Spherical Microphone Array Processing in Python with the *sound_field_analysis-py* Toolbox

Christoph Hohnerlein[1], Jens Ahrens[2]

[1] *Quality & Usability Lab, Technische Universität Berlin, Deutschland, Email: christoph.hohnerlein@qu.tu-berlin.de*

[2] *Division of Applied Acoustics, Chalmers University of Technology, Sweden, Email: jens.ahrens@chalmers.se*

Abstract

The *sound_field_analysis-py* toolbox started as a Python port of SOFiA toolbox[1] by Benjamin Bernschütz et al. [1], which performs the analysis and processing of data captured with spherical microphone arrays. SOFiA is written for Matlab with several externals in C/C++ and published under the GNU GPLv3 license.
The current implementation deals with impulse responses and headphone playback – frame-based processing, which would allow real-time manipulation, is subject to ongoing work. Furthermore, we are working towards interfacing *sound_field_analysis-py* with other existing Python audio processing tools, such as the *sound_field_synthesis-py* toolbox [2], to leverage community efforts towards baseline implementations and reproducible research.
The *sound_field_analysis-py* toolbox is available on GitHub[2].

Introduction

Spherical microphones (such as the Eigenmike[3]) as well as scanning/sequential arrays (such as the VariSphear[4]) can be used to record multi-point room impulse responses. Such a set can then be used to retroactively apply that room's reverberation to a signal, similarly to traditional Room Impulse Responses (RIRs). But in contrast to RIRs, array recordings theoretically allow for a fully dynamic reproduction of the rooms response, only limited by the spatial resolution of the array.
Figure 1 shows two possible workflows: A multi-point room IR can either be combined with a set of HRTFs to recreate a virtual scene binaurally or used to generate the corresponding driving functions of a loudspeaker based sound field synthesis approach, as for example presented in [3]. Apart from capturing impulse responses, spherical microphone arrays also allow for storing and transmitting of full dynamic sound scenes including all spatial information.

A spherical harmonics expansion of the captured sound field has shown to be a convenient representation as this finite discrete set of signals can represent a continuous spherical space. Furthermore, rotations can be performed elegantly, which is very important for head-tracked binaural playback.

Therefore, most of the work in this package concerns transformations and processing in the spherical harmonics space. Unfortunately, the larger theoretical background is out of scope for the paper at hand. As a port of the SOFiA Toolbox [1], our package implements functions covered in the corresponding thesis [4], and builds on extended literature such as [5] and [6].

Example workflow

Converting the time domain data into spatial coefficients comprises two steps: First, a standard Fourier Transform `process.FFT()` is applied, followed by either the explicit (if the quadrature is appropriate) or least-squares spatial Fourier transform (`process.spatFT()` or `process.spatFT_LSF()`). Furthermore, it is useful to pre-calculate the radial filters that effectively undo the effects of the microphone array due to its size, transducer type and scattering body (if there is one) using `gen.radial_filter_fullspec()`.

Now, the coefficients can be manipulated (for example resampled, rotated, weighted) and visualized. Furthermore, when the spherical harmonics expansion of a set of HRTFs is available, the two can be combined by means of convolution in the spherical harmonics domain (as shown in [7] and [8]):

$$S^{l,r} = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} d_n \overline{P_{nm}} H_{nm} \,, \tag{1}$$

where d_n are the radial filters, $\overline{P_{nm}}$ the complex conjugate of the sound field coefficients and H_{nm} the HRTF coefficients.
Applying the inverse of the two step transformation (`process.ispatFT()` and `process.iFFT()`) to go back to time domain yields a pair of impulse responses that represent the ear signals of a human listener that is exposed to the sound field that was captured by the microphone array. This process can be performed for different virtual head orientations and the data can then be exported for binaural rendering using the SoundScapeRenderer using `io.write_SSR_IRs()`.

Some examples are available in the `examples` folder on GitHub.

Modules

The *sound_field_analysis-py* package contains several submodules; the most important ones **gen**, **process**, **plot** and **io** are briefly introduced in the following.

[1] `http://audiogroup.web.th-koeln.de/SOFiA_wiki/WELCOME.html`

[2] `https://github.com/QULab/sound_field_analysis-py/`

[3] `https://www.mhacoustics.com/products`

[4] `http://audiogroup.web.th-koeln.de/varisphear.html`

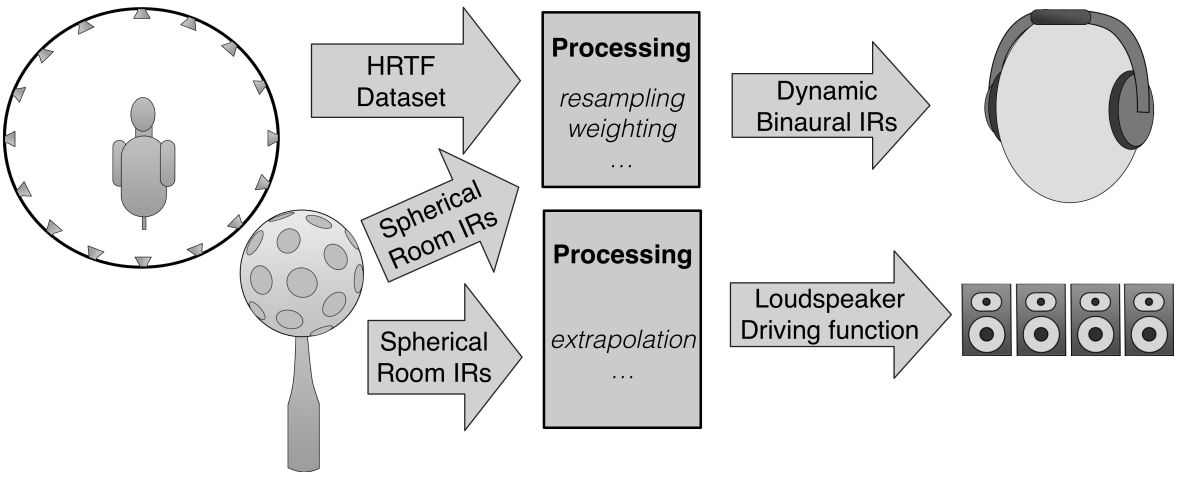

Figure 1: The sound field captured by a spherical microphone array can be combined with an HRTF dataset by means of convolution in the spherical harmonics domain to produce dynamic binaural room simulations or added to a speaker-based sound field reproduction setup, such as Wave Field Synthesis [3].

Generators

The **gen** package contains all routines that generate data based only on meta data.

Sound fields

There are two functions that directly return the coefficients of a synthesized sound field: **ideal_wave()** and **sampled_wave()**. Both simply need a description of the desired sound field, such as the configuration of the simulated microphone array, type and direction of the impinging wave.

Quadratures

Gauss and Lebedev quadratures (both explicitly integrable) can be generated using `gen.gauss_grid()` and `gen.lebedev()`. For the Lebedev grid, stable orders up to $N = 11$ (corresponding to a degrees of $L \in [6, 14, 26, 38, 50, 74, 86, 110, 146, 170, 194]$) can be satisfied. It is based on Richard P. Muller's Python implementation[5] of [9].

Radial Filters

Radial filters for three different configurations (open sphere, rigid sphere, dual sphere) using 2 different transducer types (omni and cardiod) are implemented, excluding the dual cardioid configuration.

Processing

The processing submodule contains functions that transform existing data.

Fourier Transform

The `process.FFT` and `process.iFFT` function rely on Numpy's `fft.rfft` routine to perform time ↔ frequency transformations. All frequency-domain signals are expected to be one-sided and all time-domain signals to be real.

[5] `https://github.com/gabrielelanaro/pyquante/blob/master/Data/lebedev_write.py`

Convolution

Convolution is either performed in the frequency domain (fast convolution) using `scipy.signal.fftconvolve()` or in the time domain using `numpy.convolve()`. Unless explicitly set, the mode is automatically set to the faster one (switching from time domain to fast convolution if $\forall N > 500$).

Spatial Fourier Transform

Generally, the spherical harmonics coefficients $P_{nm}(\omega)$ of order n, degree m and frequency ω that correspond to a frequency-domain function $F(\omega, \Omega)$ at positions Ω is derived through the expansion integral over a continuous unit sphere S:

$$P_{nm}(\omega) = \int_S F(\omega, \Omega)\overline{Y_n^m}(\Omega)\mathrm{d}\Omega\,, \tag{2}$$

with $\overline{Y_n^m}(\Omega)$ as the complex conjugate spherical harmonic basis functions. Because the unit sphere is not continuously measured with a real microphone array but instead sampled at discrete points Ω_i, the spherical harmonics coefficients can be determined by two different methods.

Firstly, Eq. 2 can be approximated in discrete space over an integrable spherical quadrature, as implemented in `process.spatFT()`:

$$P_{nm}(\omega) = \langle (4\pi w_i \overline{Y_n^m}(\Omega_i)), F(\omega, \Omega_i)\rangle \tag{3}$$

where $\langle\ ,\ \rangle$ denotes the inner product; $\overline{Y_n^m}(\Omega_i)$ the complex conjugate of the spherical harmonic basis functions at the discrete positions Ω_i; w_i the quadrature weights associated with each position and $F(\omega, \Omega_i)$ the corresponding frequency-domain signals.

As an alternative, a *least-square fit* of spherical harmonic coefficients on the data is implemented in `process.spatFT_LSF()`, which solves:

$$\underset{\hat{P}_{nm}(\omega)}{\operatorname{argmin}} ||\langle Y_n^m(\Omega_i), \hat{P}_{nm}(\omega)\rangle - F(\omega, \Omega_i)||_2 \tag{4}$$

for $\hat{P}_{nm}(\omega)$ in the least-square sense, where $||\cdot||_2$ is the L_2 norm.

The *inverse* spatial Fourier Transform `process.ispatFT()` is implemented as:

$$F(\omega, \Omega_i) = \langle Y_n^m(\Omega_i), P_{nm}(\omega) \rangle \quad (5)$$

Plane Wave Decomposition

Plane wave decomposition of directions Ω_i is computed as:

$$D(\omega, \Omega_i) = \langle Y_n^m(\Omega_i), d_n(kr) P_{nm}(\omega) \rangle \quad (6)$$

where $Y_n^m(\Omega_i)$ are the spherical basis functions of directions Ω_i, $d_n(kr)$ are the radial filters at wavenumber k & radius r and $P_{nm}(\omega)$ are the spherical field coefficients.

Rotation

Currently, only rotation around the vertical axis has been implemented, which is the most important rotation when head-tracking is considered. It is expressed as a complex phase at reconstruction:

$$F(\omega) = \sum_{n=0}^{\infty} \sum_{m=-n}^{n} P_{nm}(\omega) \underbrace{e^{-\mathrm{i}m\Delta\alpha}}_{\Delta\alpha \text{ rotation}} d_n(kr) Y_n^m(\Omega_i) \quad (7)$$

The implementation of arbitrary rotations is subject to on-going work.

Spherical math utilities

The `sph` subpackage contains mathematical expressions that are needed when dealing with spherical arrays. Specifically, this includes various Bessel functions, their spherical expression and their respective derivatives:

- Bessel $J_n(x)$, $j_n(x)$, $j_n'(x)$ (normal, spherical, spherical derivative)
  `besselj | spbessel | dspbessel(n, z)`
- Neumann $Y_n(x)$, ... (Weber / Bessel 2nd kind)
  `neumann(n, z) | ...`
- Hankel $H_n^{(1)/(2)}(x)$, ... (1st / 2nd kind)
  `hankel1(n, z) | ...`
  `hankel2(n, z) | ...`

Furthermore, spherical harmonic basis functions $Y_n^m(\varphi, \theta)$ up to order $N_{max} = 85$ of several types (see Eq. 8 – 10) can be generated on an arbitrary grids using the `sph.sph_harm()` function.

Plotting

Each processing stage can be evaluated via various ways of plotting data, which is internally offloaded to the `Plotly.py` package. This produces highly portable, interactive plots that render in the browser using the D3.js library.

2D

`plot.plot2D()` draws an arbitrary number of signals along a common x-axis. Several predefined types {*time, linFFT, logFFT*} are available. Exports, such as Fig. 2, are also available.

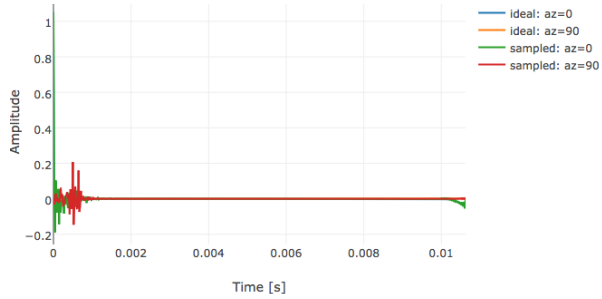

Figure 2: 2D time-domain plot of an ideal and sampled plane wave at two different directions.

3D

`plot.plot3D()` generates a 3D visualization of a sound field by displaying the normalized magnitude of it's plane wave decomposition at a 1° resolution. These are rendered using webGL, which is available in all modern browsers and therefore highly portable and fast. Figure 3 shows such a 3D plot.

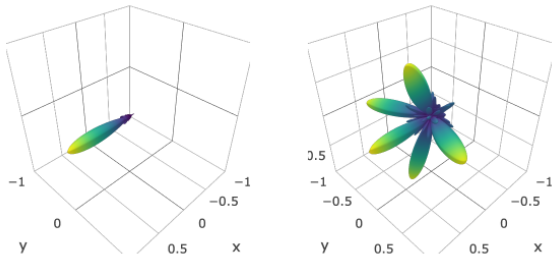

Figure 3: 3D plot of the magnitude of the plane wave decomposition of an ideal (left) and sampled (right) plane wave at $f = 7$ kHz.

Input/Output

The `io` submodule handles importing/exporting data as well as defines the four data containers used internally:

- `io.TimeSignal(signal, fs, delay)`
- `io.SphericalGrid(azimuth, colatitude, radius, weight)`
- `io.ArrayConfiguration(radius, type, transducer)`
- `io.ArraySignal(io.TimeSignal, io.SphericalGrid, io.ArrayConfiguration)`

miro

The `miro` datatype[6] for Matlab can be read using the `io.read_miro_struct()` function. However, this only works if the `.mat` file was exported as a `struct` or in the older 7.2 format, as this function relies internally on `scipy.io.loadmat`.

SOFA

Spatially Oriented Format for Acoustics (SOFA, [10]) is a file format that stores a variety of spatial acoustic data

[6] `http://audiogroup.web.th-koeln.de/FILES/miro_documentation.pdf`

such as HRTFs, BRIRs or array recordings and is standardized as AES69-2015[7]. It is based on the efficient HDF5 format and currently only provides a C++ and Matlab API. It can however be read into Python using the `netCDF4` package. A small tutorial was made available as an example on GitHub[8].

SoundScapeRenderer

The function `io.write_SSR_IRs()` exports impulse responses into a `.wav` file compatible with the binaural renderer of the SSR which allows for dynamic evaluation with head-tracking [11].

Conventions

Signal data structure

Python/Numpy's arrays can be dimensionless, contrary to e.g. Matlab. Internally, such an array is assumed to be a single signal. If more than one signal are combined into a `[M x N]` matrix, it is treated as M signals of length N (row-major).

Spherical Harmonics

In order to be compatible with the SH definitions most commonly found in the literature, three different spherical harmonic basis functions are implemented: Complex (Eq. 8), real (Eq. 9) and so called "legacy" (Eq. 10, without *Condon–Shortley phase*). The complex definition is used internally.

$$N(m,n,\theta) = \sqrt{\frac{2n+1}{4\pi}\frac{(n-m)!}{(n+m)!}} P_n^m(\cos\theta)$$

$$Y_n^m(\varphi,\theta) = (-1)^m N(|m|,n,\theta)\cdot e^{\mathrm{i}m\varphi} \tag{8}$$

$$Y_n^m(\varphi,\theta) = (-1)^m N(|m|,n,\theta)\cdot \begin{cases} \sqrt{2}\cos(m\varphi), & m>0 \\ 1, & m=0 \\ \sqrt{2}\sin(m\varphi), & m<0 \end{cases} \tag{9}$$

$$Y_n^m(\varphi,\theta) = N(m,n,\theta)\cdot e^{\mathrm{i}m\varphi} \tag{10}$$

Future Development

Currently, all implementations are carried out in terms of impulse responses. This means that *sound_field_analysis-py* reads room impulse responses captured by a spherical microphone array and produces ear impulse responses. Ways of applying the same processing to signal streams in a frame-based fashion are investigated, which would allow for real-time processing. This would likely be accomplished by interfacing with `sounddevice`[9] and `jackclient`[10] packages. This would allow for fast evaluation of sound fields directly from Python.

[7] `http://www.aes.org/publications/standards/search.cfm?docID=99`

[8] `https://github.com/QULab/sound_field_analysis-py/blob/master/examples/Exp3_Import_SOFA.ipynb`

[9] `https://python-sounddevice.readthedocs.io/`

[10] `https://jackclient-python.readthedocs.io/`

Acknowledgments

We would like to thank **Benjamin Bernschütz** for his continued support, as well as **Matthias Geier** for the many fruitful discussions.

References

[1] Benjamin Bernschütz, Christoph Pörschmann, Sascha Spors, and Stefan Weinzierl. SOFiA sound field analysis toolbox. In *Proceedings of the International Conference on Spatial Audio (ICSA)*, pages 7–15, 2011.

[2] Hagen Wierstorf and Sascha Spors. Sound field synthesis toolbox. In *Audio Engineering Society Convention 132*. Audio Engineering Society, 2012. `http://sfstoolbox.org`.

[3] Jens Ahrens and Sascha Spors. Wave field synthesis of a sound field described by spherical harmonics expansion coefficients. *The Journal of the Acoustical Society of America*, 131(3):2190–2199, 2012.

[4] Benjamin Bernschütz. *Microphone arrays and sound field decomposition for dynamic binaural recording.* PhD thesis, Technische Universität Berlin, 2016. `https://doi.org/10.14279/depositonce-5082`.

[5] Jens Ahrens. *Analytic Methods of Sound Field Synthesis.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. `http://www.soundfieldsynthesis.org`.

[6] Boaz Rafaely. *Fundamentals of spherical array processing*, volume 8. Springer, 2015.

[7] Amir Avni, Jens Ahrens, Matthias Geier, Sascha Spors, Hagen Wierstorf, and Boaz Rafaely. Spatial perception of sound fields recorded by spherical microphone arrays with varying spatial resolution. *The Journal of the Acoustical Society of America*, 133(5):2711–2721, 2013.

[8] Carl Andersson. Headphone auralization of acoustic spaces recorded with spherical microphone arrays. Master's thesis, Chalmers University of Technology, 2017.

[9] V.I. Lebedev and D.N. Laikov. A quadrature formula for the sphere of the 131st algebraic order of accuracy. In *Doklady. Mathematics*, volume 59, pages 477–481. MAIK Nauka/Interperiodica, 1999.

[10] Piotr Majdak et al. Spatially oriented format for acoustics: A data exchange format representing head-related transfer functions. In *Audio Engineering Society Convention 134*. Audio Engineering Society, 2013. `https://www.sofaconventions.org/`.

[11] Jens Ahrens, Matthias Geier, and Sascha Spors. The soundscape renderer: A unified spatial audio reproduction framework for arbitrary rendering methods. In *Audio Engineering Society Convention 124*. Audio Engineering Society, 2008. `http://spatialaudio.net/ssr/`.