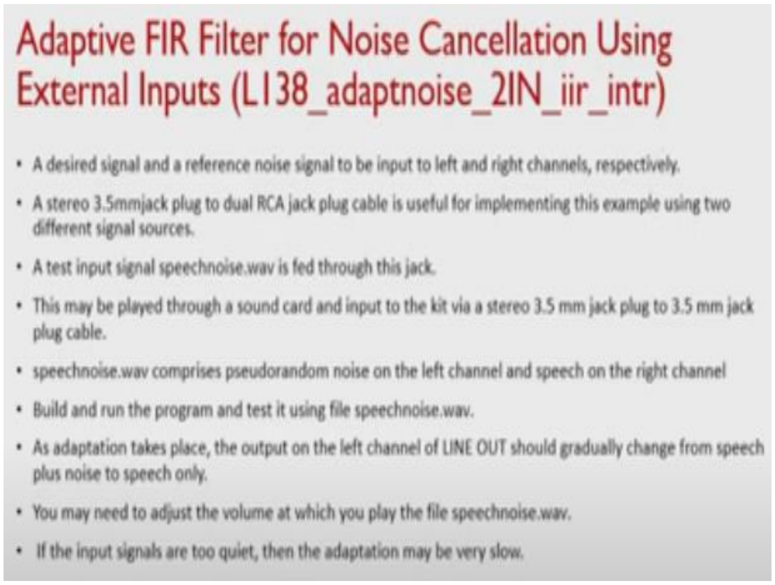
Real-Time Digital Signal Processing

Prof. Rathna G N
Department of Electrical Engineering
Indian Institute of Science, Bangalore

Module No # 11
Lecture No # 51
Adaptive Filters in CCS

Welcome back to the real-time digital signal processing lab actually, in the last class we discussed adaptive filters in MatLab. Today we will discuss adaptive filters in code composer studio, so we will see that.

(Refer Slide Time: 00:45)



Adaptive FIR Filter for Noise Cancellation Using External Inputs (L138_adaptnoise_2IN_iir_intr)

- A desired signal and a reference noise signal to be input to left and right channels, respectively.
- A stereo 3.5mmjack plug to dual RCA jack plug cable is useful for implementing this example using two different signal sources.
- A test input signal speechnoise.wav is fed through this jack.
- This may be played through a sound card and input to the kit via a stereo 3.5 mm jack plug to 3.5 mm jack plug cable.
- speechnoise.wav comprises pseudorandom noise on the left channel and speech on the right channel
- Build and run the program and test it using file speechnoise.wav.
- As adaptation takes place, the output on the left channel of LINE OUT should gradually change from speech plus noise to speech only.
- You may need to adjust the volume at which you play the file speechnoise.wav.
- If the input signals are too quiet, then the adaptation may be very slow.

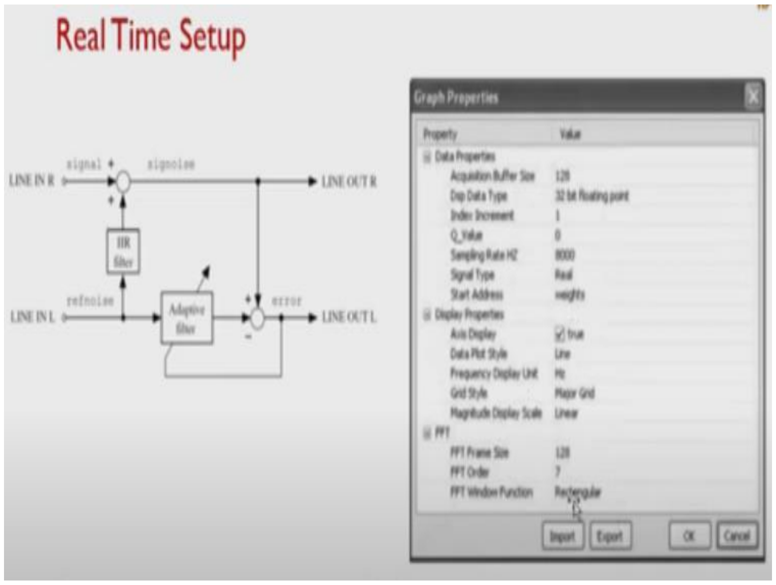
One of the things that I will do we have one of the experiments that is FIR filter noise cancellation using external inputs. This is from the book real time digital signal processing by Ronald Ray, What I have taken is the thing, which has the adaptive noise, which is a 2 in IIR-interrupt driven, so that is the left gets the noise and then the right channel gets the signal. So, using the IIR-filter, the interrupt driven, what it is the demo is because the adaptive value for this is very low.

So you will be hearing most of the type of noise, and voice is going to be very little heard in the thing because it has taken little time, so I thought I would demo this experiment first and then go back to other experiments. So what is it in this case? A desired signal and a reference noise signal to the input to the left and right channels, respectively what we are going to put it in. And a stereo 3.5 mm jack is what we will be using for inputting from the laptop to the board.

This is the plug-in dual RCA jack plug cable will be used in this case, and the test input signal speech noise dot wav is fed through this jack. And this may be played through a sound card and input into the kit via a stereo 3.5 m jack. So in our case, we are using my music basically, and this is the speech noise what will be feeding into the board, build, and we will be running this program and testing it using this wav file.

So this adaptation will be taking place the output on the left channel of line out should gradually change from speech plus noise to speech only. And it has to be adjusted with the volume at which you play the file speech noise dot wav. If the input signals are too quiet, then the adaptation may be very slow. That is what the thing, so one has to look into these things.

(Refer Slide Time: 03:14)



Real Time Setup

Graph Properties

Property	Value
Data Properties	
Acquisition Buffer Size	128
Dsp Data Type	32 bit floating point
Index Increment	1
Q_Value	0
Sampling Rate HZ	8000
Signal Type	Real
Start Address	weights
Display Properties	
Axis Display	✓ true
Data Plot Style	Line
Frequency Display Unit	Hz
Grid Style	Major Grid
Magnitude Display Scale	Linear
FFT	
FFT Frame Size	128
FFT Order	7
FFT Window Function	Rectangular

Going to the next slide we will see that this is the adaptation thing what it is a real-time setup. So line in gets the right, that is the signal, and line in left jack gets the reference noise, and you are using the IIR filter here, whose weights are going to be controlled using the adaptive filter. So you will have the signal plus noise what is going to come out as line out R and the error signal which is coming out of the left channel will be output.

And this is how you can look at what I will say is weights, basically how they are getting adjusted. So from 0, how they will be going up we will look at the thing.

(Video Starts: 04:12)

So coming to our code composer studio so this is the file that we have that is adaptive in IIR. So what we have is this uses a bilinear dot coefficient. So if I open the coefficient think so you will see then that poles and zeros have been taken in this fashion. This is a third-order filter. What it has been used, using in Mat Lab and the cutoff frequency it uses is a Chebyshev low pass filter with 2b pass band ripple and a cutoff frequency of 1500 hertz.

So when you design in Matlab, you will get the filters that are type 1. As I was discussing in the IIR filter class that Chebyshev 1 will have a ripple in the pass band and then it has steep cutoff in the stop band. So the ripple what in the pass band is used for 2 DB basically, and these are the coefficients taken from the matlab, and then it will be used in our real time implementation. So you are seeing that this has been loaded and then this thing beta, whatever this is the learning rate.

In this case, it can be alpha beta 1. Each one uses their own notation, so what it is a very slow adaptation. It has been taken to $1e^{-12}$ so and then the length of the number of adaptive filter weights what it is chosen is 128. And then you will be making variable weights and then your N and then the number of sections. What you will be taking it with 2 basically, because you need both a, and b coefficients.

So then you will be what you are going to get is reference noise you will be getting from the left channel. And then the signal is going to be getting from the right channel, so that is how it has been recorded and then stored in the thing. So we are going to have these 2 mixed, which is coming into our board. And then this is where you will be inputting reference noise, and then you are doing the filtering here basically.

You are calculating your $y(n)$ output and you are updating your coefficients $w(n)$ and then you are putting input $y(n)$, so this is signal noise has both yn plus signal. So then you are making it $y(n) = 0$, and then you have the reference noise here. And then you will be computing your error, basically signal noise- $y(n)$. So you are trying to reduce this error, and then you are calculating your weights $n + 1$ weights basically here using this equation. This is your μ adaptation parameter, which is named here as beta, and you will be updating your $x(i)$ with the previous values μ_1 will be coming from the input signal.

And output what is going on is that both the left and right channels have been put with error. It has to be reduced and then slowed down. So because if I rerunning the code again it; will have a lot of noise in the thing what I will do is because it has already been run for some time. So if you want, you can change your beta value and then look at it. So I had stopped the thing. I will be playing it and you will be hearing little voice and there will be a little error.

It will take a longer time to adapt, so you will still hear the voice. It will take a longer time to adapt so still you are still hearing the voice in between the noise in this case. As you will see that removed the thing from running basically so from the code composes studio still the input is going into board and then it is running. Only when I reset the board, you will not hear anything. So this is the first experiment because it has lot of noise initially.

So now a little bit of speech is coming, it takes a longer time. So, what you can do with the e r codes, basically you can play around with a little more on e r adaptation by changing the thing. So one of the things that I can do is I can make it 0.001 in this case. Let us see what the thing is that is going to happen. I can store it and then I have to do recompilation again. So it is telling that whether I have to stop the debugging session.

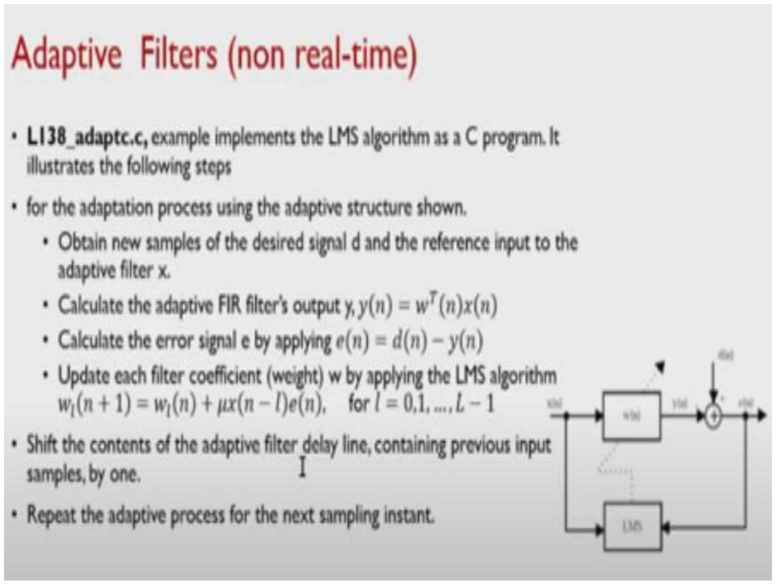
As you can see, here in the diagram at the bottom, what you are seeing is the coefficients from some value. What it is getting adjusted to, so I will restart the thing. I will be stopping it and then picking it up. There was a little error in the thing, so what I will do is stop the thing. Actually, the memory was not being cleared. So let us see if it is, some memory block that has a reading error. Let us see whether it is going to run.

I will be playing the thing in these cases. You may have to restart your board by unplugging it and plugging it. So all the software resets what it has, which is there. I have given the reset command still it is not working in that what we do is unplug and then plug in the board. So that the complete memory is going to be erased, so this is one of the things one has to do. So let me see whether it is going to have any effect because I have re-plugged my power point, so even if this occurs, one has to take care that no other devices go, you have to reboot the system for the timing. Let me see whether I can figure it out without doing that with different experiments whether I can come up with the thing. So I can close this project and then restart the thing.

(Video Ends: 12:56)

So the other experiment that I will do is the first experiment will look at it, which does not need real time.

(Refer Slide Time: 13:07)



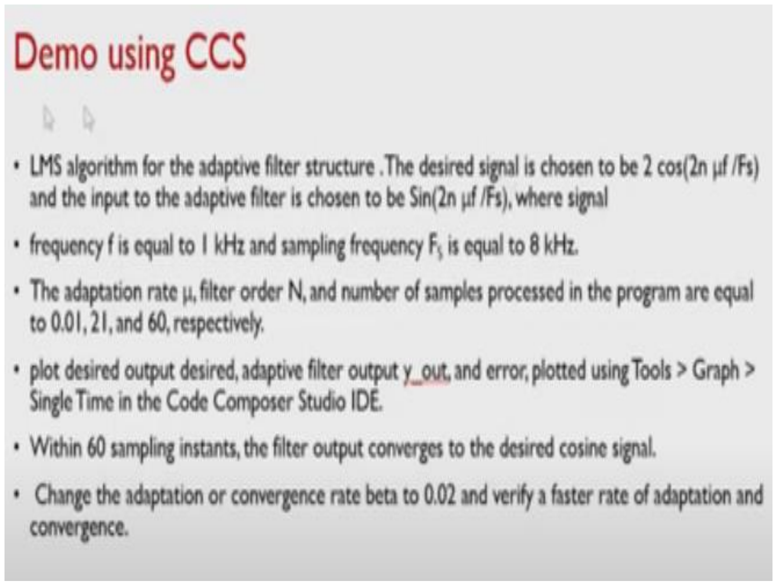
Adaptive Filters (non real-time)

- **L138_adaptc.c,** example implements the LMS algorithm as a C program. It illustrates the following steps
- for the adaptation process using the adaptive structure shown.
  - Obtain new samples of the desired signal d and the reference input to the adaptive filter x.
  - Calculate the adaptive FIR filter's output y, $y(n) = w^T(n)x(n)$
  - Calculate the error signal e by applying $e(n) = d(n) - y(n)$
  - Update each filter coefficient (weight) w by applying the LMS algorithm $w_l(n+1) = w_l(n) + \mu x(n-l)e(n)$, for $l = 0,1,\ldots,L-1$
- Shift the contents of the adaptive filter delay line, containing previous input samples, by one.
- Repeat the adaptive process for the next sampling instant.

This is non-real time what we will see it so what we have is the adaptive filter LMS algorithm as a C program, so it illustrates the following steps: what is it? For the adaptation process using the Adaptec structure as shown here? This is our weight normal adaptive filter, so I have the LMS algorithm in this case. As a result, my weights began with 0 $y(n)$. What I will be getting is the output, and then this is the desired signal.

What I have given is the thing, and then there will be a subtracting error. We are trying to minimize this being fed to our LMS algorithm to adjust our weights. So this is the equation that will be doing it. $y(n)$ is nothing but $w^T(n)x(n)$ and then $e(n)$ is nothing but $d(n-y)$(n) and our weight update $w_l(n+1)$ is given by this. μ is the adaptive step one has to follow, so $l = 0,1,\cdots,L-1$. So by one, repeat the adaptive process for the next sampling rate what we are going to do the thing.

(Refer Slide Time: 14:24)



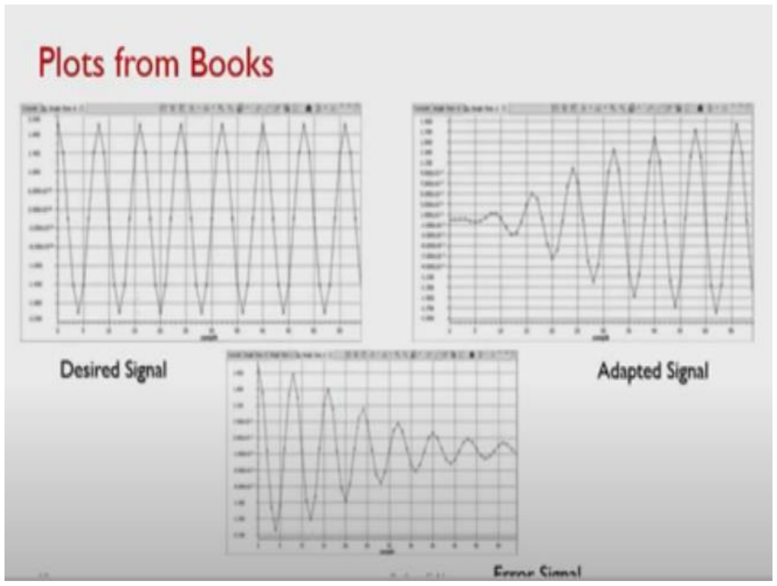
Demo using CCS

- LMS algorithm for the adaptive filter structure .The desired signal is chosen to be 2 cos(2n µf /Fs) and the input to the adaptive filter is chosen to be Sin(2n µf /Fs), where signal
- frequency f is equal to 1 kHz and sampling frequency F_s is equal to 8 kHz.
- The adaptation rate µ, filter order N, and number of samples processed in the program are equal to 0.01, 21, and 60, respectively.
- plot desired output desired, adaptive filter output y_out, and error, plotted using Tools > Graph > Single Time in the Code Composer Studio IDE.
- Within 60 sampling instants, the filter output converges to the desired cosine signal.
- Change the adaptation or convergence rate beta to 0.02 and verify a faster rate of adaptation and convergence.

So, how we are going to show all of you with the simulator how you can run this algorithm. So how we are going to show that the filter structure for the desired signal is chosen to be $2\cos(2n\,\mu f\ /Fs)$ what we have it? This should have been π basically I cannot correct the thing πf is the frequency what you want it and then Fs and the input to the adaptive filter is here also it is $\sin(2n\,\pi f\ /Fs)$. Where this is the signal frequency $f\ =\ 1$ kilohertz has been chosen and sampling frequency $Fs\ =\ 8$ kilohertz.

The adaptation rate mu so that is filter order n you can change it from that is assumed as 0.01 and then 21 is the order of the filter number of samples chosen as 60, so if we 0 to n is this thing, 60 samples. So, if we can plot this desired output, that is, output y output and error using tools, in this case, graph, and then single-time in the code composer studio. So within a 60 sample instance, the filter output converges to the desired cosine signal. So you can change the convergence rate that is 0.02 from 0.01 and verify the fast rate of adaptation and convergence.

(Refer Slide Time: 16:12)



Plots from Books

So, we will see the demo what the book gives these are the blocks. So what we will do is? We will go to this thing called composer studio and what I will do is?

(Video Starts: 16:30)

I will remove it because I wanted to show you how you can use the same thing to run your codes, basically. So I can remove these are files what I have to remove from this I will delete them and then I can add from the code. Basically, I have stored in the d directory. In this case, wherever on your laptop or computer, you can store whatever you want to have the thing. So we will see that these are the LCD cable book files that I have taken the things from, so in this case, this is from chapter 6.

So we will be seeing that this is the first code. So I will copy this c file into my directory, or I can import this project from here also. So one of the things one can do it so we will import it and then put it here I can do the paste and then I will be do first I can do the compilation to see that I have any errors. So there is a little problem in the thing because this is on a simulator of what it is going to run on. So what we have to do is I can import the project.

So what I really want is the CCS project, so I have imported. I will browse through this folder. I have imported it this has become my active this thing directory basically. Lets me see whether I am going to avoid errors in the thing, so you will be seeing the errors what are the things listed, here It is telling me that what it wants is v2 CC XML active, but what it has the thing. So I had to open this. I can create a new one.

What I want is the target configuration file, so I will be calling it a new target configuration, so in this case, what I have is a USB debug probe what I have on the Texas Instrument, we have to check on the thing xts110, what I have on the board USB debug probe, what I have it. So one has to be careful when you are doing it in real time which is the debug port what is it you are connecting to? So here it is what I have is lcdk674 input. So I will save this configuration so those who have the first time they will be doing the lab can discuss how it has to be done.

So now we will see whether by doing this compilation whether my errors are going to go. There are some of the errors that are coming I what I will do is I will fix it in the next class and then come back and tell you what the problems were with it. Because suddenly, what happened was that my active directory crashed and I had to redo the thing. Some of the errors take a little time, so I will fix them and then I will be coming back in the next class. Thank you.

(Video Ends: 21:26)