Digital Signal Processing – Lab 2

Adaptive Noise Cancellation

Automatic Control, ISY, Linköping University

Version: 2020-12-01

1 Introduction

This lab is about the design of an adaptive filter for active noise cancellation. This is a challenging task because the utility signal should be affected as little as possible while the noise should be suppressed as much as possible. For example, think of announcements in an airplane, where the sound from the engine should be suppressed but the voice of the staff should be unaffected.

In the lab, music is corrupted by a noise signal, e.g., a sum of sine signals or (bandlimited) white noise. The goal is to suppress the noise so that the music appears more clearly. One microphone is recording the mixed sound and one microphone is recording the noise signal. The signal processing, where the noise is suppressed, is done on an Arduino Due micro-controller board in real time and you can listen to the result in a pair of headphones.

The lab is designed for groups of two people during four hours in Laboteket. The schedule is tight and you must solve a number of preparation exercises (See section 6) to be able to finish on time. The lab is completed by an oral examination where the questions in this compendium are reviewed. All the files you need for the lab can be downloaded from the course home page. Download the files and place them at a suitable location (directory) on your account. Then start Matlab and go to that directory.

2 Lab setup

For the lab, you need one pair of speakers, a pair of headphones, two microphones (with built in pre-amplifier), and an Arduino Duo board with the DPS-shield. You will also need a computer with the Arduino IDE installed (including the Arduino Due support package) and Matlab.

Set up the speakers and microphones as shown in Figure 1. The noise microphone (Connected to input MIC B.) should be a few centimeters from the noise source to minimize its exposure to the music. The channel microphone (Connected to input MIC A.) should be placed about a half meter from both sound sources. The loudspeakers should be connected to the sound card of the computer and the microphones should be connected to the input of the DSP-shield; see Figure 2. The Arduino board should be connected via the USB-cable to the computer, and the headphones should be connected to output of the DSP shield. To identify which speaker is the noise and music source, respectively, you may run the function `playsound('white')`.

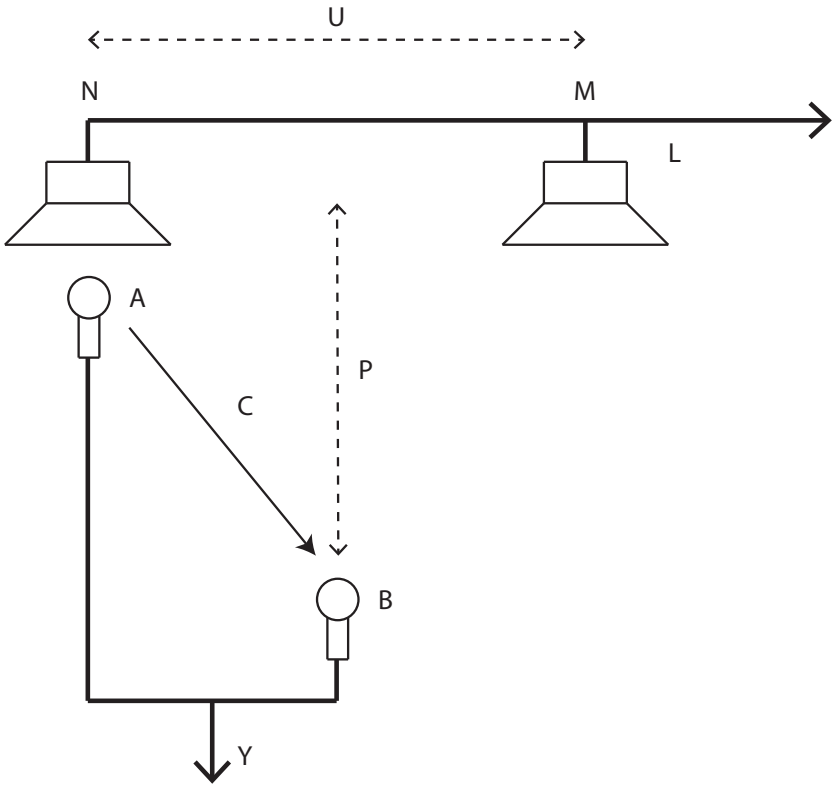


Figure 1: Microphone and loudspeaker arrangement.

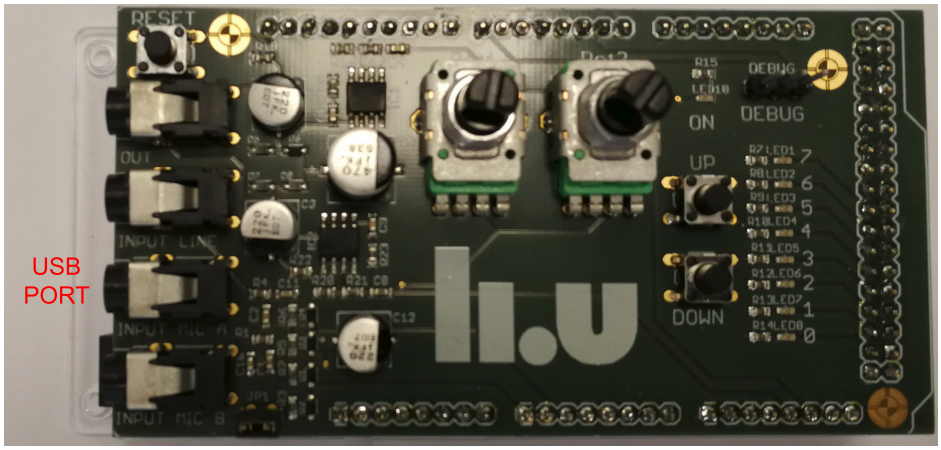


Figure 2: The DPS-shield on top of the Arduino Due board. The USB port to be used for programming and communication with the board is indicated by the read text.

3 Channel model

To design a adaptive noise cancelation system you need to develop a channel model that describes the relationship between the noises recorded by microphones B and A. The channel is the air between the source and the microphone, but also the microphone itself. It is not enough to model the channel as a pure time delay, since the sound also is reflected in surrounding objects. Thus, the signal recorded by the microphone is the sum of multiple time delayed and damped sound reflections. The reflected sound is damped and phase shifted in different ways dependent upon its frequency components. A common and physically reasonable model for describing these effects is

$$y(t) = H(q,\theta)u(t) = b_0u(t-n_k)+b_1u(t-1-n_k)+\ldots+b_{n_b}u(t-n_b-n_k). \quad (1)$$

That is, the channel is modelled as a FIR-filter of order n_b and with a time delay n_k. The time delay is caused by the finite speed of sound in air, but also delays in the electronics and the software of the processor. A first estimate of the time delay can be calculated by measuring the distance from the microphone to the speaker. More refined estimates, that also take into account the delay of the electronics and the software, can be found using for example the following two methods.

1. Assume that the delay is zero and estimate an FIR filter of high order. This will result in the first n_k coefficients of the estimated filter being close to zero.

2. Correlating the input and output signals.

You will in the lab investigate both of these methods.

4 Part I: Channel modeling

In this part of the lab you will model the channel using the FIR filter model in (1) and try to identify the model parameters using different methods and types of noises. Hence, the noise source is the only sound source used in this experiment: You will use the Arduino board to record the noise at the "input" and "output" of the channel. The estimation of the model parameters will then be done in Matlab. To get started do the following.

1. Make sure the equipment is setup as in Figure 1 and that the Arduino board is connected to the computer.

2. Open the Arduino IDE and load the project `DSPLab_part1.ino`.

3. Under the "Tools" menu select "Arduino Due (Programming Port)" as the target platform.

4. Press the upload button. This will compile and load the program onto the Arduino processor.

5. Under the tools menu, check which communication port, e.g., `'COM11'`, the Arduino is using for communicating with the computer; this port must be specified when calling the Matlab function in the next step.

6. Start Matlab and run the function `play_and_rec_noise` with the noise type set to chirp. The function will play a few seconds of noise, and tell the Arduino board to record 2 seconds ($F_s = 8$ kHz) of the sound at the microphones and then send it up to Matlab. The function will also plot the recorded signals.

7. Adjust the volume of the sound so that signal from microphone B occupies most of the dynamic range of the AD-converter of the Arduino board without saturating it.

8. Record and save data for the four types of noises that are available, that is `'sine'`, `'white'`,`'multisine'`, and `'chirp'`.

Note: It may happen that Matlab cannot find the serial port you specify. Then you need to restart Matlab. Therefore, it is important that you store your data and scripts frequently.

Next, use your `MyLMS.m` function (See the preparation exercises in Section 6.) together with the recorded noises to estimate the parameters $\theta \triangleq \{b_i\}_{i=0}^{n_b}$ for the FIR-model. If reasonable channel model has been obtained, i.e., the filter parameters $\hat{\theta}$ has been correctly estimated, it is possible to eliminate most of the noise by computing $\hat{s}(t) = y_{noise}(t) - \hat{y}_{noise}(t)$, where $\hat{y}_{noise}(t) = H(q, \hat{\theta})u(t)$.

Questions

1. Begin with the determination of n_k. Test different approaches to estimate n_k based on the preparation exercises. What are the problems with the different approaches?

2. What is the best model order n_b? Motivate in a number of different ways.

3. How much is the energy in $\hat{s}(t)$ decreased compared to $y_{noise}(t)$? Use a suitable energy ratio and examine all the different noise types.

4. Compare the estimated model from different noise signals. Are the results the same?

5. Which noise signal is the easiest to suppress? Explain why.

Hints

- Write an m-file that is stored in your account, so it is straightforward to repeat the computations.
- As always, before using a recorded signal, check that it looks "nice".

5 Part II: Adaptive noise cancellation using a LMS-algorithm

In this part of the lab you will do adaptive noise cancellation (in real-time) using an leaky LMS algorithm implemented on the Arduino board. The underlying idea is to recursively calculate the parameters of the channel model, and then use the channel model to the estimated the noise signal $y_{noise}(t)$ at microphone Y from the at microphone U measured noise $u(t)$. The estimated noise signal $\hat{y}_{noise}(t)$ can then be subtracted from the observed signal $y_{tot}(t) = y_{music}(t)+y_{noise}(t)$, which contains both the desired music signal and the noise. The remaining "error" signal $\hat{s}(t) \triangleq \hat{y}_{music}(t) = y_{tot}(t) - \hat{y}_{noise}(t)$ then appears less corrupted by the noise; ideally, it should only include the music. To get started do the following.

1. Open the Arduino IDE and load the project `DSPLab_part2.uno`.

2. In the Arduino file change the values of the defines `NR_OF_FILTER_TAPS` and `DELAY` so that they reflect the values you identified in Part I of the lab.

3. Press the upload button. This will compile and load the program onto the Arduino processor.

4. In Matlab, run the function `playsound('noise_type')`. The function will repeatedly play a piece of music and noise, until terminated by pressing "ctrl + c".

Table 1: LMS filter settings controllable via the Arduino shield.

Parameter	**Range**	**Control**	**Display**
Step length	$2^{-7} - 2^{-3}$	Pot. 1	Led 3 to 7
Leakage factor	$0 - 0.03$	Pot. 2	–
Output signal	–	Up/Down buttons	Led 0 $\rightarrow y_{tot}(t)$ Led 1 $\rightarrow \hat{s}(t)$ Led 2 $\rightarrow \hat{y}_{noise}(t)$

5. Plug the headphones into the output of the shield and listen to the results of the noise cancellation.

6. Test with different types of noises and filter settings. The (via the shield) adjustable filter settings are listed in Table 1.

Questions

1. How does different step lengths and leakage factors affect the result? Can you hear the difference?

2. Assume that we know that one of the noise signals in Section 3 is used, but we do not know which one. How should the parameters n_b and n_k be selected, based on the knowledge from the first experiment?

3. Which noise signal is easiest to cancel out? Is this the same signal that was easiest to handle in the first experiment? Motivate your findings.

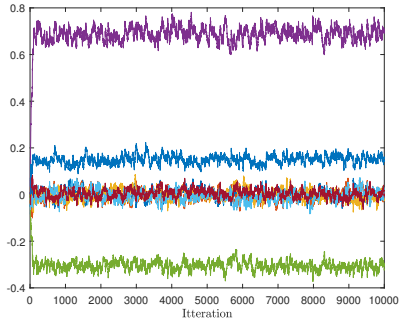
6 Preparation exercises

The following exercises should be solved/prepaired before the lab. If you have not done the preparation exercises you will not be allowed to participate in the lab secession, and you will have to do the lab the next time the course is given.

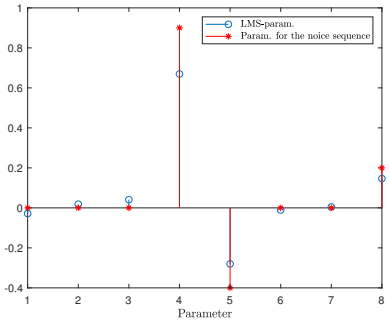
Tasks

With the experimental set-up of this lab in mind, do the following tasks.

1. Assume that the impulse response of channel is given by $h(t) = 0.9\delta(t-3) - 0.4\delta(t-4) + 0.2\delta(t-7)$. Generate a 10^4 sample long discrete time white noise sequence and filter it through the channel.



(a) Convergence of the parameters as the number of samples increases.



(b) Parameters at the last iteration and parameters that generates the noise sequences.

Figure 3: The parameters of the Leaky LMS algorithm with $n_k = 0$.

2. Do exercise 9.6 in the exercise book where you will implement your own recursive algorithms.

3. Download the Matlab function skeleton file `MyLMS.m` from the course home page and implement an leaky LMS algorithm.

4. Verify that your LMS algorithm works properly by using it to estimate your channel parameters by applying it to your generated noise sequences. Plot the filter coefficients as a function of the number of samples the LMS algorithm has processed. If you filter is working correctly you should get plots as those in Figure 3.

5. Save your LMS algorithm at a location where you can access it during the lab.

Questions

With the experimental set-up of this lab in mind, answer the following questions.

1. Is it possible to estimate the time delay by

   - using geometry and distance measurements?
   - visual examination of the signals by using Matlabs plot functions?
   - computing `[R,lags]=xcorr(y,u,M)`?

- estimating the channel parameter using a large model order n_b and a small delay n_k and the inspecting the filter coefficients?

If the answer is "yes", explain how this can be done and if there are some drawbacks with the method.

2. How does the step length affect the LMS algorithm? Does LMS work for an arbitrary step length?

3. Which measures can be used to evaluate the performance of an estimated model?

4. Draw a block diagram that explain the concept of adaptive noise cancellation as used in this lab. What is estimated by LMS?