Module 5: Record Audio Signals in MATLAB

© 2019 Christoph Studer (studer@cornell.edu); Version 0.2

The main goal of this module is to learn how to record audio signals in MATLAB from a microphone. We will later use the microphone as our "receive antenna" of the acoustic communication system. You will also see how to simultaneously play a signal from the loudspeakers and record it with the microphone, which will be used later to test our communication system. ***Remember: Whenever you are stuck or are interested in learning more details about a specific aspect, feel free to ask us—we are here to help!!***

---

7 Setting up the Microphone

So far, you have learned how to play back sampled signals from the loudspeakers and how to generate and analyze signals. In this module, you will learn how to record audio signals in MATLAB with the provided USB microphone. We will then show how to play a sampled signal through the loudspeakers and, at the same time, to record the signal with the microphone. As you will see, both of these tasks are quite simple in MATLAB. Finally, we will analyze recorded signals and demonstrate some basic properties of acoustic channels (channels in which acoustic waves travel through the air). All of these aspects will be important in building our acoustic communication system.

7.1 First Steps

The Beisiwo microphone is a cheap USB powered microphone that also receives the recorded and sampled audio data via USB. Connect the USB microphone to your computer and make sure the green LED is on—if the green LED is off, press the button next to it. If the LED is flashing (and does not stop flashing), then unplug the microphone and plug it back in. Unfortunately, if MATLAB is already running, you must restart MATLAB. Otherwise, MATLAB does not always realize that a new audio interface has been connected to the computer. After starting MATLAB, execute the following command:

```
get_audio_info;
```

If MATLAB throws an error, then, most likely, you do not have the functions we provide in the Current Folder (make sure that you can see the MATLAB functions we provide on the left side of the MATLAB desktop). If everything worked, you should see a list containing all input and all output devices:

```
Available audio inputs:
InpID=0 | Primary Sound Capture Driver (Windows DirectSound)
InpID=1 | Microphone ((LCS) USB Audio Device) (Windows DirectSound)
InpID=2 | Microphone Array (Realtek Audio) (Windows DirectSound)

Available audio outputs:
OutID=3 | Primary Sound Driver (Windows DirectSound)
OutID=4 | Speakers / Headphones (Realtek Audio) (Windows DirectSound)
```

Note that the list may differ for your computer. What is important is to see an audio *input device* called

```
Microphone ((LCS) USB Audio Device) (Windows DirectSound)
```

This is the USB microphone—please write down the associated `InpID`. Also, assign the ID to a variable called `InpID`, e.g., type `InpID=1` if the LCS audio device ID is 1. You will need to specify this ID whenever you want to record an audio signal from the USB microphone. While you are at it, also define `OutID` so that it corresponds to the index of the output device called

```
Speakers / Headphones (Realtek Audio) (Windows DirectSound)
```

7.2 Record your own Voice

You are now ready to record your own voice. Fortunately, with the function we provide, this is straightforward. If you execute the MATLAB command

```
y = record_audio(sec,FS,InpID);
```

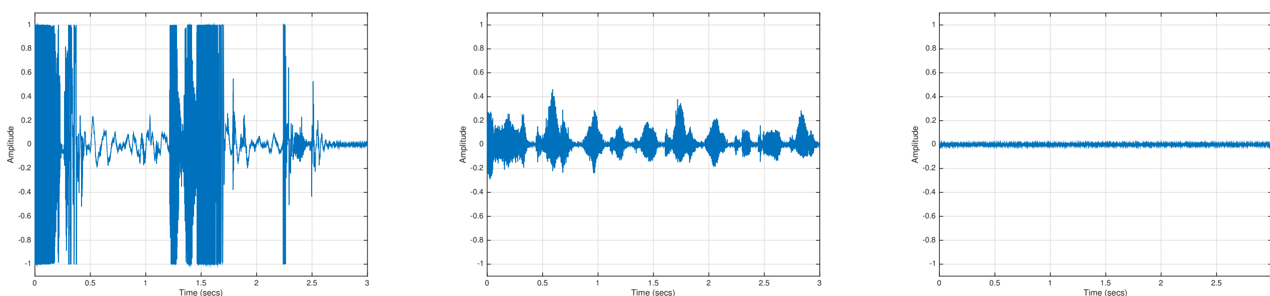
then you are recording for `sec` seconds with a sampling rate of `FS`. The result of the recording will be stored in the vector `y`. Define `sec=3` and `FS=44100` and execute the above command. After pressing enter, speak into the microphone, as this command starts recording almost immediately. To test whether you recorded something, you can play your recording back by executing the command

```
play_audio(y,FS,OutID);
```

which you have used before a couple of times. If you spoke into the microphone while recording, you should hear your voice. You can also look at the time-domain representation by using the command

```
plot_signal(y,FS);
```

Visualizing the time-domain is useful to see whether your recording was too quiet or too loud.



(a) Signal too loud. (b) Signal level acceptable. (c) Signal too quiet.

Figure 13: Time-domain visualizations of a recorded speech signals. Left: signal level was too loud; middle: signal level was acceptable; right: signal level was too quiet.

Figure 13 shows three examples of recorded speech. The figure on the left shows an example of a signal that was too loud during recording. You can see that many amplitudes are stuck at either $+1$ or -1. In fact, the signal's amplitudes were exceeding these values but the microphone performed something that is known as *clipping*. Clipping ensures that the recorded amplitudes are never outside the interval $[-1, 1]$. If you play back a heavily clipped signal, then it will sound distorted. While clipping is not desirable when

building a communication system (or when recording voice or most other things), electric guitars make heavy use of such distortion effect. The figure in the middle shows an example of a signal that was loud enough to be audible without causing distortion. You can see that no value exceeds the interval $[-1, +1]$ but you can also see changes in the amplitude, which encode your speech signal. The figure on the right shows an example of a signal that was too quiet. In fact, besides noise (or sounds produced by the power supplies, the air-conditioning system, the computer fan, other people in the room, etc.) you cannot see anything recorded. Such audio levels are too quiet and should be avoided. Remember that whenever you want to record something, make sure that the amplitude level of your recording is acceptable. Clipping and distortion will negatively affect communication systems and will result in transmission errors.

Activity 18: Record your voice

Use the functions as described above and record your voice intentionally too loud, too quiet, or exactly right. Inspect the time-domain signals to see the amplitude levels. Try to intentionally distort your voice and listen to the recorded signal. You can greatly vary the recording level by moving closer or further away from the microphone when talking. Also, try to record absolutely nothing and look at the time-domain signal. What do you see? Finally, reduce the sampling rate to $f_s = 8000\,\text{Hz}$ when recording and playing back your voice. Can you explain what happens?

Activity 19: Playback in reverse

Record your voice, reverse the signal, and play it back. To reverse the order of samples in a vector y you can use the following commands

```
index = length(y):-1:1;
y_reverse = y(index);
```

These commands first generate an index vector that starts with the last entry and counts downwards, and then reads out the samples in reverse order and assigns them to the new vector `y_reverse`. As shorter sequence of commands that produces the same output is

```
y_reverse = y(end:-1:1);
```

Try to say your name in reverse (e.g., “Christoph” would be “Hpotsirhc”), record it, and reverse it—does it sound like your name? Can you say entire sentences in reverse so that they are understandable when played back in reverse? Try it out!

7.3 Sampling of Recorded Voice

Let us look at the simplified block diagram in Figure 14, which illustrates the main components involved in recording an audio signal with MATLAB. A microphone converts acoustic sound waves (which are essentially variations in air pressure over time) into continuous electronic signals (voltages). These voltages are then filtered using a so-called low-pass filter with cut-off frequency $f_s/2$. In non-technical terms, frequencies in the analog voltage signal higher than $f_s/2$ will be removed completely—frequencies below $f_s/2$ are left untouched. The filtered signal is then sampled at a sampling rate of f_s in a so-called analog-to-digital converter (ADC, for short). Since frequencies higher than $f_s/2$ cannot be represented when sampling a continuous signals, the low-pass filter is essential in the sampling process. (Maybe you recall the activity above where you sampled your voice at $f_s = 8000\,\text{Hz}$; high frequencies were completely filtered out!) The digitized samples are then transmitted to MATLAB and stored in a vector.

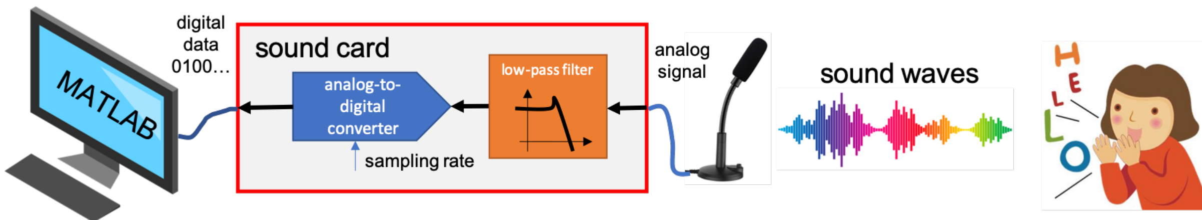


Figure 14: Simplified block diagram of a microphone, sound-card, and computer. The microphone converts air pressure into voltages, which are filtered and sampled. The samples are then transferred to MATLAB.

Important: Figure 14 represents a communication system in which your voice is the transmitter, the air is the wireless channel, and the microphone, sound-card, and MATLAB form the receiver. During this project, we will replace your mouth with loudspeakers and transmit digital information over the air.

So far, we have only inspected the recorded signal in the time-domain. Let us now look at the spectrum of the recorded voice. Use the above commands to record three seconds of voice at a sampling rate of $f_s = 44100$ and with an appropriate signal level. Play the signal back and inspect the time-domain signal to ensure that your recording is OK. Now, plot the spectrum of your recorded signal using the command

```
plot_spectrum(y,FS);
```

where we assumed that `y` was the variable that contained your recording. You should see that your voice is mostly occupying frequencies between 50 Hz and 1000 Hz (this depends on your voice), but higher frequencies exist. High frequencies in your voice are generated by so-called sibilants (hissing sounds; in English caused by pronouncing the letters s, z, sh, and zh). If you would record your voice with a sampling rate of $f_s = 8000$ Hz, then the low-pass filter in the sound-card would remove all frequencies higher than 4000 Hz and all such sibilants would be lost. This is the main reason why your voice sounds different when recording at such low sampling rates. Feel free to try it out!

Let us now look at the spectrogram of your recorded voice. Recall the command to generate a spectrogram by typing the two commands

```
figure(1);
plot_spectrogram(y,FS);
```

You should see a spectrogram that looks similar to that shown on the left side of Figure 15. Clearly, your voice consists of low- as well as high-frequency sounds, depending on what you recorded. If you type

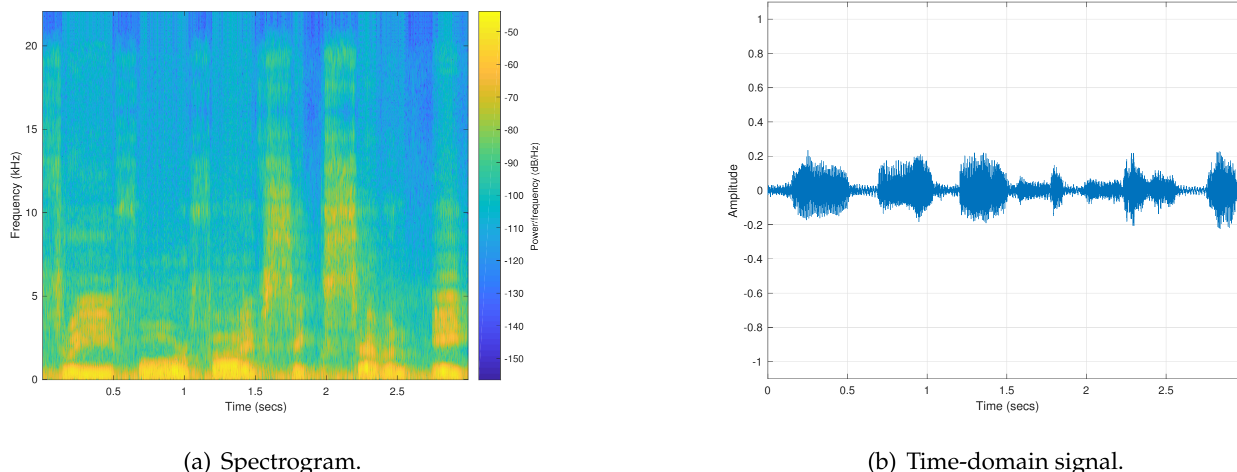
```
figure(2);
plot_signal(y,FS);
```

then you can compare the spectrogram to the time-domain signal. You should see a time-domain signal that looks similar to that shown on the right side of Figure 15. By comparing both visualizations of the same recorded signal, you should be able to see which parts of the time-domain signal correspond to the spectrogram by placing the two plots close to each other.

Activity 20: Record your voice and play it back at a different sampling rate

Take the recorded voice from before but play it back at a different sampling rate by typing

```
play_audio(y,FS*1.5,OutID)
```



(a) Spectrogram.

(b) Time-domain signal.

Figure 15: Example spectrogram and time-domain signal of recorded voice.

This command uses a sampling rate that is 1.5× higher than the one you used while recording. Your voice should have higher pitch (1.5×) and also be shorter (1.5×). Feel free to record different things and play them back using higher and lower sampling rates.

One interesting experiment is to use your normal voice but speak very slowly. You can then play it back using a faster sampling rate. If done right, the speed of your voice would be normal but the frequency would be higher. This sounds almost as if you inhaled Helium. In fact, this experiment mimics the situation of having a faster speed of sound.

7.4 Play Audio Signal and Record

You should now know how to play audio signals and how to record audio signals. Before, we explain how to *simultaneously* play and record audio signals with a *single* computer, you have to play audio signals from one computer and record them with another computer. To this end, we teamed your group up with another group. ***Talk to us if you do not remember the number of the other group. Also, if the other group is not ready yet, then postpone the following activity!*** If the other group is ready, then open MATLAB on the nearest computer to the other group and implement a script that either plays back or records (you can also load your code from the group folder on `box.com`). The other group must implement the other functionality. Then try to send a test signal of your choice from your group's computer and record it at the computer of the other group. You will experience the difficulty with synchronizing playback and recording; this is also a key problem in real-world communication systems as you will see later this week!

Activity 21: Play audio signals and record

Send test signal from one computer to another computer as described above. If you send a speech signal, then try to see whether the recorded signal is still understandable. Switch the roles, i.e., if your group implemented the transmitter, implement the receiver and ask the other group to implement the receiver. Then, try to figure out the maximum distance for which you can still understand the voice on the recording. What happens if you double the distance between loudspeaker and microphone? By how much is the received signal's loudness reduced? is it reduced by half or more? ***Talk to us if you think you know the answer!***

7.5 Simultaneous Play and Record

So far, you are able to record audio signals and play them back. However, we have not explained how to *simultaneously* record while playing something back through the loudspeakers. When building our communication system, we will be playing back a signal that contains digital information through our loudspeakers (the transmitter) and record it at the same time through the microphone (the receiver). We will then extract the digital information from the received signal. This is extremely useful to test and optimize a communication system, without transmitting signals from one computer to another (as in the previou activity). As it turns out, playing back while recording is a bit more complicated than doing both things separately—the key trick to accomplish this is to write a MATLAB script.

Generate a new MATLAB script called `test_play_and_record.m`. First, add three lines to your script that set the sampling rate, as well as the input and output device IDs:

```
FS = 44100;
InpID = 1;
OutID = 4;
```

Be careful: your IDs may be different than the ones we showed above. Now, add another line to define the signal we want to play back. As an example, use the chirp signal that sweeps a sine wave from 100 Hz to 10,000 Hz. Add the following line to your script :

```
filename = 'examples/chirp-100Hz-to-10000Hz.wav';
```

Now, add the following line that loads the chirp waveform

```
[y,FS] = load_audio(filename);
```

If you want, you can now add lines that plot the signal `y` in the time domain and the spectrogram. For example, add the following lines to your MATLAB script:

```
figure(1)
plot_signal(y,FS);
figure(2)
plot_spectrogram(y,FS);
```

Now comes the interesting part. Our goal is to record a signal using the microphone while playing back our chirp through the loudspeakers. The general approach is as follows: We start recording, we then wait two seconds, we play our audio signal, we wait for one seconds, and finally we stop recording. Here are the commands one needs to add to the MATLAB script to implement this strategy:

```
% start recording
recObj = audiorecorder(FS,16,1,InpID);
record(recObj);
% wait two seconds
pause(2)
% play audio signal
playObj = audioplayer(y,FS,16,OutID);
playblocking(playObj);
% wait one second
pause(1)
```

```
% stop recording
stop(recObj);
y_rec = getaudiodata(recObj);
```

Here, we used the comment function in MATLAB `%` to explain the individual steps. You should use the comment function in your scripts whenever possible; it is always a good idea to explain what you are doing because one quickly forgets the details. In the above example, note that the amount of pause before starting to record may vary from device to device. Hence, you may need to modify the pause value to ensure that you record the entire transmitted audio signal.

Finally, add a few lines to plot the recorded signal:

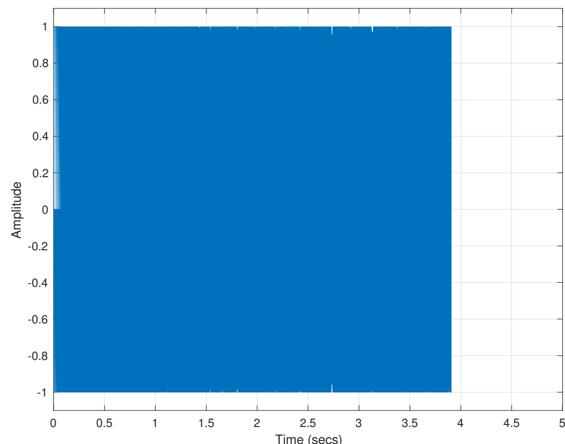
```
figure(11)
plot_signal(y_rec,FS)
figure(22)
plot_spectrogram(y_rec,FS)
```

Remember that `y_rec` is the variable (vector) containing the recorded signal. In case you would like to learn more about the details of the above method to play and record, feel free to ask us and we can explain the functionality of the individual commands. Note that the functions we provided to play and record signals were hiding all these details from you with the goal of simplifying the project. Finally, it would be great to play back the recording. To this end, add the following final line to your MATLAB script

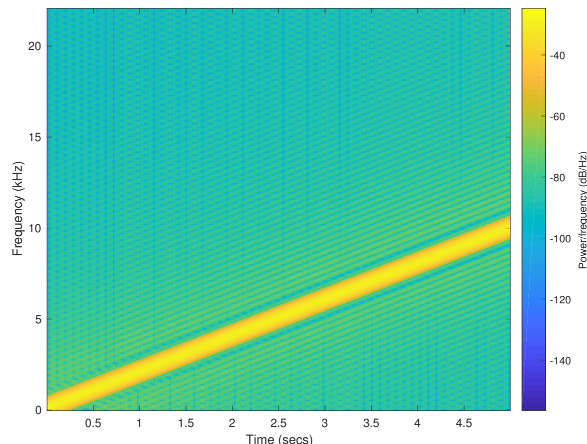
```
play_audio(y_rec,FS,OutID);
```

If you now press the "Run" icon on top of the desktop, the script should run without errors and record the chirp that is played back through the loudspeakers.

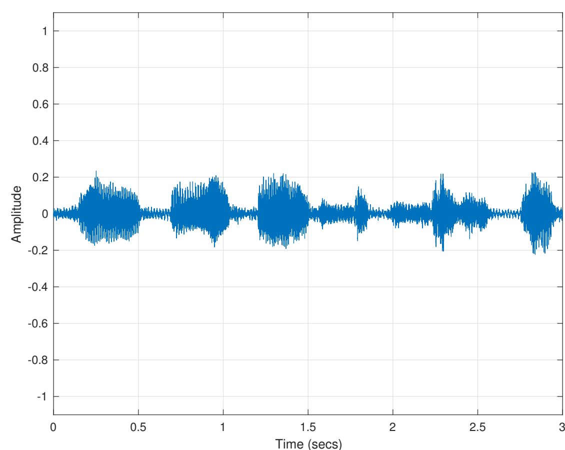
If everything worked out fine, you should see four plots that look similar to the ones in Figure 16. As you can see, the transmit waveform (the top two figures) in the time-domain and spectrogram look as expected: a clean chirp that sweeps from 100 Hz to 10,000 Hz over 5 seconds. The received waveform (bottom two figures) looks different. First, the received time-domain signal no longer has a constant amplitude. This is caused by the fact that loudspeakers and microphones are not perfect in reproducing the digitally sampled signal. In fact, loudspeakers and microphones amplify and attenuate certain frequencies, which is basically what you see here. Furthermore, the transmitted chirp is propagated through the air (a wireless channel). Wireless channels also attenuate certain frequencies more than others. Hence, what you see in the amplitude of the received time-domain waveform is the overall *frequency response* of loudspeaker, channel, and microphone. If all three components (loudspeakers, channel, and microphone) were perfect (i.e., they all would not affect the signal in any way), then the amplitude of the received signal would be constant for all frequencies in the sweep. We emphasize that this property is not only valid for acoustic communication channels but for general wireless communication, including transmission using electromagnetic waves. Combatting such non-ideal effects of the transmitter, channel, and receiver is what makes it difficult to reliably communicate over wireless channels. Finally, if you look at the spectrogram (bottom right figure), you see two important aspects: First, the waveform does not start at exactly $t = 0$ but a bit late (also at the end is some silent period). Second, there are suddenly some new frequencies visible that were not present in the transmit signal. Note that this is an issue of the microphone, which does not show particularly good quality. Well-engineered microphones (or receivers in general) should not introduce this many new frequencies (only amplify or attenuate existing frequencies). Hence, if we want to use our loudspeakers and microphones, we have to be careful to deal with all these artifacts.



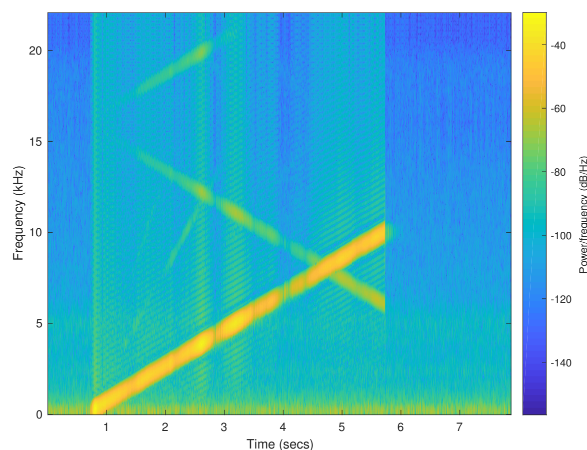
(a) Time-domain of transmitted signal.



(b) Spectrogram of transmitted signal.



(c) Time-domain of received signal.



(d) Spectrogram of received signal.

Figure 16: Transmit and receive signal visualized in the time-domain and as spectrogram.

Activity 22: Play and record the chirp that goes from 100 Hz to 22050 Hz

Modify your MATLAB script to play and record the chirp that sweeps frequencies from 100 Hz to 22050 Hz. Note that the `examples` folder already contains such a wav-file so you only need to change a single line in your MATLAB script. What do you observe for the new frequencies that we have not transmitted before? Repeat the same experiment by increasing the distance between the loudspeaker and microphone. Discuss your observations with us!