Active Noise Control: Basic Understanding

Article · August 2013

CITATIONS
2

READS
6,509

1 author:

Anshuman Swain
University of Maryland, College Park
20 PUBLICATIONS **22** CITATIONS

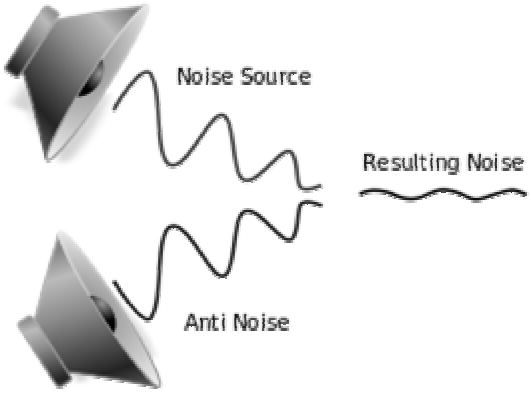
SEE PROFILE

Active noise control: Basic Understanding

Anshuman Swain, *KVPY Fellow (SA-stream)*

Introduction

In the background of today's world where machines can really be a nuisance with their burst of loud noises, our delicate ears fall prey to the loud ambience of their being. With the increase in the number of industrial equipment, the noise problem has become more and more evident. The traditional approach to control the noise was through passive methods like mechanical barriers etc… to attenuate the undesired noise. These passive silencers are valued for their high attenuation over a broad frequency range; however they are relatively costly, large and ineffective at lower frequencies. So, in recent years Active Noise Control, usually shortened to ANC, has gained quite popularity to remove loud, unwanted noise, so as to create less noisy conditions for work, using electro-acoustic or electromechanical systems. Based on the very fundamental and basic principles of wave superposition and wave cancellation, ANC, has faced numerous practical problems but nevertheless, there are a plethora of applications for which the ANC devices can possess and are even being used for. And because of their efficacy and lower cost and maintenance factors, they have been welcomed everywhere with open hands.



What is the ANC?

The Active Noise Control System is basically an entity that removes unwanted or loud noise from the surroundings. There is basically a sensor, which detects the noise, and then there is some computation and finally the noise is generated in an opposite phase by a loudspeaker, which allows it to cancel the unrequited signal or noise. It doesn't in practical life make the noise zero but reduces it by a considerable amount.

Contemporary ANC or, active noise control is usually achieved by the use of analog circuits or digital signal processing. Adaptive algorithms are prepared to examine the waveform of the noise in the background, then basing on the specific algorithm create a signal that will either phase shift or reverse the polarity of the original signal. This reversed signal is then amplified and a transducer produces a sound wave directly proportional to the amplitude of the original waveform, enabling destructive interference. This efficiently diminishes the amount of the perceivable noise.

Project Submitted under the guidance of: Dr. Debi Prasad Das, *Scientist, Institute of Minerals and Materials Technology (CSIR-IMMT), Bhubaneswar, Odisha*

A noise-removal or cancelling speaker may be placed simultaneously with the source of the sound to be attenuated. In the context of discussion, it must possess the same audio power level as the source of the unrequited sound. On the other hand, the transducer producing the cancellation signal may be placed at the site where sound attenuation is required. This needs a lesser power level for cancellation but is efficient only for a single user. Noise cancellation at other sites is more complicated as the three dimensional wave fronts of the unwanted sound and the cancellation signal could match and produce alternating regions of constructive and destructive interference. In small enclosed spaces (e.g. the passenger compartment of a car) such global cancellation can be achieved via multiple speakers and feedback microphones, and measurement of the modal responses of the enclosure.

Basic components of an ANC System and an insight into its working

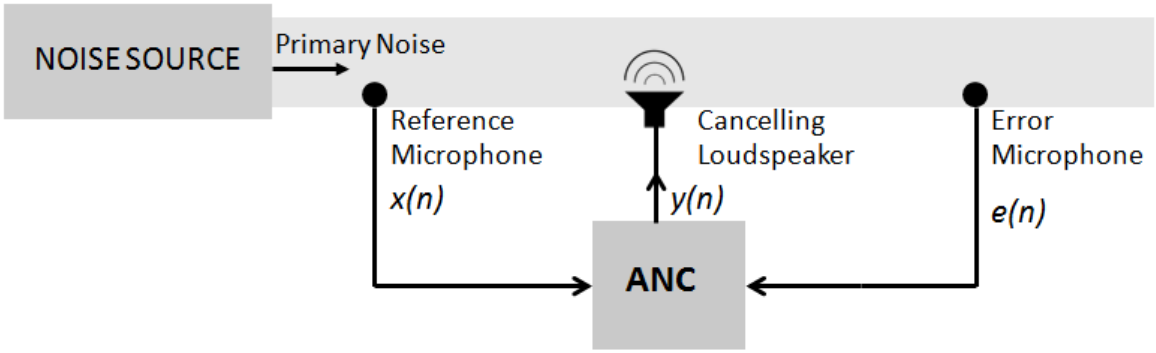


Figure 0 Basic Components of an ANC

Basically there is a noise source from which we get the primary noise signal through the Reference microphone (let us call this as ***x(n)***) which then goes to the ANC, which analyses it. Then it creates an anti-noise (say ***y(n)***) through a cancelling loudspeaker. The remnant noise is then picked up by the error microphone (let us call this as ***e(n)***) and is transferred to the ANC as a kind of feedback. Our primary goal of the system is to minimize ***e(n)***.

Now the first question that arises is that a sound signal is a function of time…so why is it written in terms of some 'n'? In actuality, when a sensor detects a signal, it is not a continuous analog detection; rather it detects the signal at discrete points in time and then forms the signal in its memory. So, there is something called **sampling frequency**, which denotes the number of times, a sensor collects information in a second (so, larger the sampling frequency the better it is.)

To show the above effect let us take a sine function, say: $x1(n)=\sin(2\pi n\frac{f}{f_s})$, where ***f*** is the frequency of the original analog signal and f_s is the frequency of sampling. So, we take four examples of the same analog sine wave with ***f*=50 Hz** and try to vary the sampling rate.

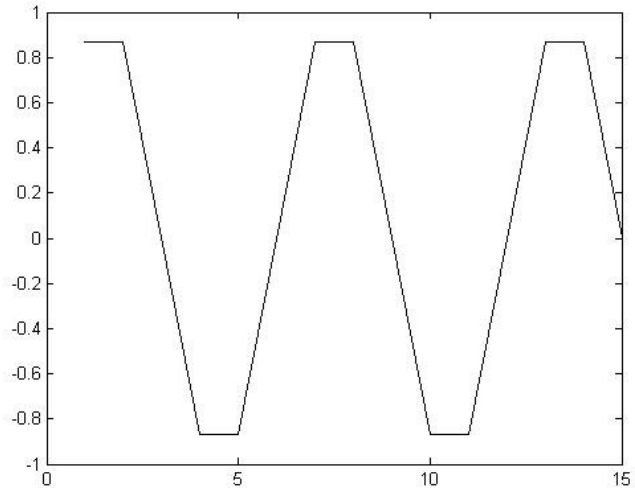


Figure 1a

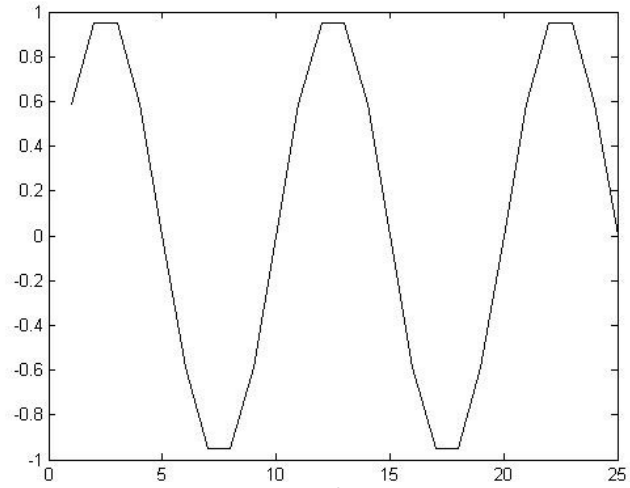


Figure 1b

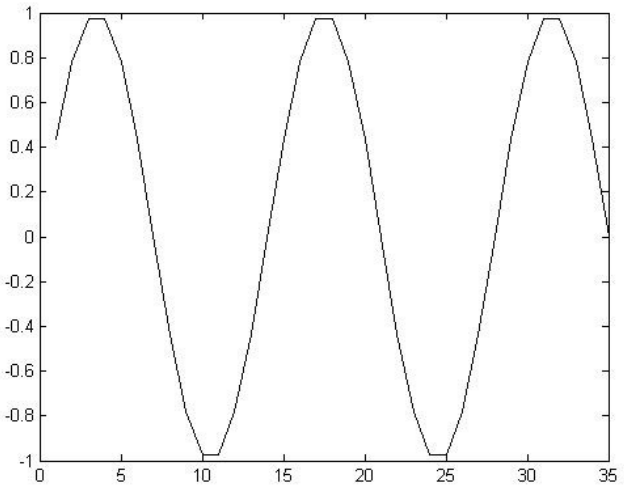


Figure 1c

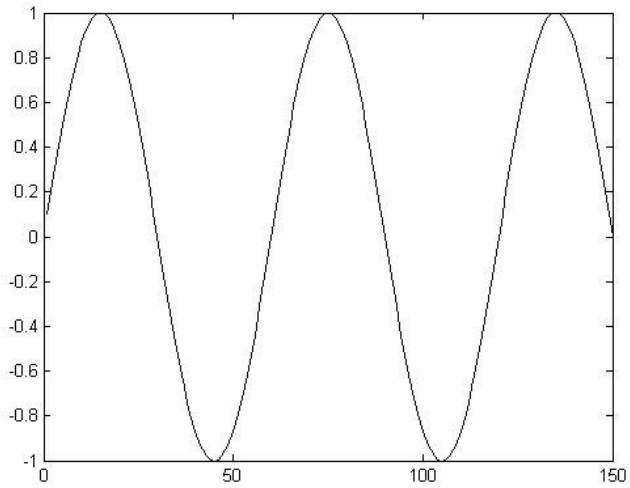


Figure 1d

These figures show the computer generated output using **Matlab**, for different sampling frequencies. Figure 1a has f_s=300 Hz; 1b has f_s=500 Hz, 1c has f_s=700 Hz and 1d has f_s=3000 Hz. It is evident from the figures, what a difference sampling frequency can make to the sensed version of the signal which is the source of information for all our computation. Thus, to obtain better results, we always need a better sampling frequency.

According to Nyquist–Shannon sampling theorem, If a function $x(t)$ contains no frequencies higher than B hertz, it is completely determined by giving its ordinates at a series of points spaced $1/(2B)$ seconds apart. But it is better to have a sampling frequency 10 times or more the sample frequency.

Another important aspect of signal processing is that most real-life signals are not just simple sine or cosine waves but a mixture of many such waves of varied frequencies and amplitudes. Moreover other than just being a mixture of a lot many simple signals, it also has mixing with the main source sound which gets delayed by various means, like echoing etc… So, taking all that in mind, we need to process the data for detecting the sound waveform.

As an example we can take a wave, which is a mixture of the following waveforms. (Matlab simulation)

```
clear
for n=1:100000
    x1(n)= 2*sin(2*pi*n*0.0001+2);
    x2(n)= 0.8*sin(2*pi*n*0.0002);
    x3(n)= 1*cos(2*pi*n*0.00015+4.223);
    x4(n)= 1.4*cos(2*pi*n*0.0006);
    x(n)= x1(n)+x2(n)+x3(n)+x4(n);
end
```

Now we can have the look at the graphs:

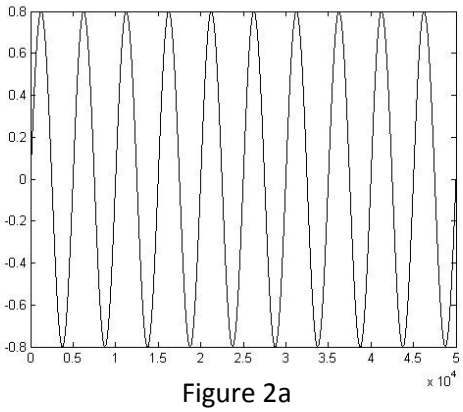


Figure 2a

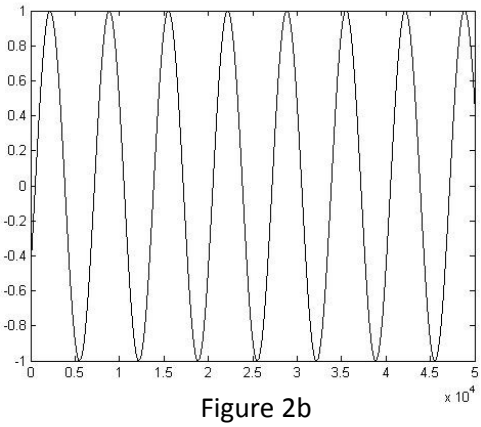


Figure 2b

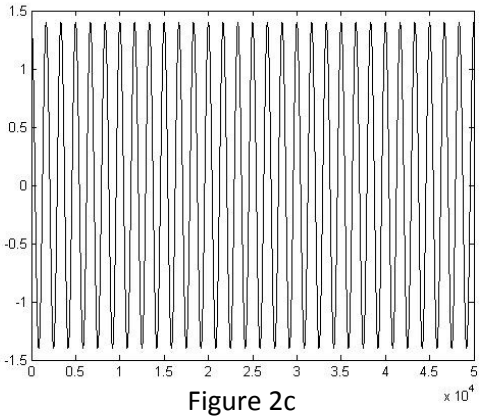


Figure 2c

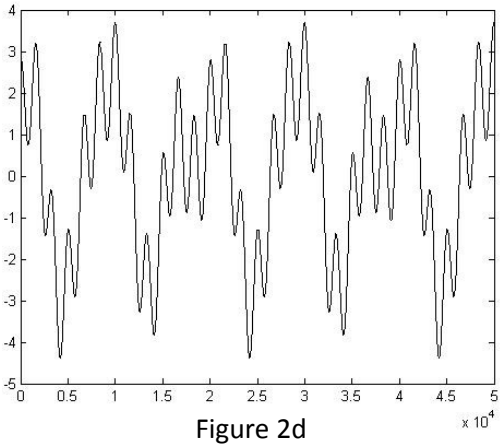


Figure 2d

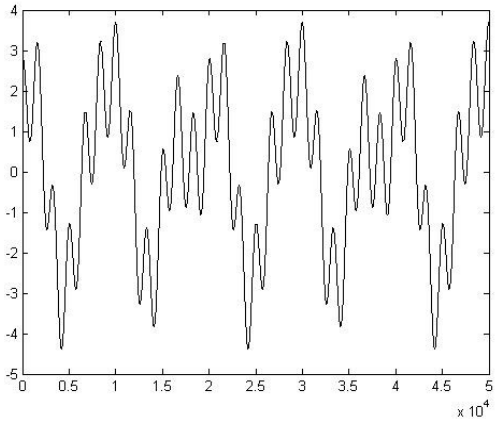


Figure 2e

Here the figures 2a, 2b, 2c and 2d represent the graphs of x1, x2, x3 and x4 respectively. And the superposition of all these waveforms results in a waveform which looks like figure 2e.

Now a real scenario can have all sorts of additions and deletions and mixing of waveforms. So, our

basic aim is to find out this scenario, which in more scientific term is called as a **plant** or a **system**. These plants or systems can be of many types. But we would not discuss them here, but would rather move on to the topic of **System Identification**.

Basically System identification is the process of transforming the real-time acoustic system to a set of mathematical equations, which can be then used to define the system as a precise mathematical model.

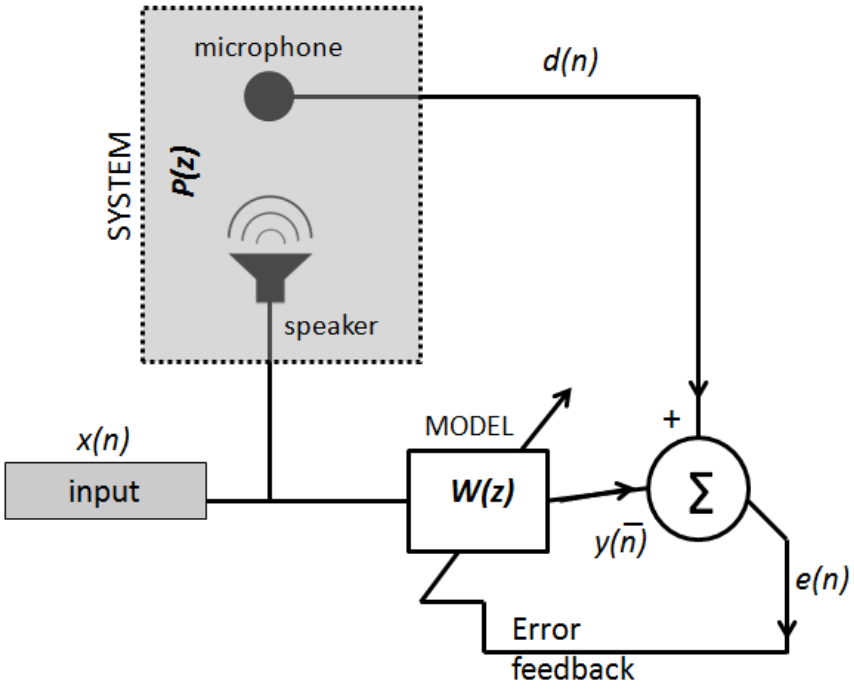


Figure 3

Figure 3 shows the main steps that are involved in the process of system identification. This is how we obtain the model of a real-life system, which may involve a lot of complex phenomena.

Basically for a linear system, with output ***y(n)*** and imminent input ***x(n)***, can be represented as;

$$y(n) = f\,[\,x(n), x(n-1), x(n-2) \ldots\ldots\ldots x(n-N+2), x(n-N+1)\,] \qquad \ldots.1$$

where **N** is the available length of memory for the electronic device or filter.

$$y(n) = a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) \ldots\ldots\ldots + a_{N-2} x(n-N+2) + a_{N-1} x(n-N+1) \qquad \ldots.2$$

(This type of equation is called a difference equation.)

Now our desired output must be equal to what the original system output ***d(n)*** is. Thus, we continuously modify our coefficients, using a weight equation and thus, obtain the required coefficients, using the error signal. The error signal is the waveform that remains in the medium after the superposition of ***d(n)*** and ***y(n)*** and is termed as ***e(n)***.

$$e(n) = d(n) - y(n) \qquad \ldots.3$$

And the weight update equation is,

$$w(n+1)= w(n)+ \mu\, e(n)\, X \qquad \text{....4}$$

where $X=[\, x(n), x(n-1), x(n-2)\ldots\ldots\ldots x(n-N+2), x(n-N+1)]$ and μ is the step-size.

Now we would get the required set of coefficients a_i after getting updated a number of times using this algorithm.

After getting the most optimized solution, through this gradient method, which relies on the continuous updating of the coefficients until we reach a maxima or minima, we can look forward to apply it in various day-to-day problems. We can look over for the application, such as curve fitting, error optimization etc… Let us start with curve fitting.

The first example that we shall take is simple. It shall involve the determination of a linear function from a predefined set of input and output. The Matlab program for the same is given below:

```
clear
x= [1 2 3 4 5 6 7];
yd=[3 5 7 9 11 13 15];
m=0.01;
c=0.01;
mu=0.01;
for n1=1:5000
for n=1:7
    Ya(n)= m*x(n)+c;
    e(n)=Ya(n)-yd(n);
    m=m-mu*e(n)*x(n);
    c=c-mu*e(n);
end
E1(n1)=mean(e.^2);
end
plot (E1);
```

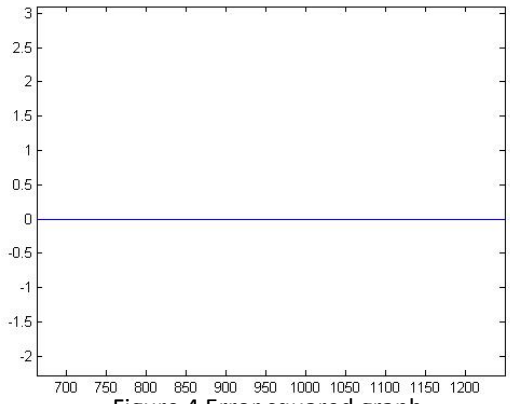


Figure 4 Error squared graph

This program will also give us the error squared graph throughout the optimization process. As, it was a very simple identification the error dropped to zero very quickly. The next example will involve the identification of a fourth degree polynomial function, and we shall discuss it in little more detail.

First let us have a look at the Matlab program which will also give us a look at the input and the output data, and the same procedure in more detail.

```
clear
x=[1 2 3 4 5 6];
y=[4 19 84 259 628 1299];
a=0;
b=0;
c=0;
d=0;
f=0;
mu=.000016999956
 for n1=1:20000
     for n=1:5
         Y(n)= a+b*x(n)+c*(x(n).^2)+d*(x(n).^3)+f*(x(n).^4);
         e(n)= y(n)- Y(n)
         a=a+mu*e(n);
         b=b+mu*e(n)*x(n);
         c=c+mu*e(n)*(x(n)^2);
         d=d+mu*e(n)*(x(n)^3);
         f=f+mu*e(n)*(x(n)^4);
    end
     E(n1)=mean (e.^2);
 end
 plot (E);
```

So, from this we can also mark another thing, which we had not discussed earlier, that the entity that is multiplied with ‘mu’, is called gradient because it is the partial differential of the squared err with respect to the coefficient we are updating.

The error and the time in which it is reduced to a minimum depend a lot on the step size and the number of iterations. So, by manipulating these two, we can achieve low errors in minimum span of time. And this is an important aspect of using this algorithm, which is also called as the LMS, or the Least Mean Square algorithm.

We can also use this to represent any polynomial as a Gaussian or Fourier function, and estimate the errors and try to reduce the errors. One such example is given below, which expresses a polynomial as a sum of various trigonometric functions.

```
clear
x= [1 2 3 4 5 ];
yd=[3 6 11 18 27 ];
b=0;
w=0;
a=0;
w2=0;
mu=0.1;
for n1=1:100000
for n=1:5
    Ya(n)= a*sin(w*x(n))+b*cos(w2*x(n));
    e(n)=Ya(n)-yd(n);
    a=a-mu*e(n)*sin(w*x(n));
```

```
    b=b-mu*e(n)*cos(w2*x(n));
    w=w-mu*e(n)*a*cos(w*x(n))*x(n);
    w2=w2+mu*e(n)*b*sin(w2*x(n))*x(n);
end
E1(n1)=mean(e.^2);
end
plot (E1);
```

Now after we have seen the various examples of function determination, which is in fact a form of system identification, by which we can model different unknown acoustic systems mathematically.

Next we shall get more into what we want ourselves to do: Acoustic control. So, basing on the same logic and algorithm, we go on to code an adaptive filter, which has been given below:

```
clear
p=34
X=zeros(1,p);
A=zeros(1,p);
f=50;
fs=3000;
mu=0.0210
for n=1:150
    x1(n)=1*sin(2*pi*n*f/fs);
    x2(n)=0.8*sin(2*pi*n*2*f/fs);
    x3(n)=0.5*sin(2*pi*n*3*f/fs);
    x(n)=x1(n)+x2(n)+x3(n);

    X(2:p)=X(1:(p-1));
    X(1)=x(n);
    y(n)= X*A';
    e(n)= x1(n)-y(n);
    A= A+mu*e(n)*X;
    g= e.^2

 end
```

in this example we took ***d(n)*** or the desired output as ***x*1**. So, basically its work is to filter out a particular signal from a set of different signals mixed together. So, it is an example of the LMS algorithm system identification and coefficient determination. Now let us alter the step sizes and see the varied results.

Here we have kept all the things constant and only changed the value of the step size ***mu*** and then plotted the error graphs, versus the number of iterations. The graphs that were obtained are given below with the value of ***mu*** that was used to plot them.

As we can see from the graphs, the error decreases more slowly with small step size but as the step size increases fine-tuning of error is not possible and it keeps on 'humming' continuously for a long time with the same error frequency. A minima is attained with least time and stabilization, somewhere in the middle (in this case for ***mu***= 0.021).

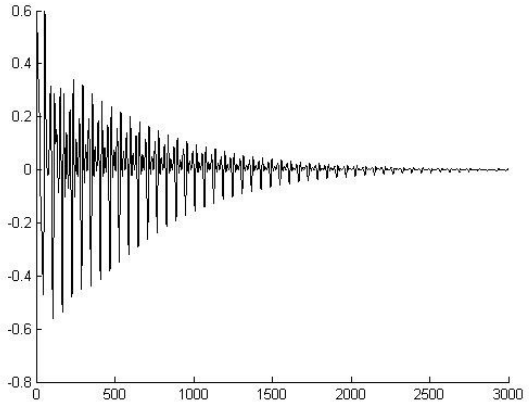


Fig 5a. mu=0.010

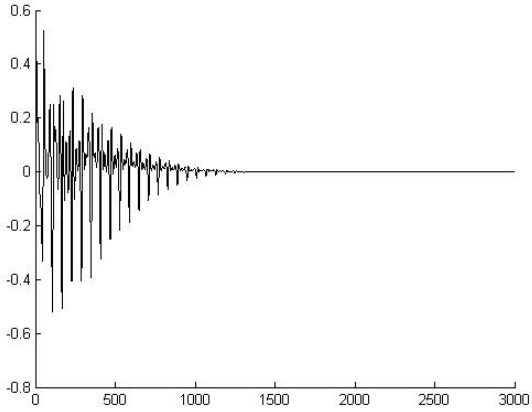


Fig 5b. mu=0.020

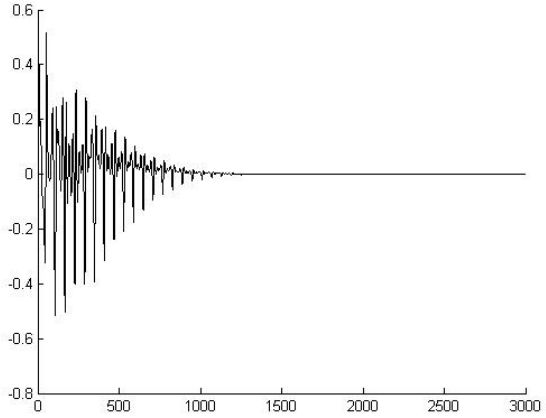


Fig 5c. mu=0.021

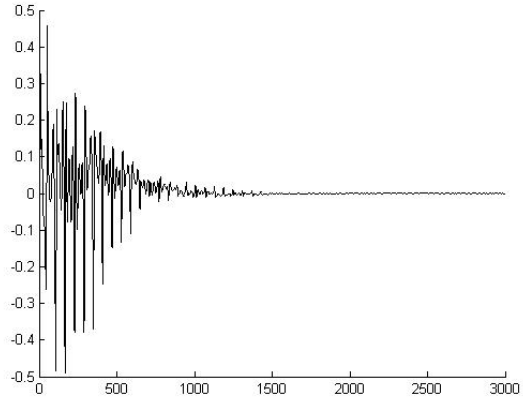


Fig 5d. mu=0.030

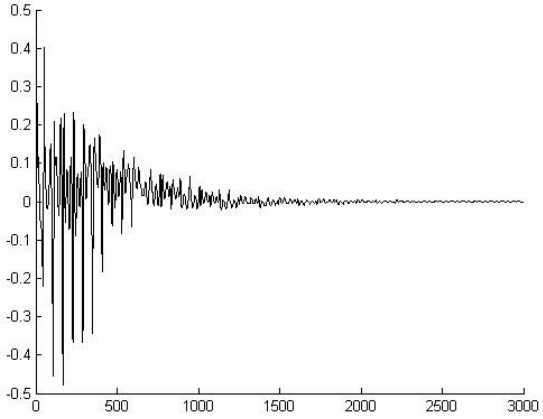


Fig 5e. mu=0.040

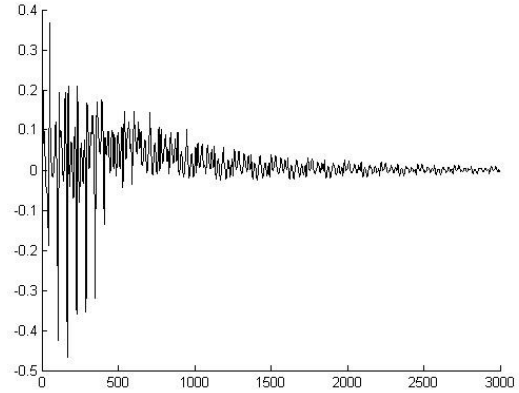


Fig 5f. mu=0.050

Another thing that has been used here is the usage of the previous history of the signal in addition to the present state of the signal, which is actually quite useful in the case of day-to-day systems. We also observe that by changing the size of the coefficient matrix, we can achieve optimization in the best possible time scale. The results for different sizes of the coefficient matrix (***p***) are given below.

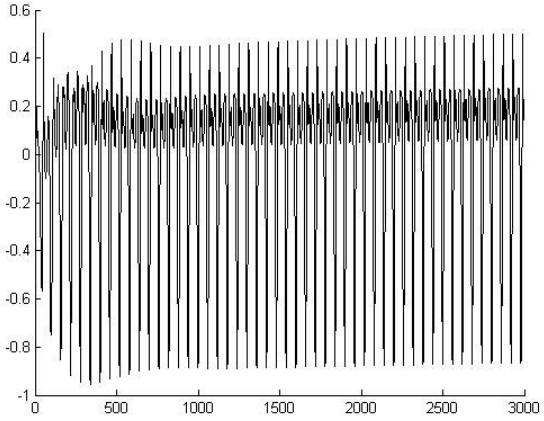


Fig 6a. p=20

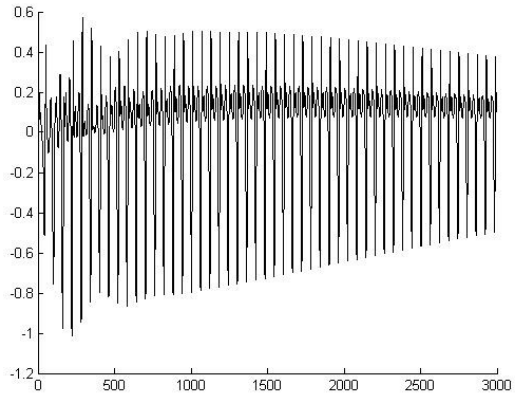


Fig 6b. p=25

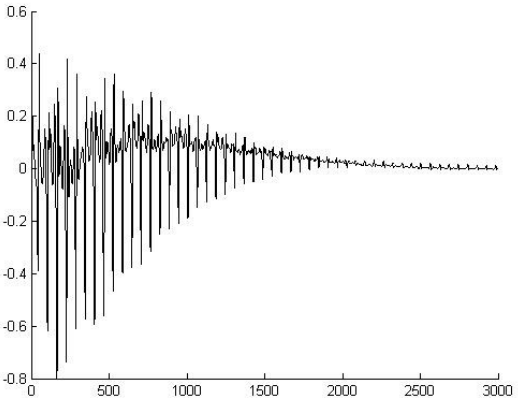


Fig 6c. p=30

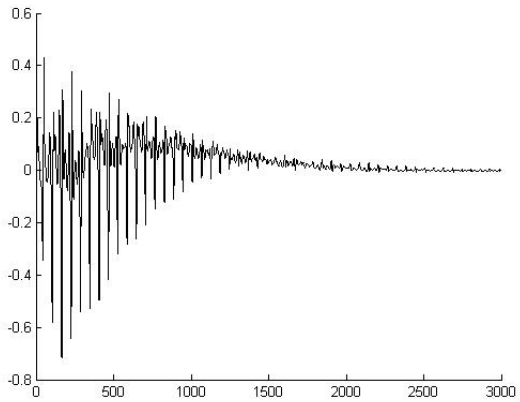


Fig 6d. p=31

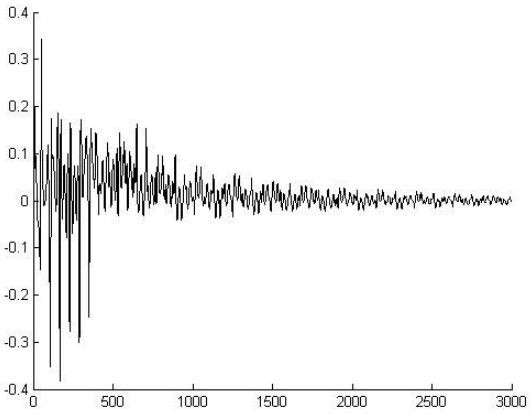


Fig 6e. p=35

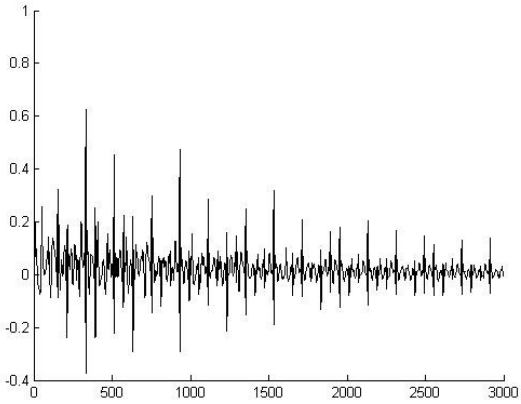


Fig 6f. p=40

Here we observe that the error is minimized in the lowest time with ***p***=31. And as we keep on still increasing the size of ***p***, the system starts to become unstable. So, using this **LMS** algorithm, we can deduce signals and perform active noise control quite effectively, but it has many shortcomings, which needed to be dealt with. Some of them have been discussed here.

Firstly the most important thing that comes into the picture is the effect of the secondary path (through the ANC system). Now the computer computes the signal and produces the signal electrically which is then converted to sound through a loudspeaker and mixes with the original sound in an acoustic medium, rather than an electronic medium. So, the effects that the speaker and the secondary path (D/A converter, power amplifier etc…) create or add to the originally calculated signal can create a lot of problems. Thus it is necessary to minimize the effects of the secondary path.

Also in the above algorithm, one has to simultaneously model ***P*(*z*)** as well as the inverse of the secondary path function, say ***S*(*z*)** so as to find the optimal transfer function ***W*(*z*)**. So, a sufficiently large order **FIR ?** filter, would require the approximation of a rational function and it would be impossible to compensate for the inherent delay without a similar delay in the primary path. Thus the solution to this problem is the addition of an equalizer in the secondary path which would be an approximate model of ***S*(*z*)** and would have been estimated before, let us call it $\hat{S}(z)$. This new type of algorithm is called as the **Filtered-X Least Mean Square** algorithm or the **FXLMS** algorithm.

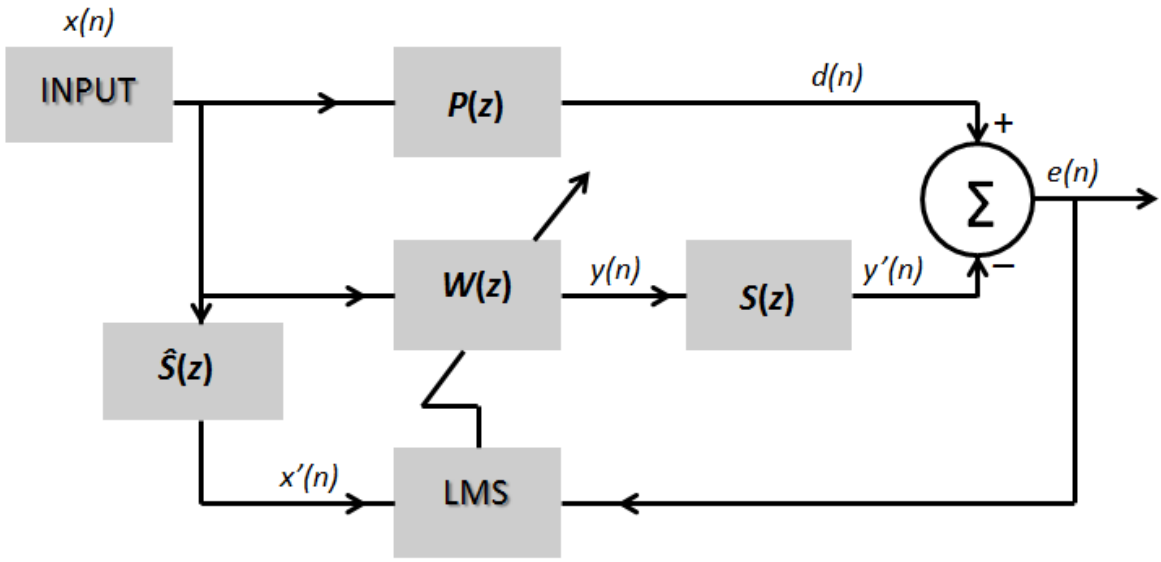


Fig 7 FXLMS Block Diagram

This precisely shows what this FXLMS algorithm is all about. The only difference that we mark here is the introduction of a filter $\hat{S}(z)$ and hence the name filtered-x LMS. In this, we first estimate the secondary path changing system ***S*(*z*)**. Then we keep on updating both the estimate and our main LMS based algorithm program and hence give out desired output ***d*(*n*)**, that is when the error becomes negligible.

The Matlab program for FXLMS algorithm has been mentioned under, along with the explanations of why each step was coded.

```
% Setting the simulation duration (normalized)
clear
T=1000;
p=10
```

```
% We do not know P(z) and S(z) in reality. So we put to make dummy
paths
Pw=[0.01 0.25 0.5 1 0.5 0.25 0.01];
Sw=Pw*0.5;

% The first task is to estimate S(z). So, we can generate a
% white noise signal,
x_iden=randn(1,T);

% send it to the actuator, and measure it at the sensor position,
y_iden=filter(Sw, 1, x_iden);

% Then, start the identification process
Shx=zeros(1,p);     % the state of Sh(z)
Shw=zeros(1,p);     % the weight of Sh(z)
e_iden=zeros(1,T);   % data buffer for the identification error

% and apply least mean square algorithm
mu=0.1;                          % learning rate
for k=1:T,                       % discrete time k
    Shx=[x_iden(k) Shx(1:p-1)];  % update the state
    Shy=sum(Shx.*Shw);           % calculate output of Sh(z)
    e_iden(k)=y_iden(k)-Shy;     % calculate error
    Shw=Shw+mu*e_iden(k)*Shx;    % adjust the weight
end

% Checking the result 1
subplot(2,1,1)
plot([1:T], e_iden)
ylabel('Amplitude');
xlabel('Discrete time k');
legend('Identification error');
subplot(2,1,2)
stem(Sw)
hold on
stem(Shw, 'r*')
ylabel('Amplitude');
xlabel('Numbering of filter tap');
legend('Coefficients of S(z)', 'Coefficients of Sh(z)')


% The second task is the active control itself. Again, we need to
simulate the actual condition. In practice, it should be an
iterative process of 'measure', 'control', and 'adjust'; sample by
sample. Now, let's generate the noise:
X=randn(1,T);

% and measure the arriving noise at the sensor position,
Yd=filter(Pw, 1, X);

% Initiate the system,
Wx=zeros(1,p);       % the state of W(z)
Ww=zeros(1,p);       % the weight of W(z)
Sx=zeros(size(Sw));   % the dummy state for the secondary path
```

```
e_cont=zeros(1,T);     % data buffer for the control error
Xhx=zeros(1,p);       % the state of the filtered x(k)

% and apply the FxLMS algorithm
mu=0.1;                                % learning rate
for k=1:T,                             % discrete time k
    Wx=[X(k) Wx(1:p-1)];               % update the controller state
    Wy=sum(Wx.*Ww);                    % calculate the controller output
    Sx=[Wy Sx(1:length(Sx)-1)];        % propagate to secondary path
    e_cont(k)=Yd(k)-sum(Sx.*Sw);       % measure the residue
    Shx=[X(k) Shx(1:p-1)];             % update the state of Sh(z)
    Xhx=[sum(Shx.*Shw) Xhx(1:p-1)];    % calculate the filtered x(k)
    Ww=Ww+mu*e_cont(k)*Xhx;            % adjust the controller weight
end

% Checking the result 2
figure
subplot(2,1,1)
plot([1:T], e_cont)
ylabel('Amplitude');
xlabel('Discrete time k');
legend('Noise residue')
subplot(2,1,2)
plot([1:T], Yd)
hold on
plot([1:T], Yd-e_cont, 'r:')
ylabel('Amplitude');
xlabel('Discrete time k');
legend('Noise signal', 'Control signal')
```

The graphs that were plotted using this set of data conditions are given below for reference.

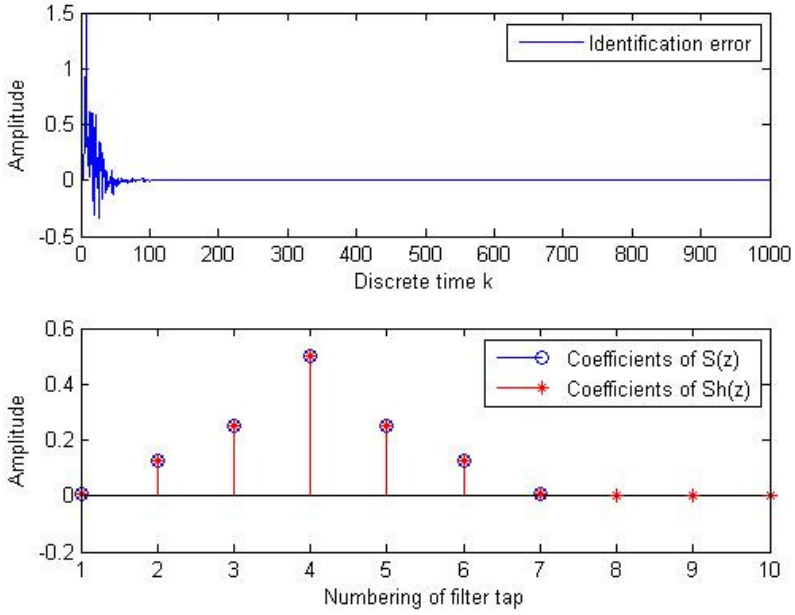


Fig 8 FXLMS Program Graph 1

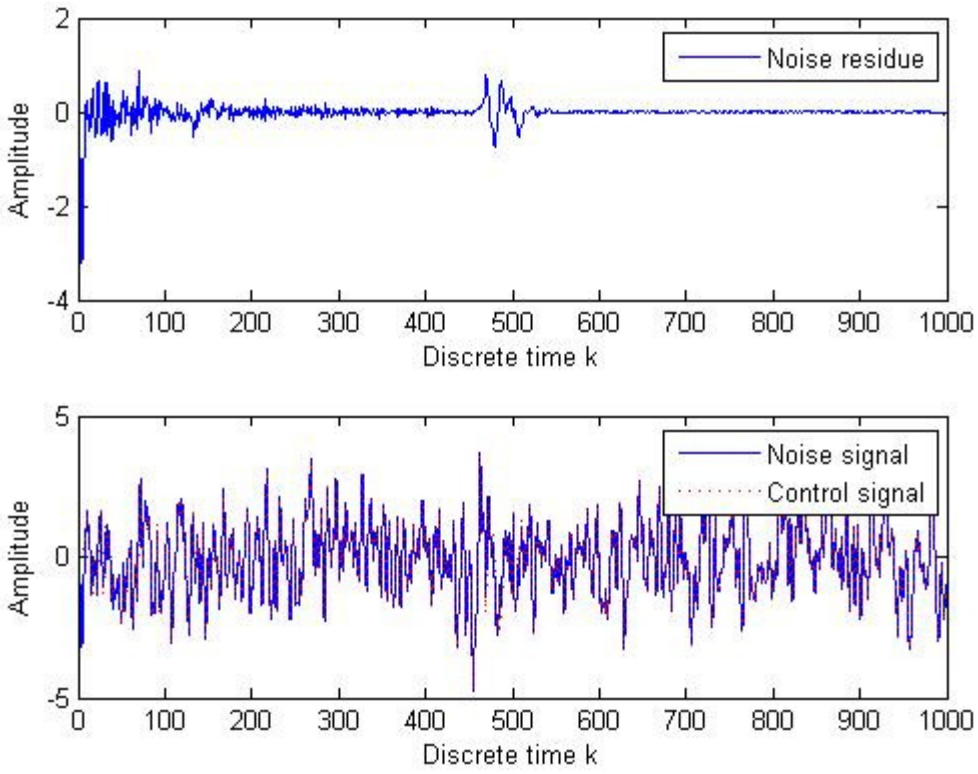


Fig 9 FXLMS Program Graph 2

This program will give us an insight into how active noise control can be achieved using the FXLMS algorithm, which was our main idea. We can alter various things in the programs like iterations, the coefficient matrix size etc… and view the changes similar to those which were observed while using the LMS algorithm.

Now after having understood the simulation of FXLMS algorithm based noise control our next aim is to perform a real-time monitoring and testing of this. The photographs of the setup has been given below; the source code used for this is similar to the one which we have discussed above.

Fig 10 Collection of various photos of the setup

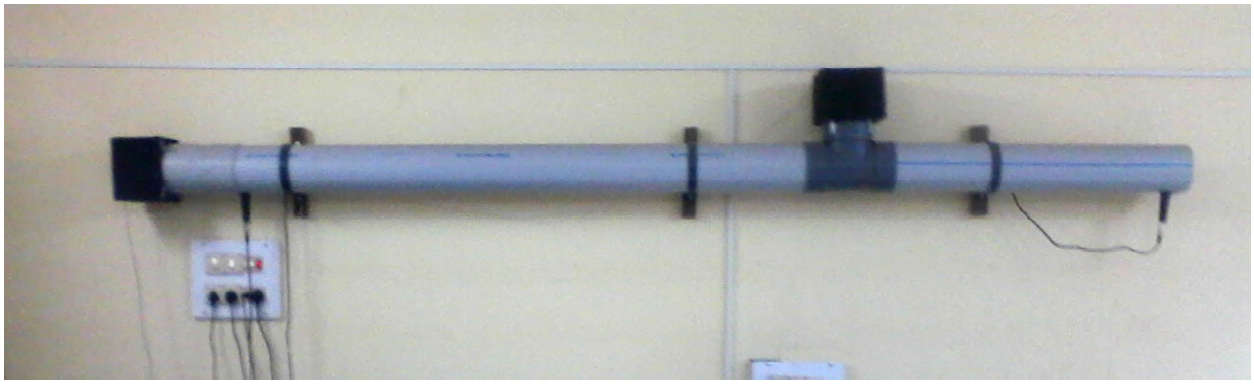


Fig 11 The Complete Sound duct

There was Simulink and dSpace pre-loaded in the computer where this experiment was performed.

What we did was that we produced a noise of 300Hz frequency and then did some simple experiments in real-time, like secondary path identification in open and closed ducts, the noise control in them etc… The results that we obtained are produced as graphs below:

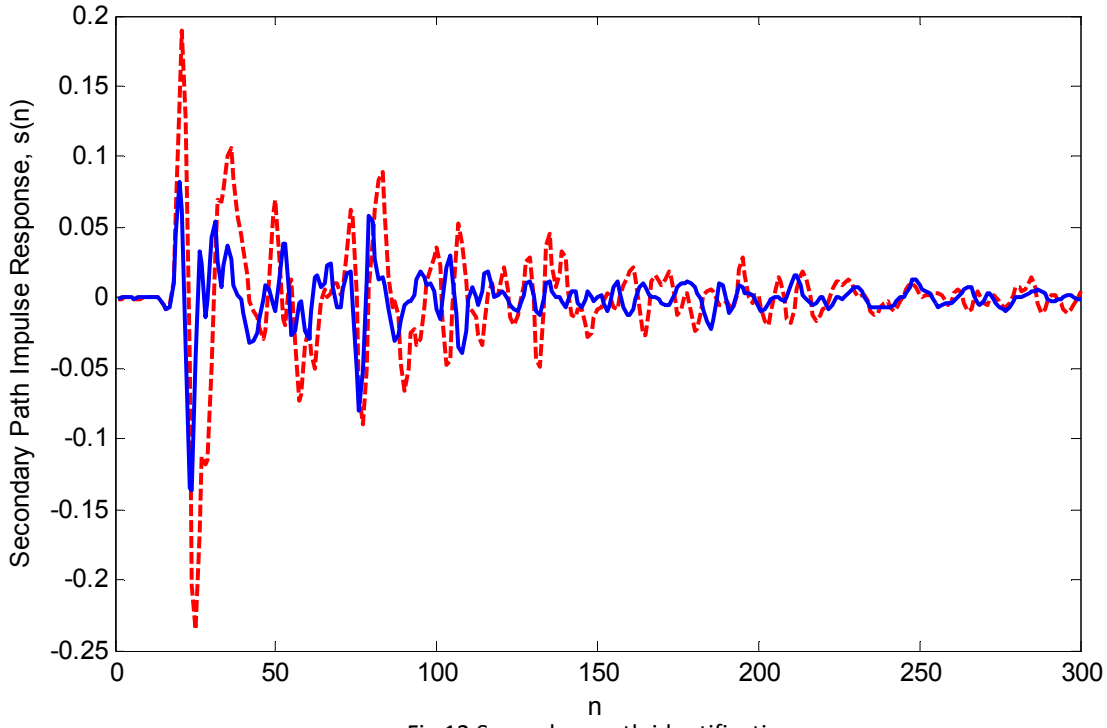


Fig 12 Secondary path identification

The Secondary path identification results have been shown above. The solid blue line represents the identification in open duct and the dashed red line represents that of a closed duct acoustic system under similar conditions and sources. We can mark here that the amplitude with the same source in

open system is less than that of closed system but, the delay period is same in the starting for both the cases.

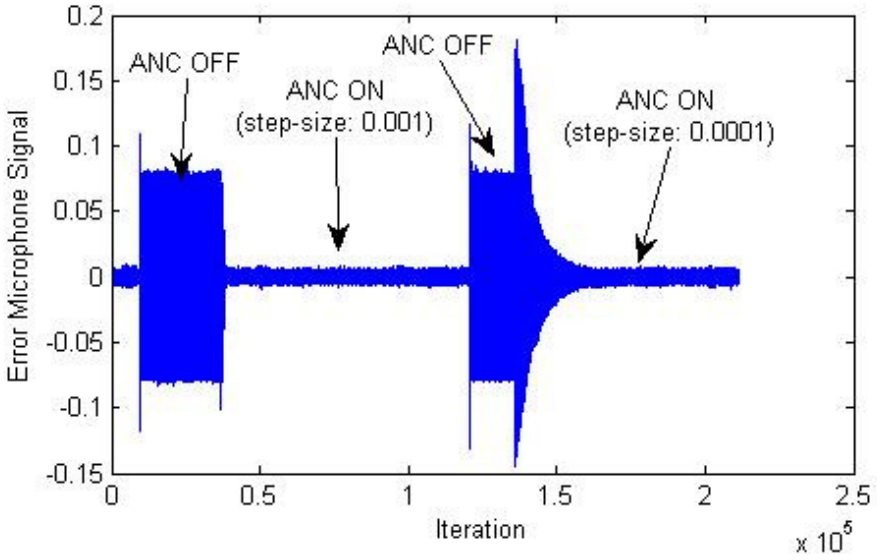


Fig 13 e(n) Signal

Figure 13 shows an interesting practical thing, the same thing in more detail than that we had observed in the simulation using various step sizes in LMS. The time that is required to reduce the noise with a lower step size was greater than that with a larger step size but as observed the residual noise was higher albeit.

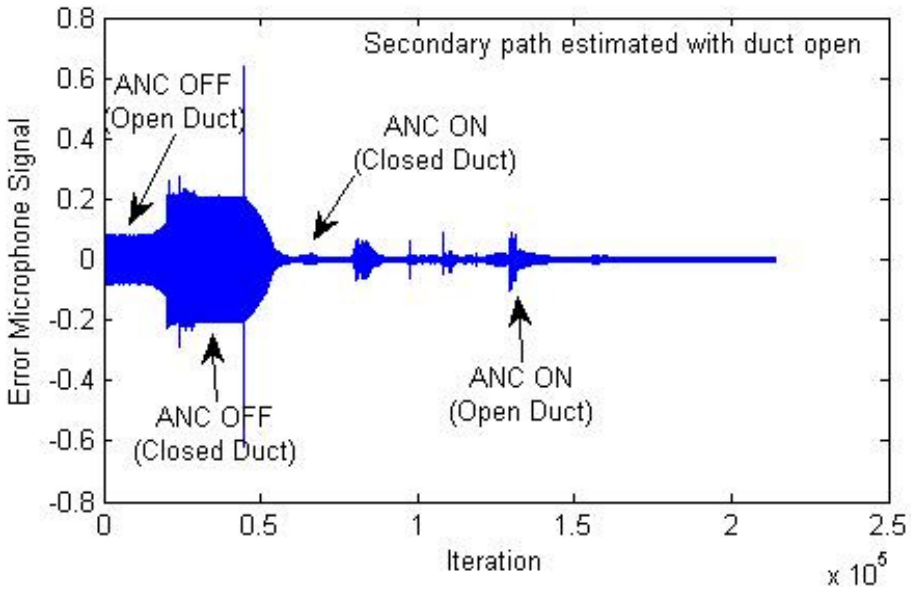


Fig 14 e(n) Signal with Secondary path estimated using open duct

Figure 14 shows the importance of secondary path estimate and also the relative loudness of closed duct as compare to open duct system with same presets. When we estimate the secondary path using an open duct and then use the same filter to have active noise control in a closed duct system, we observe that the filter is not able to remove the noise to a very good extent. These is perceived high residual noise everywhere and when the system is made open again, the ANC system works perfectly fine. This shows us the importance of the secondary path estimate for an ANC to work perfectly.

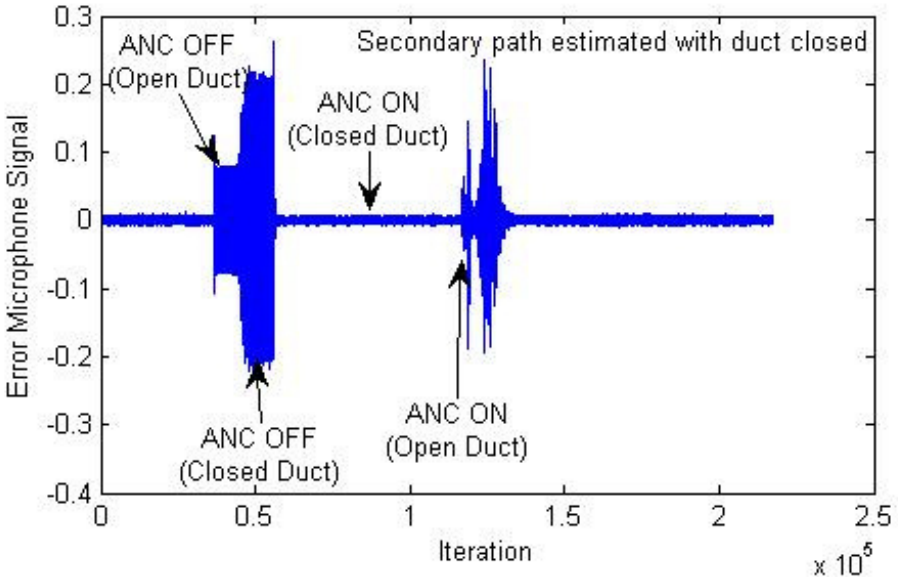


Fig 15 e(n) Signal with Secondary path estimated using closed duct

Figure 15 shows us the same thing as the previous figure but the only difference this time is that the path was estimated using closed duct. Even though the noise amplitude was higher in closed duct system without the ANC system. After the secondary path was estimated for it, the ANC worked in an highly efficient manner to reduce the noise but when the same filter was used for a lower amplitude open duct noise, it was unable to completely nullify the signal and had higher residual noise. Thus, the secondary path estimate for a system is unique and highly necessary.

Conclusion

From the set of experiments that we did we observe the various things that are necessary for the construction of an Active Noise Control system. The knowledge of the optimization technique and the use of coding using Matlab, can help us a lot in this process. This project was basically aimed at achieving the basing understanding of active noise control, the optimization techniques, the LMS algorithm (and its applications) and the FxLMS algorithm. Finally the observation of real time active noise control was observed using a system having dSpace and Simulink.

References

1. Sen M Kuo and Dennis S Morgan, '*Active Noise Control: A Tutorial Review*', Proceedings of the IEEE, Vol. 87, No. 6, June 1999, Pgs 943-973
2. Wikipedia.org, '*Active Noise Control*'. Retrieved 11-07-2013
3. Wikipedia.org, '*FxLMS Algorithm*'. Retrieved 11-07-2013
4. L. L. Beranek and I. L. Ver, *Noise and Vibration Control Engineering: Principles and Applications*. New York: Wiley,1992.
5. Simon Haykin and Thomas Kailath, *Adaptive Filter Theory*, Pearson Education, 2003
6. D P Das, S R Mohapatra, A Routray, & T K Basu. '*Filtered-SLMS algorithm for multi-channel active noise control of nonlinear noise processes'*. IEEE Transactions on Audio, Speech and Language Processing (2006), 14(3), 1875–1880.
7. D P Das & G Panda. '*Active mitigation of nonlinear noise processes using a novel filtered-S LMS algorithm*'. IEEE Transactions on Audio, Speech and Language Processing (2004), 12(3), 313–322.