ESE 531 Term Project
Fall 2006

Noise cancellation using adaptive digital filtering

This project was done towards the partial fulfillment of the ESE 531 course offered in Fall 2006.

Submitted by

Gupta, Abhishek
Jumanov, Baurzhan
Rajhans, Akshay

Noise cancellation using adaptive digital filtering

Introduction:

In theory we often model noise or interference using deterministic models, which make mathematical treatment of noise possible. However, often in practice, noise can have complicated mixture of different frequencies and amplitudes. What's worse is it can even change from time to time. In such cases, it is difficult to write precise equations and often unnecessary to model noise. Even if we succeed in writing good enough approximations, it would become very difficult to build filters that would suppress this noise. With the advent of Digital Signal Processing algorithms, and availability of fast computing power has made it possible for us to use the digital treatment of noise cancellation effectively and in a cost effective manner.

In this project we tried to implement adaptive noise cancellation on different signals. We considered two scenarios- one in which we have access to the original noise creating source and one in which we don't. In both cases, the algorithm works fairly well. A few sample observations, and conclusions are noted below.

The setups:

Setup A: Access to noise source

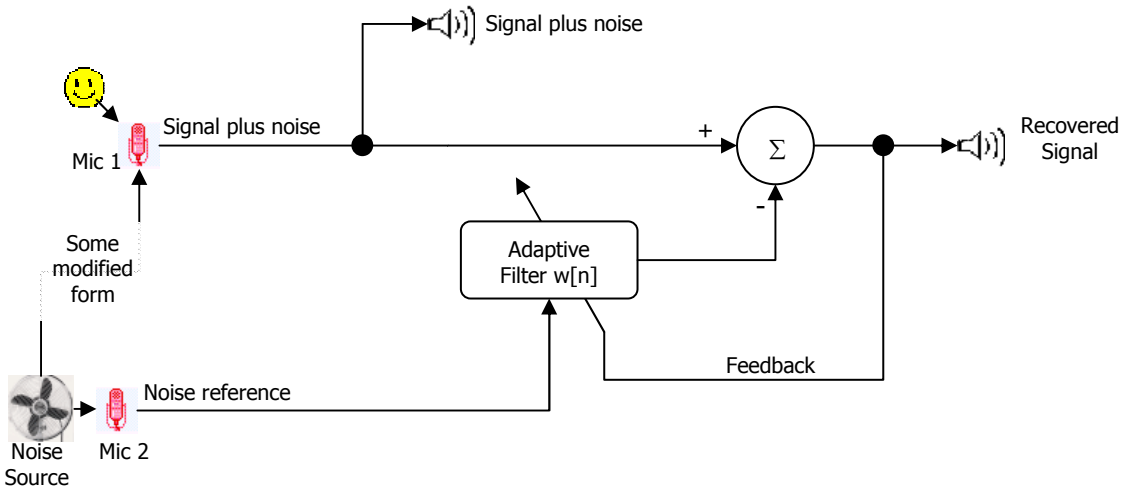

Figure 1: Block diagram for the first setup: Two sources

In this case, there are two signals that are being received. One has the signal plus noise, while the other one contains the reference noise. The noise content that is mixed with the signal is a *modified* form of this reference noise.

In this scenario, we make use of the fact that the noise getting added to the signal is highly correlated with the reference noise. One simple possibility that can be thought of as the noise that actually got recorded was a linear combination of the delayed versions of the noise reference.

Setup B: No access to noise source

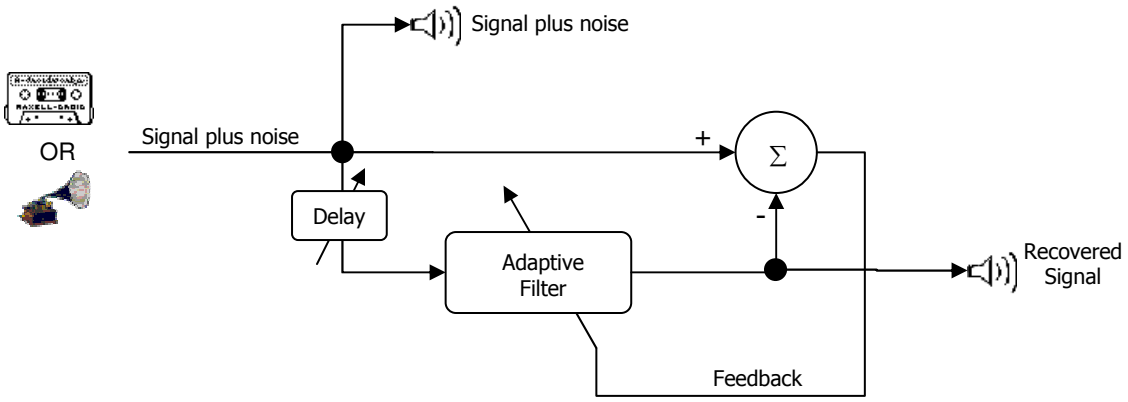

Figure 2: Block diagram for the first setup: Single source

In a scenario similar to the one shown in the figure, we have no access to the reference noise as before. Here we have to rely on only one input which contains signal mixed with noise.

In this case, we make use of the fact that the signal will be somewhat correlated with the delayed version of itself because it comes from the same setting, e.g. same speaker if speech signal, same singer and instruments if music. However the noise, being random in nature would not be correlated. If the precise amount of time delay is properly chosen, we get good noise reduction.

The 'adaptation' part: **LMS / Gradient Descent Algorithm:**

In both the above cases, we take an 'initial guess' as to what our filter coefficients should be, and keep on 'learning' them as time goes. In order to learn the coefficients, we feed back the error (i.e. the difference between the signal-plus-noise and the output of the adaptive filter) to our adaptive filter, and the filter tries its best to minimize it. Since the algorithm tries to make the coefficients that make the mean squared error the least, it is called Least Mean Squared error (LMS algorithm).

The algorithm feeds back the rate of change of error, i.e. the error 'gradient'. In order to find the least error, we need to approach a minimum, i.e. where gradient will become zero. Since the algorithm also tries to reduce the gradient all the time, it is also (popularly) known as 'gradient descent' algorithm.

Gradient Descent update rule:

$$\Delta\vec{w} = -\eta\nabla E[\vec{w}]$$

$$i.e.\ \Delta w_i = -\eta\frac{\partial E}{\partial w_i}$$

Here, w is a matrix of coefficients of the digital filter; E is the error (a function of weights w) and η is the learning rate.

Actual implementation: changes, difficulties and modifications

1. For the first scenario, we initially tried recording signal and noise, and noise reference using two different microphones, as shown in the diagram. However, due to lack of precise synchronization of start of recording of the two, and due to the large sampling frequencies of the recorded files, the two files were becoming intractable.

2. To overcome this problem, or rather to find a way around, we had to revert to a more implementable method. In the simpler method, we took a noise file from the user, and simulated a noise-to-be-added by filtering the original noise. This was done by means of taking any filter coefficients h[n] from the user, and generating the noise-to-be-added by convolving the reference noise and the filter coefficients. Then we added this 'synthetic' noise to the signal, (again, taken from the user) and generated a sample signal-plus-noise file. Apart from simplicity, the added advantage of using this method was the tractability. Synthesizing noise this way from given h[n] coefficients made it possible for us to crosscheck whether our algorithm was going in the right direction, and tracking these filter coefficients or not.

3. During practical implementation of the LMS algorithm, we needed to feed back $-2 \cdot \eta \cdot \varepsilon_k$, as the approximation of the gradient (ε_k here, is the instantaneous error). The derivation of this calculation of the gradient has been explained in the book Adaptive Signal Processing by Bernard Widrow and Samuel D. Stearns, 1985. In practice, we fed back $-\eta \cdot \varepsilon_k$, since changing η will take care of the constant 2.

4. While generating synthetic noise, we have used the convolution function, which does the job of flipping and shifting the h[n] values. The adaptive filter, however, tracks just the linear combination. So filter coefficients that the algorithm converges to are oriented from last coefficient to the first.

5. We kept the initial guess of w's to be all zeros and all ones. Since the algorithm learns on its own, the initial guess is not too important.

Observations:

In order to check the performance of the algorithm, we tried different combinations, i.e. different learning rates (η's) and different h[n] values while generating synthetic noise. A few graphs depicting these observations are given below.

1. Effect of changing different learning rate on the convergence of the algorithm:

   Case I: Considerably low learning rate: η kept at 0.0001

   The instantaneous values of signal plus noise, noise and restored signal are plotted as a function of time.

(Figure on the next page)

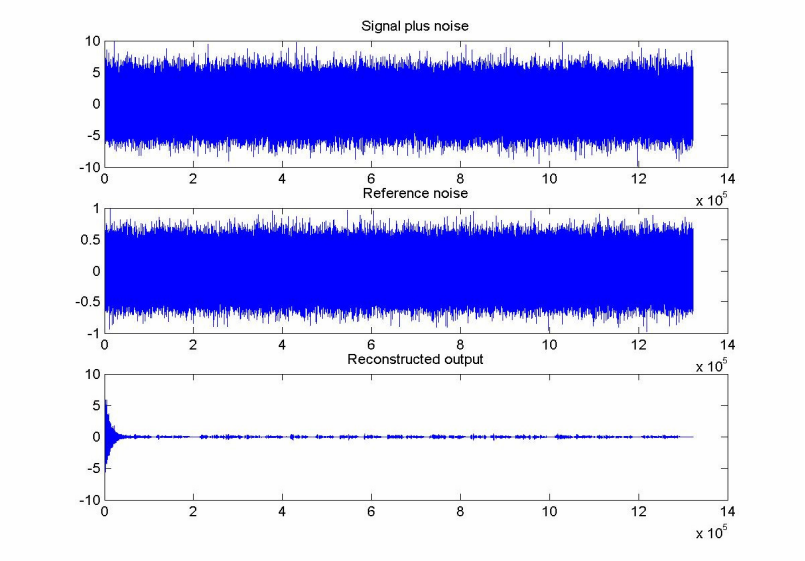

Figure 3: Progress of the algorithm with time for low η

h[n] for generating noise: [.1,.2,.3,.4,.5,.6,.7,.8,.9,0,1,2,3,4,5,6];
Length of reconstruction filter (L)=16;
η =0.001

Corresponding adaptive filter coefficients 'w' that the algorithm generated:
w = [6.0001 5.0001 4.0002 3.0001 2.0000 1.0001 0.0002 0.9002 0.8002 0.7000
0.6000 0.5001 0.4001 0.3001 0.2000 0.1000]^T

Case II: Moderately big learning rate: η kept at 0.1

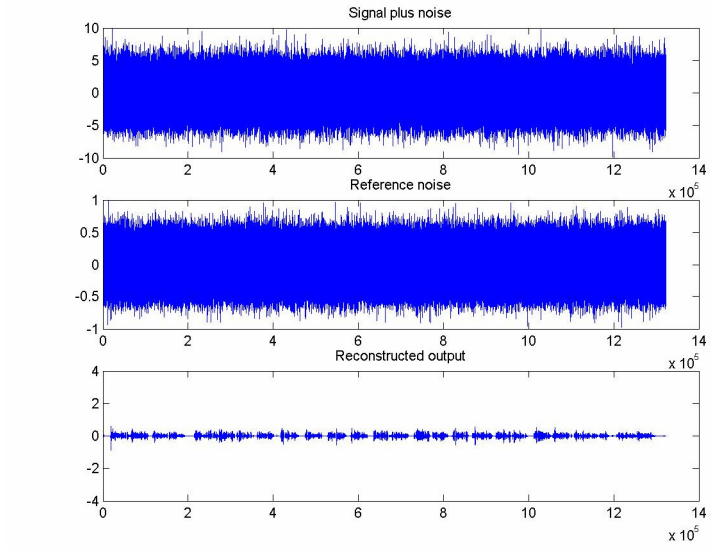

Figure 4: Progress of the algorithm with time for medium η

h[n] for generating noise: [.1,.2,.3,.4,.5,.6,.7,.8,.9,0,1,2,3,4,5,6];
L=16; η =0.1

w = [6.0001 5.0002 4.0004 3.0004 2.0003 1.0003 0.0003 0.9004 0.8005 0.7005 0.6005
0.5005 0.4004 0.3003 0.2002 0.1002]^T

Case III: Very high learning rate: η kept at 1.1

Here we can see that the algorithm diverges instead of converging. The reconstructed audio plays very loud noise.

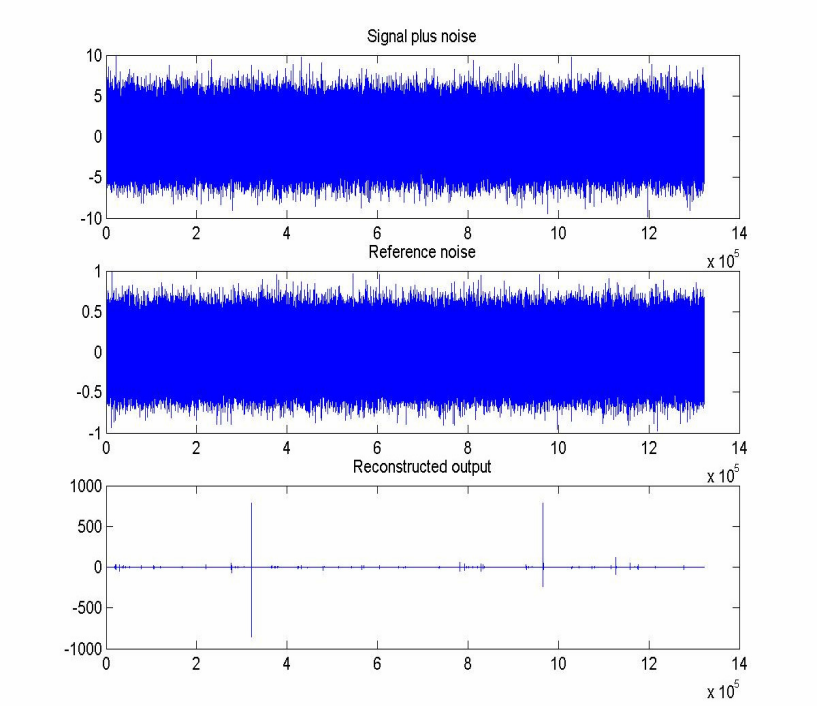

Figure 5: Progress of the algorithm with time for very high η

h[n] for generating noise: [.1,.2,.3,.4,.5,.6,.7,.8,.9,0,1,2,3,4,5,6];
L=16; η =1.1

w = [6.0034 5.0038 4.0022 3.0066 2.0047 1.0041 0.0037 0.9026 0.8013 0.7050 0.6047 0.5056 0.4070 0.3038 0.1996 0.1039]^T

The usual .wav magnitudes lie between +/- 1. Note the shoot up in the magnitudes of reconstructed output.

2. How the algorithm updates the h[n] values as time goes by:

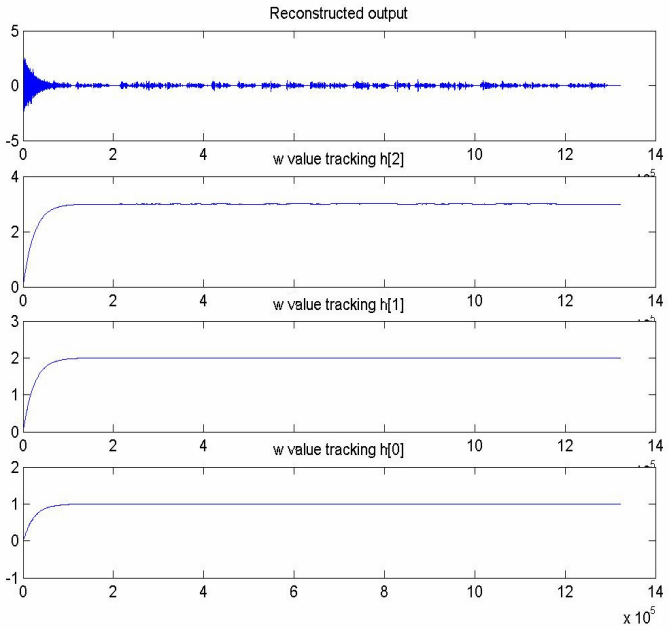

Figure 6: Tracking of the filter coefficients over the course of time

Here η was kept considerably low (0.005), so as to observe the h[n] parameters being updated.

h[n] for generating noise: [1, 2, 3]; L=16; η =0.005

w = [-0.0001 0.0001 0.0002 0.0000 -0.0002 -0.0002 0.0001 0.0002 -0.0000 -0.0003 -0.0002 -0.0000 0.0001 3.0000 1.9998 0.9998] ^{T}

3. Different h[n] combinations as tracked by the algorithm:
(L and η kept constant at 16 and 0.01 respectively)

- h[n] for generating noise: [.1,.2,.3,.4,.5,.6,.7,.8,.9]

w = [0.0001 0.0001 0.0002 0.0002 0.0002 0.0002 0.0002 0.9002 0.8002 0.7002 0.6002 0.5002 0.4002 0.3001 0.2001 0.1001] ^{T}

- h[n] for generating noise: [.1,.2,.3,.4,.5,.6,.7,.8,.9,0,1,2,3,4,5,6]

w = [6.0001 5.0001 4.0002 3.0001 2.0000 1.0001 0.0002 0.9002 0.8002 0.7000 0.6000 0.5001 0.4001 0.3001 0.2000 0.1000]^{T}

- h[n] for generating noise: [1,2,3]

w = [-0.0001 0.0001 0.0002 0.0000 -0.0002 -0.0002 0.0001 0.0002 -0.0000 -0.0003 -0.0002 -0.0000 0.0001 3.0000 1.9998 0.9998]^{T}

Comments and conclusions:

1. The rate of convergence depends upon the value of the learning rate (η). As the value of η becomes smaller the rate of convergence becomes slower, but the tracking is more precise. If η is made bigger, the tracking is not as precise, but the convergence is faster. This could be compared in figures 3-5.
2. There is an optimal value for η. If we increase η beyond this limit, the algorithm diverges instead of converging. This can be seen in figure 5.
3. If η is kept close to the optimal value, the coefficients of the adaptive filter track the coefficients used to synthesize noise. This could be seen from figure 6.
4. As said above, the coefficients tracked by the adaptive filter are flipped as compared to the h[n] coefficients. This was because we used convolution in case of h[n], and linear combination in case of w's.
5. The generation of noise using different h[n] coefficients was just for testing purposes. If precise synchronization between the recording of the reference noise and signal plus noise, the algorithm could be directly implemented.
6. In the single source implementation, we tested the algorithm on four old songs, which were possibly recorded from the old gramophone records or audio cassettes. The reconstruction is satisfactory. However, the learning rate and the time delay (amount by which the original file needs to be delayed) need to be tuned precisely. We got best results for delay=100, length of reconstruction filter 128, and η=0.005.
7. The length of the reconstruction filter and the learning rate values are user programmable. In real world applications, the length of the reconstruction filter should be decided by the precision needed.

Reference:

[1]. Widrow B., Stearns D.S. (1985). Adaptive Signal Processing (pp. 99-114; 302-350) Englewood Cliffs, N.J. Prentice-Hall, c1985

Appendixes:

1. MATLAB code for the two settings:

Setting 1: Two source implementation:

```
% Two Channel implementation of the LMS algorithm
clear all;close all;

sigfile='bella.wav'; % Read the recorded signal
noisefile='noise.wav'; % Read the recorded noise

[sig_noise,fs_sig,nbits_sig,refnoise]=sig_plus_noise(sigfile,
noisefile); % Generate a synthetic noise from this reference noise,
mix it with signal, and return the mixed file
file_len=length(sig_noise); % Length of the file

L=input('Enter the order of the reconstruction filter: '); % Order
of the filter

e=zeros(file_len,1); % Put place-holder zeros for error vector
w=zeros(L,1); % Put place-holder zeros for weight vector
```

```
eta=input('Enter the learning rate for LMS: '); % Gain constant that
regulates the speed and stability of adaptation

for i=L+1:file_len
   e(i)=sig_noise(i)-refnoise(i-L+1:i)'*w; % Calculation of Error
vector
   w=w+2*eta*e(i)*refnoise(i-L+1:i); % Calculation of the Weight
vector
end;
subplot(311);
plot(sig_noise); title('Signal plus noise');
subplot(312);
plot(refnoise); title('Reference noise');
subplot(313);
plot(e); title('Reconstructed output');

echo on;
% The final values of converged w of the filter:
echo off;
w

wavwrite(e,fs_sig,nbits_sig,'restored.wav') % Write the output
signal to a music file
```

```
function noiseout = gennoise(noisein,h,fs,nbits)

% This fucntion takes in the reference noise sample, and the h[n] to
% generate a synthesized version of noise. This noise could be
realistic as to what could get added in practice
% compared to just the reference noise.

noiseout=conv(noisein, h); % Generate a synthetic noise
noiseout1=noiseout/max(abs(max(noiseout)),abs(min(noiseout))); %
Normalize the magnitudes
wavwrite(noiseout1,fs,nbits,'addnoise.wav'); % Write the synthetic
noise into a .wav file
```

```
function
[sig_noise,fs_sig,nbits_sig,ref_noise]=sig_plus_noise(sigfile,noisef
ile)

% This function takes in the reference signal, the noise to be added
and
% generates a synthesized signal plus noise file.

[ref_sig,fs_sig,nbits_sig]=wavread(sigfile); % Read the pure signal
to be added
[ref_noise,fs_noise,nbits_noise]=wavread(noisefile); % Read the
reference noise

h=input('Enter the h[n] for generating noise: '); % Get the h[n]
from user
add_noise=gennoise(ref_noise,h,fs_noise,nbits_noise); % Generate a
synthetic noise signal to be added
```

```
sig_noise_len=min(length(add_noise),length(ref_sig)); % Get the
length of the larger of the two files to be mixed

sig_noise=zeros(sig_noise_len,1); % Put place-holder zeros

sig_noise= (ref_sig(1:sig_noise_len)+ add_noise(1:sig_noise_len)); %
Mix the two files

wavwrite(sig_noise,fs_sig,nbits_sig,'sig_noise.wav'); % Write signal
mixed with noise output to the music file
```

%%End of two source implementation%%

Setting 2: Single source implementation:

```
% One channel implementation of the LMS algorithm
clear all;close all;
[sig,fsd,nbitsd]=wavread('johnson.wav'); % Reading music file with a
noise
sig2=sig(1:1000000,1); % Processing only first 1000000 samples to
reduce the time, Input to the primary channel
ylen=length(sig2);
d=150; % Large delay to make noise uncorrelated as possible
sig_del=zeros(ylen,1);
sig_del(d:ylen)=sig2(1:ylen-d+1); % Delayed signal, Input to the
reference channel
e=zeros(ylen,1);w=zeros(ylen,1); w1=zeros(ylen,1); L=151; % Filter
order
eta=0.00005; % Learning rate that regulates the speed and stability
of adaptation
for i=L+1:ylen
   e(i)=sig2(i)-transpose(sig_del(i-L+1:i))*w(i-L+1:i); %
Calculation of Error
   w(i-L+2:i+1)=w(i-L+1:i)+2*eta*e(i)*sig_del(i-L+1:i); %
Calculation of the Weight vector
   w1(i)=transpose(sig_del(i-L+1:i))*w(i-L+1:i); % Output signal of
our code
end;
subplot(411);
plot(sig2); title('Signal');
subplot(412);
plot(sig_del); title('Delayed signal');
subplot(413);
plot(e); title('Error');
subplot(414);
plot(w1); title('Restored signal');
echo on;
%complete!
echo off;
w1=w1/max(abs(max(w1)),abs(min(w1))); % Normalization of output to
prevent data clipping
wavwrite(w1,fsd,nbitsd,'restored_johnson_w4.wav'); % Write the
output signal to the music file
```