Adaptive Noise Cancellation

James Fosburgh, Scott Bradley and Claire Wenner

University of Rochester
AME272: Audio Signal Processing

Abstract

In this project, we researched adaptive filters and their applications to noise cancellation. We explored different adaptive filtering methods and implemented our own versions of these filters in MATLAB. We also created a live simulation of our adaptive filtering system in Simulink.

1. Adaptive Filters

An adaptive filter is a system with a filter whose transfer function is controlled by a set of weights that change over time. An adaptive filter takes in two inputs, x[n] and d[n], and the weights of the filter are changed so that the difference between the two signals, the error e[n], is minimized. The output of the filter y[n], represents the filtered version of x[n] that is similar to d[n], and the error is found by simply subtracting y[n] from d[n]. The resulting weights represent the filter applied to the signal x[n], before being added to a source signal to create d[n]. Figure 1 shows the block diagram for adaptive filters.

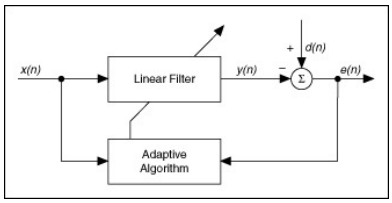

Figure 1: Adaptive Filter Block Diagram [2]

2. MATLAB Implementation

For this project, we explored two common adaptive filters: the Least Mean Squares (LMS) filter, and the Recursive Least Squares (RLS) filter. We implemented both filter algorithms in MATLAB, as described below.

2.1 LMS

The simplest active noise cancellation filter is the Least Mean Square filter. This algorithm works by finding a filter represented by coefficients that creates the least mean square when applied to the noise of the system and subtracted from the total input. This result is known as the error signal. For example, in the case of noise cancelling headphones, the total input of the system is taken from a mic on the inside, and the noise of the system is taken from a mic on the outside. These are then passed frame by frame through the algorithm, which generates a filter that represents the filter created by the headphones, and applies it to the noise taken from the outside mic. When this is subtracted from the total noise on the inside, the resulting error signal should be whatever is being actively played by the headphones.

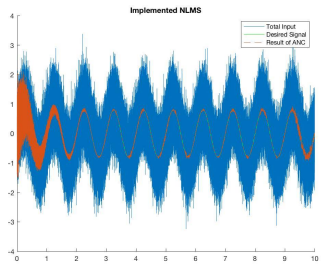

Figure 2: NLMS

One problem with LMS is that it does not take into account the power of the noise input. This can be solved by adding a normalization to the updating of the weights. We have implemented our own Normalized Least Mean Square (NLMS) algorithm in Matlab. Figure 2 shows the result of passing a 1Hz sine wave with randomly generated noise through our implemented NLMS and Matlabs built-in NLMS function.

The two main factors that go into the NLMS implementation are the learning rate and the filter coefficient size. As can be seen in Figure 3, if the learning rate is too low the algorithm will not be able to filter out the noise effectively. However, if the learning rate is too high, the algorithm over corrects. When using a filter vector size of 128, we found that a learning rate of around .005 worked best. As can be

seen in the second image below, the filter vector size has an impact on both how fast the algorithm learns and how precise it is. A smaller vector will allow the filter to act more quickly, but will not filter as well as a larger vector which will take more time to converge.

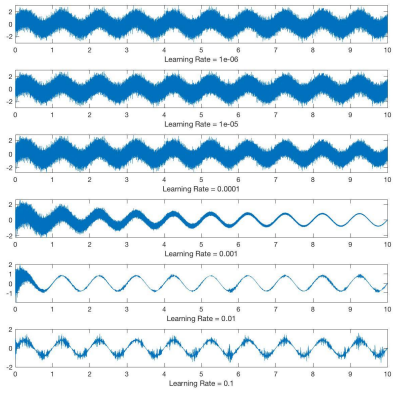

Figure 3: NLMS Learing Rate

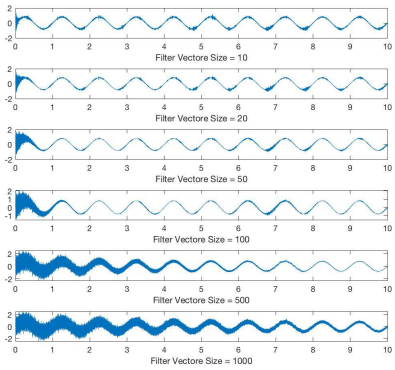

Figure 4: NLMS Filter Vector Size

2.2 RLS

An alternative algorithm to LMS is the Recursive Least Squares (RLS) algorithm. The RLS algorithm is overall fairly similar to LMS, but updates the weight coefficients recursively. The main computational difference is that where the LMS algorithm uses the noise buffer and the current error to update its weights, RLS uses a recursively processed version of the noise buffer **g** combined with another matrix **P** that begins as the identity matrix with an order equal to the order of the RLS filter. **P** is in turn updated with by the noise buffer and **g**. In these calculations are where the learning rate comes into effect. The learning rate determines by how much of **P** and **g** can be updated in each pass of the algorithm.

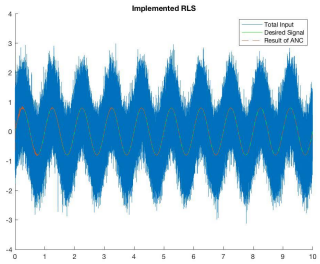

Figure 5: RLS

Overall, the RLS algorithm converges faster and is more precise than LMS. With LMS, there was a tradeoff between getting the filter to converge quickly and having it remain stable once it had converged. In comparing the picture below to that of the LMS algorithm learning under the same conditions, it is clear that the RLS algorithm works both more quickly and more effectively than the LMS algorithm.

3. Simulink Implementation

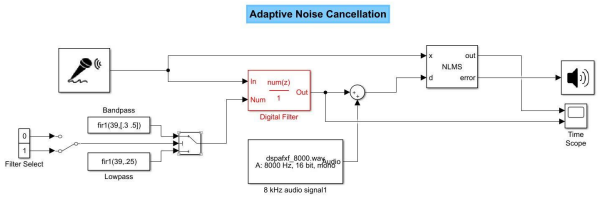

Figure 6: Simulink Block Diagram

For our live implementation, we decided to use MATLAB Simulink. Simulink provides us the tools to incorporate our implemented adaptive filter algorithms, and use them in a real-time system. Our model stimulates an Adaptive Noise Cancelling system. An example use of our system is for a hands-free car phone call. In this example, there would be one microphone recording the exterior noise, x[n], and one microphone in the car recording the driver talking. The microphone on the inside of the car will record a record a noisy signal, and it will be a combination of the driver talking and a filtered version of the exterior noise. In this situation an ANC system is desirable to keep a low noise level in the recorded audio. Figure 6 shows our implemented Simulink system.

In our system, we take a microphone input to record the noise, and we simulate the filter and the corrupted signal from that. The output of the system is the error of our implemented NLMS adaptive filter, and should be the noise canceled input signal. Although we have this source signal

being a pre-recorded audio file for demonstration purposes, this input can be replaced with another microphone representing a corrupted signal.

4. Results

Our implemented NLMS adaptive filter algorithm was able to significantly reduce the noise found in the corrupted signal. We were able to change the filter applied to the noise representing different acoustic environments, and yield the same noise reduced source signal. Figure 7 shows our resulting simulink system. We used an audio file of a drum loop to represent the source signal. The first graph shows the corrupted signal represented as filtered noise combined with a source signal, and the second shows the NLMS error, which should be the isolated source signal. (**?**)

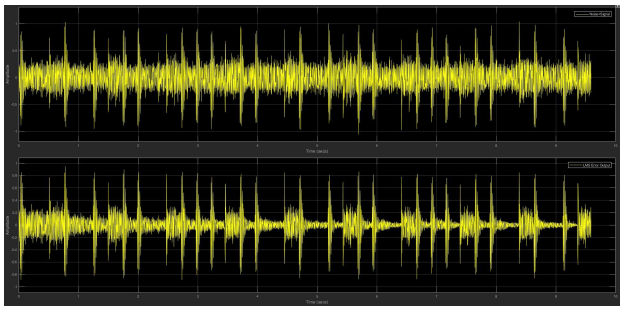

Figure 7: Simulink Live Output

We observed that the NLMS algorithm had a delay period before it fully took effect. In this example that period was about 1 second. This is related to the filter weights needing more input data before they fully represent the noise filter. After that period, you can see the noise from the signal almost completely eliminated, and the shape of the source signal starts to become more apparent.

5. Future Work

Further work for this project includes developing a live Simulink system using the RLS filter rather than the NLMS filter. Live implementation onto a DSP board is another further development.

References

[1] A. Singh. *Adaptive Noise Cancellation.* (2001). Avaliable: http://www.cs.cmu.edu/ aarti/pubs/ANC.pdf.

[2] *Least Mean Square (LMS) Adaptive Filter - National Instruments*. Ni.com, 2013. [Online]. Available: http://www.ni.com/example/31220/en/.

[3] Mathworks.com. (2018). *Overview of Adaptive Filters and Applications.* [Online]. Available: https://www.mathworks.com/help/dsp/ug/overview-of-adaptive-filters-and-applications.html.

[4] S.C. Douglas. *Digital Signal Processing Handbook.* Ed. Vijay K. Madisetti and Douglas B. Williams. Boca Raton: CRC Press LLC, 1999.