**RTDSP Final Report**

**Content**

**Introduction**

Basic Algorithm

Noise Removal Basic Principle

Basic Performance Evaluation

**Enhancement evaluation**

Enhancement1

Enhancement2

Enhancement3

Enhancement4

Enhancement5

Enhancement6

Enhancement7

Enhancement8

Enhancement9

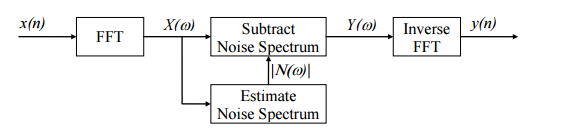
**Additional Enhancement**

**Other Possible Enhancement**

**Reference and Bibliography**

**Introduction**

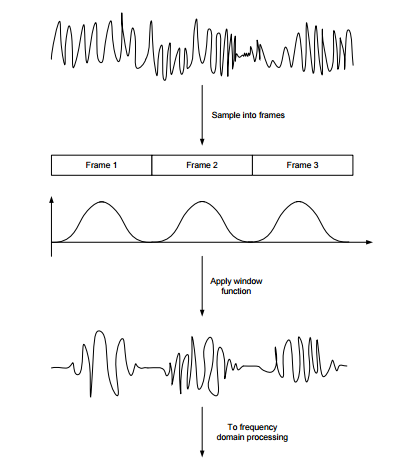
**Basic function**

The role of this program is to remove disturbance from the speech signal while leaving the signal itself intact. Since the speech or the noise is random for us, we can’t simply use a filter to knock out the noise frequencies. The clever way we chose is to do a Fast Fourier Transform (FFT) on our input signal which is received by DSK and then estimating the noise spectrum to be removed from our obtained spectrum (the assumption we made here is that the spectrum of the input signal can be expressed as the sum of the speech spectrum and noise spectrum). After an Inverse Fast Fourier Transform (IFFT) is executed on the resulting spectrum, the output signal will be propagated to the audio port of the DSK as shown on Figure 1.

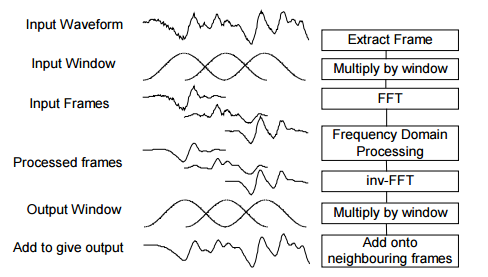
**Figure 1.** Noise subtraction signal flow graph

The original algorithm we used was in which we split our input signal into several non-overlapping frames so that we could fills up our input buffer, do processing in an intermediate buffer and output samples from an output buffer. The hazard with the technique we described above is that discontinuities will be generated at the frame boundary which give rise to special artefacts. We have introduced unwanted frequencies by splitting input signal into frames.

We could use a window function which smoothly reduces the amplitude of the frame to zero at the edges. However, by using this method some information might be erased (Figure 2). In order to avoid spectral artefacts and reserve our original input, we use an idea called overlap add processing.



**Figure 2.** Windowing function on frame

Once we overlap the frames, the envelope of the overlapping windows will always add to 1. We can built up our frame in this way without having spectral artefacts and losing power.

Processed frames may have discontinuities in the time domain which could lead crackles in the output sound, so we prefer to multiply the processed time domain frames by another window.

**Figure 3.** Overlap add algorithm

Since frequency resolution and time resolution are two critical factors for our processing speed, a trade-off based on the above factor giving the length of 256 for our frame (FFT more efficient if the length is power of two).

**Nosie removal basic principle**

We assume that the speaker will stop at least once every 10s which let us have enough time to measure the noise during this period. This is done in detail by measuring the minimum spectra during every 2.5 seconds, we extract four sections at this part (the window we chose is the square root of the Hamming window which has the property that the windows always add to 1 for 4 times frame oversampling) which accumulates to 10s. However, this will only return a minimum noise and thus the spectrum should be enlarged by a factor to estimate the average noise.

Throughout the 2.5 second frame, our spectrum is continuously updated by comparing the minimum among the most recent 256 samples (only absolute value is calculated) against recorded value in buffer, mathematically described as . We shift the buffer for every 2.5 seconds to store the new incoming value. The noise is estimated by the equation:

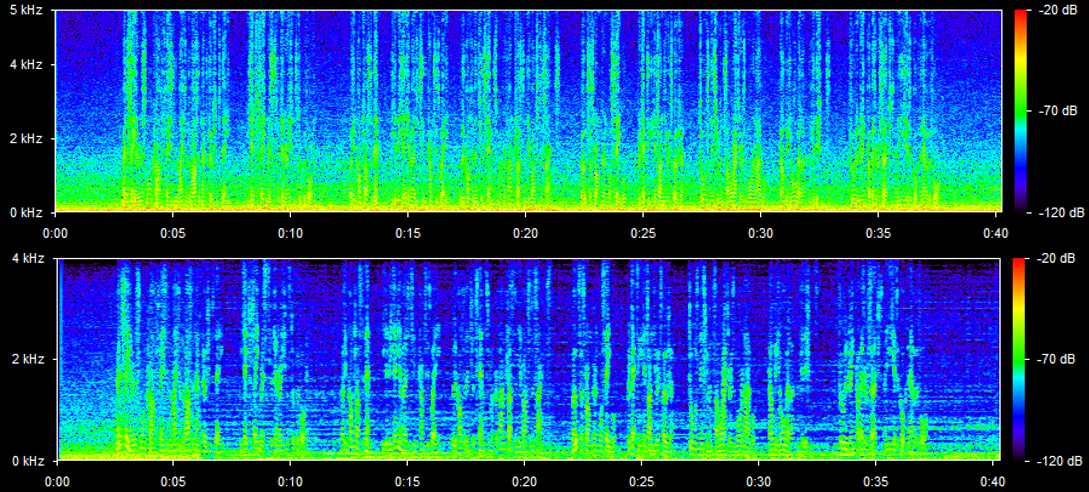
The most recent input signal will be store in buffer.

|  |
| --- |
| **for** (i=0;i<FFTLEN;i++)  {    N\_i[i] = M1[i];  **if**(N\_i[i]>M2[i]){N\_i[i]=M2[i];}  **if**(N\_i[i]>M3[i]){N\_i[i]=M3[i];}  **if**(N\_i[i]>M4[i]){N\_i[i]=M4[i];}  } |

Once we know the noise exactly, we can subtract the noise (after FFT processing) from the original signal (). However, the phase of our noise signal is still deficient, we can only reserve the minimum magnitude value which means that we need to use the absolute value in our calculation ().We can achieve this by rearrange our equation:

Where G (k) is known as frequency-dependent gain factor.

When is greater than, G (k) can sometimes become negative. In order to resume whole function working properly, we prefer to insert a parameter ( is around 0.1) which limit the boundary (G (k)>0) by using function .

**Basic Performance Evaluation**

**Figure 3.** Spectrogram before (top) and after (bottom) noise subtraction

Figure 3 shows the effect of basic noise subtraction algorithm. Noise at low frequency (yellow region close to 0 Hz) and middle frequency (green region between 0 to 2 Hz) is significantly attenuated. The frequency components which exists alone all the time contributes to musical noise.

**Enhancement evaluation**

**Enhancement 1**

In the original design without enhancement, buffer M1 to M4 always store the lowest value among past 2.5 seconds intervals. This algorithm requires a sufficiently large alpha for all frequency bins to perform effective noise subtraction. Typically, alpha is selected between 15 and 20. However, by using a low-pass filter, which averaging over past samples, alpha value required can be reduced to around 2.

The implemented low-pass filter is expressed as:, where . This filter takes average over the previous and current frequency components and prevents any sudden change in magnitude. For example, if input signal become zero at some point during transmission due to unexpected reason, original algorithm will store zero, which is the lowest possible value in reality. Hence, noise estimation of that frequency component will remain zero for the next 10 seconds and no noise subtraction is performed. While the low-pass filter prohibits the buffer stores zero directly and saves an averaged value instead.

|  |
| --- |
| lpf\_weight\_speech =exp(-TFRAME/tau1);  mag=cabs(buffer[k]);  P[k]=(1-lpf\_weight\_speech)\*mag+lpf\_weight\_speech\*P[k];  **if**(P[k]<M1[k]){  M1[k]=P[k];  } |

The moving average filter implemented above has the time constant of 10 ms, and coefficient ,which is within the general range of filter coefficients [1]. Intuitively, this coefficient determines how fast buffers react to current value. This can also be considered as a trade-off between speech and noise estimation since smaller cut-off frequency means larger amount of current input will be treated as noise and saved for future estimation.

Result of the first enhancement is significant that a suitable value of alpha now is around 2 and noise is significantly reduced.

**Enhancement 2**

The second enhancement is pretty much similar to enhancement 1. Rather than perform the power pass on magnitude domain, it is now performed in the power domain, with mathematical expression:

Compared the first enhancement, power-domain filtering is more sensitive to input change after squaring it since amplitude difference is enlarged. Also, human ear is a complex system and intuitively, loudness can be approximated by sound power. So filtering in power domain may be closer to what we actually hear.

|  |
| --- |
| M1[k]=(1-K)\*cabs(cmul(buffer[k],buffer[k]))+K\*M1[k]; |

In our listening tests, performance is improved slightly as they are following the same principles in general. However, musical noise is more significant since musical noise components have large amplitude in the beginning and squaring even makes it larger. This function includes expensive functions in the loop like *sqrt(), cabs(),* and *cmul()*, which easily overload CPU.

**Enhancement 3**

The working principle of this part is similar to the previous two. Noise buffer always saves the smallest value among M1 to M4. Every rotation of M buffers results in a possible sudden change in noise buffer since there is a buffer dropped out and a new buffer coming in. If noise level is varying quickly, this sudden change every 2.5 seconds is noticeable. Thus, a low-pass filter can be used to smooth out this change.

|  |
| --- |
| lpf\_weight\_noise=exp(-TFRAME/tau2);  N[i]=alpha\*(1-lpf\_weight\_noise)\*N\_i[i]+lpf\_weight\_noise\*N[i]; |

In implementation, the above code follows after selecting N\_i[i] as the smallest among four buffers. Here time constant is chosen to be 80 ms so that *lpf\_weight\_noise*  is approximately 0.9. Past value weighs more than present input. Similarly, this is a trade-off between the weights of current and past input. Typical value of *lpf\_weight\_noise*  is between 0.5 and 0.9[2]. We expect past noise estimator to weigh more than the current since noise estimation over long time provides better result.

The effect of smoothing out signal is observed in this enhancement. Although the occasion that noise varies quickly is not often seen, this enhancement does not cost much and can be easily implemented in DSP.

**Enhancement 4**

In our previous enhancements, gain factor . is the spectrum floor, with a value from 0.01 to 0.1. It is used to prevent gain from becoming negative. Also, increasing reduces the musical noise, at the cost of increasing background noise. This is because human ear is more sensitive to musical noise, which is essentially a single unremoved peak in noise spectrum. While white noise is often ignored by ear.

In this enhancement, many different ways to implement gain factor. The averaged input signal is also in the list. It estimates the total input spectrum, including both noise and signal by taking the moving average. It prevents the sudden change in input signal and smooth out the gain factor. However, to some extent, using moving average is not necessary since what we are looking for is an instant response to remove the estimated noise signal. For example, if SNR of the incoming signal suddenly increase, while gain factor is estimated as, in which past input signal is used to estimate current input. The resultant gain factor will be larger than theoretical value since the expression does not respond fast enough.

Regarding the spectrum floor, a varying spectrum floor depending on input frame or noise estimation is suggested. However, this improvement not only adds to CPU load, but also shows no obvious effect. The changes in spectrum floor can hardly be noticed by human ear. Therefore, none of the suggestions are used in our final design.

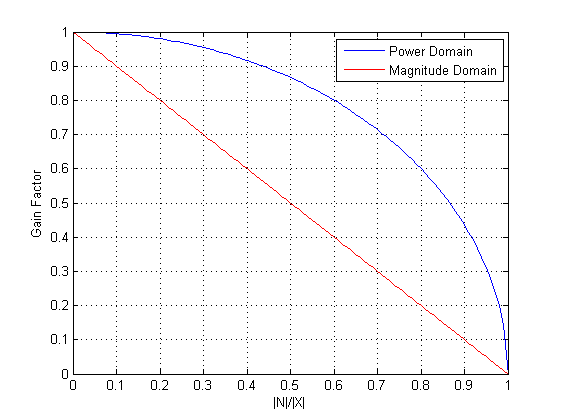
|  |  |  |
| --- | --- | --- |
| **No** | **Gain Factor** | **Observation** |
| **1** |  | Little noticeable change. Speech sound vibrates |
| **2** |  | Little noticeable change. Speech sound vibrates |
| **3** |  | Little noticeable change. |
| **4** |  | Little noticeable change. |

In general, these four methods do not make much contribute to noise subtraction, but increase the CPU load. During the implementation, sometimes compiler optimisation is necessary to avoid CPU overloading. While additional noise is observed after compiler optimisation, which changes the code structure and removed ‘unnecessary’ code determined by compiler.

**Enhancement 5**

|  |
| --- |
| mag=cabs(buffer[k]);  a=lambda;  **#if** lpf\_power\_switch ==1  b=sqrt(1-N[k]\*N[k]/cabs(cmul(buffer[k],buffer[k])));  **#else**  b=1-N[k]/mag;  **#endif** |

Instead of calculating *g* in magnitude domain, it is calculated in power domain in this enhancement, using .

According to our observation, calculating gain factor in power domain does not make noticeable change on sound intelligibility. If we compare the gain factor in power domain and magnitude domain, the following graph is plotted:

**Figure 4.** Comparison of gain factor in power or magnitude domain

The difference of the two domain is nothing but larger gain factor for most input. Intuitively, calculating gain in power domain would agree with the mechanism of human ear. However, in our tests, enhancement 5 only produces a louder output and introduces more musical nodes since noise subtraction is not performed effectively. Therefore, enhancement 5 is not used in our final design.

**Enhancement 6**

This enhancement particularly focuses on the low frequency bins, which normally have a poor signal-to-noise ratio due to the fact that human voice usually does not exist frequency region below 80 Hz. For a frame with length 256 and Nyquist frequency 4 kHz, each frequency bin represents Hz.

|  |
| --- |
| **float** s\_factor;  **for** (i=0;i<FFTLEN;i++)  {  s\_factor=1;  **if**(i<=2){s\_factor=2;}  N[i]=s\_factor\*alpha\*N[i];  } |

In the above code snippet, we perform over-subtraction on the first two bins, resulting attenuating signal below 66.5 Hz.

It is observed that musical noise with low frequency is reduced without affecting sound quality. We then found that there are many other ways of further improvement based on this example, which are explained in detail in Additional Enhancement.

**Enhancement 7**

|  |
| --- |
| **#define** FFTLEN 256  **#define** OVERSAMP 4  **#define** FSAMP 8000.0  **if**(i== 2.5\*FSAMP\*OVERSAMP/FFTLEN) |

After modifying FFTLEN, the condition to rotate buffer needs to be changed correspondingly, while time period used to estimate noise is kept as 2.5 seconds.

Reducing FFTLEN would improve the response speed of program with the cost of losing sound intelligibility. It is observed that short frame length introduces musical noise and speech sounds rough. This is because shorter FFTLEN means more quantisation on FFT signal, which means that for the same width of frequency band less frequency bins are available.

Increasing FFTLEN makes signal sounds slurred. It also increase CPU load since all the loop-statement in this program have multiples of work to finish, which increase the risk of CPU overload. Moreover, sound intelligibility is reduced. Longer frame length is means more quantisation in time domain since each frame now represents a larger time inteval.

As a result, the frame length of 256 samples is actually a good compromise between quantisation level in time and frequency domain.

**Enhancement 8**

Residual noise reduction is suggested in this enhancement [3]. If is greater than some threshold, is chosen to be the minimum calculated in three adjacent frames. This methodology takes advantages of the randomness between frames. The assumption here is that, given a frequency bin, noise residual randomly varies at each frame. It can be attenuated by replace the bin with the minimum value among the adjacent bins.

If signal amplitude is below noise residue and varies fast, the frequency bin is likely to be dominated by noise. Therefore, taking the minimum in adjacent frames would reduce noise. If signal is below noise residue but maintained in a constant level, signal is mainly composed of speech spectrum, choosing the minimum will not attenuate signal much. If signal amplitude is greater than the maximum, it is high likely to be caused by speech. Therefore, noise subtraction is sufficient to obtain a good speech estimation.

|  |
| --- |
| residue\_red1[i]=residue\_red0[i];//delay commands  residue[i]=abs(N\_i[i]-N[i]);//determine current residue  residue\_red0[i]=0;  **if**(residue[i]< cabs(buffer[i])){  residue\_red0[i]=1; //1 is the command to choose minimum among adjacent frames  } |

Many difficulties are encountered in implementing residual noise reduction. This algorithm requires us to store frames and commands until next frame in order to select the minimum among the three adjacent frames. Also, this is a complicated process, which introduces many steps involving complex number calculations. In our implementation, CPU overload is unavoidable and no valuable observations are made. But it is expected that this method will work at the cost of space and time complexity.

**Enhancement 9**

In this enhancement, modifying the time period to estimate noise is suggested.

|  |
| --- |
| **float** time  **if**(i== time\*FSAMP\*OVERSAMP/FFTLEN) |

The code above is the condition to rotate M1 to M4. By modifying *time* while keep *FSAMP*, *OVERSAMP* and *FFTLEN* constant, time period used to estimate noise is changed correspondingly.

In theory, shorter estimation time leads to faster reaction of program before noise subtraction performs effectively. While longer estimation time might produce a more accurate estimation of noise since not all noise spectrum would appear within a 2.5 second period.

Our observation approximately agrees with our prediction. It takes less time to hear noticeable noise subtraction with short estimation time. Since in the noise sample provided, noise spectrum is quite stable, the disadvantage of using short estimation time is not obvious and choosing a reasonable short estimation time shows a similar performance as long estimation time. However, the choice of 2.5 seconds is already a good compromise between noise estimation quality and reaction time.

**Additional Enhancements**

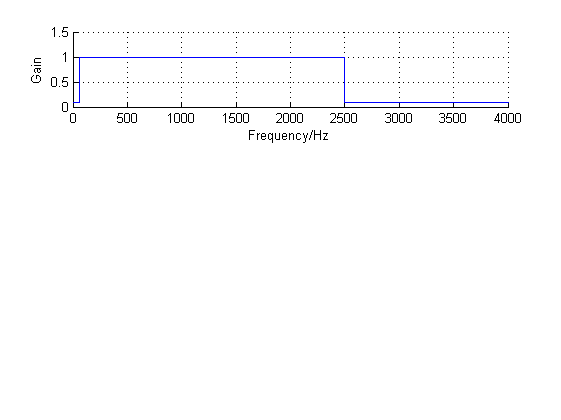
**Further development on over-subtraction**

This idea is one of the further development from enhancement 6. Given the human voice spectrum, it can deducted that noise is dominating in most other frequencies. Regarding our speech sample, the fundamental frequencies of a typical adult male is between 85 Hz and 180 Hz, while human voice is also composed of high harmonics. Therefore, it is reasonable to perform over-subtracting in low-frequency region and attenuate very high frequency region. Parameters need to be chosen carefully to avoid reducing sound intelligibility.

Chosen FFT length is 256, which should represent all frequencies below Nyquist frequency, 4 kHz. Each frequency bin represents Hz in continuous frequency domain. A reasonable low frequency region to perform over-subtraction is the first 2 bins. While in higher frequency region above a threshold, we attenuate signals without removing them.

|  |
| --- |
| **float** s\_factor;  **for** (i=0;i<FFTLEN;i++)  {  s\_factor=1;  **if**(i<=2||(i>=80&&i<=176)||i>=253){s\_factor=10;}  N[i]=s\_factor\*alpha\*N[i];  } |

A sample modification in the for-loop is shown above. By adjusting alpha value with an additional gain factor, a band-pass filter is implemented on spectrum with cut-off frequency at around 80 Hz and 2550 Hz and stop-band gain is half. Approximated frequency response is show below. The higher boundary of 2550 Hz ensures at least 10 harmonics are not attenuated.

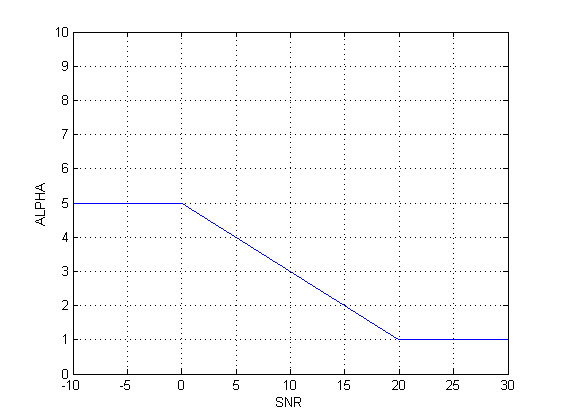


**Figure 5.** Band pass filter

After this enhancement, it is observed that musical noise in high frequency region is significantly reduced. We select 60 Hz and 1250 Hz as pass-band in order to balance voice intelligibility and noise subtraction. The former would be affected if fundamental spectrum of harmonics is over-attenuated. This enhancement is then include in our final design.

**Over-subtraction depending on SNR**

This part focuses on adjusting the alpha value according to different input signals. By over-subtracting the frequency bins with poor signal-to-noise ratio, musical noise artefacts are expected to be reduced. Intuitively, if we apply large alpha value when SNR is low, the spectrum peak which causes musical noise will be significantly reduced, or even attenuated to spectrum floor level.

****

**Figure 6.** Relation between Alpha and SNR

A reasonable relationship between alpha and SNR is suggested by the graph above [4]. Multiple sets of parameter is attempted by us in order to find the best balance between noise cancellation, sound intelligibility and musical noise. Eventually, we decided to use for SNR between -5 dB and 10 dB.

|  |
| --- |
| **float** apporx,mag,SNR;  **int** alpha;  mag=cabs(buffer[i]);  approx=2-mag/N\_i[i];//first order maclaurin series appoximation  SNR= approx\*20;  **if**(SNR>20){SNR=20;}  **if**(SNR<-5){SNR=-5;}  alpha=-0.2\*SNR+5; |

Notice that in order to reduce calculation complexity, first order MacLaurin series is used to approximate logarithm and SNR is set to integer to reduce CPR work load. These approximations turn out to make ignorable difference in performance.

This enhancement turns out to be useful. It keeps the frequency component when SNR is high and attenuates the bins where SNR is low. Such process improves the sound intelligibility and reduces musical noise if parameters are carefully selected. This enhancement is then included in our final design.

**Other possible improvements**

Apart from the enhancements successfully implemented by us, there are many other ways to further improve speech quality. The noise residue theory mentioned in enhancement 8 might be a good approach.

Firstly, low-pass filter used in enhancement 1 and 3 have a fixed time constant. Although we carefully choose the time constant, different time constant will still perform differently on various occasions, depending on frame length and environmental noise. Also, short frame length provides more space for noise spectrum to change faster even in same background noise. How rapidly the environmental noise spectrum has direct impact on requirement of time constant. If possible, a dynamically-changing time constant depending on environmental noise is preferred.

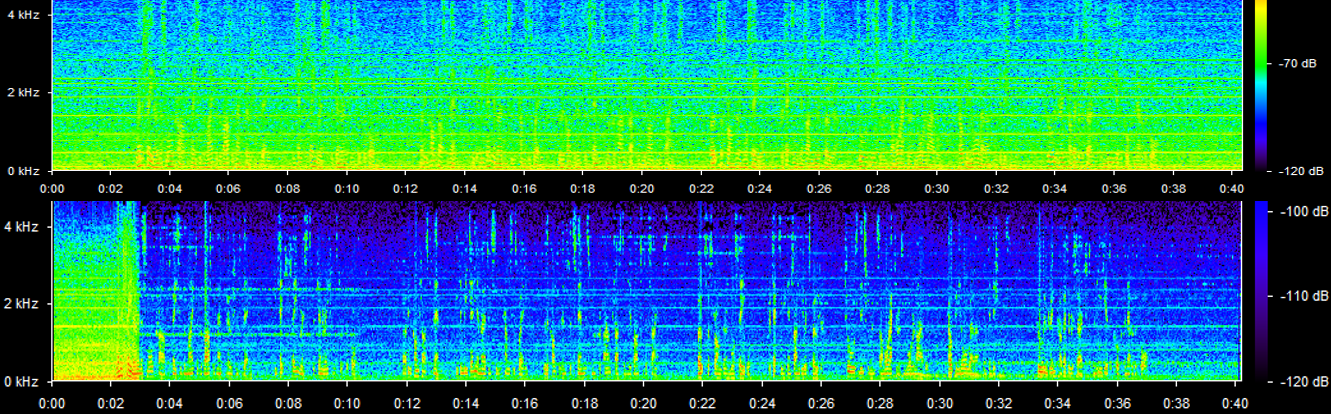
Secondly, parameters in the last two additional enhancements are especially chosen for the sample speaker, who is an adult male with low and deep voice. If object is a child or female, we expect the pass-band of filter implemented in the second additional enhancement to shift to higher frequency region. If given more information on either noise, speaker or occasion, there are many potential improvement. Apart from the gender of speaker, speech pattern of speaker can also be important, including loudness, speed or even emotion. Each speaker speaks differently on different occasions. In this test, however, the occasion and environmental noise is really limited and hence filter is not designed for a general case. If possible, changing fixed parameters to variable, which depends on occasions would be a good idea.

Thirdly, in this experiment, we only process the magnitude of signal. In the past, phase spectrum is considered not important and contain not much information in short frames. However, in this lab clean speech sample is known and it is possible to find the phase spectrum and use it as the output phase spectrum.

Finally, we come up with another possible method to perform noise estimation. With the assumption that human voice is varying fast while environment noise is almost constant. Noise can be calculated through a low-pass filter without tracking the lowest value in 2.5 seconds intervals.

|  |
| --- |
| **for**(k=0;k<FFTLEN;k++){  M1[k]=(1-lpf\_weight\_speech)\* cabs(buffer[k])+lpf\_weight\_speech\*M1[k];  } |

This method works well in noise sample like lynx2. Musical noise bin which overlaps human voice region, are effectively attenuated at the cost of sound intelligibility.

**Final Performance Evaluation**

**Figure 7.** Spectrogram of before (top) and after (bottom) speech enhancement

Figure 7 is an example of noise subtraction with our final design. It is obvious that back ground noise is reduced significantly while speech spectrum is mostly maintained same. However, throughout time axis, there are some frequency components always exists (shown as green horizontal line), which are musical noise. In general, our noise subtraction performs pretty well, majority of noise is removed while musical nodes are small enough to be ignored.

**Reference**

[1] Experiments with a Nonlinear Spectral Subtractor (NSS), Hidden Markov Models and the projection, for robust speech recognition in cars. By P.Lockwood and J boudy

[2] Experiments with a Nonlinear Spectral Subtractor (NSS), Hidden Markov Models and the projection, for robust speech recognition in cars. By P.Lockwood and J boudy

[3] Suppression of Acoustic Noise in Speech Using Spectral Subtraction. By STEVEN F.BOLL

[4] Enhancement of speech corrupted by acoustic noise. By M. Berouti, R. Schwartz, and J. Makhoul

**Bibliography**

[1] Paul D. Mitcheson, Real Time Digital Signal Processing, Section 8 – Speech Enhancement

[2] Paul D. Mitcheson, Real Time Digital Signal Processing, Section 7 – Frame Processing

[3] Rainer MARTIN, Spectral Subtraction Based on Minimum Statistics

[4] [Kuldip Paliwal](http://www.sciencedirect.com/science/article/pii/S0167639310002086), [Kamil Wójcicki](http://www.sciencedirect.com/science/article/pii/S0167639310002086), The importance of phase in speech enhancement