Final Project Report

Jason Voccia

December 20, 2002

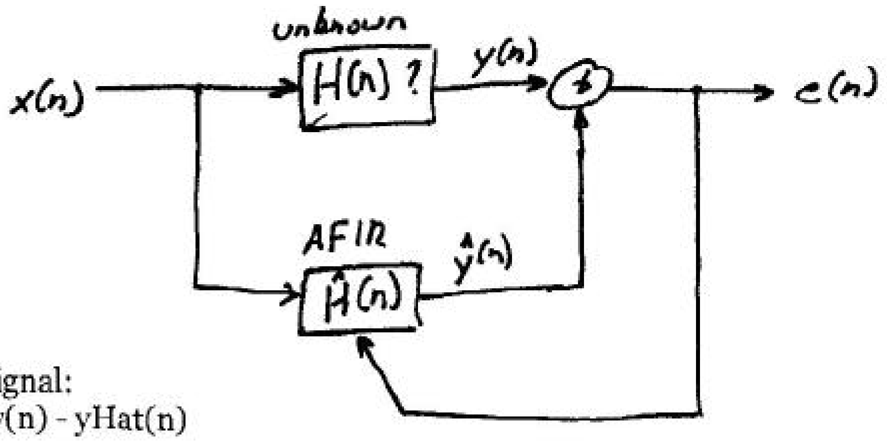
Physic 398EMI

Introduction

I originally set out to create an adaptive system to match the phase of sound signals in an arbitrary location in a room. Given a stereo signal, 2 speakers and a microphone, I wanted to create an adaptive system that adjusted the output of the 2 speakers such that the microphone heard an in phase signal. I set out to get a simulation working in Matlab, and then try and implement a real device using either C on a computer, or assembly on a Texas Instrument's DSP. As it turned out this problem was prohibitively hard to solve, and I ended up creating a much-crippled implementation in Matlab.

Background

Most of my code depends on a least mean squares (LMS) adaptive filter. The LMS algorithm tries to match a FIR filter to an unknown transfer function, by trying to minimize an error signal that is used to the Adaptive FIR (AFIR) filter.



Error signal:
e(n) = y(n) - yHat(n)

The AFIR filter coefficients are updated using the following relationship:
hNew(i) = hPrev(i) + μ/2*(-(d/dh(i))*|e|^2)

Where μ is the amount the coefficients are updated every step, and (-(d/dh(i))*|e|^2) is the squared error at the current coefficient. After each iteration, the filter incrementally updates itself by adjusting the coefficients so if the same data was filtered again, the there would be less error between the output of the real filter and the output of the adapted filter.

After a bit of math -(d/dh(i))*|e|^2 can be rewritten so it becomes 2*(-x(n-i))*e and the coefficient update relationship becomes:
hNew(i) = hPrev(i) + μ*e*x(n-i)

For a reasonable value μ of LMS filter will converge in most situations to a FIR filter that approximates the unknown filter. Generally, as the number of taps increases in the AFIR filter, the better the match will be to the unknown filter.

However, in some situations the LMS algorithm, will not converge, or even worse become unstable and diverge.

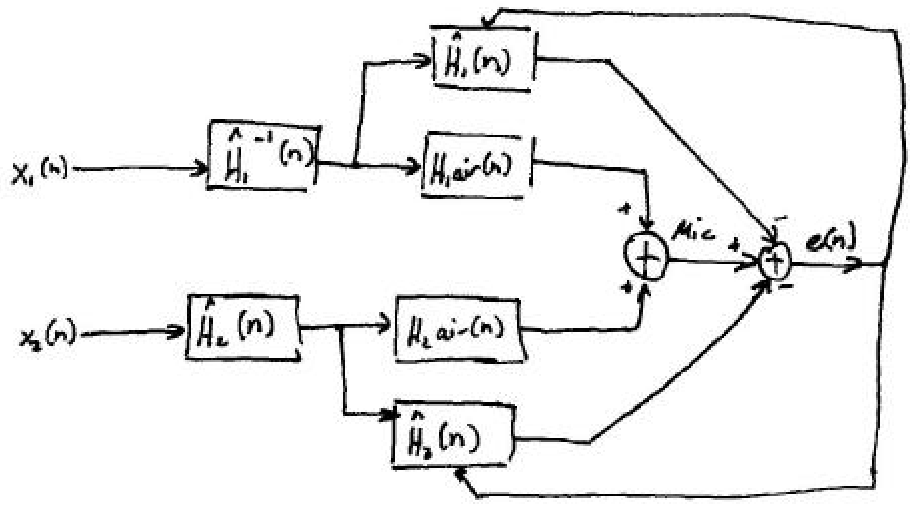
One situation where the LMS filter will not converge is when the value of μ is too large. Too large of a μ will result in a filter that almost always overcompensates when it updates filter coefficients. If this is occurring the filter will not converge, but actually jump around the properly converged adaptive filter. In an extreme situation, when the μ picked is well out of the range of reasonable values, the system will actually diverge. Another situation where the filter will not converge is when μ is too small. This will make the AFIR filter too slow to converge. In some cases, so it will move so slowly that if the unknown filter or input signal changes with time, the AFIR filter will not be able to follow the changes fast enough and the filter will never converge.

The Project

The first couple of weeks I worked on this project, I was planning at taking windowed FFTs of the stereo input signals of the system, and the FFT of the input to the microphone and then working with those to determine delay. After taking the FFT of the signals, I was planning at looking at the FFT of the microphone input and identifying key frequencies in the spectrum. I was then going try to find two input signals (one from each channel) that when added were the most correlated to the microphone signal. From there I would have an approximate delay, and I could then proceed to use that delay in the output of one channel to approximately match the other channel.

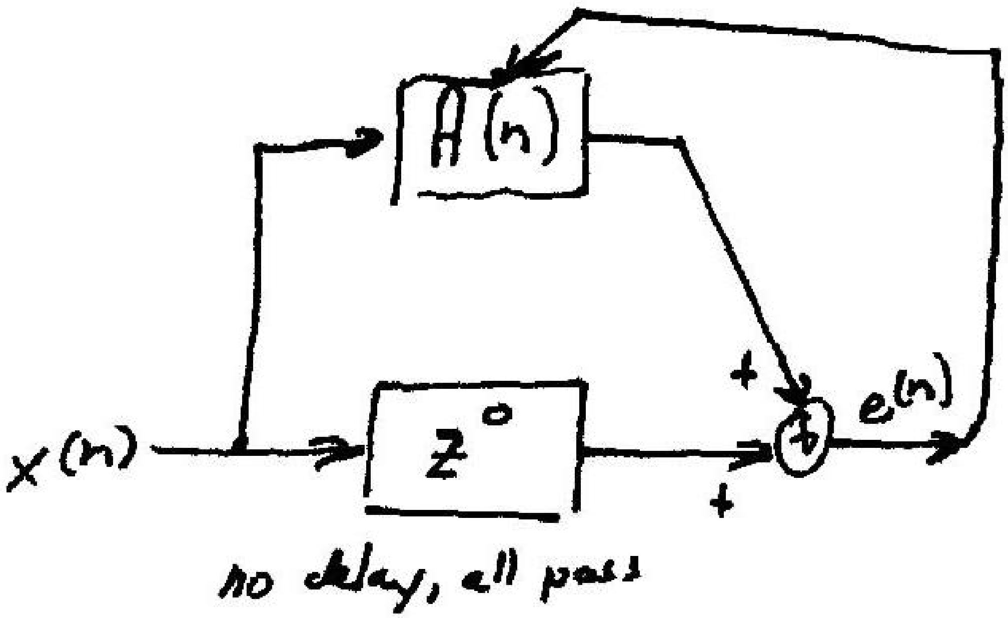
I quickly realized this was not going to work. Identifying FFTs that when added together would result in the microphone input becoming almost impossible to find. If I used short windows, there would most likely be many almost equally good matches, and it is very possible that I would find pick up the wrong match, and the delay calculated would be off. Using large windowed FFTs would also have problems. A large FFT window would prevent finding good matches if the delay was not a multiple of the FFT window. More importantly, the coarseness of the FFT would result in large changes in delay, and an inaccurate delay calculation. Also, trying to adjust the delay of one speaker, would involve implementing a time-stretching/compression algorithm to avoid a pop every time the delay changed. Time-based algorithms would operate similarly, and have the same type of problems.

After consulting with a professor, it was suggested I look into using a LMS algorithm for the system. This would allow for a delay resolution of one sample instead of the resolution of one window, and it would also alleviate problems associated with changing the delay because the AFIR filter would smoothly change the delay of the system.



Original Project Algorithm using LMS

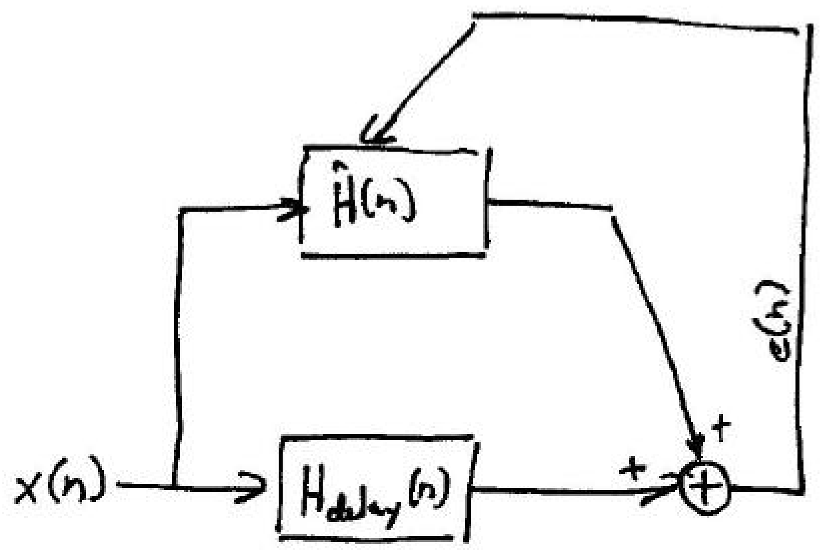
I first wrote Matlab code to test that I could get an LMS algorithm working in a regular manner. Once I was positive my code was working, I started to try to implement a more complex system. I wrote code to simulate a system that took a mono input and output it to one speaker, and took that same input and ran it through a AFIR filter that was hooked up to another speaker. The LMS algorithm then tried to minimize the microphone signal.



Block Diagram to Minimize Microphone Signal

This worked great for a sum of three sine waves, and worked decent for a clip of music. Both successfully converged and cancelled each other with good degrees of success.

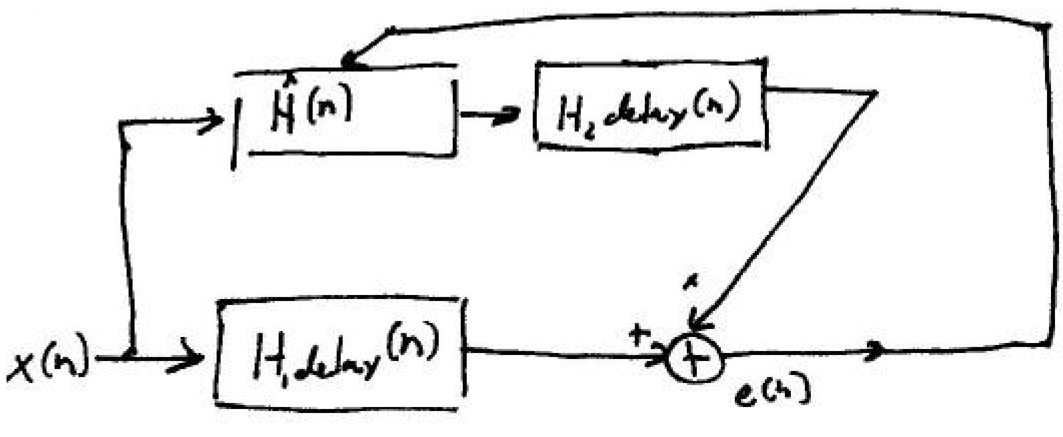
I then went on to try to get this same process to work when the output to one loudspeaker was delayed.



Delayed Loudspeaker Block Diagram to Minimize Microphone Signal

This system preformed well, also correctly minimizing the signal. As with the system with no delay, both the sum of sines and the clip of music both converged. However, convergence on both the sines and the music clip was slowed by approximately 20%.

Next I tried to delay both loudspeakers.



Two Delayed Loudspeaker Block Diagram to Minimize Microphone Signal

This is where problems started to develop. The system grows unstable because of the lack of immediate feedback in the LMS loop. The AFIR filter adapts to resonate the input signal, which in turn resonates the error. The cycle continues and the system diverges. Since changes to the system are not immediately reflected in the output the system is unable to train itself. Even with absurdly small values of μ the system still would eventually explode. I could not get the system to converge properly without knowing the delay. If I already knew the delay I was able to set up a system that works, but this defeated the purpose of the project.

I attempted to get a system that tries to match the input at the microphone to the input to the system, but this too failed when a large delay was introduced on the adaptive filter's output. The system would simply resonate and diverge. Again I could not get proper behavior for the system unless I already knew the delays.

At this point I gave up with the scheme to work the way I envisioned it. I changed the constraints on the project so I could measure the delay of the loudspeaker to the microphone by using a specific signal. This would be done once and then the system would output sound so that it all arrived in phase to where the microphone was when the delays were measured.

I set up a system that outputs a hanning windowed sinc periodically. It outputs the sinc no more than once per filter length. I run the sinc through a FIR filter that simulates both delays and reverberations in a simple room. I also run it through gaussian noise to simulate ambient noise in the area. I then feed the original sinc into the AFIR filter and use yhat-y as the error signal to update the filter. After 20 sincs have been output, I identify the first significant, peak in the AFIR filter and assume that corresponds to the direct sound path between the microphone and the loudspeaker. I clear out the whole AFIR filter and then put a value of .4 in the AFIR at the peak that I found previously. I repeat the same sinc sound again, this time spacing them slightly farther apart. After finding the peak, I compare the two values. If they are identical, or almost identical, I figure that is the delay, and use that. If they are different I repeat the steps until duplicate peaks are found.

Conclusion

This project was much more difficult and time consuming than I had originally anticipated. I was unable to get my original project to work, but I am not convinced that it cannot be done. It seems the original project may work if the two loudspeaker outputs are decorrelated, however I have not tested this yet to be sure. If this is the case, it may be possible to use two microphones to try and separate the outputs of the two loudspeakers and use the separated signals update the AFIR filters. It also may be possible to try and partially decorrelate the two loudspeaker outputs without it being audibly noticeable. If this could be done, the correlated parts could be filtered out, then the decorrelated parts could probably be used to update the filter without it diverging.

The changed project works pretty well as it is now in Matlab. I plan on working on it more over winter break and getting a five-speaker implementation working on a TI-54x DSP.