA

Inria
inventeurs du monde numérique

Multi-channel BSS Locate User Guide V 2.0

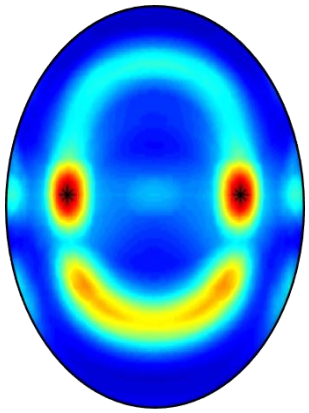


Table of Content

1 Generality

1.1 Document history

Version	Date	Authors	Comments
1.0	25/11/2015	Ewen Camberlein Romain Lebarbenchon	Initial release
2.0	11/09/2018	Ewen Camberlein Romain Lebarbenchon	New API (allowing wiener filtering of sources, block processing approach, new results evaluation figures) and new examples

1.2 References

Object	Version	Author	Comments
Multi-channel BSS Locate	2.0	Ewen Camberlein Romain Lebarbenchon	Related software
BSS Locate	2.0	C. Blandin E. Vincent A. Ozerov	http://bass-db.gforge.inria.fr/bss_locate/
Roomsimove	-	E. Vincent DR. Campbell	http://www.irisa.fr/metiss/members/evincent/Roomsimove.zip

1.3 Document purpose

This document describes the functionalities that can be realized through Multi-channel BSS Locate software (Matlab implementation) along with some explanations about how it is achieved.

1.4 Document organization

Section 2 an overview of Multi-channel BSS Locate functionalities;

Section 3 describes the API of the different tools present in this release;

Section 4 explains the available examples;

Section 5 describes how contributions of all microphones pairs are aggregated for localization purpose.

2 Multi-channel BSS Locate overview

This software aims at localizing audio sources in 3D audio scene based on recorded signals using a small array of N microphones (“small array” means that sources have to be outside of the space defined by all microphones). Results are expressed in terms of directions (azimuth and elevation) using the centroid of this microphones array as reference.

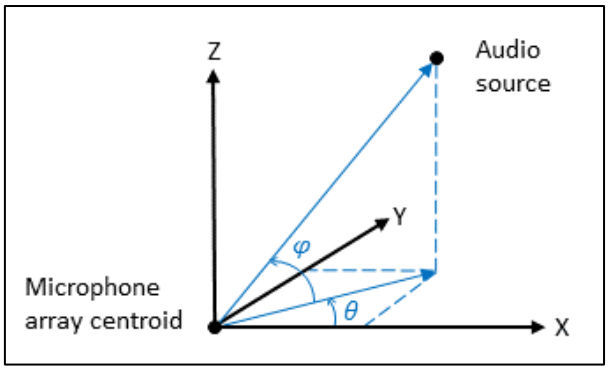


Figure 1: Azimuth (θ) and elevation (φ) of one source in the microphone array referential

Localization is achieved under far field assumption based on a choice between several local angular spectrum methods. Chosen method is applied to each microphone pair and resulting contributions (of all microphones pairs) are then aggregated following the algorithm described in section 5.3. Available local angular spectrum methods are those used in BSS Locate Toolbox [1]: GCC-PHAT, GCC NON LIN, MVDR, MVDRW, DS, DSW and DNM. No clustering based methods is available in Multi-channel BSS Locate software.

As this software is implemented for 3D audio scenes, a minimum of four microphones not located in the same plane should be used to avoid ambiguities that could lead to bad direction estimations. The toolbox covers the following use cases:

- Single or multiple source(s) localization;
- Static or moving source(s) localization;
- Static or moving microphone array processing;
- Source localization enhancement under noise condition

These use cases are illustrated with the help of example scripts released with the toolbox (see section 4). Examples are based both on provided audio mixture and/or simulated mixture. The latter is achieved thanks to the Roomsimove software [3] based on following information:

- Room configuration (empty shoebox reverberant room): dimensions and frequency dependent absorption coefficients of walls ;
- Microphones positions, directivity and orientation;
- Source positions;
- Close field recordings of the sources.

Currently, the toolbox does not provide any tracking information for the multiple sources scenario, this should be one of the functionalities added in future releases.

3 API description

Functionalities of this software can be achieved through the functions located in following folders under *mFiles* directory:

- *Localization tools*: Multi-channel BSS Locate functions;
- *Evaluation tools*: functions to evaluate estimated directions of sources vs ground truth (BSS Locate [1] evaluation functions adapted to this software);

- *Roomsimove tools*: functions for simulated convolutive audio mixture computation (Roomsimove toolbox [3] functions adapted to this software).

The API of each tool (functions) is described below.

3.1 Localization_tools

The *localization_tools* folder contains all functions needed for source localization processing. We only describes in this section the three useful functions required by the user:

- MBSS_InputParam2Struct()
- MBSS_locate_spec()

3.1.1 MBSS_InputParam2Struct()

sMBSSParam = MBSS_InputParam2Struct (angularSpectrumMeth, c, fftSize_sec, blockDuration, blockOverlap, pooling, azBound, elBound, gridRes, alphaRes, minAngle, nsrc, fs, normalizeSpecInst, specDisplay, enableWienerFiltering, wienerMode, freqRange, micPos, isArrayMoving, subArray, sceneTimeStamps, angularSpectrumDebug)

Description:

This function is dedicated to check and transform the user localization tuning and microphone position parameters into a structure used by *MBSS_locate_spec().*

Mandatory (i.e. different from empty matrix) input parameters:

Parameter name	Size / type	Description
nsrc	1 x 1 scalar	Number of sources to be located
fs	1 x 1 scalar	Sampling frequency of processed signal (in Hertz)
micPos	3 x N x T scalar	Cartesian positions of the N microphones over time T
isArrayMoving	1 x 1 scalar	Flag to set if the microphone array is moving

Optional input parameters:

If one of the following parameter is set to the “empty” value, a default value is applied.

Parameter name	Size / type	Description
angularSpectrumMeth	String or empty	angular spectrum method : ‘GCC-PHAT’, ‘GCC-NONLIN’, ‘MVDR’, ‘MVDRW’, ‘DS’, ‘DSW’, ‘DNM’. Default : ‘GCC-PHAT’
speedOfSound	1 x 1 scalar or empty	Speed of sound (m/sec) Default: 343
fftSize_sec	1 x 1 scalar or empty	Frame size / FFT size expressed in seconds - Default: 0.064 sec

blockDuration_sec	1 x 1 scalar or empty	Analysis window duration (in seconds). Default: inf (entire signal)
blockOverlap_percent	1 x1 scalar or empty	Requested Block overlap in % Default: 0 (no overlap)
pooling	String or empty	Pooling function, 'max' or 'sum'. Default: 'max'
azBound[1]	1 x 2 or 1 x 1 or empty	Azimuth boundaries to define the grid search (in degrees). Default: [-179 180]
elBound[1]	1 x2 or 1 x 1 or empty	Elevation boundaries to define the grid search (in degrees). Default : [-90 90]
gridRes	1 x 1 scalar or empty	Grid search resolution (in degrees). Default: 1
alphaRes	1 x 1 scalar or empty	Sampling resolution applied to each microphone pair referential (in degrees) Default: 5
minAngle	1 x 1 scalar or empty	Minimum distance (in degrees) between two peaks. Default: 1
applySpectInstNormalization	1 x 1 scalar or empty	Flag used to enable instantaneous local angular spectra normalization. Default: 0
specDisplay	1 x 1 scalar or empty	Flag used to display the global angular spectrum and directions of sources found. Default: 0
enableWienerFiltering	1 x 1 scalar or empty	Flag used to enable Wiener filtering to attenuate or emphases a source signal into the mixture used for localization. Require an excerpt of the source signal. Default: 0
wienerMode	String or empty	Selected Wiener mode if *enableWienerFiltering* flag is set. In the later case, this parameter controls how the signal excerpt provided to *MBSS_locate_spec* is used. This parameter could be 'Attenuation' or 'Emphasis'. - Attenuation: the signal corresponding to the provided excerpt is attenuated into the mixture in order to enhance the other source(s) signals - Emphasis: the signal corresponding to the provided excerpt is emphasized into the mixture in order to be better localized Default: 'Attenuation'
freqRange	1 x 2 or empty	Frequency range (minimum and maximum value in Hertz) to sum the local angular spectra. Default: [0 fs/2]
subArray	1 x K or empty	Vector containing indices of used microphones. If empty, all microphones are used
sceneTimeStamps	1 x T or empty	Microphone position time stamps in seconds. Must be defined if the third dimension of *micPos* differs from 1.

angularSpectrumDebug[2]	1 x 1 scalar or empty	Flag to enable additional visualization to debug the angular spectrum aggregation. Default: 0

(1) *azBound* and *elBound* parameters can lead to reduce the overall processing time by adding information known by the user. For example :

- As an example, If you know that the sources cannot be located at a negative elevation, so you can reduce the elevation boundaries to [0 90] ;
- As an other example, if you know that the sources cannot be located in the (xy^-) plane, you can reduce the azimuth boundaries to [0 180];
- You can also try to locate at a specific azimuth or elevation, in this case you have to specify this value. For this special use case, the angular spectrum map displayed by setting *specDisplay* will only be a 1-D curve. This use case can be use for example with a two or three microphone array to avoid inherent ambiguities on results due to small number of microphones.

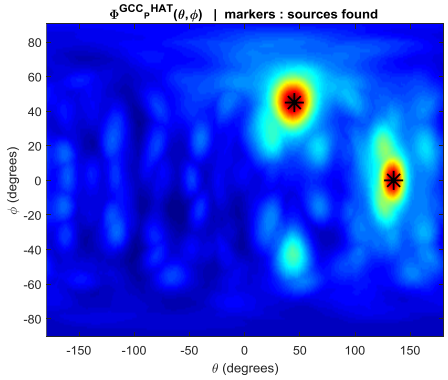


Figure 2: Displayed angular spectrum map with azBound = [-179 180] and elBound = [-90 90]

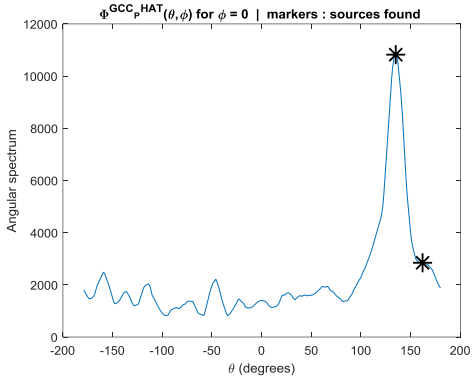


Figure 3: Displayed angular spectrum map with azBound = [-179 180] and elBound = 0

(2) Setting this flag provides a visualization of how the aggregation part of the algorithm is achieved by showing the sampled angular spectrum into the current microphone referential as a function of TDOA (a), the angular spectrum interpolated over the entire search grid (b) and the pooled angular spectrum over pairs (c).

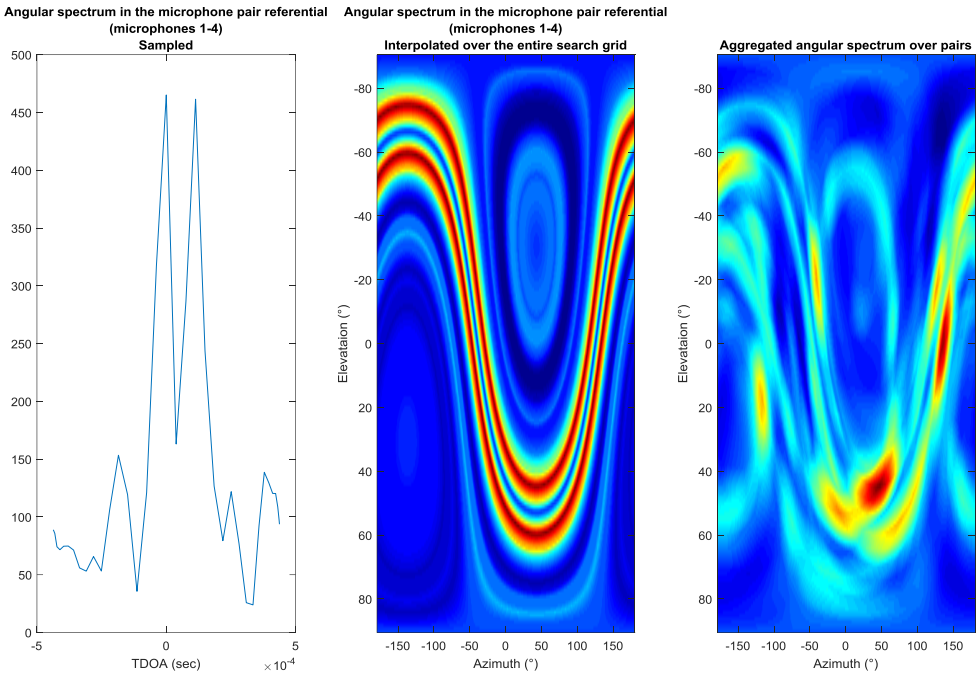


Figure 4: (a) Left (b) Center (c) Right

Output parameters:

Parameter name	Size / type	Description
sMBSSParam[1]	Structure	Structure encoding the user parameter choices and used by MBSS_locate_spec()

(1) This structure especially encodes the microphone positions for the dedicated sub array of microphones (*sMBSSParam.micPos*). In addition, the structure provides the associated microphone array centroid (*sMBSSParam.arrayCentroid*) which could be directly used in the evaluation tools (3.2).

3.1.2 MBSS_locate_spec()

[azEst, elEst, blockTimestamps, sElapsTime, figHandle] = MBSS_locate_spec (x, wienerRefSignal, sMBSSParam)

Description:

This function is the main entry point for the source direction estimation tool. It estimates the directions of sources recorded with a microphone array (audio mixture analysis) by implementing the algorithm described in (1).

Input parameters:

Parameter name	Size / type	Description

x	nSampl x nChan	Matrix containing the input audio signal (nChan: number of channels, nSampl: number of samples per channel).
wienerRefSignal	nExcerptSampl x nChan or empty	Matrix containing an excerpt of the source signal to be attenuated or emphasized by the Wiener filtering process. Required if *enableWienerFiltering* is set by the user in sMBSSParam.
sMBSSParam	Structure	MBSS Parameter structure. See 3.1.1 .

Output parameters:

Parameter name	Size / type	Description
azEst	nBlocks x nrc	Estimated directions of arrival (resp. azimuth and elevation) at each time stamps. The number of block depends on the analysis window *blockDuration* and the overlap *blockOverlap*. If the algorithm locates only k < nrc sources, the last estimated (nsrc – k) directions are set to a NaN value.
elEst	nBlocks x nrc	
	nBlocks x 1	Time instant in second (middle of the analysis window) at which the estimated direction is computed
sElapsTime	Structure	Informative computational time structure with the following fields: • xTFComp: Time spent to compute the time-frequency representation of x • xnTFComp: Time spent to compute the time-frequency representation of xn (if *enableWienerFiltering* is set, [] otherwise) • xnCovComp: Time spent to compute the noise covariance matrix (if *enableWienerFiltering* is set, [] otherwise) • blockLoc: nBlocks x 1 vector, time spent to compute the estimated directions on each block Note: All times are expressed in seconds
figHandle	1 x 1	figure handle indices if *specDisplay* is set, otherwise the vector is set to "-1" value.

3.2 Evaluation_tools

3.2.1 MBSS_true_aoa()

[azRef, elRef] = MBSS_true_aoa (arrayCentroid, srcPos)

Description:

This function computes the ground truth angle directions (azimuth and elevation) of the sources defined by its cartesian coordinates. The ground truth referential is defined by the microphone centroid position.

Input parameters:

Parameter name	Size / type	Description
arrayCentroid[1]	3 x T	Cartesian microphone array centroid positions at each time stamp. If the third dimension (i.e. time) equals 1, a static microphone array is considered.
srcPos[1]	3 x nsrc x T	Cartesian source positions of each source at each time stamp. If sources are statics, the last dimension (T) can be equal to 1.

Output parameters:

Parameter name	Size / type	Description
azRef	nsrc x T	Ground truth azimuth and elevation related to the microphone array referential at each time stamp t.
elRef	nsrc x T	

(1) Warnings:
- Both array centroid positions and source positions must be defined into the same referential.
- If both array centroid positions and source positions are moving over the time, we consider that both are sampled at the same time stamps.

3.2.2 MBSS_eval()

[R, P, F, Acc] = MBSS_eval_angle (evalMode, angleThreshold, srcPos, azEst, elEst, blockTimeStamps, sceneTimeStamps)

Description:
This function evaluates estimated source directions results (azimuth or elevation) in terms of recall, precision, F-measure and accuracy according to a tolerance error threshold. Some explanations about these metrics can be found in [4].

Input parameters:

Parameter name	Size / type	Description
evalMode	String	Evaluation method, define the way of how the metrics are computed. This parameter could take of these values : • 'az_only': Metrics are computed by only computing the localization error in azimuth. The error is then compared to *angleThreshold* parameter.

		• 'el_only': Metrics are computed by only computing the localization error in elevation. The error is then compared to *angleThreshold* parameter. • 'cartesian': Metrics are computed by evaluating the localization error in azimuth and elevation. Both are compared to *angleThreshold* parameter. • 'curvilinear'[1]: Metrics are computed by evaluating the localization error as a curvilinear distance on a 1-meter sphere. The error is then compared to *angleThreshold* parameter.
angleThreshold[2]	1 x 1 scalar	Tolerance error threshold in degrees, "inf" if no threshold applied
arrayCentroid[2]	3 x T	Cartesian microphone array centroid positions at each time stamp t. If the second dimension (i.e. time) equals 1, we consider a static microphone array.
srcPos	3 x nsrc x T	Cartesian true source positions at each time stamp t. If the sources are statics, the last dimension, T, could be equal to 1.
azEst	nBlocks x nrc	Estimated directions of arrival (resp. azimuth and elevation) at each time stamp returned by *MBSS_locate_spec()*
elEst	nBlocks x nrc	
blockTimeStamps	1 x nBlocks	Time instant in second at which the estimated direction is computed. Correspond to the *blockTimeStamps* parameter returned by MBSS_locate_spec()
sceneTimeStamps	1 x T or []	Mandatory if true source positions are defined at different time stamps (i.e T > 1 for srcPos parameter). In this case, this parameter encodes the audio scene time stamps in seconds.

(1) The curvilinear distance.

Let's define $\{\theta_e\ ,\ \varphi_e\}$ the estimated directions (azimuth and elevation) and $\{\theta_{ref}\ ,\varphi_{ref}\}$ the ground truth directions. The curvilinear error *e* is computed as follow:

$$e = \cos^{-1}\big(\sin(\varphi_e) * \sin\big(\varphi_{ref}\big) + \cos(\varphi_e) * \cos\big(\varphi_{ref}\big) * \cos\big(\theta_{ref} - \theta_e\big)\big)$$

(2) Warnings:

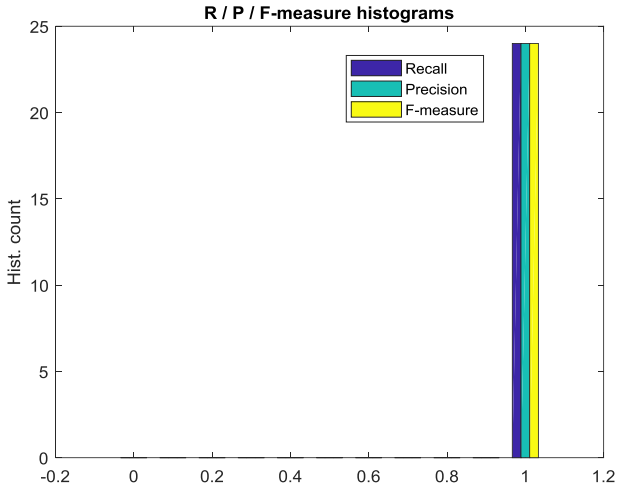
- Both array centroid positions and source positions must be defined into the same referential.
- If both array centroid positions and source positions are moving over the time, we consider that both are sampled on the same time grid.

Output parameters:

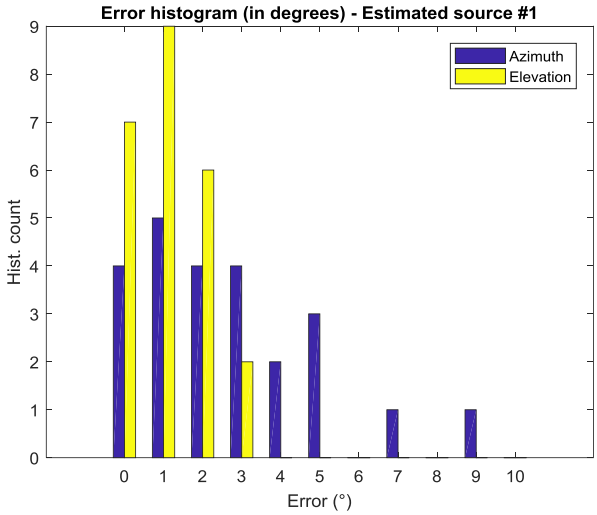
Parameter name	Size / type	Description
R	1 x nBlocks	Recall computed at each estimated direction time stamp
P	1 x nBlocks	Precision computed at each estimated direction time stamp
F	1 x nBlocks	F-measure computed at each estimated direction time stamp
Acc	nBlocks x nsrce x 2 (*) or nBlocks x nsrce x 1 (**)	Accuracy of each estimated direction at each time stamp. (*) For *evalMode* = 'cartesian' (**) For other *evalMode* values

As output, the function provides some visualization plots helping the interpretation of the results.

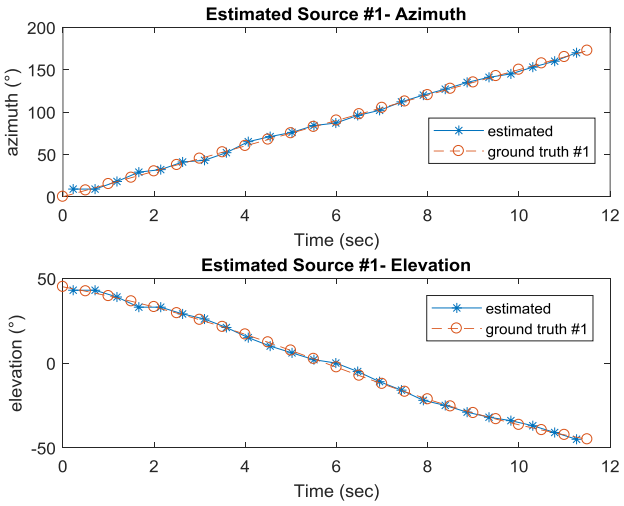
- The first plot is a histogram of R, P and F-measure:



- For each estimated localization, an histogram is plotted showing the accuracy (error) results for the chosen evaluation mode:



- In the case of a block processing approach (i.e. nBlocks > 1) an additional figure is provided to plot the true and estimated direction over time.



3.3 Roomsimove_tools

3.3.1 MBSS_roomsimove()

[time, HH] =
MBSS_roomsimove(fs, room_size, F_abs, A, sensor_xyz, sensor_off, sensor_type, ptime, source_xyz)

Description:
This function computes impulse room filters for a moving source and/or moving array described by their positions at discrete instants in a “shoebox” empty room.

Comment:
Roomsimove software is used to generate simulated spatial source image of fixed sources for simulated audio mixture computation. This software can be used to generate such images for moving sources too. However no moving source tracking functionality is implemented in Multi-channel BSS Locate.
Input parameters:

Parameter name	Size / type	Description
fs	1 x 1 scalar	Sampling frequency (Hertz)
room_size	1 x 3 scalar	Room dimensions (in meters)
F_abs	nfreq x 1 scalar	Frequencies to define absorption coefficients;
A	nfreq x 6 scalar	Frequency-dependent absorption coefficients for each surface
sensor_xyz	3 x nchan x T scalar	Cartesian coordinates of the nchan microphones
sensor_off	3 x nchan scalar	Sensor directions (azimuth, elevation and roll offset in degrees)
sensor_type	1 x nchan scalar	Sensor type (1 = omnidirectional / 2 = cardioid)
ptime	1 x T scalar	Time stamps array for corresponding source position (in seconds)
source_xyz	3 x T scalar	Corresponding source position over ptime

Parameter name	Size / type	Description
time	n_samples x 1 scalar	Vector listing the sampling instants
HH	H_length x channels x n_samples scalar	Matrix containing the associated filters

3.3.2 MBSS_roomsimove_apply()

x=MBSS_roomsimove_apply(time, HH, s, fs)

Description:
This function filters a signal using precomputed simulated room impulse responses. Given some room impulse responses computed with *roomsimove()* and close field recording of a source, this function generates the simulated spatial source image corresponding to the source recorded by some microphones in a room.

Input parameters:

Parameter name	Size / type	Description
time	n_samples x 1 scalar	Vector listing the sampling instants
HH	H_length x channels x n_samples scalar	Matrix containing the associated filters
s	1 x s_length scalar	Vector containing the signal to be filtered
fs	1 x 1 scalar	Sampling frequency (Hertz)

Output parameters:

Parameter name	Size / type	Description
x	nchan x s_length	Matrix containing filtered signals

4 Example description

The toolbox is provided with some examples that illustrate different use case scenarios. Those examples are located in the *examples* folder.

4.1 Example 1: Run MBSS Locate on a recorded audio file

Description:

This use case consists of applying Multi-channel BSS Locate on an audio file recorded with a microphone array to estimate source directions.

The script MBSS_*example1.m* illustrates the calling sequence necessary for its realization using a file previously simulated with the roomsimove software with Example 3 simulation settings (see 4.3).

Calling sequence:

- Open the input audio file;
- Define algorithm tuning parameters and and microphone position parameters and call *MBSS_micPosParam2Struct();*
- Call MBSS_locate_spec() function to estimate direction(s) of source(s) in recorded 3D audio scene.

4.2 Example 2: real mixture with one static speaker + one static noise source

Description:

This use case consists in using an audio mixture of the voiceHome-2 corpus [5]. The provided mixture is composed of 5 seconds of noise only signal followed by an utterance (wake up word + command) spoken by a static speaker under this noise condition.
This example illustrates also the use of the *enableWienerFiltering* flag to enhance the speaker over noise localization into the utterance. This is achieved by using 4 seconds of noise only signal to attenuate its localization into the mixture. Using proposed Wiener filtering for such record allows the speaker to be correctly localized, otherwise only the noise localization is found.
The script MBSS_*example2.m* illustrates the calling sequence necessary for its realization. In order to execute this example you need to download and unzip the voiceHome-2 corpus available at the following link: https://zenodo.org/record/1252143 .

Calling sequence:

- Define the corpusPath to point towards the unzipped voiceHome-2 corpus or copy it in dedicated folder;
- According to voiceHome-2 corpus, choose the processing {home, room, speaker, position, noise, utterance} condition;
- Define tuning of localization algorithm. We recommend to enable the *enableWienerFiltering* flag for this use case;
- Call *MBSS_InputParam2Struct()*;
- Call *MBSS_locate_spec()* to estimate directions of sources in the mixture;
- Call MBSS_*eval()* to compute evaluation metrics.

4.3 Example 3: simulated mixture with two static speakers

Description:

This use case consists in using Multi-channel BSS Locate to generate a simulated audio mixture, applies the multi-channel localization algorithm to this mixture and evaluates the results. The script MBSS_*example3.m* illustrates the calling sequence necessary for its realization.

Calling sequence:

- Define the simulation parameters corresponding to your audio scene:
  Fill the different data structures necessary for the mixture generation (3D audio scene parameters). Those structures are defined in MBSS_*audioScene_parameters() function* for *MBSS_example3.m* script (see 4.5 for further details):
  - roomStruct: parameters structure containing room dimensions with absorption coefficients of each surface (walls, floor, ceiling) at given frequencies;
  - sensorsStruct: parameters structure containing sensors positions, directivities and orientations;
  - sourcesStruct: parameters structure containing sources positions, filename of close field recordings for sources to be mixed.

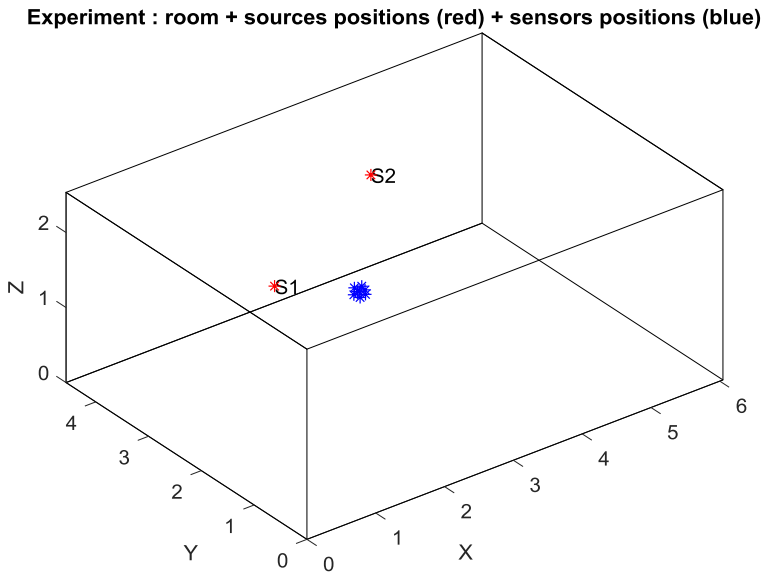


Figure 5: 3D audio scene example with 8 microphones (blue) and 2 sources (red)

- Call *roosimove() and MBSS_roomsimove_apply()* to generate the simulated audio mixture described with previous parameters;
- Define algorithm tuning parameters and microphone position parameters, then call *MBSS_InputParam2Struct()*;
- Call *MBSS_locate_spec()* to estimate directions of sources in the mixture;
- Call MBSS_*eval()* to compute evaluation metrics.

4.4 Example 4: simulated mixture of one moving source recorded by a moving microphone array

Description:

This use case consists in using Multi-channel BSS Locate to generate a simulated audio mixture of one moving speaking source with a moving array of microphones as well, apply the multi-channel

localization algorithm to this mixture and evaluate the results. The script MBSS_*example4.m* illustrates the calling sequence necessary for its realization.

Calling sequence:

- Define the simulation parameters corresponding to your audio scene:
  This part was already introduced in 4.3. The MBSS_*audioScene_parameters()* (see 4.5 for further details) is modified to simulate a moving source with an uniform circular motion.

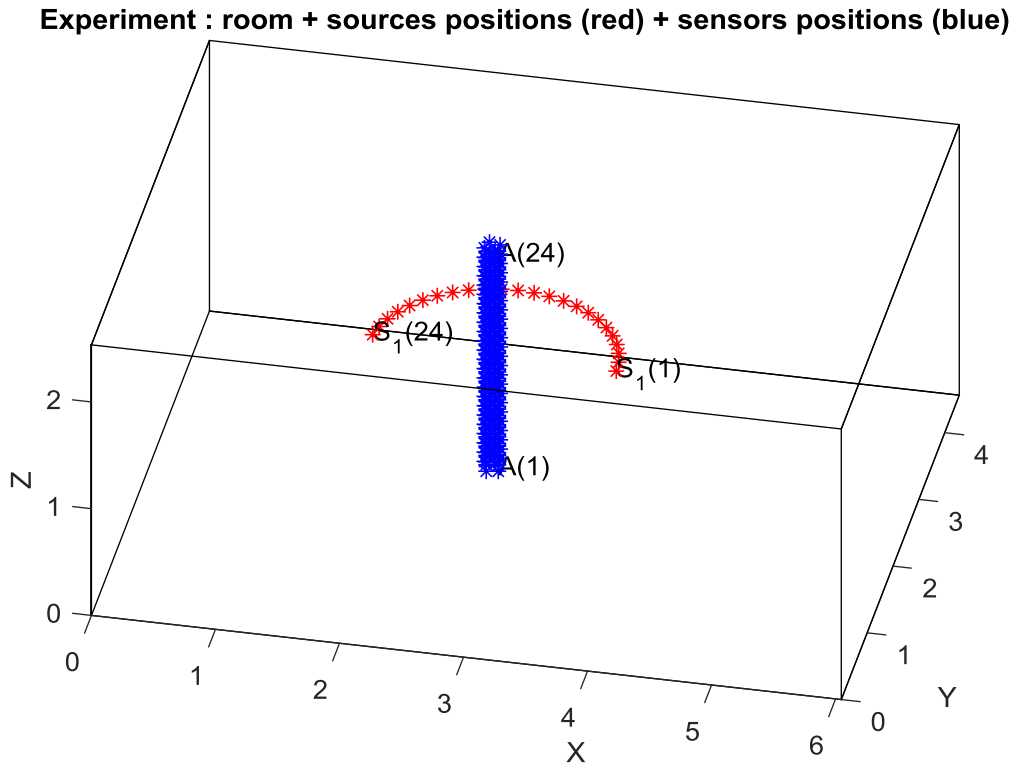


Figure 6: 3D audio scene example with a 8 -microphones moving array (blue) and one moving source (red)

- Call *roosimove() and MBSS_roomsimove_apply()* to generate the simulated audio mixture described with previous parameters;
- Define algorithm tuning parameters and microphone position parameters then call *MBSS_InputParam2Struct()*;
- Call *MBSS_locate_spec()* to estimate directions of sources in the mixture;
- Call MBSS_*eval()* to compute evaluation metrics.

4.5 MBSS_audioScene_parameters()

[roomStruct, sensorsStruct, sourcesStruct, sceneTimeStamps] = MBSS_audioScene_parameters()

Description:

This function used in MBSS_*example3.m and MBSS_example4.m* script contains audio scene parameters necessary for mixture generation using *roomsimove*.

Output Parameters:

Parameter name	Size / type	Description
roomStruct	Structure containing these fields:	Structure of room parameters
	room_size: 1 x 3 scalar	Room dimensions (in meters)
	F_abs: 1 x nfreq	Frequencies used for frequency-dependent absorption coefficients
	A: 6 x nfreq	Frequency-dependent absorption coefficients for each surface (walls, floor, ceiling)
sensorsStruct[1]	Structure containing these fields:	Structure of sensors parameters
	sensor_nb: 1 x 1 scalar	Number of microphones
	sensor_xyz: nchan x 3 x T scalar	Cartesian coordinates of the nchan microphones
	sensor_off: nchan x 3 scalar	Sensor directions (azimuth, elevation and roll offset in degrees)
	sensor_type: 1 x nchan scalar	Sensor type (1 = omnidirectional / 2 = cardioid)
sourcesStruct[1]	1 x nsrcMixture structures, with the following fields:	structures of sources parameters (one for each source)
	filename: string	Audio file name recording of the source to be mixed
	ptime: 1 x T	Time stamps array for corresponding source position (in seconds)
	source_xyz: 3 x T	Corresponding source positions over ptime
sceneTimeStamps[1]	1 x T scalar	Audio scene time stamps, corresponds to the ptime variable

(1) If the array and the source are moving, both must be synchronized on the same time stamps, which are called the audio scene time stamps. In the multiple source case, all sources must have the same time duration, if not the audio file to be mixed are truncated to the minimum source time duration.

5 Multi-channel localization algorithm

The multi-channel localization algorithm consists in aggregate the angular response of a 2-microphones localization algorithm over all the pairs. Such aggregation technique was first introduced

in [6]. All the 2-mics localization algorithm needs a time-frequency representation of the processed signal. This section is decomposed as follow:

1. Time-Frequency representations details
2. Wiener filtering for localization enhancement (optional)
3. Aggregation algorithm overview

5.1 Time – Frequency representations

Depending on the 2-mics localization technique used, two time – frequency representations has been defined in BSS Locate [1]. The following table mentions the time – frequency used according to the chosen 2-mics localization technique.

2-mics localization technique	Time-Frequency representation
'GCC-PHAT' , 'GCC-NONLIN'	Linear
'MVDR', 'MVDRW', 'DS', 'DSW', 'DNM'	Quadratic

5.1.1 Linear time-frequency representation

In the linear case, the multichannel input x is transformed into the frequency domain by applying a short-time Fourier transform (STFT) using half-overlapping sine window. The size of the STFT is controlled by the *fftSize_sec* parameter which is typically 64ms. For a sampling frequency of 16kHz, that's resulting a STFT of 1024 points.

5.1.2 Quadratic time-frequency representation

In the quadratic case, the multichannel input x is transformed into the frequency domain by using a quadratic linear-scale time-frequency transform based on the local covariance of a STFT with sine window.

5.2 Wiener filtering for localization enhancement (optional)

In order to attenuate or emphasize the localization of a type of signal into a mixture, the toolbox provides some Wiener filtering tools. The typical scenario is a mixture composed of one speaker and a noise source, both located at different positions (see **Erreur ! Source du renvoi introuvable.**). If an excerpt of this noise is available for example, it can be used to enhance either the speaker localization or the noise localization on the mixture. This technique is only available for 2-mics localization methods based on linear time – frequency transform.

5.2.1 Averaged covariance definition

Let us define x the multi-channel mixture (Ks samples and I channels) and xn the multi-channel signal excerpt to be attenuated or emphazed into the mixture (Kn samples and I channels). We denote as $X(n,f)$ and $X_n(n,f)$ the corresponding I-dimensional complex valued vectors of TF coefficients.

First, let us define the averaged covariance of a signal $Y(n,f)$:

$$R_Y(f) = \frac{1}{N}\sum_n Y(n,f) * Y(n,f)^H$$

With $.^H$ denotes the Hermitian transpose.

5.2.2 Attenuation mode

In the "attenuation" mode, the signal excerpt is used to be attenuated into the mixture. The filtered version of X, named $\tilde{X}$, is computed as follow:

$$\tilde{X}(n,f) = \left(\left(R_X(f) - R_{X_n}(f)\right) * R_{X_n}^{-1}(f)\right) * X(n,f)$$

With R_X and R_{X_n} respectively the averaged covariance of X and X_n.

5.2.3 Emphasis mode

In the "emphasis" mode, the signal excerpt is used to be emphasized into the mixture. The filtered version of X, named $\tilde{X}$, is computed as follow:

$$\tilde{X}(n,f) = (R_{X_n}(f) * R_{X_s}^{-1}(f)) * X(n,f)$$

With $R_{X_s}^{-1}(f)$:

$$R_{X_s}^{-1}(f) = V * S * U^H$$

With U (left singular vectors), S (singular values), V (right singular vectors) the SVD of $(R_X(f) - R_{X_n}(f))$.

Note: Negative singular values in S are forced to "0".

5.3 Aggregation algorithm overview

The algorithm computes a function $\Phi(\theta,\varphi)$ with θ and φ representing respectively azimuth and elevation parameters. This particular function aims at representing actives sources directions as local maxima.

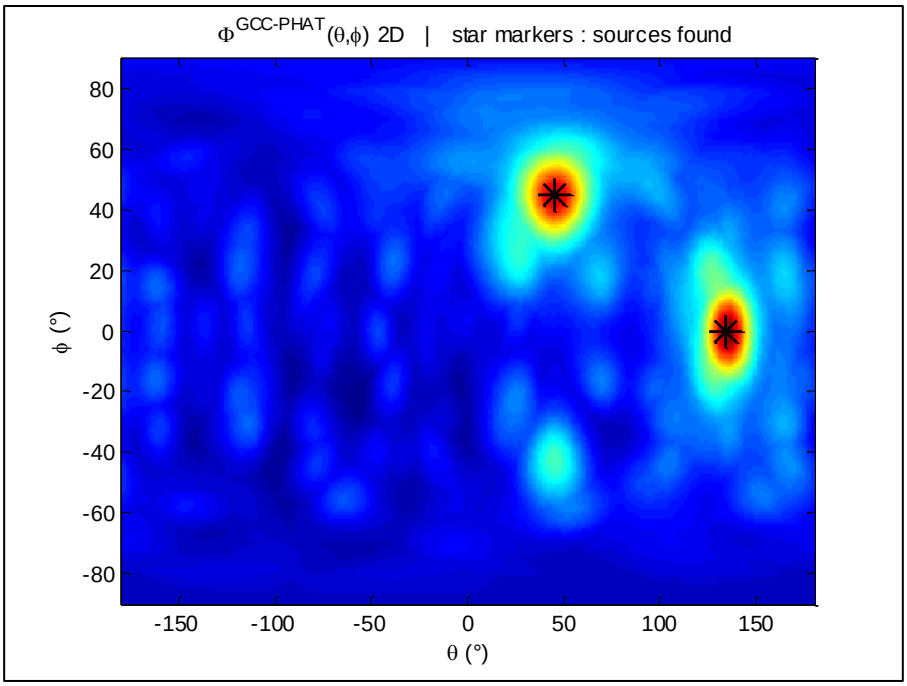


Figure 7: Example of function $\Phi(\theta, \varphi)$ and sources found (star markers) obtained for two audio sources with the GCC-PHAT angular spectrum method.

$$\Phi(\theta, \varphi) = \underset{t}{\text{poolingFunction}}\left(\sum_{n}^{N} \Phi_n^{AngularSpectrumMeth}(t, \theta, \varphi)\right)$$

With:

- t: time frames index;

- $n \in [1 \dots N]$: microphones pair index, with N the total number of microphones pairs;

- $\underset{t}{\text{poolingFunction}}$: a function to integrate results over time frames;

- $\Phi_n^{AngularSpectrumMeth}(t, \theta, \varphi)$: The local angular spectrum in a $\{\theta, \varphi\}$ direction of the nth pair of microphones;

Pooling functions:

Pooling function can be chosen between $\max_t(.)$ and $\text{sum}_t(.)$ functions.
One could prefer $\max_t(.)$ function for sources that are only active within a few time frames in analyzed signal due to integration of irrelevant information when the sources are inactive.

Local Angular spectrum functions:

Available local angular spectrum functions are the following ones: GCC-PHAT, GCC NON LIN, MVDR, MVDRW, DS, DSW and DNM.

All these methods were available in BSS Locate [1] and are detailed in [4].

5.4 Detailed implementation

The $\Phi_n^{AngularSpectrumMeth}$ notation becomes Φ_n below for readability purpose.

Current implementation can be decomposed in following steps:

1. Determine the whole set of combinations S_{jk} of potential azimuth/elevation pairs and compute corresponding set of angles of arrival $\{\alpha_{jk}\}_n$ in each microphones pair referential ;
2. Resample $\{\alpha_{jk}\}_n$ to $\{\alpha_{s_i}\}_n$ for each microphone pair in order to limit the number of values to build $\Phi_n(t, \alpha_{jk})$;
3. Compute the set of delays $\{\tau_i\}_n$ corresponding to the set of angles $\{\alpha_{s_i}\}_n$;
4. Compute the angular spectrum function $\Phi_n(t, \tau_i)$ for each microphone pair ;
5. Compute the angular spectrum function $\Phi_n(t, \alpha_{jk})$ for each microphone pair interpolating $\Phi_n(t, \alpha_{s_i})$ function ;
6. Convert $\Phi_n(t, \alpha_{jk})$ to $\Phi_n(t, \theta_j, \varphi_k)$;
7. Compute $\Phi(\theta, \varphi)$;
8. Search global maximum (one source) or local maxima (multiple sources) of $\Phi(\theta, \varphi)$ and finally compute estimated source(s) azimuth and elevation parameters.

Those steps are detailed below. $\{\alpha_{jk}\}_n$, $\{\alpha_{s_i}\}_n$ and $\{\tau_i\}_n$ sets are always dependent of a microphones pair (index n). This index is not mentioned everywhere in following explanation for readability purpose.

Step 1:

This step consists in sampling the whole space of potential directions and express those directions into each microphone pair referential as proposed in [7]. This set of potential directions can be for example expressed as a combination of azimuth and elevation values.

In this algorithm the whole set of combinations S_{jk} is obtained by computing the Cartesian product between the whole set of azimuths $\{\theta_j\}$ and elevations $\{\varphi_k\}$. Typically, we define $\theta_j \in [-180° \; 180°]$ and $\varphi_k \in [-90° \; 90°]$ with a specific resolution.

The $\{\alpha_{jk}\}_n$ angles in the nth pair referential, composed by microphones M_{n1} (x_{n1}, y_{n1}, z_{n1}) and M_{n2} (x_{n2}, y_{n2}, z_{n2}), are the angles formed between the directional vector of M_{n1} and M_{n2} microphones and the vectors pointing to a $\{\theta_j, \varphi_k\}$ direction.

$$\alpha_{jk} = \arccos\left(\frac{M_{n1n2}.P(\theta_j, \varphi_k)}{||M_{n1n2}|| \times ||P(\theta_j, \varphi_k)||}\right)$$

With $P(\theta_j, \varphi_k)$ the vector representing the cartesian coordinates of a $\{\theta_j, \varphi_k\}$ direction on a unit sphere.

$$P(\theta_j, \varphi_k) = \begin{pmatrix} cos(\varphi_k)\cos(\theta_j) \\ cos(\varphi_k)\sin(\theta_j) \\ sin(\varphi_k) \end{pmatrix}$$

And M_{n1n2} :

$$M_{n1n2} = \begin{pmatrix} x_{n2} - x_{n1} \\ y_{n2} - y_{n1} \\ z_{n2} - z_{n1} \end{pmatrix}$$

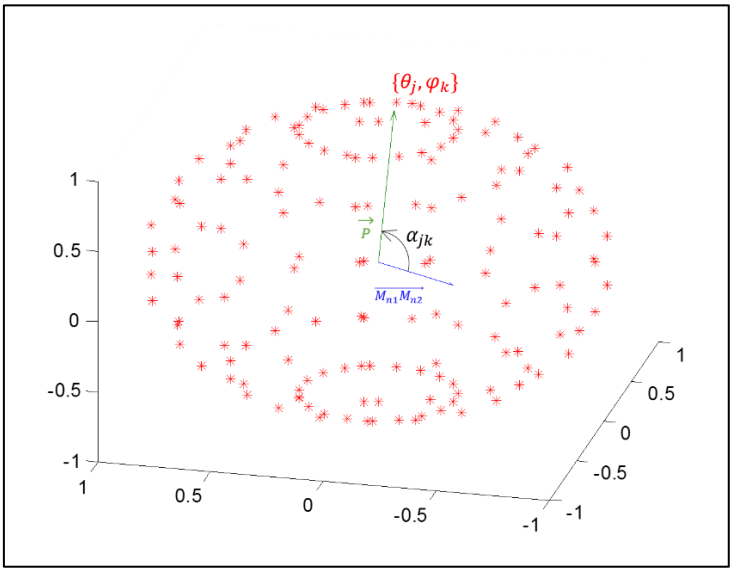


Figure 8: Illustration of α_{jk} computation

Step 2:

The number of potential angles of arrival $\{\alpha_{jk}\}_n$, evaluated for each microphone pair, is equal to the number of $\{\theta_j, \varphi_k\}$ combinations. The amount of combinations increases by reducing the search grid resolution of $\{\theta_j, \varphi_k\}$, consequently some $\{\alpha_{jk}\}$ values are close. Following the idea used in [1] where TDOA search space is sampled, one can use sampled values of potential angles of arrival to decrease the CPU time needed for angular spectra values computation.

In this algorithm, for each microphone pair $n \in [1 \dots N]$, the whole set of $\{\alpha_{jk}\}_n$ is resampled with a resolution of α_{res} in order to limit the set of angles used for angular spectra values computation. $\{\alpha_{s_i}\}_n$ is the sampled version of $\{\alpha_{jk}\}_n$.

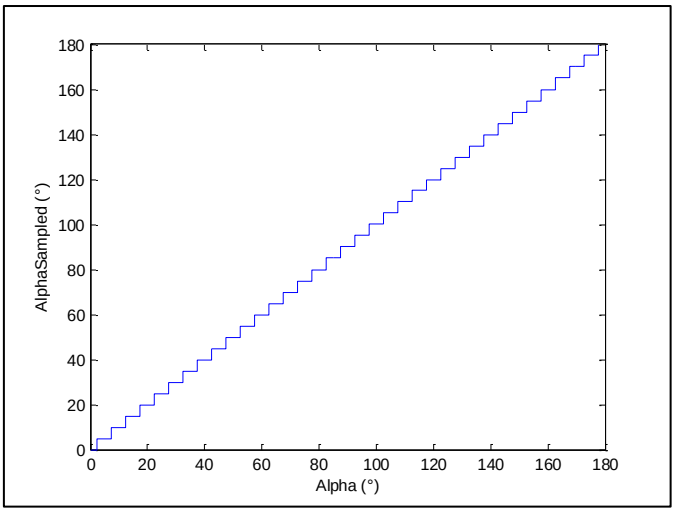


Figure 9: Sampling function of $\{\alpha_{jk}\}_n$ with a 5 degrees resolution

Step 3:

This step consists in converting angle of arrival of sound wave in a delay τ. This delay, named TDOA (Time Difference Of Arrival), can be computed from the angle of arrival of the sound wave under far field assumption:

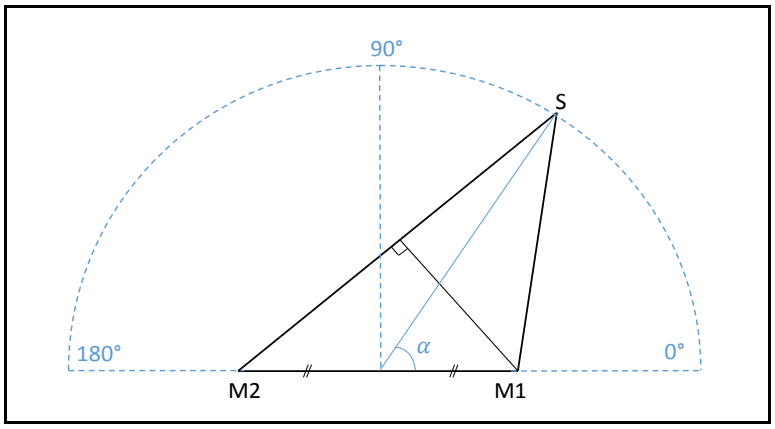


Figure 10: Angle of arrival of a sound source by TDOA estimation

Let us define S as an audio source and M1 and M2 two microphones for recording purpose. The sound wave generated by source S reaches the two microphones with a delay τ (TDOA) due to microphone spacing and source position. This delay τ is equivalent to an angle of arrival α under the far field assumption (allowing to consider the sound wave as a "plane wave"):

$$\alpha = arccos\left(\frac{c\,\tau}{d}\right)$$

c is the sound speed, typically 343m/s in the air at 20 °C;

d is the microphones spacing

For each microphone pair and each angle of arrival α_{s_i} TDOA is then computed with following formula:

$$\tau_i = \frac{d_n}{c}\cos(\alpha_{s_i})$$

Step 4:

This step consists in computing local angular spectrum $\Phi_n(t,\tau_i)$ for each microphones pair using one of the local angular spectrum methods detailed in [4].

Step 5:

$\Phi_n(t,\tau_i)$ is computed on a sampled set of values (see Step 2). In order to compute an angular spectrum value for the whole $\{\alpha_{jk}\}_n$ set of potential angles, $\Phi_n(t,\tau_i)$ is interpolated with an order 1 interpolation function.

$$\Phi_n(t,\alpha_{jk}) = \Phi_n(t,\alpha_{s_r}) * \left(\frac{\alpha_{s_{r+1}} - \alpha_{jk}}{\alpha_{res}}\right) + \Phi_n(t,\alpha_{s_{r+1}}) * \left(\frac{\alpha_{jk} - \alpha_{s_r}}{\alpha_{res}}\right)$$

With r the first index respecting the following condition: $\alpha_{jk} \geq \alpha_{s_r}$.

Step 6:

α_{jk} is equivalent to $\{\theta_j, \varphi_k\}$ by construction. We can then write:

$$\Phi_n(t,\alpha_{jk}) \Leftrightarrow \Phi_n(t,\theta_j,\varphi_k)$$

Step 7:

The global angular spectrum function $\Phi(\theta,\varphi)$ is computed by summing local angular spectrum contributions over all microphone pairs. The result is then integrated over time frames using one of the pooling functions described in **Erreur ! Source du renvoi introuvable.**.

$$\Phi(\theta,\varphi) = \underset{t}{\text{poolingFunction}} \sum_n \Phi_n(t,\theta,\varphi)$$

An option is also available to apply pooling on normalized versions (on each time frame) of local angular spectra. Above function is then modified with corresponding option activated:

$$\Phi(\theta,\varphi) = \underset{t}{\text{poolingFunction}} \left[\Phi_{norm}(t,\theta,\varphi)\right]$$

With:

$$\Phi_{norm}(t,\theta,\varphi) = \frac{\Phi_{pos}(t,\theta,\varphi)}{\max_{\theta,\varphi}\left[\Phi_{pos}(t,\theta,\varphi)\right]} \qquad \Phi_{norm}(t,\theta,\varphi) \in [0\ 1], \forall \{t,\theta,\varphi\}$$

$$\Phi_{pos}(t,\theta,\varphi) = \Phi_{sum}(t,\theta,\varphi) - \min_{\theta,\varphi}[\Phi_{sum}(t,\theta,\varphi)] \qquad \Phi_{pos}(t,\theta,\varphi) \geq 0, \forall \{t,\theta,\varphi\}$$

$$\Phi_{sum}(t,\theta,\varphi) = \sum_n \Phi_n(t,\theta,\varphi)$$

One could prefer using this normalization when sources active on a few time frames are exhibiting different levels of peak when computing $\Phi_{sum}(t,\theta,\varphi)$.

Step 8:

Estimated directions $\{\tilde{\theta}, \tilde{\varphi}\}$ are computed through local peaks finding method on $\Phi(\theta, \varphi)$:

- Find all local peaks (maxima) of $\Phi(\theta, \varphi)$: $\{\theta_j, \varphi_k\}$ is considered as a peak if $\Phi(\theta_j, \varphi_k)$ is the maximum value compared to its local neighborhood values composed of its 8 closest neighbors;
- Sort all local peaks in decreasing order according to their values;
- Select the first K highest peaks values with respect to a minimal distance condition between peaks;
- Deduce the K estimated directions $\{\tilde{\theta}, \tilde{\varphi}\}$ corresponding to the K selected local maxima.

6 Bibliography

[1] C. Blandin, E. Vincent and A. Ozerov, "BSS Locate : A toolbox for source localization in stereo convolutive audio mixtures," [Online]. Available: http://bass-db.gforge.inria.fr/bss_locate/.

[2] A. Deleforge and F. Forbes, "Rectified binaural ratio: a complex T-distributed feature for robust sound localization," in *Signal Processing Conference (EUSIPCO) IEEE*, 2016.

[3] E. Vincent and D. Campbell, "Roomsimove : A Matlab toolbox for the computation of simulated room impulses responses for moving sources," [Online]. Available: http://www.irisa.fr/metiss/members/evincent/software.

[4] C. Blandin, A. Ozerov and E. Vincent, "Multi source TDOA estimation in reverberant audio using angular spectra and clustering," *Signal Processing 92,* pp. 1950-1960, 2012.

[5] N. Bertin, E. Camberlein, R. Lebarbenchon, E. Vincent, S. Sunit, I. Irina and F. Bimbot, "VoiceHome-2, an extended corpus for multichannel speech processing in real homes," *Speech Communication, Elsevier [SUBMITTED],* 2017.

[6] J. DiBiase, H. Silverman and M.-S. Brandstein, "Robust localization in reverberant rooms," in *Springer*, vol. Microphone Arrays, Signal Processing Techniques and Applications, 2001, pp. 131-154.

[7] J.-M. Valin, F. Michaud and J. Rouat, "Robust localization and tracking of simultaneous moving sound sources using beamforming and particle filtering," *Robotics and autonomous systems,* vol. 55, pp. 216-228, 2007.