Signal & System Project
Adaptive Noise Canceler

Yuyang Rong
School of Information Science and Technology
Shanghaitech University
Student ID: 69850764
Email: rongyy@shanghaitech.edu.cn

Jianxiong Cai
School of Information Science and Technology
Shanghaitech University
Student ID: 67771603
Email: caijx@shanghaitech.edu.cn

Anqi Pang
School of Information Science and Technology
Shanghaitech University
Student ID: 956814403
Email: pangaq@shanghaitech.edu.cn

1. Introduction

As a course project of *Signal and System*, we worked adaptive noise controller.

The basic assumption is, when sample rete is large enough, we can assume that the samples form a line. Based on this assumption, we can predict the next sample, in this way we can build an adaptive noise controller.

After reading serval paper we split our three to work on proof of FXLMS, Simulink approach and MATLAB approach. At the end we realized that, although theoretically it worked since we proved it, we can't make it online due to IO latency of PC. However, we managed to finished a offline prototype in both Simulink and MATLAB.

2. Proof of FXLMS

It is clear in Gan and others paper [1], [2] how FXLMS should be implemented and used in ANC, but they didn't provide a clear proof of why should ω_l be updated in the following way:

$$\omega(n+1) = \omega(n) + \frac{\mu}{\widehat{P}_x(n) + c} e^T x$$

Here we will give a simple proof, or an explanation for this update. Now that we recorded certain samples of errors, we can calculate it's cost function value:

$$C = \frac{1}{2} e^T e$$

Here, let's assume that e is a vector with history records. However, notice that:

$$e(n) = d(n) - \omega_l(n)^T x$$

So to minimize C, we need to take derivatives of ω_l, which gives:

$$\nabla_{\omega_l} C = \frac{1}{2} 2 e^T \nabla \omega_l e = e^T(-x) = -e^T x$$

Now we know that the gradient, all we need to do is update ω:

$$\omega(n+1) = \omega(n) - \nabla_{\omega_l} C = \omega(n) + e^T x$$

This result is close to that shown above, but unfortunately not enough. From experiment we noticed that ω doesn't converge, it goes to infinity as time goes by.
After trials and errors we decided that LMS is not enough for this task since ω is not converging. This may happen when the variance is large [3], which would be the common case for white noise.

The solution is to change learning rate(μ) according to x we get. We took Dhiman's advice and changed the update function to:

$$\omega(n+1) = \omega(n) + \frac{\mu}{x^T x} e^T x$$

The result is, that μ goes to infinity even faster. After a few toughs we realized that at the very beginning, x can be really sparse and close to zero. As a result, $x^T x$ can be really small(smaller than 1), thus leads to a greater μ, which is not desirable. The solution to that, is to add a constant after $x^T x$, then we have:

$$\omega(n+1) = \omega(n) + \frac{\mu}{x^T x + C} e^T x$$

The constant we choose is $C == 1$

Another thing to consider is, should all history records be considered equal when they act as a Still, this is not the same as learning step controller. Actually, not. We need the closes history record to be more important than others. So we do a scale according to history. That's how the the formula defined above came around. If we take a look back how $\widehat{P}_x(n)$ is defined:

$$\widehat{P}_x(n) = \alpha x(n)^2 + (1-\alpha)\widehat{P}_x(n-1)$$

$$\alpha \approx 1$$

Then it's clear that in $\widehat{P}_x(n)$ the closest history is the most important.

Now we have explained step by step, from LMS to NLMS to FXLMS, that the update formula of ω_l is reasonable and valid. Next we will implement them in two approaches: Simulink approach and MATLAB approach.

3. Implementation

Implementation details goes here.

3.1. Simulink approach

The Simulink approach is shown in Figure 1.Note that this is an offline simulation, so we use noise generator instead of one microphone. mixture of noise and music instead of another microphone.

We use a random generator to simulate the white noise. Then there will be a LPF to filter the high frequency noise. The Digital Filter1 and Lowpass1 in the Figure 1 do this thing.

And for the LMS Fliter part , there are two input , one is the noise, another is the the mixture of noise and music. As proofed above, the LMS Fliter can then get the Error, easy to see the Error should be close to the music.

The Audio Device is the Error, which should be music without noise. And in the scope we can see the waveform of noise , music , mixture , and Error.

But there are still some bugs, for example , the simulink do not support high frequency , and there will be more noise when the input "music" is speech rather than music.

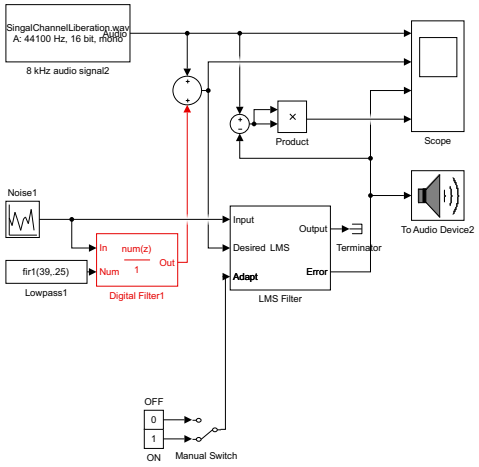

Figure 1. Simulink ANC

3.2. MATLAB approach

The MATLAB approach support two channel audio. The implementation is similar to Simulink approach.

For simulation, we use a two channel audio file and a pre-generated two channel noise file to simulate the real-time environment. The MATLAB code use dsp.AudioFileReader to read in signal and noise files. Here, we simulate one microphone to get the mixture of signal and noise from the environment.

In the system, first we subtract the noise from the mixture as noise = mixture - signal. Then the FXLMS algorithm uses noise and music to get the output like what has been mentioned above. In the code, the dsp.LMSfilter is used to perform the algorithm. Also, the low-pass filter used in simulink approach is also used here. As for the two-channel part, the input is separated into two ANC systems, and the output is consist of outputs from both ANC systems.

4. I/O latency

For a real-time system, I/O latency has a huge impact on the performance of the system.

MATLAB audio latency test program audioLatencyMeasurementExampleApp.m is used to test the audio latency. The test is run on window 10 with Intel i7-5500U processor, 2.4ghz, and results are shown in the table.

The test read in an audio file, play it with soundbar and record back using a microphone. Then it calculate the latency by calculate the delay between the original signal and the recorded signal.

As is shown in the table, the average latency on PC is 0.2358 seconds. As has mentioned above, the LMS filter

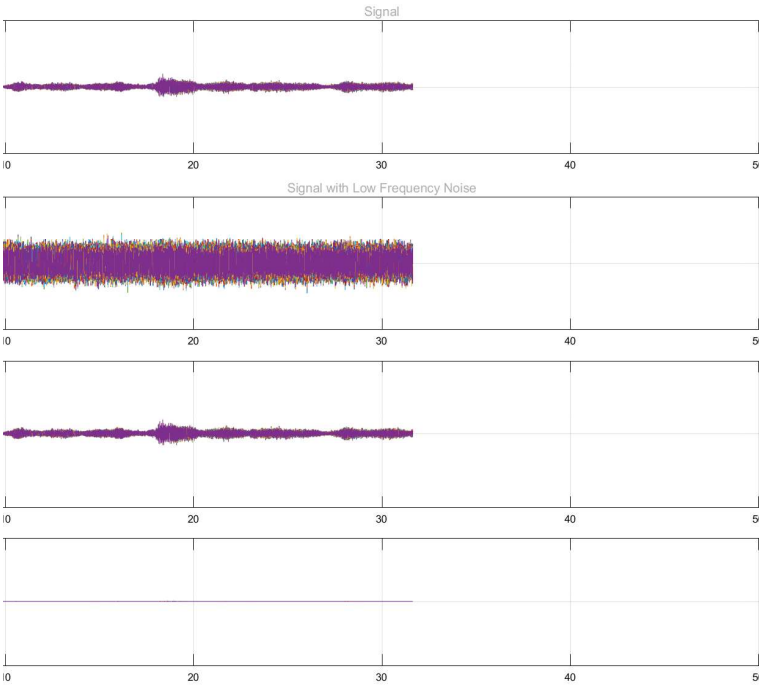

Figure 2. ANC Scope

TABLE 1. AUDIO I/O LATENCY

Test NO.	Latency(s)
1	0.2361
2	0.2364
3	0.2323
4	0.2380
5	0.2366

has an assumption that the training data and predicted data is in one period thus can be regarded as a line. However, with such an audio I/O latency, the predicted data will be several periods away from the training data. Thus, the LMS filter can not be applied on PC with such a latency.

5. Future Work

6. Conclusion

The conclusion goes here.

Acknowledgments

We would like to thank for Prof. Luo's great lectures and paper he recommended, which gave us many hints. We would also thank Teaching Assistances' help. Their effort and devotion helped us a lot.

References

[1] W. S. Gan, S. Mitra, and S. M. Kuo, "Adaptive feedback active noise control headset: implementation, evaluation and its extensions," *IEEE Transactions on Consumer Electronics*, vol. 51, no. 3, pp. 975–982, Aug 2005.

[2] Y. Song, Y. Gong, and S. M. Kuo, "A robust hybrid feedback active noise cancellation headset," *IEEE Transactions on Speech and Audio Processing*, vol. 13, no. 4, pp. 607–617, July 2005.

[3] J. Dhiman, S. Ahmad, and K. Gulia, "Comparison between adaptive filter algorithms (lms, nlms and rls)," *International Journal of Science, Engineering and Technology Research (IJSETR)*, vol. 2, no. 5, pp. 1100–1103, 2013.