Tufts University

School of Engineering
Department of Electrical and Computer Engineering

EE 98 - Senior Design
Spring 2014

Active Noise Cancelling iPhone Application
Final Report

Michael Abboud
Francesco Pittaluga
Jesse Zhang

Abstract

For our senior design project, we attempted to design an iPhone application, which using only the standard Apple headset, suppresses the low-frequency components of ambient audio noise. Tackling this problem was a three-step process: First, we identified the hardware limitations of our system. We then used MATLAB to design, prototype, and simulate a noise-cancelling algorithm that adhered to these limitations. Lastly, we built an iPhone application according to our design.

Introduction

We live in a world of signals and noise. Low-frequency ambient noise from sources such as airplane engines, fans, and ac units can be a hinderance to everyday life. For our senior design project, we attempted to design an iPhone application, which using only the standard Apple headset, suppresses the low-frequency components of ambient audio noise. Our team consisted of two senior electrical engineering majors and one senior electrical engineering/computer science major. Combined, we had basic experience with digital signal processing, probability, linear algebra, control theory, and C programming. New concepts we had to learn included autoregressive models, Kalman filtering, Objective C, the iOS programming environment, and the iPhone 5 hardware.

Theory

The success of this project relied on three critical assumptions:

1. The ambient noise is significantly correlated with itself (i.e. given a set of samples, we can accurately predict the next sample).
2. The ambient noise that reaches the ears is approximately the same as the ambient noise that reaches the headset microphone.
3. The ambient noise is wide-sense stationary, meaning that
    a. Its mean does not change with time.
    b. The correlation between samples A and B is the same as the correlation between samples C and D given the time between A and B was the same as the time between C and D.

DSP

Traditional noise-cancelling systems rely on one or more microphones located next to each of the user's ears. These microphones enable such systems to sample noise, at the exact position of each ear, before ambient noise reaches each ear. Since our goal was to build an iPhone App that relies solely on the the standard Apple headset, we were forced to develop an alternative algorithm that only uses a single mic located by the mouth of the user. I.e. an algorithm that does not have access to noise before it reaches each ear and cannot sample noise at the exact position of each ear. The premise of our algorithm was the following: 1) Sample ambient noise via the headset mic, 2) Model the noise as a system of linear combination of past samples, 3)

Use the model to predict future noise samples, 4) Adjust the prediction to account for the difference between noise near the headset mic and noise near each ear.

iOS Programming

The iPhone 5 processes audio as follows: the microphone gives blocks of samples (minimum of 64) called *buffers* to a digital signal processing (DSP) block. This is the only place where we interact with the data before our signal of 64 samples is sent to the earbuds. This means that we can not process samples one at a time, but rather 64 at a time. We also knew that the microphone sampled at 44.1 kHz, and its max sampling rate was 48 kHz.

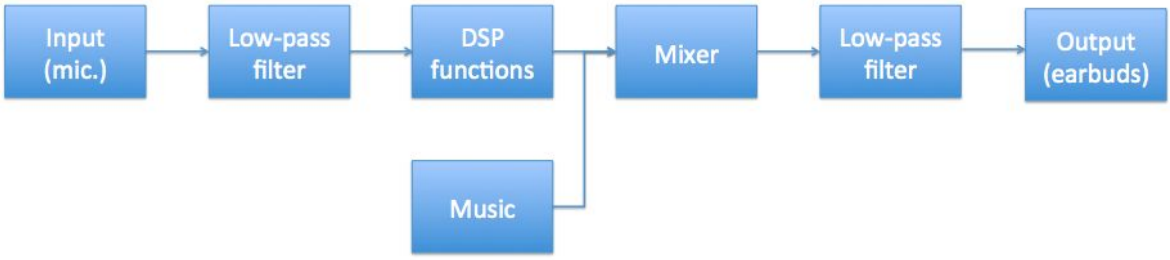
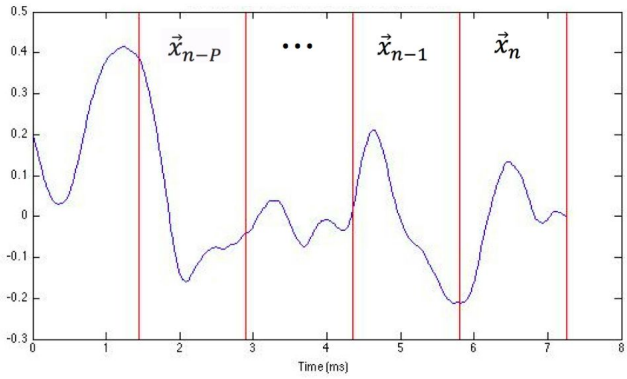


Figure 1. iPhone audio processing chain

Because our noise was correlated with itself and generally not changing with time (wide-sense stationary), we could create an autoregressive model that we could use to predict future samples of the noise. Each sample was represented as a linear combination of p past samples, and therefore using p samples we could predict the next sample with relative accuracy. Because we had to predict the next 64 samples into the future at once, we needed to use the first predicted sample to predict the second one, etc. This method propagated any errors very quickly, resulting in inaccurate predictions. To alleviate this problem, we first downsampled the 64 samples to 4 samples, effectively reducing our sample frequency from 44.1 kHz to about 1.38 kHz. Using Nyquist's Sampling theorem, this told us that the highest frequency component left in our signal will be about 690 Hz. The frequencies we were interested in are below 600 Hz, and therefore downsampling to 4 samples was not been a problem, given that we first applied an anti-aliasing low-pass filter with a cutoff frequency of 600 Hz.

Rather than iteratively predicting one sample four times, we instead predicted four samples at once. We modified the autoregressive model to make each block of four samples a linear combination of P previous blocks of four samples. The Yule-Walker equations had to be modified accordingly:



$$\vec{x}_n = -\sum_{i=1}^{P} A_i \vec{x}_{n-i}$$

$$\begin{bmatrix} \vec{x}_n\vec{x}_n^T & \cdots & \vec{x}_n\vec{x}_{n-P}^T \\ \vdots & \ddots & \vdots \\ \vec{x}_{n-P}\vec{x}_n^T & \cdots & \vec{x}_{n-P}\vec{x}_{n-P}^T \end{bmatrix} \begin{bmatrix} I \\ A_1^T \\ \vdots \\ A_P^T \end{bmatrix} = \begin{bmatrix} C_u \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Figure 2. Modified autoregressive noise model equations

After predicting a block of four samples, we interpolated the signal back up to 64 samples using linear interpolation. This set of 64 samples is the noise we expect the ear to experience approximately one millisecond in the future. By inverting this signal, we attempted to take advantage of destructive interference. If the earbuds played out a signal that was the perfect inverse of the ambient noise that came in, the signals would cancel out and the ears would experience less noise.

Because the microphone might record a different phase-shifted signal than the ears, simply playing the inverted signal is not enough, and we therefore allow the user to adjust the phase through our application interface.

Method of Solution

While building and rebuilding our algorithm, we used the following:

1. MATLAB R2012a software: used to prototype our algorithms, simulate the iPhone environment and restrictions, and evaluate the validity of our C implementation
2. iPhone 5: the platform on which our application is written for. The hardware limitations we encountered are specifically for this model of the iPhone
3. Apple Earpods: has stereo-audio and a microphone for recording
4. Xcode software: programming environment in which we can build our application using C programming language

After further research and consulting with various professors, we drafted a system-level diagram for our algorithm using the theory we developed from above. Our general algorithm is as follows:

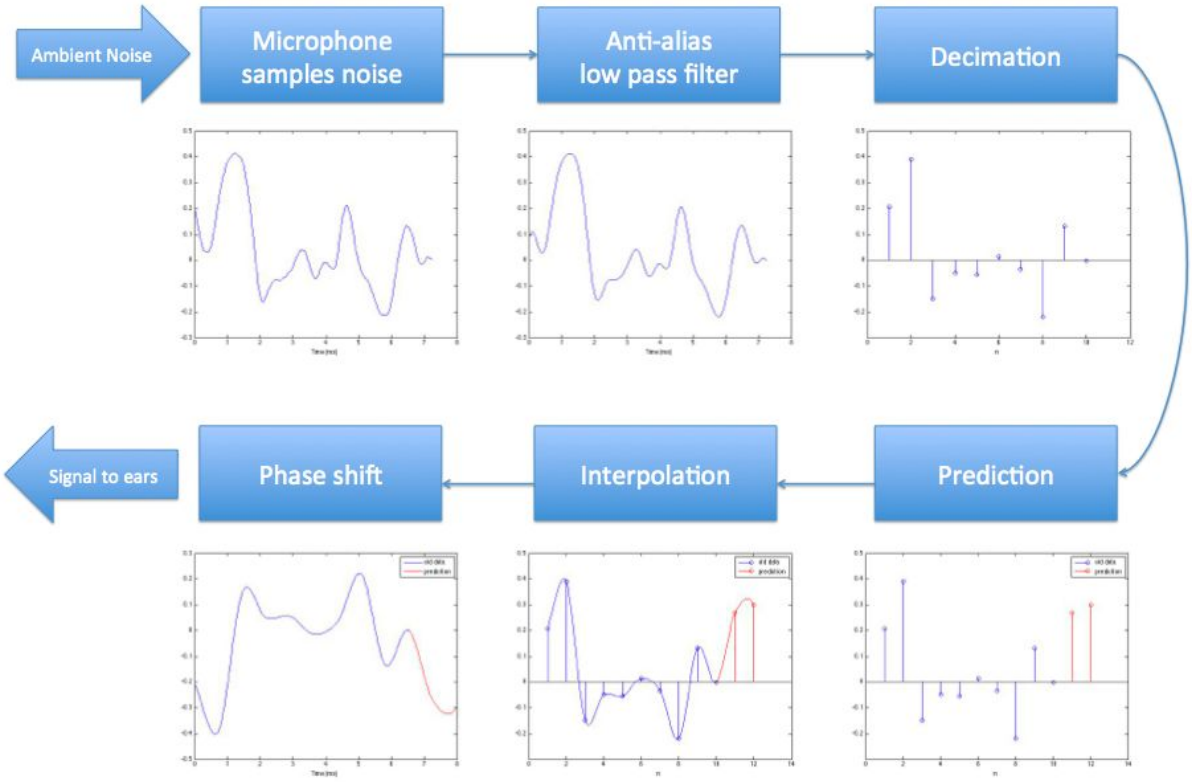


Figure 3. Block diagram of algorithm

Because our project was so mathematically intensive, we first prototyped the algorithm in MATLAB. We read in sample airplane noise and attempt to predict and cancel subsequent parts.

After obtaining satisfactory results in our MATLAB simulations, we implemented our algorithm in Xcode. One member of the group already has a background in C programming, and after taking a Stanford course on iOS, he was able to build an application that could sample noise from the mic while simultaneously outputting sound from the earbuds.

Using the logic that all ambient noise was a linear combination of different-frequency sine waves, we decided that before we attempted to cancel ambient noise, we should first cancel a simple tone, or a single sine wave. For this experiment, we implemented phase-shifting (no prediction necessary). We created a slide bar in our application interface to adjust the outgoing phase in real time. We then played out a sine wave from our laptops using MATLAB, and upon adjusting the phase bar, we could audibly hear the tone become quieter and then louder, deeming this experiment a success.

While implementing phase-shifting, we encountered two discontinuity issues:

1. The first discontinuity issue occurred when interpolating from 4 samples to 64 samples. During downsampling, we took the 16th, 32nd, 48th, and 64th samples of our input buffer. Because we could only interpolate between points. we needed a starting point at the front of the buffer to keep our signal continuous. We used the last point from the previous buffer to fix this problem.
2. The second discontinuity issue occurred when we concatenated our phase-shifted signal with the buffer we previously played out. To keep our buffers relatively smooth, we deleted the first beginning samples of our new buffer and re-interpolated, beginning with the last sample played.

We then implemented the rest of our prediction algorithm in C, and we encountered another problem. We did not know the delay between when we would tell the iPhone to play a sound, and when the iPhone would actually play a sound. Using time-stamps in code, we discovered that this delay time was not consistent. To account for this, we had to introduce another adjustable parameter into our application interface: timing offset.

Ultimately, our application has 4 adjustable parameters:

1. Volume of earbuds output (used to match amplitude of output signal with ambient noise)
2. Timing offset (due to iPhone processing time)
3. Phase shift of Left Ear
4. Phase shift of Right Ear

Because the microphone was not equidistant from both ears, we had to consider the fact that each ear may be experiencing a slightly different phase of the ambient noise.

Results and Analysis

After implementing filtering, downsampling, prediction, interpolation, and phase-shifting, our MATLAB simulation produced the following results:

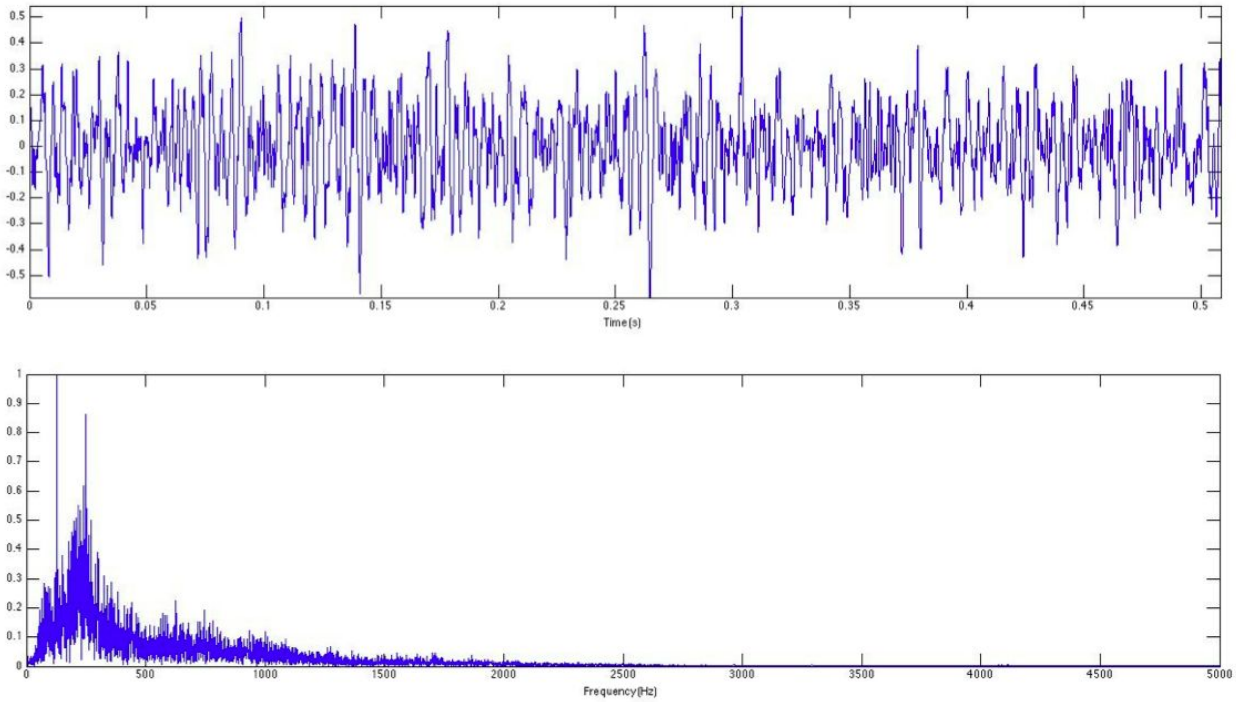


Figure 4. Airplane cabin noise

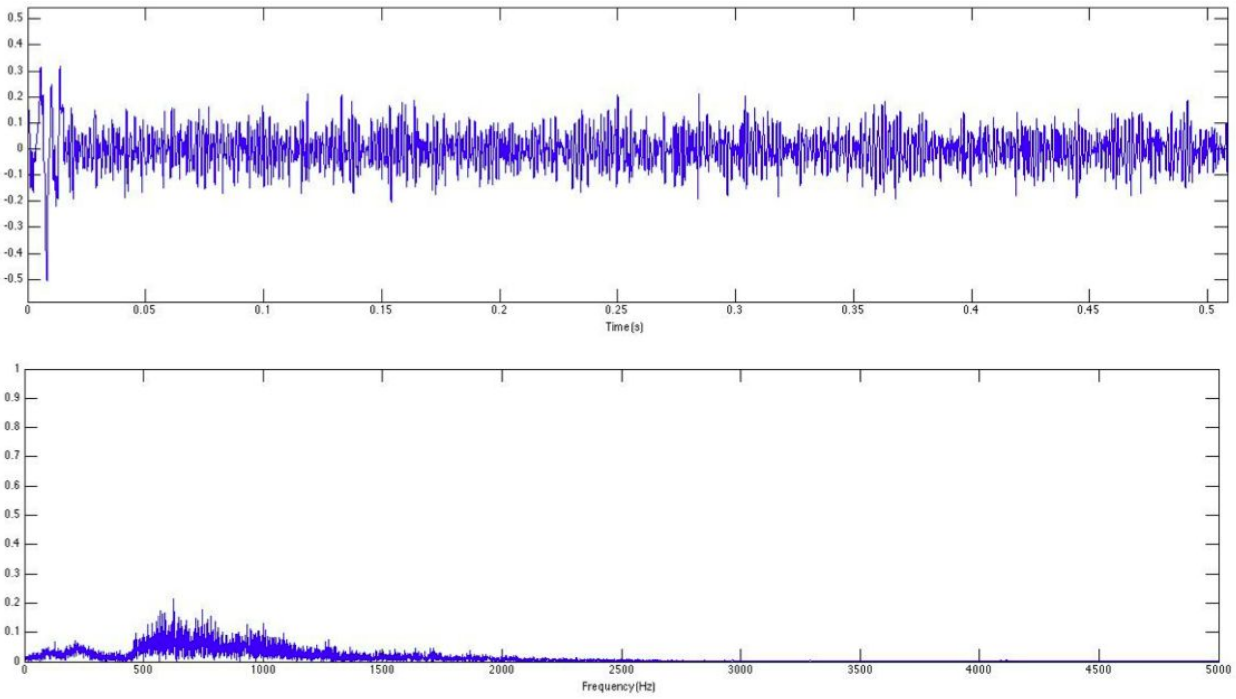


Figure 5. Difference between airplane cabin noise and the predicted noise

As seen in Figure 4 and 5, the low-frequency components of airplane cabin noise were successfully suppressed. These results were especially noticeable when we listened to the original and reduced versions of the airplane cabin noise. The less prominent higher-frequency

components (> 600 Hz) were barely affected, as these components were filtered out and ignored during processing in order to optimize performance.

After implementing our entire algorithm in Xcode, we finally had an application interface with four easy-to-adjust parameters.

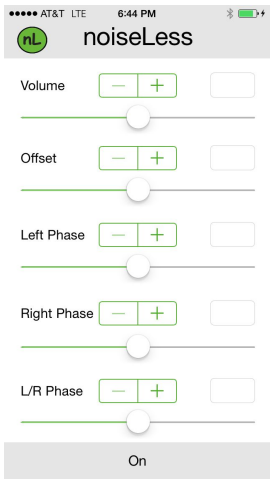


Figure 6. iPhone application interface

Conclusions

When we first started this project, our initial goal was to create a noise-cancellation application with results that could rival those of Bose noise-cancelling headphones. As expected, we had to scope down repeatedly due to time and hardware constraints.

Our greatest achievement was in successfully implementing the algorithm in MATLAB despite multiple complications. This accomplishment demonstrates that our theory was correct, and that after sampling, low-pass filtering, downsampling, predicting, interpolating, and phase-shifting, we have successfully canceled lower frequency components of the ambient noise. Creating this algorithm has taught us about the power of efficient teamwork, communication, and organization. It also showed us the value of tapping into other people and papers for insight.

We realized that if we only predict one or two samples into the future using a 64-sample buffer (no downsampling), we could achieve near-perfect prediction. In other words, at a sampling frequency of 44.1 kHz, we could safely and accurately predict samples 1/44100 seconds into the future. Predicting 64 times that, the time of a full buffer, proved to be more difficult. Perhaps our project would be easier to implement in the future, when the hardware on the iPhone improved to the point where we could sample at significantly higher frequencies. If the sampling frequency was 64 times higher while each buffer remained 64 samples big, we could predict ambient noise almost perfectly without having to decimate, interpolate, and low-pass filter.

Another iPhone hardware quirk that proved obstructive was the fact that we did not know how long the iPhone takes to play a buffer after we a command is sent. We attempted to solve this problem by introducing another adjustable parameter into our application, but this made tuning all the parameters much more tedious and difficult.

We have several ideas for improving our application, the most important of which are:

1. Filtering the input signal to isolate ambient noise so that the application can train itself even in less-ideal conditions (such as noisy rooms or environments with various sounds)
2. Removing the initial-offset parameter once we learn more about the iPhone hardware
3. Removing the earbud-output volume parameter after we develop a way to relate the amplitude of the ambient noise to the amplitude of the signal we should play out
4. Replacing the left and right phase-shifting parameters with a single phase-shifting parameter after we find a stable relationship between the two earbuds and the mic
5. Having our application continue to train itself in the background while playing out cancelling signals. This would allow the application to deal with noises that change over time.

Given what we know now, we would have done a few things differently. For example, we would have sought help from experts in control theory and linear system earlier rather than spend a large amount of time trying to figure out advanced algorithms on our own without sufficient background. We would have also all attempted to learn Xcode and C programming so that we could create code for our application much faster.

Ultimately, we are very satisfied with our progress and the knowledge we have gained since starting this project. Embarking on this gargantuan task has given us a feel for applying our knowledge to real-world situations, and we are excited to tackle all the engineering problems that the our futures hold.

References

1. Oppenheim, A. V., Weinstein, E., Zangi, K. C., … Gauger, D. (1994). Single-Sensor Active Noise Cancellation. IEEE Transactions on Speech and Audio Processing, 2(2), 285-290. DOI: 10.1109/89.279277
2. iOS Developer Library. (2014, April). *Using Audio*. Available from https://developer.apple.com/library/ios/navigation/