SASP Homework Assignment No. 1 (HWA1)

Mattia Montanari, Flavio Ingenito

Contents

0.1 Code implementation discussion

The resolution of the proposed homework is organized in the most modular way possible. Indeed, we can begin the analysis of the results by understanding the structure of the project, and then analyze each of its modules in the detail: homework.m is the main file, compute_psudospectum.m is the function that makes it possible to computes the source's directions of arrival (DOAs) returning the time-varying pseudospectrum, multichannel_stft.m and my_STFT.m are used together to calculate the Short Time Fourier Transform without using the matlab's built-in function, while my_plots.m and my_videos.m are the functions to plot all the required graphs and video showing the source arrival direction for each time instant.

Homework.m Within the main file, as said before, it's possible to find all the various operations that, starting from the reading of the audio sample, leads to the estimation of the source's DOAs. The code starts initializing the variable containing the multichannel audio sample, then we find at line 9 the normalization of it, with the purpose of having the volumes of all 16 channels similar so as to have more homogeneous and obvious results.

Afterward also the variables that will be needed during the course of the program flow are initialized, such as the range of theta, the number of microphones, the distance between them and the speed of sound.

The frequency *freq_max* is also defined, which is the frequency for which no aliasing is created, neither in the frequency domain, so that it is less than the Nyquist frequency, nor in the spatial frequency domain.

At this point, the main file continues calling one after the other all the functions above-mentioned in the introductory paragraph.

multichannel_stft.m and my_STFT.m These two functions can be commented together as strictly connected to obtain a common goal. The function *my_STFT.m*, implements a standard STFT algorithm in matlab language without using any built-in function.
First, the function initializes the output variable *audio_stft* as a matrix of zeros. This matrix will store the STFT coefficients. Next, it initializes a Hann window which will be used for windowing the segments of the input audio. Then, the function iterates over each time segment of the input audio signal. Within each iteration, it extracts the current segment based on the window length and overlap. It applies the Hann window to the segment, then computes its FFT with the specified number of points (nfft). Finally, it stores the first half of the FFT result in the corresponding column of the *audio_stft* matrix.
The function *multichannel_stft.m*, on the other hand, simply calls the just commented *my_STFT.m* over all the channel of the audio sample taken as input. The main variable returned by this function will be indeed the 3D matrix *audio_multichannel_stft*, 3D because it will contain a 2D matrix obtained with *my_STFT.m* for each channel (the third dimension of the matrix).

After this the main file is ready to calculate the pseudospectrum.

compute_pseudospectrum.m We can say that this function is the heart of our code: let us see its behavior. First the matrix p, which will contain the calculated

pseudofrequency spectrum, is initialized. Then we start a first loop on the frequencies and a second on the angles, which in detail, range from -90 to +90 degrees.

Then we calculate, for each of the sixteen microphones the propagation vector, *a*, by first calculating *omega_s* (spatial frequency), which is the frequency of the complex sine wave that makes up the propagation vector.

At this point, with an additional third nested loop, for each time interval created by the STFT, we calculate *R_hat*, or the sample estimate of the covariance matrix. In the specifics of the calculation, a sub-matrix is extracted from *audio_stft* by specifying the indices *i_freq* and *i_time*. These indices select a specific frequency and time instant in the matrix containing the STFT coefficients of the audio for all the channels. The function *squeeze* removes the dimension of length 1, producing a matrix of a smaller dimension (the matrix becomes a vector). This vector is then multiplied by its conjugate transposition. The result, namely the matrix *p*, is a 3D matrix which represent the power related to each possible direction assumed by the source (angle) at different time instants considering individually all the frequencies of the spectrum.

This isn't the desired result yet. In fact we are interested in having a matrix of only two dimensions which shows the power associated to every source's direction of arrival at different time instants but considering all the frequency together in their totality. We end up with such a 3D matrix only because the adopted algorithm works only with narrow band signals, but since we are working with a wide band signal it's then computed the geometric mean over the frequencies. Finally the function *squeeze* is used another time to remove the third dimension (the one of the frequencies), not relevant anymore.

In summary, we can say that, by first analyzing all the frequencies, sweeping each corner, for each of the 16 microphones; this function calculates *p*, that is, the power coming from these dimensions of analysis, allowing us to understand, based on where the power peaks are located, where the sound source is located in space for each time instant.

After the *compute_pseudospectrum.m* function, in the main file are now easily extracted from the matrix *p* the source's DOAs, simply saving in a vector the indexes of the peaks of power associated to the range of angles that goes from -90 to 90 degrees. This vector will tell us the direction estimated of the source for each instant. Finally the main file is concluded calling the last two function to produce plots and videos to show the results estimated

my_plots.m and my_videos.m There's no point in going deeper in the explanation of these functions since the results will be commented in the following chapter.

0.2 Results and graphs discussion

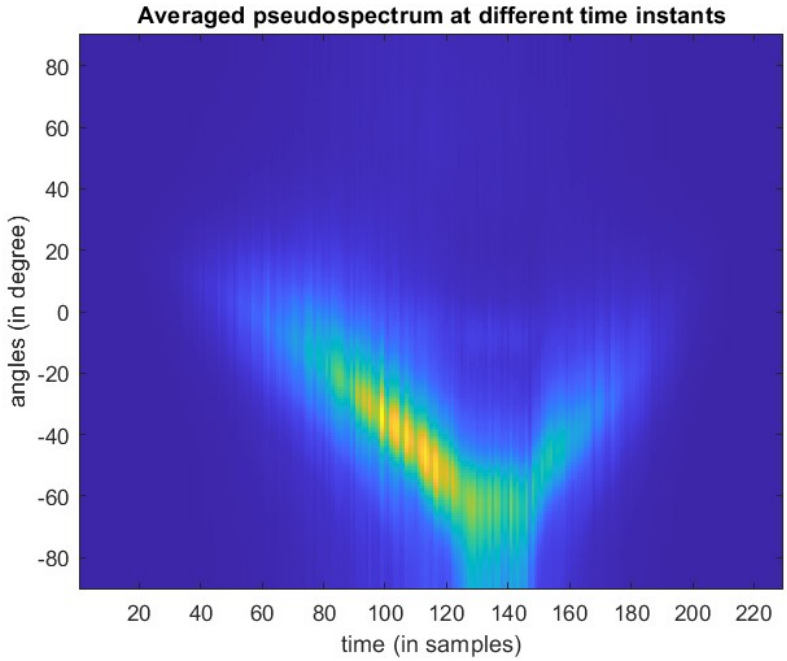

Figure 1. Averaged pseudospectrum at different time instants

In the graph showed in Figure 1 we have, for each time instant in samples, the relative power at each possible DOA. The areas of the graph tending toward yellow represent the DOAs at which the algorithm has estimated a higher power. To give an example, at time instant 100, thanks to this algorithm, we expect the direction of arrival of the source to be an angle of about -40 degrees respect the uniform distribution of microphones.

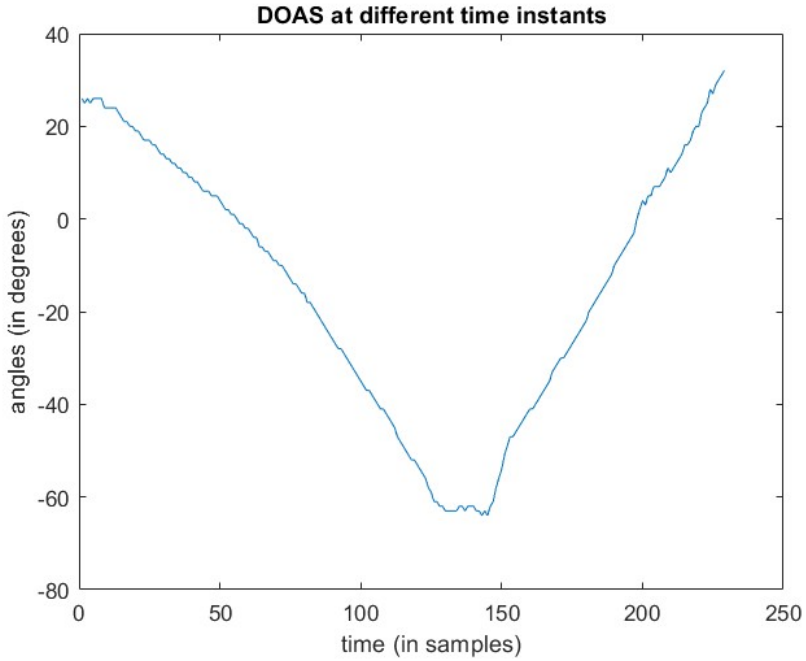

Figure 2. DOAs at different time instants

Figure 1 represents the same concept as Figure 2, the only difference is that it shows only the power peaks, going to exclude all angles with lower power. In fact, thanks to this graph, it is easier to understand the path taken by the source in the audio sample. We can in fact notice that the source starts from an angle of about 25 degrees, reaches about -60 degrees, and then backtracks to an angle of about 30 degrees.

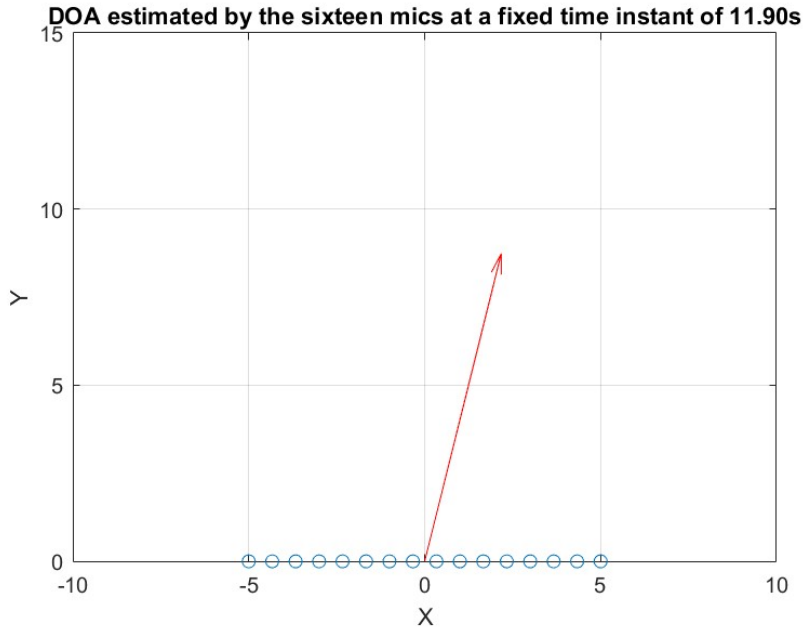

Figure 3. DOA estimated at a fixed time instant

The graph present in Figure 3 instead can be used to better understanding the concrete meaning of all this simulation that we have done. The figure in fact shows the 2D scene in space expliciting how the sixteen microphones are arranged (represented by the celestial circles on the x-axis), with respect to the arrow, that indicates the estimated direction from which the source comes at a fixed time instant. The arrow start from the center of the microphone array since the whole distribution can be considered as a single point in space. This is possible since one of the assumptions of the model adopted is that the sound source is located in the far-field. This graph, in particular, points out that the direction of the source at 11.90 seconds (time instant randomly chosen) is around -15 degrees.

The video *DOAs_estimation.avi* (present in the folder of the homework) simply extend this concept putting together consecutively graphs like the one showed in Figure 3 for all the time instants, going from the start to the end of the audio sample. The result will be indeed an arrow centered in the origin of the microphones array distribution which dynamically point at the estimated direction of arrival of the source.

Note that because of the way the model has been implemented in our code, both in the video and in Figure 3, 0 degrees coincides with the direction perpendicular to the x-axis, so for positive angles the arrow moves to the left and vice versa for negative angles (as if the standard Cartesian system would been rotated by 90 degrees).