Acoustic modem project

Session 1: Audio playback, recording and analysis

Alexander Bertrand, Paschalis Tsiaflakis, Hanne Deprez[1]

September 2018

Goal: a) Using a simulink model to simultaneously play and record audio signals, and to analyse these signals. b) Estimating the Shannon bound or channel capacity of the acoustic channel used by the acoustic modem.

Requirements: Matlab/Simulink in Windows, a sound card, a loudspeaker and a microphone.

Required files from DSP-CIS website: *recplay.mdl*

Required files from previous sessions: none

Outcome: 3 m-files (+2 optional): *initparams.m*, *test_playrec.m*, *analyze_rec.m*, *compute_shannon.m* (optional), *plot_shannon_vs_distance.m* (optional)

Deliverables: See Session 2

Important remarks:

- Students who are not familiar with Matlab should first go through the Matlab tutorial, which can be found on the DSP-CIS website. It is recommended to make all the exercises in this tutorial. If you have problems, ask your TA for help. Students who are already familiar with Matlab can skip this tutorial. However, it is highly recommended to at least go through Section 14, where a number of commands are listed that may prove to be very useful during this project.
- Throughout the entire project, you will work under Windows.
- Make sure you set the Matlab path to your home-folder (and **not** on the local hard-drive, since this will be erased after you log off).
- Work in m-files, and save your work regularly, in order to be able to repeat your experiments later on.
- If you use a stereo loudspeaker set, it is recommended to switch off one of the two loudspeakers. Try to optimize the dynamic range of loudspeakers and microphone, but **scale your signals to avoid clipping!**
- Important: In Windows 10, there is an internal software component that applies an automatic gain control to the microphone inputs, which may be harmful when implementing the acoustic modem. Please make sure this automatic gain control is disabled (also for all sessions that will follow). To disable this, first plug in your microphone, then go to control panel, 'Hardware and sound', 'Sound', 'Recording', 'Front Microphone', 'Advanced',

[1]For questions and remarks: `jeroen.verdyck@esat.kuleuven.be`

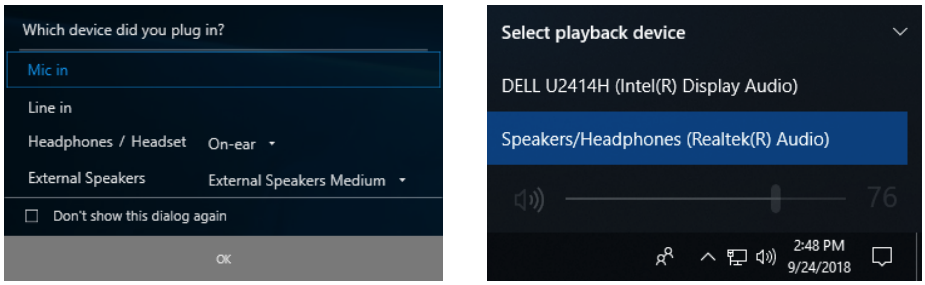

Figure 1: Audio input/output settings.

and disable the check box for 'Allow applications to take control of this device'. This should disable the automatic gain control.

- In all experiments in this session, use a sampling frequency $f_s = 16000$ Hz, if not specified otherwise.
- All magnitude plots should be visualized on a dB-scale (for the y-axis).
- Try not to re-invent the wheel, i.e., do not implement every functionality from scratch: if Matlab has it, you can use it!
- Make sure to select the correct audio device type when connecting the microphone, as well as the correct audio output device (see Figure 1).

1 Exercise 1-1: Audio playback/recording

In this exercise, you will use a Simulink model that allows for simultaneous audio playback and recording. The playbacked audio signal is taken from a Matlab workspace variable (*simin*). The simultaneous recorded audio signal is saved in another workspace variable (*simout*). This simple Simulink scheme will be the backbone for the Matlab algorithms that you will develop during the upcoming exercise sessions.

1. Download the Simulink model *recplay.mdl* from the DSP-CIS website.
2. Open the Simulink model, but do not change any setting.
3. Create a Matlab function *[simin,nbsecs,fs]=initparams(toplay,fs)*, where the input *toplay* is a vector that contains the samples of an audio signal, and *fs* is the sampling frequency at which the playback/recording must operate. The function should return the following variables:
   - *simin*: This is a 2-column matrix. In case of mono transmission, which we will use during the first six sessions, both columns contain the vector *toplay*, with 2 seconds of silence in the beginning, and one second of silence at the end.

- *nbsecs*: The number of seconds the playback/recording must run (should be at least as long as the signal in *simin*).
- *fs*: This is the same *fs* that is given as an input.

The output variables *fs*, *simin* and *nbsecs* are used by the Simulink model *recplay.mdl*, and should always be set before running the model by using the command
`[simin,nbsecs,fs]=initparams(toplay,fs);`.

4. Make an m-file *test_playrec.m* that
   (a) sets *fs* to 16000.
   (b) generates a 2-seconds long sine wave of 1500 Hz, saved in the variable *sinewave*.
   (c) feeds *sinewave* and *fs* to the function *initparams* to set the Simulink variables.
   (d) runs the Simulink model (use the command `sim('recplay')`).
   (e) creates a variable *out* that collects the Simulink output *simout* by using the command `out=simout.signals.values;`
   (f) plays the audio vector *out* at the correct sampling frequency.

   If you run *test_playrec.m*, and if you hear the same sine of 1500Hz (twice), you can go to the next exercise.

2 Exercise 1-2: Time-frequency analysis of recorded signals

1. Create an m-file with the name '*analyze_rec.m*'. In the beginning of the file, define 3 variables that can be set by the user: *sig*, *fs* and *dftsize*. Do not define *analyze_rec* as a function, i.e., these input parameters must be hardcoded in the m-file. The m-file should perform the following tasks (try to re-use some code from the previous exercise):
   (a) Play and record the audio signal in *sig* at sampling frequency *fs* with the Simulink model from the previous exercise.
   (b) Plot the spectrogram (with a DFT size equal to *dftsize*) of the signal in *sig* and the signal recorded by the microphone. Both should be plotted in one figure below each other (using subplot). Note that spectrogram $\neq$ spectrum (google if you do not know the difference). Make sure the time-axis and frequency axis display the correct scales (i.e., seconds and Hz). Use a dB scale for the magnitude (i.e., the colors of the spectrogram).

(c) In a separate figure, do the same for the PSD of the transmitted and recorded signal (again a dB scale). Calculate the PSD by averaging multiple periodogram spectrum estimates, all with a DFT size equal to *dftsize*. This way of estimating the PSD is referred to as *Bartlett's method* or *Welch's method*. What is the difference between these two methods, and how does their result differ from the actual PSD?

2. Set *sig* in *analyze_rec.m* to a sine signal of 400 Hz, and execute the file. Take a look at the PSD and the spectrogram of the *transmitted* signal (i.e., what is sent to the loudspeakers, *not* what is recorded). Is this what you expected? Are there other frequencies besides the 400 Hz present in the PSD/spectrogram? Why (not)?

3. What is the influence of the DFT size? Try to answer first and then verify experimentally.

4. Now compare the spectrogram/PSD of the transmitted signal and the recorded signal. Is this what you expected?

5. Repeat the previous experiment, but add a DC component to the sine wave. What do you observe now? Is this what you expected?

6. Did you properly scale the signal in the previous experiment to avoid clipping (i.e., Matlab expects audio samples to be in the interval $[-1, 1]$)? If you did: nice! What is the influence of clipping in the spectrum? Try it out. Is this harmful for the acoustic modem?

7. Add a line of code in *initparams.m* that automatically scales the input signal to the interval $[-1, 1]$ to avoid clipping. Note that clipping can still occur due to non-linearities in the speakers/microphone, in particular when operating at a high volume. Hence, be careful!

8. Define *sig* in *analyze_rec.m* as a signal containing a sum of sines of 100, 200, 500, 1000, 1500, 2000, 4000 and 6000 Hz (with equal amplitude), and execute the file. Take a look at the PSD and the spectrogram. Can you locate the original frequencies in the spectrogram? Is there a difference between the PSD/spectrogram of the transmitted and the recorded signal? Can you explain this?

9. Define *sig* in *analyze_rec.m* as a white noise signal, and execute the file. What relevant information for the acoustic modem can be found in the spectrogram of the recorded signal? This white noise experiment is important for later sessions.

10. In the previous white noise experiment, does the spectrogram of the recorded white-noise signal change significantly in time? Why is this question relevant for the acoustic modem?

11. Redo the white noise experiment, but move the microphone around while recording. What do you observe now in the spectrogram?

3 Exercise 1-3 *(Extras)*: Your best friend Shannon

1. What is the useful frequency range that can be used by the acoustic modem? (*Hint:* google your hardware!)

2. Create an m-file *compute_shannon.m* that automatically conducts an experiment to estimate the capacity of the channel that you will use for the acoustic modem. Note that the sampling frequency determines the bandwidth (reminder: Nyquist), and this must be a user-defined input, hardcoded in the beginning of the file. You may assume during this class that the background noise has a stationary Gaussian distribution. Notice that the noise is *not* white, and the channel is *not* flat.
   The Shannon capacity formula (with discretization of an integral over the bandwidth) is given by

$$C_{\text{channel}} \approx \frac{f_s}{2N} \sum_{k=1}^{N} \log_2(1 + \frac{P_s(k)}{P_n(k)})$$

   where N is the number of frequency bins (DFTsize=2N), and where $P_s(k)$ and $P_n(k)$ are the recorded signal and noise power in bin k (can you explain the scaling factor?). Noise power can be determined by recording without playing a signal. Signal power can be determined by playing the signal and recording the signal+noise, then subtract the noise PSD from the signal+noise PSD (why is this equal to the signal PSD?).

3. With this m-file, determine the capacity of the acoustic channel, assuming $f_s = 16000$ Hz, with distance between microphone and loudspeaker < 10 cm.

4. How does this number (channel capacity) need to be interpreted, i.e., what does it mean?

5. Do the same for $f_s = 44100$ Hz.

6. *Extra (for the die-hards)*: Determine how the capacity changes with distance between loudspeaker and microphone, and make a dependency plot. Save the plot as a mat-file (and an m-file *plot_shannon_vs_distance.m* to actually plot it) and show it during the milestone demo (next week).