SpectAcoular! An Extension for Acoular

Adam Kujawski, Gert Herold, Ennes Sarradj, Simon Jekosch
FG Technische Akustik, Technische Universität Berlin, Deutschland, adam.kujawski@tu-berlin.de

Introduction

The separation of sound sources with microphone arrays has been widely investigated in the field of engineering acoustics. In this regard, publicly available software for obtaining and analyzing microphone array data can help to expand scientific knowledge. The use of openly available source code saves implementation time and improves the comparability of calculation results between the users.

To date, various frameworks exist for the processing of acoustic signals. Among those frameworks, Acoular [1] has been developed specifically for the processing and analysis of microphone array data and is published under the terms of the BSD license. It is written in Python language and follows an object-oriented software design. Acoular includes several state-of-the-art methods for acoustic source mapping, such as CMF [2] or DAMAS [3]. Costly algorithms are accelerated by the use of JIT-compiled low-level routines, available from the Numba library [4]. Moreover, results obtained from computationally demanding calculations can be saved to cache files for effortless reuse. So far, Acoular allows to build efficient processing scripts. However, unlike commercial software, no graphical user interface (GUI) application exists that simplifies the handling for inexperienced users. Instead, programming knowledge is required to apply the methods of the Acoular framework.

GUIs can be of great value in microphone array applications. They are well suited for sensor signal monitoring during acquisition to prevent measurement errors or for simplifying the data evaluation workflow. Several graphics packages are available in Python that interact with the common GUI frameworks (e.g. matplotlib [5]). However, the building of sophisticated GUI applications is often cumbersome, since event callbacks need to be set up manually, resulting in many lines of specialized code. This issue can be solved by providing a high-level design based on the Model-View-Controller pattern (MVC) [6], as implemented in the Enthought Traits and TraitsUI library [7, 8]. The Traits library provides classes with typed attributes representing the model. The TraitsUI library offers the view, which is the GUI representation of model attributes. Interactions between the model and the view are realized by the handler objects building the controller. Updates to the GUI elements are automatically forwarded to the data model and vice versa. In many cases, this concept avoids defining event callbacks for the user. Acoular's objects derive from the classes of the Traits library. By means of the TraitsUI library, a GUI can be called to edit their attributes. However, the functionality can behave differently depending on the operating system and is therefore an unsupported feature of the framework. Newly developed libraries make use of web browsers to build interactive dashboard applications (e.g. Bokeh [9]). Usually, the rendering and event handling is performed by a high-level JavaScript library offering improved performance compared to native Python code. Using web browsers for the visual representation makes applications independent from the operating system. The only drawback associated with these libraries is the absence of the data models. The necessary mappings between data objects and their view have to be created manually.

So far, none of the available visualization libraries provide seamless and easy integration of Acoular's objects and methods into GUI applications. In order to change this, a new library has been designed that is introduced in the following.

Implementation

The SpectAcoular library [10] is an extension module for Acoular for the realization of microphone array GUI applications with minimal programming effort. The following sections provide insight into the design of the library and its fundamentals.

Bokeh Library

SpectAcoular is designed to be used in conjunction with the visualization library Bokeh [9]. Bokeh is a third-party module of the Python ecosystem and targets web browsers for visualization. It consists of a low-level Python API and a JavaScript library. The latter is named BokehJS. Rendering in the web browser is achieved by serializing plotting related Python objects into JavaScript Object Notation (JSON) format. The latter can be consumed by the BokehJS library to be displayed in the web browser.

In addition to basic plotting capabilities, Bokeh enables the creation of web applications based on a client-server architecture. Thereby, the application runs within a Python environment hosted by the server. One or multiple clients can connect to the server via accessing a URL. Bokeh models of the environment are rendered by the BokehJS library and and are displayed in the browser. Synchronization between the server and the client is maintained by the BokehJS library. The synchronization allows to stream data for visualization to the client and to react on user inputs or interactions with glyph and widget elements. Widgets are interactive control elements such as buttons or input fields. A glyph is a visual shape displayed by the figure object.

Building applications with Bokeh offers several advantages. Rendering and event handling are relatively fast since they are executed in JavaScript. This is particularly useful for running real-time applications and for

the visualization of high-dimensional data. Furthermore, the user does not need any JavaScript knowledge to create a web application since models defined in the BokehJS library are accessible via the Python API. Finally, the client-server architecture can be of great value for applications where no physical access to the computational resources exists. (This includes the use of a high-performance cluster (HPC) to leverage large computational resources.)

SpectAcoular Library

The library concept is based on the MVC pattern [6]. From an application perspective, objects of the Acoular framework are well suited data models. Besides, Bokeh offers objects for the creation of screen representations (view) and the handling of UI events for a wide range of data types. SpectAcoular is responsible for establishing a connection between a data models attributes and a view by setting up methods to react on user inputs or changes of the model.
The SpectAcoular library consists of the following modules

- factory module (`factory.py`)
- default view definition module (`bokehview.py`)
- consumer module (`consumer.py`)
- data processing module (`dprocess.py`)
- streaming module (`lprocess.py`)

The **factory** module is the core module that acts as an interface between Acoular and Bokeh by providing a factory that translates model attributes into widgets. Acoular relies on the Enthought Traits library [7] to define specific data types (traits) for each attribute of a class. The purpose of the factory is to return a suitable widget based on a given trait. Furthermore, callbacks that are necessary to maintain synchronization between the widget available in the browser and the trait are set up by the factory method. Figure 1 illustrates some important mappings between trait objects of the Enthought Traits library and the Bokeh library implemented by the factory module.
The **bokehview** module defines the default view for most of Acoular's classes. Here, a view consists of several instantiated widgets without a layout. Classes that have a default view can be imported from SpectAcoular. They inherit the `get_widgets()` method, which uses the factory to produce a view of the object.
The remaining modules, including the **consumer**, **dprocess** and **lprocess** module provide additional classes to simplify the processing and visualization of data in applications.

Examples

The following code snippet shows how an application can be created with just a few lines of code. The example includes the interactive visualization of 64-channel microphone array.
In line 1-4 of Listing 1, package content from

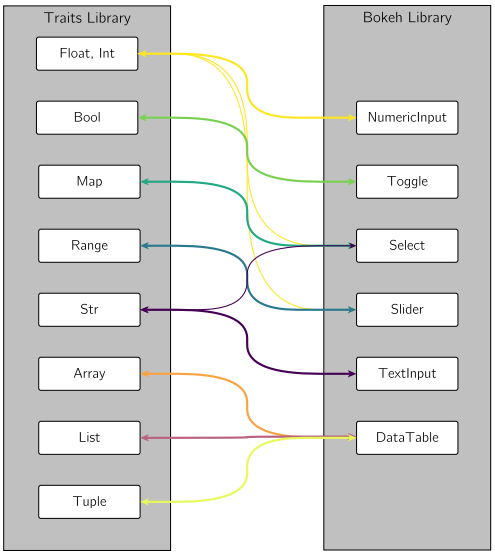

Figure 1: Implemented mappings between traits of the Enthought Traits library and widgets from the Bokeh library. Thick lines represent the default mapping used by the factory module of SpectAcoular.

SpectAcoular and Bokeh is imported to the environment. The `MicGeom` class originates from Acoular and provides coordinates of microphones in the array. The import from the SpectAcoular library enables the use of the widget factory. In line 7-8, the `MicGeom` class is instantiated and the arrangement, stored in the array_64.xml file, is specified via the `from_file` attribute. The `get_widgets()` method returns a dictionary containing the `MicGeom` attribute names as keys and their

Listing 1: Code snippet for the visualization of a microphone geometry in an interactive web application

```
1  from spectacoular import MicGeom
2  from bokeh.plotting import figure
3  from bokeh.layouts import column, row
4  from bokeh.io import curdoc
5
6  # instantiate MicGeom object
7  mg = MicGeom(from_file="array_64.xml")
8  widget_dict = mg.get_widgets()
9
10 # add widgets to column layout
11 widget_column = column(
12 widget_dict["from_file"],
13 widget_dict["num_mics"],
14 widget_dict["mpos_tot"],
15 )
16
17 # create figure and add microphone data
18 fig = figure(title="Microphone Geometry")
19 fig.circle(x="x", y="y", size=10,
20 source=widget_dict["mpos_tot"].source)
21
22 # define layout
23 layout = row(fig, widget_column)
24
25 # add figure and widgets to Document
26 curdoc().add_root(layout)
```

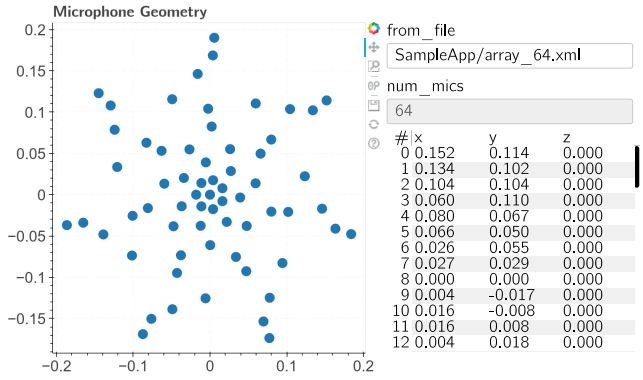

Figure 2: Rendered user interface related to the application code from Listing 1.

corresponding widget object as values. The widgets are then arranged in a column layout by line 11-15. In line 18, a figure object is created to display the microphone geometry. The coordinates of the geometry are stored as array data type in the `mpos_tot` attribute of the `MicGeom` instance. The corresponding data table widget holds a synchronized copy of the coordinates, which is linked to the figure by means of a circle glyph in line 19-20. Finally, the figure and the widget column are arranged in a row layout. The final layout is added to the Bokeh document. A document is a class that collects Bokeh models to be rendered in the browser.
Figure 2 displays the application window after serving the document. The application is fully interactive, meaning that the microphone locations can be selected and adjusted by editing the data table (right column). A different microphone geometry can be provided by changing the file name. Since the datatable and the figure are linked to the same data model, they are always synchronized. A change of the geometry file results in an update of the datatable and the figure. The tools available in the plot toolbar allow interactions with the figure (e.g. to zooming in or out).

Applications

Pre-built applications, intended to simplify microphone array measurements and data evaluation workflow, are part of the SpectAcoular repository. They are well suited for educational purposes since programming knowledge is not required.
The files belonging to an application are organized in a predefined directory format. Within the application directory, Bokeh allows to provide a customized HTML template for the styling of the web browser interface.
Currently, six different applications are available and are mentioned in the following.

microphone array measurement app is an application which can be used for data acquisition with a 16-channel MEMS microphone array[1]. The GUI allows to monitor the sound pressure level of each array channel and to perform live beamforming. Moreover, a calibration functionality is implemented.

sound level meter app is a streaming application which allows to analyze data collected with any connected input sound device. It offers live monitoring of the sound pressure level (SPL) at different frequency and time weightings.

frequency domain beamforming app can be used to calculate source maps from measured microphone array data with different frequency domain methods. The default data was collected during a wind tunnel experiment on trailing edge noise. The app is well suited for the comparison of different microphone array algorithms for the characterization of aeroacoustic sources.

virtual rotating source mapping app offers similar functionality to the frequency domain beamforming app. However, it is designed for the comparison of virtual rotating methods for the characterization of rotating sources. Simulated microphone array data is provided.

time signal exploration app gives insight into data stored in HDF5 format. The application can be used to explore collected time signals and allows to calculate the auto power spectra. A playback of the audio signals can be realized by any connected output sound device.

microphone array geometry app is described in detail in the following to give an example of the level of interactivity for an application.

The microphone array as a spatial filter can be characterized by its spatial response to a point source with unit strength, known as the point-spread function (PSF). Assuming a point source at location $\mathbf{x}_s$, the PSF describes the contribution of the source at any other spatial position $\mathbf{x}_t$ in the region of interest. The PSF depends on the microphone geometry, the signal frequency, and the steering vector formulation [11]. The characteristics of the PSF, especially the beam width and side lobe level, are important to assess the source separation performance that can be expected from an array aperture. Numerous studies have focused on finding a geometry with optimal properties, by applying parametric [12] or statistical optimization approaches [13].
The microphone geometry application helps to compare different configurations and to investigate the influence on the PSF when individual sensor positions are changed, removed or added. Figure 3 displays the rendered interface in the web browser. The GUI comprises several visualization components that are arranged in three columns. The left column consists of a subfigure showing the positions of the microphones as blue circles. The figure is fully interactive, meaning that the circles can be dragged to other locations. New microphones can be added by tapping into the window. A toolbar provides further tools, e.g. to select multiple channels at once. The middle column consists of a subfigure showing the PSF for the given array configuration. By tapping into the image,

[1] https://www.minidsp.com/products/usb-audio-interface/uma-16-microphone-array (accessed: 2021-09-04)

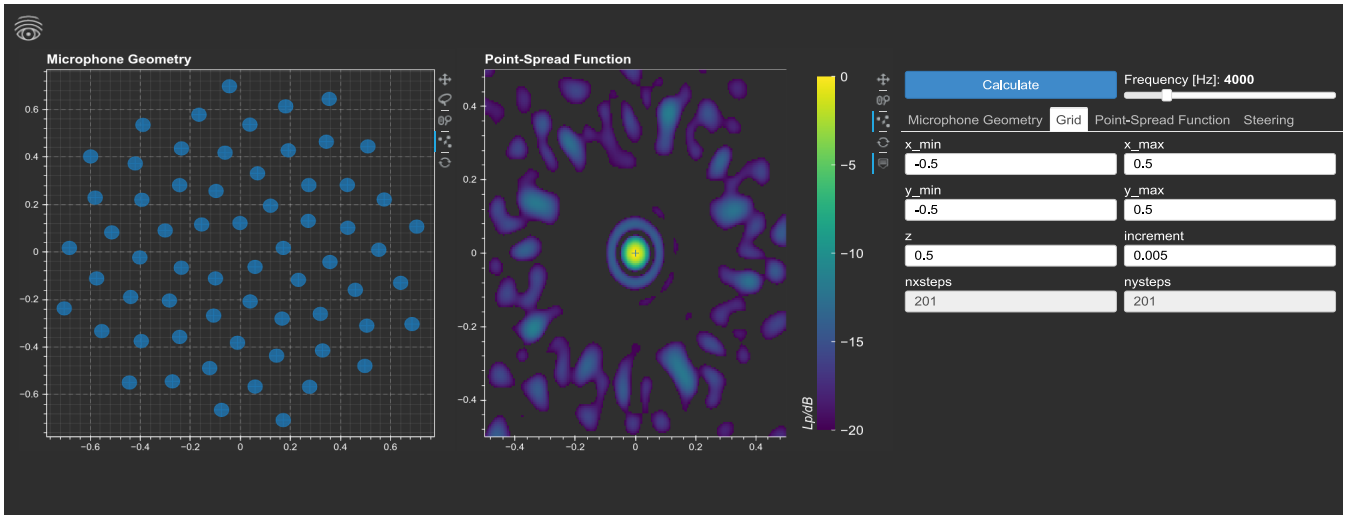

Figure 3: Interactive application for the exploration of the spatial response of a microphone array geometry to a point-source.

the unit source position can be changed. The right column provides access to the simulation parameters, e.g. the considered frequency. The four different tabs comprise mapped widgets (view) of Acoular objects in the Python environment.

Conclusive Summary

In this contribution, the SpectAcoular library has been introduced as an extension of the Acoular framework. It has been shown that the library allows the creation of individual dashboard applications with low implementation effort. The use of web browsers for visualization allows to create real-time applications and to use the capabilities of Acoular independently from the operating system. SpectAcoular comes with pre-built applications designed for data acquisition and exploration. Furthermore, they serve as a template for the development of new applications with SpectAcoular.

References

[1] Ennes Sarradj and Gert Herold. A Python framework for microphone array data processing. *Applied Acoustics*, 116:50–58, 2017.

[2] Tarik Yardibi, Jian Li, Petre Stoica, Nikolas S. Zawodny, and Louis N. Cattafesta. A covariance fitting approach for correlated acoustic source mapping. *The Journal of the Acoustical Society of America*, 127(5):2920–2931, 2010.

[3] Thomas F. Brooks and William M. Humphreys. A deconvolution approach for the mapping of acoustic sources (DAMAS) determined from phased microphone arrays. *Journal of Sound and Vibration*, 294(4):856–879, 2006.

[4] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. Numba: A LLVM-based Python JIT Compiler. In *LLVM '15: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*, pages 1–6, Austin, Texas, 2015. Association for Computing Machinery.

[5] John D. Hunter. A 2D graphics environment. *Computing in Science and Engineering*, 9(3):90–95, 2007.

[6] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, 1994.

[7] Enthought Inc.. TraitsUI: Traits-capable windowing framework. `https://github.com/enthought/traitsui` (accessed 2021-09-09).

[8] Enthought Inc.. Traits: observable typed attributes for Python classes. `https://github.com/enthought/traits` (accessed 2021-09-09).

[9] Bokeh Development Team. Bokeh: Python library for interactive visualization. `https://docs.bokeh.org/` (accessed 2021-09-09).

[10] Adam Kujawski, Gert Herold, Simon Jekosch, Ennes Sarradj, and Arne Hölter. acoular/spectacoular: v21.08, aug 2021. `https://doi.org/10.5281/zenodo.5242826`

[11] Ennes Sarradj. Three-dimensional acoustic source mapping with different beamforming steering vector formulations. *Advances in Acoustics and Vibration*, pages 1–12, 2012.

[12] Ennes Sarradj. A Generic Approach To Synthesize Optimal Array Microphone Arrangements. In *Proceedings of the 6th Berlin Beamforming Conference*, pages 1–12, 2016.

[13] Florent Le Courtois, Jean-Hugh Thomas, Franck Poisson, and Jean-Claude Pascal. Genetic optimisation of a plane array geometry for beamforming. Application to source localisation in a high speed train. *Journal of Sound and Vibration*, 371:78–93, jun 2016.