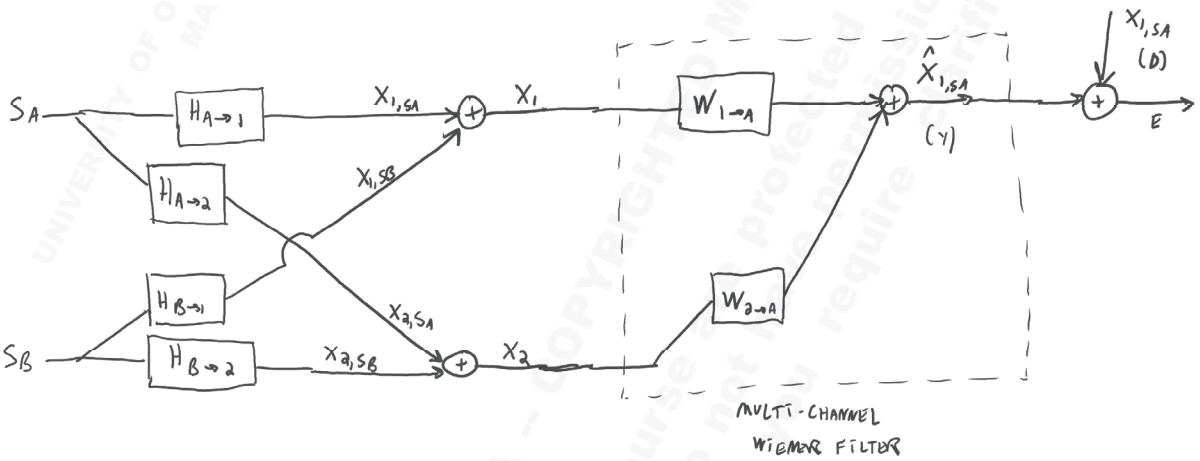
Assignment #3

Consider a noise canceling system (source separation, source extraction, de-mixing system) with two sensors (microphones here) measuring acoustic mixtures from two speech sources S_A and S_B (talkers). The system is shown below to perform the extraction of X_{1,S_A} (which is the component from speech source S_A received at sensor 1) from the mixtures X_1 and X_2 measured at the sensors. A similar system can also be designed for extracting X_{1,S_B} (or X_{2,S_A}, X_{2,S_B}).



The Wiener filter to be used here is a multichannel filter, using more than one input signal to produce an output signal.

Components from the two sources S_A and S_B appear in each mixture X_1 and X_2 (each microphone signal). In the mixtures, the components from the same source are fully correlated with each other (X_{1,S_A} and X_{2,S_A} can be predicted 100% from each other, X_{1,S_B} and X_{2,S_B} can be predicted 100% from each other), but the components from different sources are not correlated (e.g. X_{1,S_A} and X_{1,S_B}), as the two sources S_A and S_B are uncorrelated. The total mixtures $X_1 = X_{1,S_A} + X_{1,S_B}$ and $X_2 = X_{2,S_A} + X_{2,S_B}$ at the two microphones are not fully correlated, and we cannot fully predict one mixture (or one source component in a mixture) directly from the other mixture. Therefore, the performance of a single channel Wiener filter would be limited.

A multichannel solution can improve the solution here. Intuitively, in the frequency domain or z-transform domain, from the sources to the mixtures we have a 2x2 mixing system, and from the mixtures to the separated/extracted sources we have another 2x2 de-mixing system, and the 2x2 de-mixing solution is the inverse of the 2x2 mixing matrix. If we are only extracting one signal (as here), then the de-mixing becomes a 2x1 system, i.e., only the top row of the 2x2 inverse matrix is kept. So, in principle, there exists an ideal multichannel solution for this problem. This justifies why the use of a multichannel Wiener filter with two input signals will outperform the use of a

single channel Wiener filter. Here we assume that the ideal multichannel Wiener filter solution is also causal and can be approximated by FIR filters.

The transfer functions / frequency responses / impulse responses for the LTI mixing systems $H_{i\to j}$ are unknown, so the de-mixing coefficients cannot be obtained by a direct inversion of the 2x2 mixing system in the transform domain. We can rely instead on the estimated statistics (correlations) of the signals $x_1(n)$, $x_2(n)$ measured at the microphones to compute a time domain FIR causal Wiener solution.

The system can be in 3 states:

- Only Speaker A (S_A) is talking ("interval 1"),
- Only Speaker B (S_B) is talking ("interval 2"),
- Both Speaker A (S_A) and Speaker B (S_B) are talking ("interval 3").

The objective is that the filtering should be able to isolate/extract/separate/de-mix the speech of Speaker A and Speaker B as received at one of the sensors (called the reference sensor), when both sources are simultaneously talking ("interval 3"). To achieve this, you need to formulate two multichannel Wiener solutions (one for Speaker A extraction, one for Speaker B extraction), where the "desired" signal $d(n)$ is the speech of the target speaker received at the reference sensor (in one case Speaker A during "interval 1" and in the other case Speaker B during "interval 2"). The reference microphone is (arbitrarily) chosen to be the first microphone. The filter input signals $x_1(n)$, $x_2(n)$ are the mixtures from the microphones. The correlations that you will use to compute the multichannel Wiener solutions will involve statistics that need to be measured in different states or intervals, because the required signals $d(n)$ for the two solutions are only available when one of the two speech sources are inactive ("interval 1" or "interval 2"). So the cross-correlations involving $d(n)$ must be computed using "interval 1" or "interval 2", depending on which filter is designed. However, the auto-correlations involving only the $x_1(n)$, $x_2(n)$ signals must be computed when all sources are active ("interval 3"), so that the statistics of the auto-correlations include all sources. To achieve this, you are provided with recordings for the 3 different states or intervals:

- mic1_SA_only_interval1.wav
- mic2_SA_only_interval1.wav
- mic1_SB_only_interval2.wav
- mic2_SB_only_interval2.wav
- mic1_SA_and_SB_interval3.wav
- mic2_SA_and_SB_interval3.wav

Question 1

For each speaker, compute an optimal multichannel Wiener filter and use it during "interval 3" to extract the speaker component as received at the reference sensor, from the measured mixtures $x_1(n)$, $x_2(n)$ of the two speakers. In each case, save the extracted speaker signal to a .wav file, so that you can listen to it and perceptually evaluate the performance of your solution.

You can use 10 coefficients for each filter/channel inside the multichannel Wiener filter, i.e., 20 coefficients in total. You should be able to nearly perfectly extract the signals from the two speakers. You need to submit the .wav files of the extracted speakers, in addition to your results (objective scores, below) and your code.

As an objective performance indicator, compute the following Signal to Interferer SIR gains:

- For the filter designed to extract the Speaker A component:
    - Using the Speaker A-only sound files (interval 1), compute the output of the filter, and compute the ratio between the output average power and the input average power for microphone 1. The is the "power gain" for the Speaker A source.
    - Using the Speaker B-only sound files (interval 2), compute the output of the filter, and compute the ratio between the output average power and the input average power for microphone. The is the "power gain" for the Speaker B source.
    - Compute the ratio between the power gain for Speaker A source over the power gain for Speaker B source. This gives you the Signal to Interferer SIR gain, where Speaker A is the signal and Speaker B is the interferer. Convert this SIR gain in dB (10 log10()) and provide it in your report.
- Repeat the same computations (filtering and computation of SIR gain) using the multichannel filter designed to extract Speaker B, where Speaker B becomes the source and Speaker A becomes the interferer.

Note: some code provided at the end of Question 2 may also be used for Question 1.

Note: why aren't we using the MMSE to evaluate performance here, since the Wiener solution minimizes the MSE? The MMSE is of course also a valid metric to compute, but arguably it would be less informative here. For example, if the power of $d(n)$ and $e_{opt}(n)$ are evaluated when both sources are active, then the normalized MMSE may be only -3 dB (i.e., reduction by 50% from the power of $d(n)$ to the power of $e_{opt}(n)$, as one of the sources is removed in $e_{opt}(n)$ and the other source remains). On the other hand, if the power of $d(n)$ and $e_{opt}(n)$ are evaluated when only one source is active, then the normalized MMSE does not reflect the attenuation of the

interferer (since there is no interferer), instead it only reflects how accurately the system predicts the desired component and if some distortion is introduced by the filtering process.

Question 2

Add some uncorrelated zero-mean white Gaussian noise to each microphone signal (each .wav file provided), with an SNR level relative to the signal level at each microphone. Use the following code for this. Perform the experiment for different SNRs: 30dB, 20dB, 10dB, 0dB. For each SNR, compute the same SIR gains as those calculated in Question 1. You should observe a gradual decrease of performance. For this part, it is not required to submit .wav sound files with your report.

```
SNR=30; % note: if SNR is very large (ex. >100 dB), it is as if there is no noise (Q1)
rng(13); % set random generator seed, so that the noise sequences remain the same between
         % different runs, allowing a more direct comparison

% reading signals and adding noise (when appropriate)
[speechA_mic1_interval1, fs]=audioread('mic1_SA_only_interval1.wav');
[speechA_mic2_interval1, fs]=audioread('mic2_SA_only_interval1.wav');
speechA_mic1_interval1_rms=sqrt(sum(abs(speechA_mic1_interval1).^2)/length(speechA_mic1_interval1));
noise=randn(length(speechA_mic1_interval1),1);
noise=noise*speechA_mic1_interval1_rms/(10.^(SNR/20));
speechA_mic1_interval1=speechA_mic1_interval1+noise;
speechA_mic2_interval1_rms=sqrt(sum(abs(speechA_mic2_interval1).^2)/length(speechA_mic2_interval1));
noise=randn(length(speechA_mic2_interval1),1);
noise=noise*speechA_mic2_interval1_rms/(10.^(SNR/20));
speechA_mic2_interval1=speechA_mic2_interval1+noise;

[speechB_mic1_interval2, fs]=audioread('mic1_SB_only_interval2.wav');
[speechB_mic2_interval2, fs]=audioread('mic2_SB_only_interval2.wav');
speechB_mic1_interval2_rms=sqrt(sum(abs(speechB_mic1_interval2).^2)/length(speechB_mic1_interval2));
noise=randn(length(speechB_mic1_interval2),1);
noise=noise*speechB_mic1_interval2_rms/(10.^(SNR/20));
speechB_mic1_interval2=speechB_mic1_interval2+noise;
speechB_mic2_interval2_rms=sqrt(sum(abs(speechB_mic2_interval2).^2)/length(speechB_mic2_interval2));
noise=randn(length(speechB_mic2_interval2),1);
noise=noise*speechB_mic2_interval2_rms/(10.^(SNR/20));
speechB_mic2_interval2=speechB_mic2_interval2+noise;

[speechAB_mic1_interval3, fs]=audioread('mic1_SA_and_SB_interval3.wav');
[speechAB_mic2_interval3, fs]=audioread('mic2_SA_and_SB_interval3.wav');
speechAB_mic1_interval3_rms=sqrt(sum(abs(speechAB_mic1_interval3).^2)/length(speechAB_mic1_interval3));
noise=randn(length(speechAB_mic1_interval3),1);
noise=noise*speechAB_mic1_interval3_rms/(10.^(SNR/20));
speechAB_mic1_interval3=speechAB_mic1_interval3+noise;
speechAB_mic2_interval3_rms=sqrt(sum(abs(speechAB_mic2_interval3).^2)/length(speechAB_mic2_interval3));
noise=randn(length(speechAB_mic2_interval3),1);
noise=noise*speechAB_mic2_interval3_rms/(10.^(SNR/20));
speechAB_mic2_interval3=speechAB_mic2_interval3+noise;
```