Sorbonne Université
Département des Masters de Sciences de l'Ingénieur
Mention Automatique/Robotique, Parcours Ingénierie des Systèmes Intelligents

Advanced Audio Processing
Part 1: Acoustics and Sound Localization

S. Argentieri and H. Boutin

Version 0.2, 2023/2024

The acquisition system

For the following 2 practicals, we will work with a linear, evenly spaced, array made of $N = 8$ MEMS (MicroElectroMechanical Systems) microphones, see Figure 1. The array is connected with an ethernet cable to an acquisition system that can handle 4 identical arrays. Consequently, 4 pairs of student will work with one acquisition system only.

Each acquisition system in then connected to one computer through USB. This computer emits a Wifi signal (the SSID is written on the top of each box) which you must connect to start the acquisition. In order to use this all system, you must install some Python specific libraries, in addition to the code which is available on Moodle. The installation of these dependencies is explained in the next section.

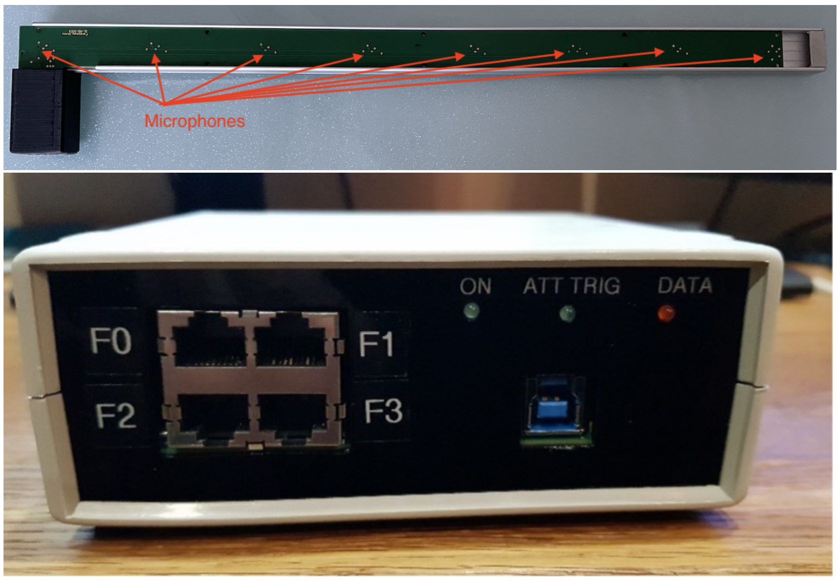


FIGURE 1 – Overview of the audio system : (up) the microphone array (down) the acquisition system.

Preparation and installation of the software

We will consider that you already have a working Python installation on your own computer, which is using the `conda` package manager. One example is the `Anaconda` distribution, which can be installed on most operating systems. You might also consider using `conda-forge` with `miniforge`, which also allows you to install the `conda` package manager with some nice pre-configured features. From `https://github.com/conda-forge/miniforge` you can find the latest installers (search for `Miniforge3`, and select the corresponding operating system you are working with). *Be carefull : multiple installations of different Python managers are generally not advised, and you should stick with your current working installation as long as the required Python packages are available.*

In all the following, we will show how to use the `Anaconda` distribution, with examples coming from Windows 11. The very same procedure can also be used for other operating systems.

Python environment

To begin, we will create a dedicated Python virtual environment where all the required packages and libraries will be installed. The steps are the following :

1. launch `Anaconda Navigator`, and select the "Environments" menu on the left. Then click on the "Channels" button, click on "Add..." ;
2. next, enter the following address in the empty field that appears : `https://conda.anaconda.org/conda-forge/`. Hit Enter, and then click on "Update channels" ;

3. once the new `conda-forge` repository has been added, click then on the "Create" button, at the bottom of the current page. Now you can specify the "Name" of your new environment ("array", for example), and then select the "Python" checkbox (the version number is automatically chosen, there is no need to change it). Click then on "Create".
4. the new environment is now created, with a minimal Python installation. We will now need to install a list of packages and libraries. To that end, click on the name of your newly created environment to activate it. Then click on the small "Play" button near to the environment name, and click on "Open Terminal". A terminal is now launched, in which your current environment is also activated (its name should appear between parentheses) ;
5. in this terminal, copy and past the following command :
   `conda install matplotlib numpy scipy h5py websockets notebook ipywidgets ipyfilechooser ipympl paramiko nbclassic ipympl`
   You might have to confirm the installation of these packages. Obviously, confirm it.
6. once all required packages are installed, you can close your terminal. Then, come back to `Anaconda Navigator` and select the "Home" menu on the left. From here, on list above select "Installed applications" on "array" (or the name you selected for the environment you created during step 3 above). From this screen, you can now launch `Jupyter Notebook`, which should then launch your web browser.
7. that's it ! You are now ready to work with the acquisition system.

All these steps have to be followed carefully only once. The next time you want to work on this system, you only have to select the appropriate environment in the "Envrionments" menu, and then to launch `Jupyter Notebook` from the same environment in the "Home" menu.

Configuration and tests

Download now the files available on Moodle. First, unzip the file `TP.zip` to a location of your choice on your disk. *This location will be the root for your developments.* Next, download the notebook `Test.ipynb` and place it at this root location. Open the notebook in Jupyter Notebooks, and execute the first cell containing all the import directives :

Code Listing 1 – Import of the required libraries

```
%matplotlib widget
import matplotlib.pyplot as plt
import numpy as np
from client import array
import time
```

If this first cell is correctly executed, without errors about missing libraries or dependencies, then you should be ready to go !

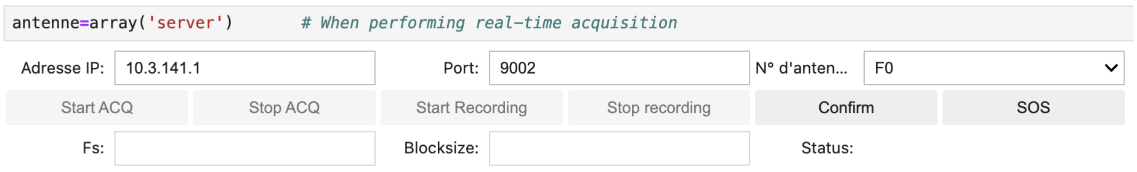
When working with the acquisition system

First, you must connect to the Wifi network emitted by the acquisition system. Its SSID is indicated on it, and your teacher will give you the corresponding password. Next, execute the cell with the following code :

Code Listing 2 – Initialization of the acquisition interface

```
antenne=array('server')
```

This code then displays the GUI shown in Figure 2, in which some parameters from the acquisition system are reported. In particular, you must specify the IP address of the acquisition system ; the default value should be ok. You must also specify which array you are using (the array is connected to one of the 4 ports named `F0`, `F1`, `F2` or `F3`). Then, click on the "Confirm" button. From now on, you can :



```
antenne=array('server')        # When performing real-time acquisition
```

FIGURE 2 – Graphical interface for configuring the acquisition system.

— start the acquisition (`Start ACQ` button)

— start the recording of an audio file in the `.h5` file format (`Start Recording` button)

— related stop buttons are also available to stop acquisition or recording.

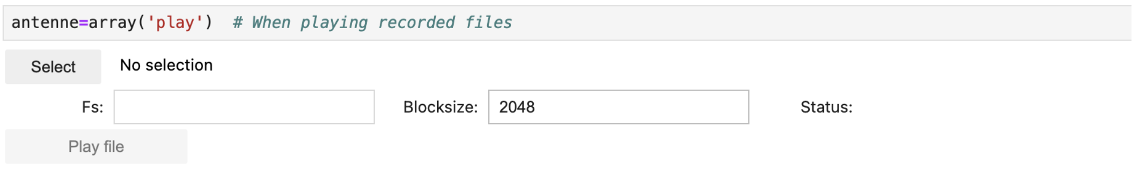
When working on recorded files

You simply have to use the following code :

Code Listing 3 – Initialization of the acquisition interface

```
antenne=array('play')
```

This code then displays the GUI shown in Figure 3, in which some parameters Òloaded from the selected file are reported. Your first have to click on the `Select` button to open a window



```
antenne=array('play')  # When playing recorded files
```

FIGURE 3 – Graphical interface for playing recorded data.

allowing you to browse to the `.h5` audio file you want to play. Then, you can click on `Play file`.

How to read data

The variable `antenne` you created to access the GUI can be used to manipulate and access data any time. For that purpose, you can use :

— `antenne.blocksize` : to get the buffer length used during the acquisition ;

— `antenne.mems_nb` : to get the value of N, the number of microphones used in the array ;

— `antenne.d` : to get the microphone interspace value d ;

— `antenne.fs` : to get the sampling frequency value ;

— `m = antenne.read()` : to capture the current audio buffer in `m` ; `m` is of size `Blocksize` $\times N$.

How to emit sounds

You will need in the two following practicals to emit specific sounds, like pure sinus tone at a specific frequency. The easiest way to do that is actually to install some free application able to generate such signals on your phone. A lot of them exist on the App Store (iOS) or Google Play (Android). Select one of them (any free version of an app should be sufficient) and install it.

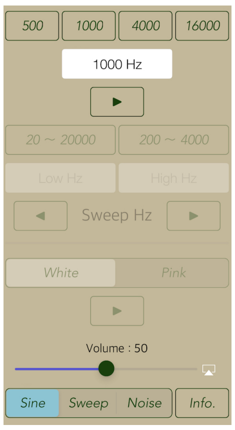


FIGURE 4 – Example : the “Audio Tone Generator Lite” iOS app.

1 TP1 : acoustics is absolutely fantastic !

2 TP2 : sound source localization with a microphones array : beamforming approaches

You have characterized and analyzed the sound propagation in the previous practical. We will now exploit theses properties to infer one sound source position w.r.t. a linear microphone array made of $N = 8$ omnidirectional MEMS microphones. The system you will be using is the same as before; thus, most of the code you already wrote to acquire signals, plot them, etc. will remain the same.

In all the following, we will work with a sampling frequency $F_s = 20$kHz, and with a buffer of size `BLK` $= 2048$.

1. To begin, start the acquisition of the audio system, and capture one audio buffer. Plot the resulting signals as a function of time.

Beamforming consists in applying a filter on each microphone signals and to sum their outputs to form the beamformer signal $y(t)$, as showed in Figure 5. As explained during the course, one

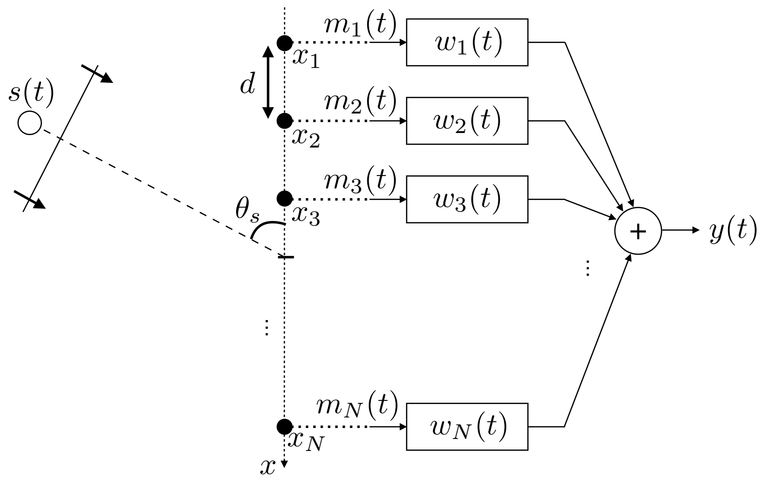


FIGURE 5 – TODO.

will use filters with a frequency response $W_n(f)$ given by

$$W_n(f) = \text{TF}[w_n(t)] = e^{-j2\pi \frac{f}{c} x_n \cos\theta_0},$$

where x_n is the position of the n^{th} microphone along its axis, and θ_0 is the angular direction in which the beamformer is focalized.

2.1 Coding the beamformer filters and analyzing their properties

2. Write the position x_n as a function of n and interspace d. As a convention, the first microphone number is selected as 0, and the origin of the frame is placed at the center of the array.

3. Propose a function `beam_filter` returning the filter frequency response for one microphone number `mic_nb`. The function prototype must be :

```
def beam_filter(array, freq_vector, theta0=0, mic_nb: int = 0):
"""Compute the filter frequency response of a DSB beamformer for one
                                            microphone

Args:
array (array_server obj): array structure controlling the acquisition
                                            system.
freq_vector (np.array): frequency vector.
theta0 (int, optional): focusing angular direction (in degrees). Defaults
                                            to 0.
mic_nb (int, optional): microphone number. Defaults to 0.
```

```
Returns:
np.array: the filter frequency response. Shape is (len(freq_vector),).
"""

    # Microphone position x
    x = # TO BE COMPLETED
    # Filter's frequency response
    return # TO BE COMPLETED
```

4. Plot the two frequency responses obtained for two filters associated to two different microphone outputs when $\theta_0 = 0°$ and for frequencies between 0 and 5kHz. Explain the effect of these filters on the signals.
5. Compare again the filters obtained when $\theta_0 = 90°$. Explain the differences.

2.2 Using the filters : coding of the beamforming

Basically, the beamforming algorithm is the following :

(a) acquire an audio frame
(b) compute the corresponding FFT
(c) analyze the FFT to define which frequency(ies) you would like to localize
(d) restrict the FFT to the frequencies of interest
(e) for one given θ_0, for the frequencies selected before, and for each microphone :
   - compute the corresponding filters frequency responses with the `beam_filter` function
   - apply these filters to the microphone outputs

(f) compute the beamformer output associated to the angular polarization θ_0
(g) repeat all these last steps for each θ_0 you want to test
(h) finally, decide of the angular position of the source by detecting for which θ_0 the beamformer output is maximum.

For now, we will try to localize only one frequency. We will select here $F_0 = 1\text{kHz}$ the frequency of the sinus tone to localize (and emitted by your phone through one of the application listed in the introduction).

6. **Step (a) and (b) :** After acquiring an audio buffer, compute its FFT in an array `M_fft`. Plot the result of this analysis as a function of the frequency when emitting a pure sine tone with a frequency $F_0 = 1\text{kHz}$.
7. **Step (c) and (d) :** Among all the frequencies you obtained from the FFT, select the one corresponding to the source frequency. Give its exact value and index k_0 in the frequency array, and collect the corresponding FFT values of each microphone outputs in one vector `M` of length N.
8. **Step (e) :** In a loop among all microphones, compute each filters for the position θ_0 and the frequency value you obtained in the previous step. Apply then these filters to the array `M` defined before.
9. **Step (f) :** Combine then the filters outputs to form the beamformer output $Y_{\theta_0}[k_0]$. $Y_{\theta_0}[k_0]$ *is obviously a complex value which corresponds to the frequency contribution of the source to the* k_0^{th} *frequency component of the beamformer output when focalized in the direction* θ_0. Compute then the corresponding power $P(\theta_0)$ at k_0 of the beamformer output.
10. For a direction θ_0 of your choice, compute $P(\theta_0)$ for (i) a source emitting from a direction close to θ_0, or (ii) far from it. Compare the two values.
11. **Step (g) :** Repeat now the previous code in a loop for θ_0 values ranging from 0 to 180°. You should then obtain an array P where each value corresponds to the power of the

beamformer output at F_0 for each angular polarization. Plot the array P as a function of the angle θ_0.

12. **Step (h) :** Find the θ_0 value corresponding to position of the maximum in P and compare it with the actual (but approximate) position of the sound source.

2.3 Analyzing the beamformer performances

From now on, you can use your own code written in Section 2.2, or use the provided `beamformer` function which exactly reproduces the beamformer algorithm. You might then add `from beamformer import beamformer` in your Notebook before being able to use the `beamformer` function.

```
def beamformer(buffer, theta, F0, Fs):
    """Compute the energy map from a Delay-And-Sum beamforming

    Args:
        buffer (np.array): audio buffer, of size N x BLK_SIZE
        theta (np.array): array of angular value in degrees listing the
                                                polarization angle of the
                                                beamformer
        F0 (float): source frequency to localize
        Fs (float): sampling frequency
    """
```

13. Plot the energy maps you obtain when using source frequencies $F_0 = 400$Hz, $F_0 = 1$kHz, $F_0 = 2$kHz and $F_0 = 4$kHz emitting from a fixed arbitrary position. Comment and explain carefully the differences between these curves.
14. For a frequency $F_0 = 1$kHz and a source moving aroud the array, plot the estimated position as a function of time. Comment the effectiveness of the approach and its limits.