Imperial College London

Real-Time Digital Signal Processing Speech Enhancement Project Report

Calvin Chan (CID: 01048905)

Zheyuan Li (CID: 01181358)

Imperial College London

EE3-19 Real-time Digital Signal Processing (2018-2019)

Content

1. Introduction

In this lab session, we are required to design a real-time noise cancellation of a noisy speech. However, we cannot simply design a filter because we do not know the exact frequency components of the noise. The basic methodology we use is to estimate the noise spectrum and deduct it from the noisy speech spectrum. Several speech enhancements are used to further suppress the noise and improve the signal-to-noise ratio (SNR).

2. Frame Processing

2.1. Basic Ideas

Rather than do processing in every input and output sample, Fast Fourier Transform (FFT) requires computations of samples in bulk. But if the data is long, we need to wait the last sample to be read and thereafter do the FFT. This may cause a very long latency.

However, we can split the entire array of samples into shorter blocks of samples, do FFT in those blocks and then add them together. This technique is called Frame Processing, or Block Filtering. As a result, several procedures are required to perform a desirable noise cancelling task.

Figure 1 illustrates some fundamental elements of doing one frame processing.

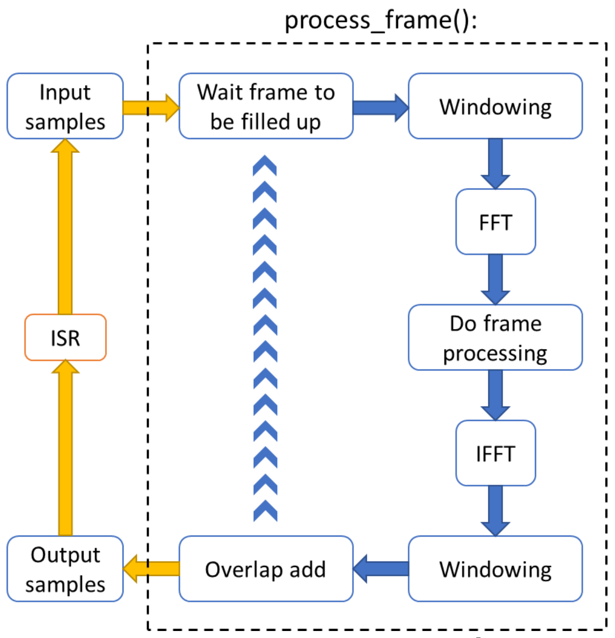


Figure 1: Frame processing procedure

2.2. Frame Length

The frame here means the length of FFT we are going to perform or the numbers of samples in every FFT procedure. Considering all factors, we know that longer frame length means better frequency resolution because of more frequency bins in frequency domain, but worse time resolution in time domain due to latencies. We choose 256 points FFT frame as a moderate frame length and also comply with the project description. Further discussion of using different frame lengths is in Section 4.7.

2.3. Windowing

As discussed in Section 2.1, splitting samples into blocks is equivalent to multiplying them with a window, and specifically a rectangular window. However, rectangular windows introduce disruptions of original samples regardless in time or frequency domain. Multiplying a signal in time domain with a window is equivalent to convolute them in frequency domain. The Fourier Transform of a rectangular window is a sinc function and it adds other unwanted frequency components to the original signals.

Hence, we use the square root of a hamming window, shown in Figure 2.

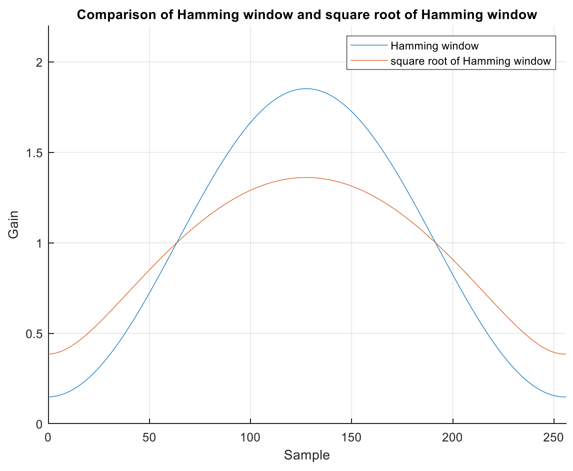


Figure 2: Comparison of windows

According to lecture notes, "the square root of the Hamming window has the property that the windows always add to 1 for 4 times frame oversampling" [1].

And the windowing function w(k) is:

$$w(k) = \sqrt{1 - \frac{0.46}{0.54}\cos\left((2k+1) * \frac{\pi}{N}\right)},$$

where N is the FFT frame length 256.

And this windowing function is done in C:

```
...
for (k=0;k<FFTLEN;k++){
  inwin[k] = sqrt((1.0-WINCONST*cos(PI*(2*k+1)/
    FFTLEN))/OVERSAMP);
  outwin[k] = inwin[k];
}
...
```

Code 1: Windowing function

After multiplying the windowing function, the signals near the edges of windows are suppressed and results in discontinuity in signals, then the Overlap Add procedure is introduced in Section 2.4.

2.4. Overlap Add

In this project, we do FFTs 4 times in every 256 samples. In another word, we do an FFT or a frame processing every 64 samples but the FFT size is still 256 samples. This can be illustrated in Figure 3.

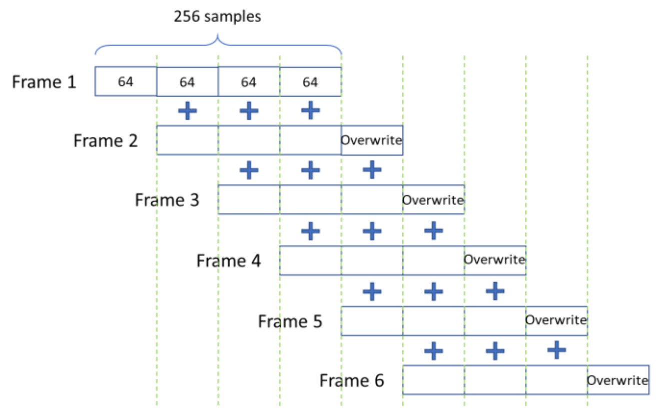


Figure 3: Overlap add

We can easily deduce that there are 3×64=192 samples overlapping and adding together in any two adjacent frames. Every quarter of an output frame is the sum of 4 quarters of 4 different frames.

C implementation:

```
...
for (k=0;k<(FFTLEN-FRAMEINC);k++){
  /* this loop adds into outbuffer */
  outbuffer[m] = outbuffer[m]+outframe[k]*outwin[k];
  if (++m >= CIRCBUF) m=0; /* wrap if required */
}
for (;k<FFTLEN;k++){
  outbuffer[m] = outframe[k]*outwin[k];
  /* this loop over-writes outbuffer */
  m++;
}
...
```

Code 2: Overlap add

In C, pointers and buffers are used instead of copying data and shifting frames as illustrated in Figure 3, in order to reduce computational complexity and improve efficiency.

2.5. Latency

The sampling rate is always 8kHz, so we do 8k/64 = 125 frame processing in one second. To fill up one quarter of a frame (64 samples), it takes 8ms. Figure 4 shows a moment when the ISR fills up a ¼ frame.

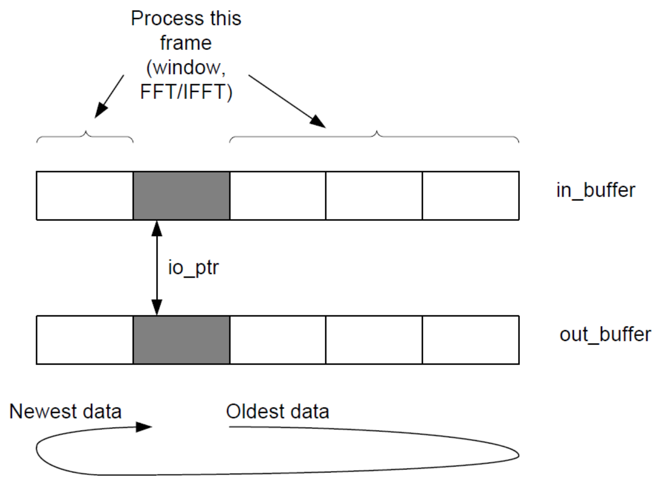


Figure 4

From Figure 4, we can see that ¼ frame is used in 4 successive FFT calculations. There is an additional delay when ISR write samples in the "out_buffer" to the H_Codec. So there are totally five ¼ frames or 5×8ms=40ms delay between output and input signals.

3. Basic Noise Cancelling

3.1. Methodology

Assuming the noise is purely additive to the speech signal, the original speech spectrum Y(ω) theoretically would be:

$$\mathrm{Y}(\omega) = \mathrm{X}(\omega) - N(\omega) \qquad (1)$$

where X(ω) and N(ω) stand for noise corrupted speech spectrum and estimated noise spectrum (Section 3.2) respectively. However, the exact phase of the noise spectrum is unknown. So we cannot directly deduct noise spectrum from the noisy speech spectrum. Instead, we just deduct the amplitude of noise spectrum from the amplitude of the noisy speech spectrum:

$$|\mathrm{Y}(\omega)| = |\mathrm{X}(\omega)| - |N(\omega)| \qquad (2)$$

We know that:

$$\frac{Y(\omega)}{X(\omega)} = \frac{|Y(\omega)|}{|X(\omega)|},$$

and from Equation (1) and (2) we can get:

$$Y(\omega) = X(\omega) \cdot \frac{|X(\omega)| - |N(\omega)|}{|X(\omega)|}$$
$$Y(\omega) = X(\omega)\left(1 - \frac{|X(\omega)|}{|N(\omega)|}\right)$$
$$Y(\omega) = X(\omega)g(\omega)$$

where g(ω) is defined as:

$$g(\omega) = \max\left(\lambda, 1 - \frac{|X(\omega)|}{|N(\omega)|}\right) \qquad (3)$$

and λ is a constant between 0.01 and 0.1, it will be further discussed in Section 3.3.

3.2. Noise Spectrum Estimation

As described in the lab instruction, normally a person would not talk for 10 seconds without a break. By using this short pause during speech, we can estimate the background noise.

We use 4 pointers M1, M2, M3 and M4, each of them points to a dynamically allocated array with size of half of the FFT length (HALF_FFTLEN) which is 128. Reason for this is discussed in Section 3.4. Each of the array stores the minimum amplitude spectrum of 2.5s, and four arrays store the minimum amplitude spectrums over the past 10s.

Basically, there are 4 steps in calculating the estimated noise spectrum:

i. Update M1 in every frame:
$$M_1(\omega) = \min\big(|X(\omega)|, M_1(\omega)\big)$$
ii. Shift pointers every 2.5s. We introduce a temporary pointer like a buffer pointer in order to circulate all four pointers. And also let M1 to be the amplitude spectrum of the current FFT frame, i.e.:
$$M_1(\omega) = |X(\omega)|$$
iii. Estimate the noise spectrum N(ω):
$$|N(\omega)| = \alpha \min_{i=1\sim4}\big(M_i(\omega)\big)$$
where α is a constant.
iv. Calculate the output frame:
$$g(\omega) = \max\left(\lambda, 1 - \frac{|X(\omega)|}{|N(\omega)|}\right)$$
$$Y(\omega) = X(\omega)g(\omega)$$
where λ is a constant.

Because of the algorithm of estimating the noise spectrum, the noise level is always underestimated. The purpose of α is to deliberately overestimate the amplitude of estimated noise spectrum and to further reduce the noise from the noisy speech. The purpose of λ is to prevent g(ω) from being negative.

Straightforwardly, we can implement the algorithm in C:

```c
// in frame processing function noise_processing():
...
int i = 0;
float g = 0;

for(i=0;i<HALF_FFTLEN;i++){
  X[i] = cabs(inframe[i]);  // calculate the amplitude

  // get the maximum amplitude of any frequency bin
  XMAX = max(XMAX, X[i]);

  // when M1-M4 are shifted:
  if(frames_shifted == 1) M1[i] = X[i];
  // when M1-M4 are not shifted:
  else M1[i] = min(X[i], M1[i]);

  N[i] = ALPHA*min(min(M1[i],M2[i]),min(M3[i],M4[i]));
  g = max(LAMBDA, 1-N[i]/X[i]);
  inframe[i] = rmul(g, inframe[i]);
}
...
```
```c
// in ISR function ISR_AIC():
...
//in every 2.5 seconds
if(++samp_count > MSHIFTTIME){

  // shift of pointers
  Mtmp = M4;
  M4 = M3;
  M3 = M2;
  M2 = M1;
  M1 = Mtmp;

  M_initial(M1); // initialize M1
  samp_count = 0; // reset samp_count every 2.5 seconds
  M_shifted = 1;
  // set flag indicating M1-M4 shifts are performed
}
...
```

Code 3: Basic noise cancelling

We use a counter in ISR to count the number of samples read and when it counts 20k we shift pointers. The value 20k is equivalent to 2.5s in time because the sampling frequency is 8kHz. Also we use a flag M_shifted to indicate whether the pointers are shifted and its value is used in the noise_processing() function.

Figure 5 shows the estimated noise spectrum of the factroy1.wav file, and this noise spectrum matches with the factory1 noise spectrum our spectrograms.

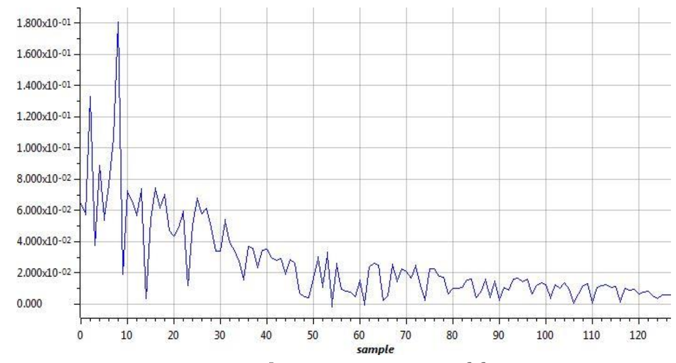


Figure 5: Estimated noise spectrum of factory1.wav

3.3. α and λ trade-off

Theoretically, α should takes a value much larger than one to overestimate noise. However, this is not enough. According to Berouti,M [2], this noise elimination method will introduce a "new" noise called **musical noise**. And the properties of this noise "have valleys and peaks in the short-term power spectrum of noise [2]". After the noise subtraction, valleys are shifted down and set to a floor value. In order to fill in the valleys, a method called spectrum floor needs to implement, which means that a factor λ with much smaller than one should use.

After running the program and testing with all noisy speech .wav files, we find that there is no unique pair of optimal parameters α and λ for every file. Therefore, compromises have to be made.

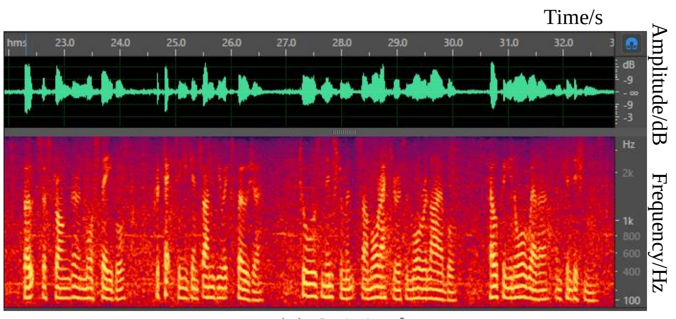
.wav files	α	λ
car1	40	0.1
factory1	60	0.01
factory2	60	0.01
lynx1	60	0.01
lynx2	40	0.01
phantom1	50	0.01
phantom2	50	0.02
phantom4	50	0.01

Table 1: Optimal values of α and λ for each .wav file

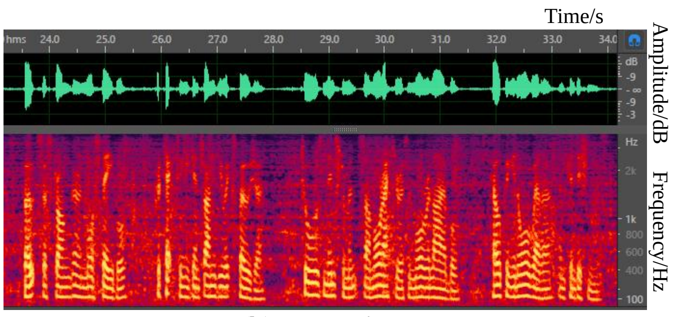
Table 1 shows the optimal values of the parameters α and λ we choose for each of the .wav file. The criteria we use for choosing the optimal values are:

i. How much noise is removed, the more the better.
ii. How much the speech is distorted, the less the better.
iii. How much additional unwanted noise, e.g. musical noise residue or intermittent cracking sound, the less the better.

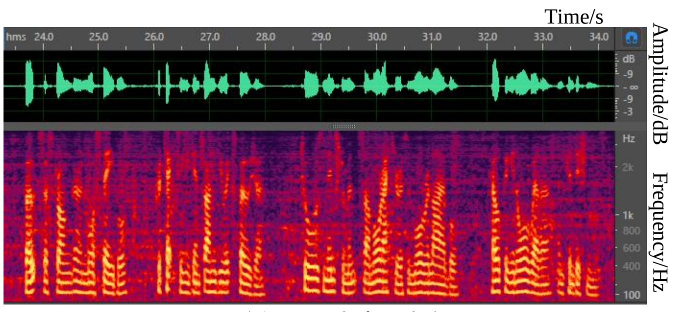
The choices of these parameters are quite subjective and largely influenced by the quality of our earphones. So combining with personal opinions and spectrograms, we derive the data in Table 1.



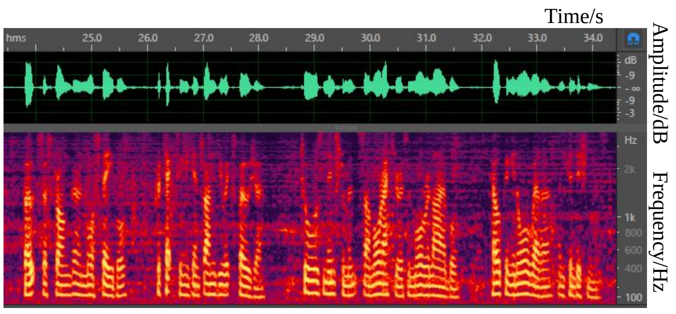
(a) Original



(b) α = 20, λ = 0.1



(c) α = 40, λ = 0.1



(d) α = 40, λ = 0.01

Figure 6: Waves and spectrograms of "Boats are stronger … conditions" before and after basic noise cancelling for factory1.wav with different parameters

The four diagrams in Figure 6 display the waves and spectrograms of the speech "Boats are stronger … conditions" in "factory1.wav" with different α and λ settings. In spectrograms, higher brightness of the colour indicates higher amplitude at that frequency at that time. Compare the four spectrograms in Figure 6, we confirm that the basic noise cancelling program does reduce the background noise because the brightness in spectrograms during non-speech intervals is lower after basic noise cancelling.

We can also see that with same λ but the higher α value, the more background noise is reduced. However, if we continue to raise the α value, the voice would be distorted. Similarly, with the same α value, more noise is reduced if using smaller λ, but higher musical noise is introduced. The musical noise will be discussed in Section 4.8.

Note that the frequency axes are semi-logarithmic (neither linear nor completely logarithmic), because this would give us clearer view for both low and high frequencies.

3.4. Symmetric Property

```
// in the end of noise_processing():
...
for(i=HALF_FFTLEN;i<FFTLEN;i++){
  inframe[i].r = inframe[FFTLEN-i].r;
  inframe[i].i = -inframe[FFTLEN-i].i;
}
...
```

Code 4: Upper 128 frequency bins using symmetric property

Considering the nature of FFT and z-transform, we know that the frequency bins above the Nyquist frequency are the complex conjugates of corresponding frequency bins below the Nyquist frequency, symmetrical with the Nyquist frequency. By implementing this property in our code, we can improve our code efficiency by a factor of 2. Therefore, we only calculate the lower 128 frequency bins 0-127, and the upper 128 bins can be easily calculated as shown in Code 4.

3.5. Limitations

From Figure 7, we conclude that there are 2 major limitations for this basic noise cancelling mechanism. By experimenting several times using the same audio file, we find out that there is a 7.5s to 10s delay before the noise cancelling to be functional. This is due to our algorithm and cannot be eliminated, but it can be reduce using shorter noise detection time as discussed in Section 4.9. The other limitation is that the background noise is still very noticeable, which can be seen by comparing Figure 7 and Figure 8. As a result, several speech enhancement attempts are made in Section 4.

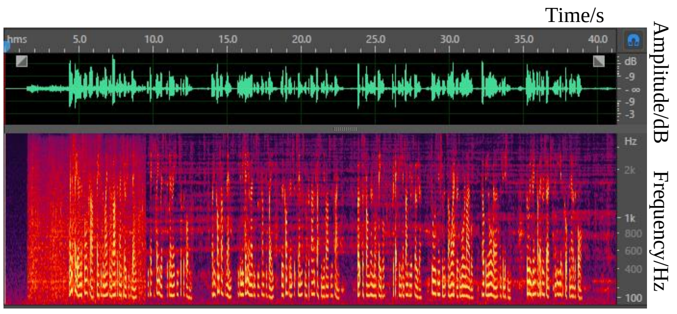


Figure 7: Overall wave and spectrogram after basic noise cancelling for factory1.wav

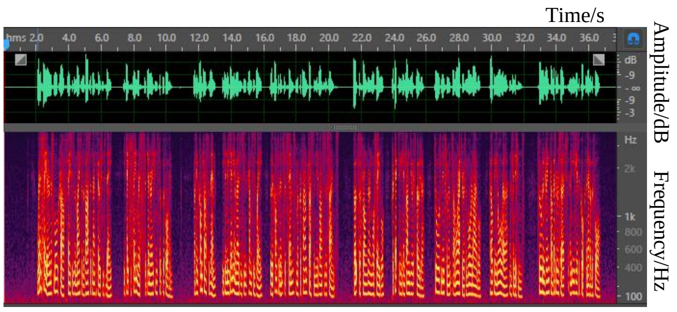


Figure 8: clean.wav from DSK with no signal processing

We also notice that, at the beginning of Figure 7, there is already very slight background noise. This noise is systematic due to the imperfection of the sound card of our PC which we use to record the sound from the DSK board. This noise cannot be reduced and it is always present (see Appendix 8.2), causing the spectrograms to have an overall background noise. Figure 8 is the spectrogram of the clean speech passing through the DSK board without any signal processing. And we can see even during the non-speech interval, the spectrogram is not completely dark because of this systematic noise.

4. Speech Enhancement

4.1. Enhancement 1 (E1)

This enhancement is using a low-pass filtered version of the input frame P(ω) when do the noise cancelling by following the equation:

$$P_t(\omega) = (1 - K) \times |X(\omega)| + K \times P_{t-1}(\omega)$$

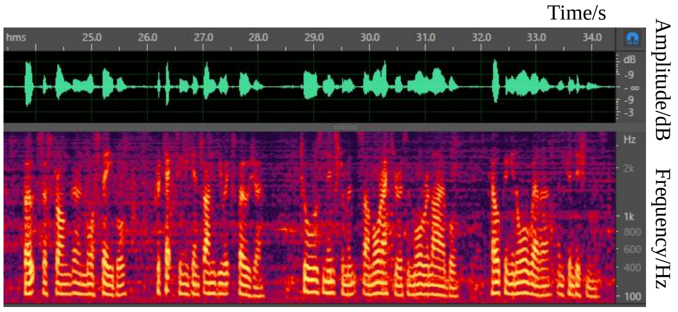
where t means the current frame, t-1 means the frame delayed by one frame, K is a constant which is calculated by $K = e^{-T/\tau}$, where T is TFRAME which is defined at the beginning of the code, τ is a manually set time constant from 20ms to 80ms.

And we implement the formulae in C directly:

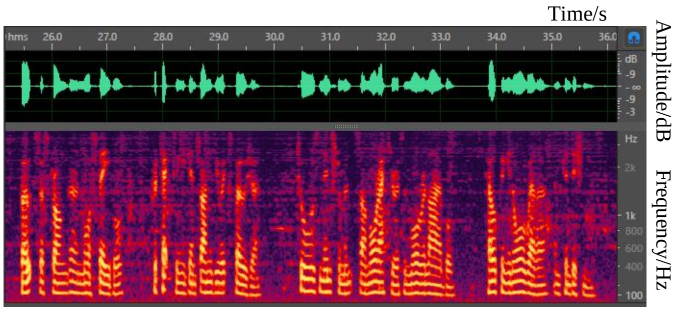
```
// in function main():
...
K = exp(-TFRAME/TAU);
...
// in function process_frame():
...
if(E1) p[i] = (1-K)*X[i]+K*p[i];
// low-pass filtered X in amplitude domain
...
```

Code 5: Enhancement 1

The motivation behind this enhancement is that the noise spectrum we designed for the basic noise cancellation always captures the minimum values from input frames. And these minimum values in input frames are often not the true and fair estimation of the noise spectrum. Theoretically, P(ω) is similar to a moving average filter which removes the abrupt change of the amplitude of each frequency bin from frame to frame. It may give us more accurate noise estimation and presumably require lower α value.



(a) Basic noise cancelling, $\alpha = 40$, $\lambda = 0.01$



(b) Enhancement 1, $\alpha = 4$, $\lambda = 0.02$

Figure 9: Waves and spectrograms of "Boats are stronger … conditions" for factory1.wav before and after Enhancement 1

By experimenting with Enhancement 1, we can reduce the α value down to 4 but increase the noise reduction level, retaining the same speech quality as shown in Figure 9. 80ms is a very favourable value for τ which gives us the clearest voice. Higher values would increase the noise and lower values would result in corruption of the voice.

4.2. Enhancement 2 (E2)

Instead of doing the low-pass filtered version of the input frame in magnitude domain, we try it in power domain by taking the square of |X(ω)| and the square root of the calculated P(ω):

$$P_t^{sqr}(\omega) = (1-K) \times |X(\omega)|^2 + K \times P_{t-1}^{sqr}(\omega)$$

$$P(\omega) = \sqrt{P_t^{sqr}(\omega)}$$

Implementing the above formulae in C:

```
...
else if(E2){
  // low-pass filtered X in power domain
  psqr = (1-K)*X[i]*X[i]+K*psqr;
  p[i] = sqrt(psqr);
}
...
```

Code 6: Enhancement 2

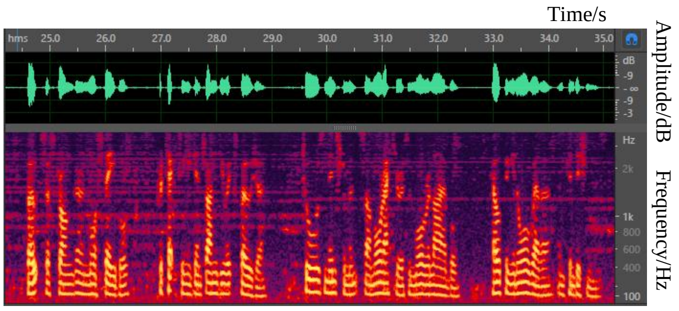


Figure 10: Waves and spectrograms of "Boats are stronger … conditions" for factory1.wav after Enhancement 2

By listening the output and comparing the spectrograms (Figure 10), we conclude that Enhancement 1 and 2 have similar noise reduction level, and the voice in them has equal intelligibility.

However, Enhancement 2 introduces more undesirable musical noise in the background. Because Enhancement 1 and 2 are not compatible (they cannot be used at the same time), we decide to choose Enhancement 1 as our priority over Enhancement 2.

4.3. Enhancement 3 (E3)

Using the same technique as in Enhancement 1, we low-pass filter the estimated noise spectrum by applying:

$$N_t(\omega) = (1-K) \times N_t(\omega) + K \times N_{t-1}(\omega)$$

and implementing in C:

```
...
a = ALPHA;    // do nothing for ALPHA
Ntmp = min(min(M1[i],M2[i]), min(M3[i],M4[i]));
// take minimum value in M1-M4 for the current
frequency bin
...
if(E3) N[i] = (1-K)*a*Ntmp + K*N[i];
// low pass filter the noise estimate
...
```

Code 7: Enhancement 3

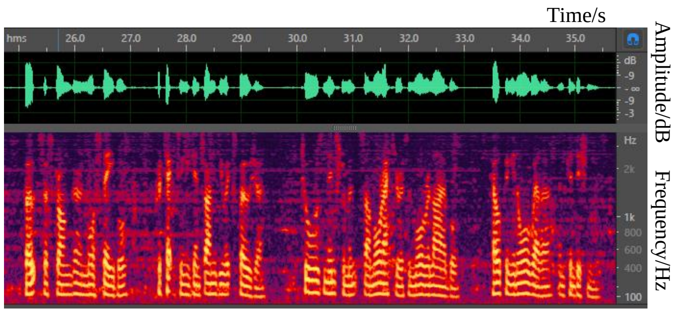


Figure 11: Wave and spectrogram of "Boats are stronger … conditions" for factory1.wav after Enhancement 1 and 3

When we turn on Enhancement 1 and 3 together, we find out that the noise level and the spectrogram stay pretty much the same. However, as the lab instruction noted, the reduction of noise is effective only when the noise level is fluctuated. When we test the factory2.wav which has high noise level and high noise fluctuations, there is indeed a noticeable reduction in the noise level.

4.4. Enhancement 4 (E4.1-E4.4)

In this enhancement, we replace g(ω) by 4 different possible expressions, which are:

$$E4.1: \quad g(\omega) = \max\left(\lambda\frac{|N(\omega)|}{|X(\omega)|}, 1 - \frac{|N(\omega)|}{|X(\omega)|}\right)$$
$$E4.2: \quad g(\omega) = \max\left(\lambda\frac{|P(\omega)|}{|X(\omega)|}, 1 - \frac{|N(\omega)|}{|X(\omega)|}\right)$$
$$E4.3: \quad g(\omega) = \max\left(\lambda\frac{|N(\omega)|}{|P(\omega)|}, 1 - \frac{|N(\omega)|}{|P(\omega)|}\right)$$
$$E4.4: \quad g(\omega) = \max\left(\lambda, 1 - \frac{|N(\omega)|}{|P(\omega)|}\right)$$

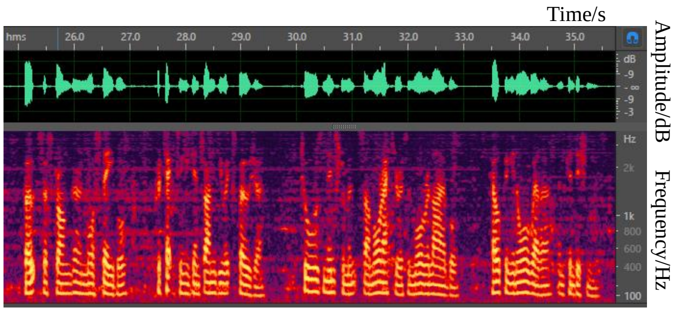
We transform these formulae into C code:

```
...
if(E4==1)      g = max(LAMBDA*N[i]/X[i], 1-N[i]/X[i]);
else if(E4==2) g = max(LAMBDA*p[i]/X[i], 1-N[i]/X[i]);
else if(E4==3) g = max(LAMBDA*N[i]/p[i], 1-N[i]/p[i]);
else if(E4==4) g = max(LAMBDA, 1-N[i]/p[i]);
...
```

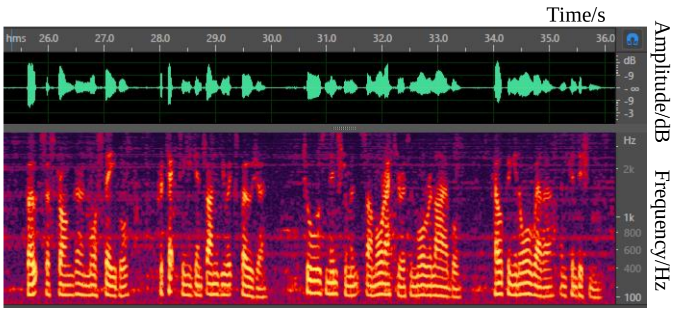
Code 8: Enhancement 4

We let the value of "E4" indicate which version of g(ω) we want to choose and put it in the watch window, so that we can change it in real time.

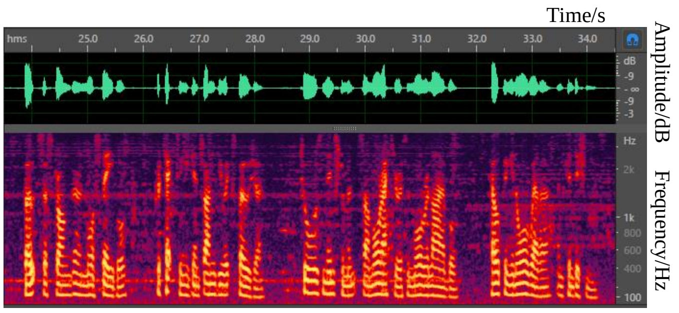
By running these settings with Enhancement 1 and 3 open, we find that none of these gives significant improvement for either noise suppression or speech intelligibility. E4.1 and E4.2 both make no significant changes. E4.3 and E4.4 have greater noise subtraction but they introduce some reverberations. We can hear a short echo sound after every articulation in the speech. This can also be visualized in the spectrogram (Figure 12) that there are some kind of small tails or obscureness after high amplitude frequency bins, especially at high frequencies. These reverberations are more significant if we use smaller α value or larger time constant τ.



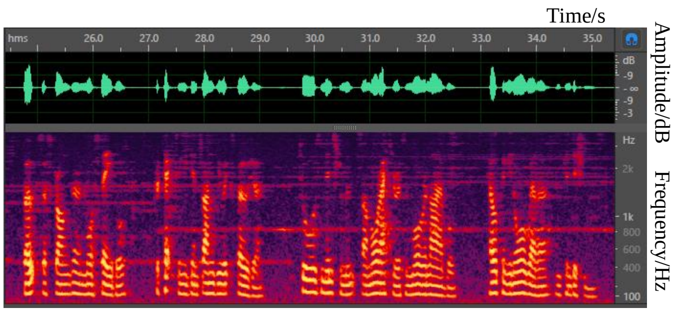
(a) Enhancement 1 and 3, α = 4, λ = 0.02



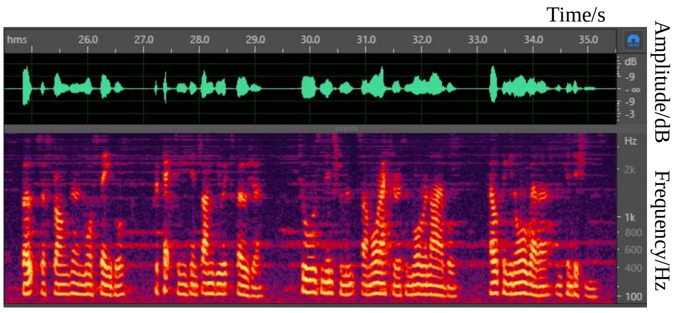
(b) Enhancement 1,3 and 4.1, α = 4, λ = 0.02



(c) Enhancement 1,3 and 4.2, α = 4, λ = 0.02



(d) Enhancement 1,3 and 4.3, α = 4, λ = 0.02



(e) Enhancement 1,3 and 4.4, α = 4, λ = 0.02

Figure 12: Waves and spectrograms of "Boats are stronger … conditions" for factory1.wav after Enhancement 1, 3 and 4

4.5. Enhancement 5 (E5.1-E5.5)

Follow the similar formulae as in Enhancement 4, but this time we calculate g(ω) in power domain rather than in amplitude domain by taking square and square roots like we did in Enhancement 2:

$$E5.1:\quad g(\omega) = \max\left(\lambda\frac{|N(\omega)|}{|X(\omega)|}, \sqrt{1-\frac{|N(\omega)|^2}{|X(\omega)|^2}}\right)$$

$$E5.2:\quad g(\omega) = \max\left(\lambda\frac{|P(\omega)|}{|X(\omega)|}, \sqrt{1-\frac{|N(\omega)|^2}{|X(\omega)|^2}}\right)$$

$$E5.3:\quad g(\omega) = \max\left(\lambda\frac{|N(\omega)|}{|P(\omega)|}, \sqrt{1-\frac{|N(\omega)|^2}{|P(\omega)|^2}}\right)$$

$$E5.4:\quad g(\omega) = \max\left(\lambda, \sqrt{1-\frac{|N(\omega)|^2}{|P(\omega)|^2}}\right)$$

$$E5.5:\quad g(\omega) = \max\left(\lambda, \sqrt{1-\frac{|N(\omega)|^2}{|X(\omega)|^2}}\right).$$

Transforming them into C:

```
...
if(E5==1)       g = max(LAMBDA*N[i]/X[i],sqrt(1-
  (N[i]*N[i])/(X[i]*X[i])));

else if(E5==2)  g = max(LAMBDA*p[i]/X[i], sqrt(1-
  (N[i]*N[i])/(X[i]*X[i])));

else if(E5==3)  g = max(LAMBDA*N[i]/p[i], sqrt(1-
  (N[i]*N[i])/(p[i]*p[i])));

else if(E5==4)  g = max(LAMBDA, sqrt(1-
  (N[i]*N[i])/(p[i]*p[i])));

else if(E5==5)  g = max(LAMBDA,sqrt(1-
  (N[i]*N[i])/(X[i]*X[i])));
...
```

Code 9: Enhancement 5

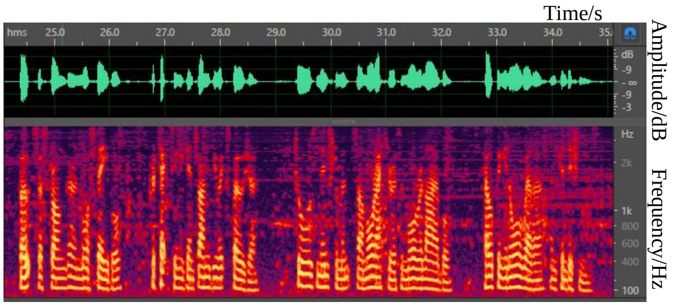


Figure 13: Wave and spectrogram of "Boats are stronger … conditions" for factory1.wav after Enhancement 1, 3 and 5.5

There is a slight improvement for E5.1~E5.4 with larger and clearer voice but less reverberation. E5.5 gives the most stable and clear voice overall with no reverberation heard. We can also confirm this with the spectrogram in Figure 13. The amplitude of the speech is larger and the high energy frequency bins look sharper using Enhancement 5, meaning the voice is much more stable. There is not a noticeable change in background noise level which may result in better SNR ratios.

4.6. Enhancement 6 (E6)

In previous versions of enhancements, we used the same α value at all frequencies. In this enhancement, we intentionally overestimate the noise spectrum at low frequencies under the assumption that, at low frequencies, the SNR is low. Therefore, the α now is a variable that varies with frequency ω.

Thinking of the following function:

$$\alpha(\omega) = \alpha_0 + \frac{\alpha_{max} - \alpha_0}{b\omega + 1}$$

where α_0 is the original α constant value, α_{max} is the maximum value of α(ω), b is the decreasing rate of α(ω) versus ω.

The graph of this function would look like this:

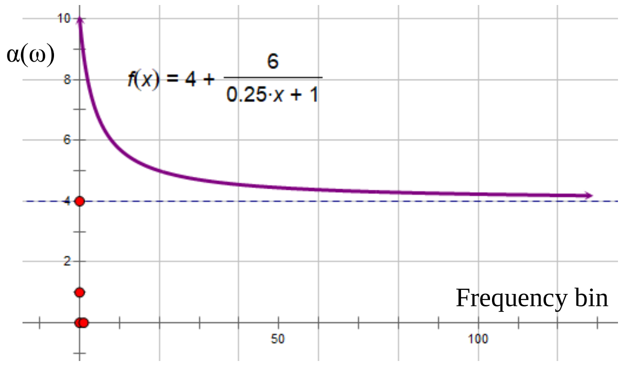


Figure 14: Enhancement 6 graph for α function

Implementing this in C:

```
...
float ALPHA_gradient = 0.25;
...
if(E6) a = ALPHA+(ALPHAMAX-ALPHA)/(ALPHADECRATE*i+1);
// deliberately overestimate the noise level in low
frequency bins
...
```

Code 10: Enhancement 6

Actually, this enhancement does not work very well because in different situations, the SNR at low frequencies are different, and we cannot simply and brutally overestimate the noise level. Under-estimation may cause noise not being reduced enough, while overestimation may result in corruption in voice.

4.7. Enhancement 7 (E7)

By default, our frame length "FFTLEN" is 256. According to Berouti,M et al [2] and combining with our experiment, we conclude that decreasing the frame length would make the speech sounds rougher and more musical noise would be introduced. On the other hand, increasing the frame length would make it sound slurred.

In reality, there is an upper limit for the frame length which is 512. If a larger frame length than 512 is chosen, e.g. 1024, there will be silence at the output because the DSK stores too much samples and information beyond its memory.

4.8. Enhancement 8 (E8)

Due to the uncorrelation of noise, after doing the noise subtraction, there are random frequency peaks in the frequency spectrum. After the IFFT, they may sound like tone generators switch at a very fast rate, about 1/0.008=125Hz where 0.008s is the time interval of two successive FFTs. This is referred as the musical noise.

Interestingly, we can utilize the uncorrelation or randomness of noise. When calculating each frame, if the current N(ω)/X(ω) ratio exceeds some threshold, we let the current output frame to be the minimum from three adjacent frames, i.e. last frame, current frame and the next frame (Boll,S.F. [3]). This can be written in formula:

$$Y_i(\omega) = \begin{cases} Y_i(\omega) & \left(if\ \frac{N_i(\omega)}{X_i(\omega)} > E8_{threshold}\right) \\ \min\limits_{j=i-1,i,i+1}\left(Y_j(\omega)\right) & (otherwise) \end{cases}$$

where i represents the current frame, i−1 and i+1 represents the next frame and last frame.

Writing the equation above in C:

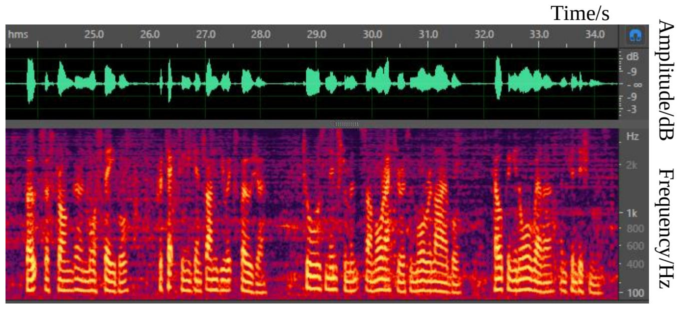
```c
...
if(E8){
  // perfrom delay of frames
  fftframe2[i] = fftframe1[i];
  fftframe1[i] = fftframe[i];
  fftframe[i]  = inframe[i];

  if(N1[i]/X1[i] > E8_threshold){
    inframe[i] = min_cmplx(min_cmplx(fftframe[i],
  fftframe1[i]), fftframe2[i]);
    // take minimum value of frequency bin index i of
  3 adjacent frames
  }

  N1[i] = N[i];
  // perfrom 1 frame delay of noise spectrum
  X1[i] = X[i];
  // perfrom 1 frame delay of noise spectrum
}
...
```

Code 11: Enhancement 8

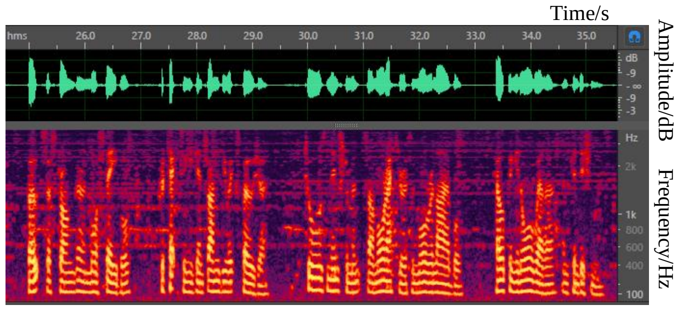
This enhancement works well, gives us lots of music noise suppression and increased volume of the voice.

However, since the noise is deliberately overestimated in Enhancement 6, the N(ω)/X(ω) ratio at low frequencies will always exceed the threshold while the high frequencies will not. This would result in a cracking sound when listen to the output and speech is severely distorted. So, we make a trade-off to keep Enhancement 8 on and disable Enhancement 6 for better overall quality. However, this method introduces an extra but ignorable ¼ frame delay because the calculation for current frame would need the information in the next frame.

After combining Enhancement 1, 3, 5 and 8, the spectrogram is shown in Figure 15 which is much better than using basic noise cancelling but there is still room for improvement.



(a) After basic noise cancelling



(a) After Enhancement 1, 3, 5, and 8

Figure 15: Wave and spectrogram of "Boats are stronger … conditions" for factory1.wav after basic noise cancelling and after Enhancement 1, 3 ,5 and 8

4.9. Enhancement 9 (E9)

As mentioned in Section 3.5, there is a 7.5s~10s delay at the beginning of each speech for the noise cancelling to be functional. We can shorter this delay by decreasing the parameter NOISETIME which defines how long the period that the program estimates the noise spectrum. For example, after setting the NOISETIME to 5s, the delay is reduced to 3.75s~5s as we expected. And there is not much difference compare to 10s. But if we further reduce the NOISETIME, the noise spectrum is no longer accurate and being well suppressed.

5. Additional enhancement

5.1. Extra Enhancement 1 (Ex1)

If we analyse the frequency of the voice without any noise, we can see that the frequency spectrum of the voice is quite complex. But the main information of a speech is at the harmonics or high frequency components rather than low frequency components. And that is why in telephony only a very small fraction of frequencies is extracted from human voice without affecting the intelligibility [4]. Figure 16 shows the frequency spectrums of three different moments in the clean speech. The lowest or the fundamental frequency in the voice is roughly from 80Hz to 110Hz.

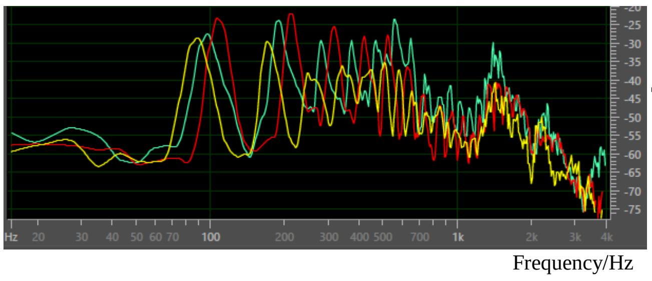


Figure 16: Frequency spectrum of the clean speech clean.wav at three different moments

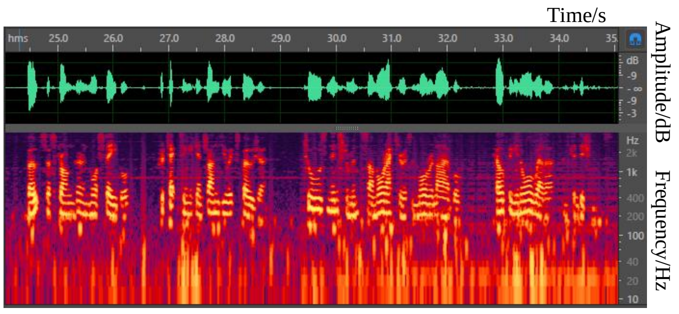
According to this, all the frequency components under 70Hz in any noisy speech are presumably pure noise. We can brutally cut all the frequencies below 70Hz:

```
...
float HPFreq = 70;
// high-pass filter frequency (in Hz)

float HPF = 0;
// high-pass filter frequency (in frequency bin index)
...
HPF = HPFreq*2/FSAMP*HALF_FFTLEN;
...
if(Ex1){
  if(i < HPF) inframe[i] = cmplx(0,0);
  // deliberately eliminate low frequency components
}
...
```

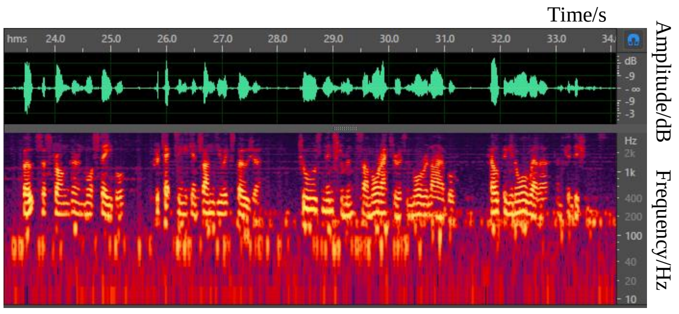
Code 12: Extra Enhancement 1

This enhancement is obvious especially for speech with noise that has high energy at low frequencies. More importantly, it does no harm for the voice at all and has no influences on the intelligibility of speech.

Figure 17 illustrates the spectrograms before and after the Extra Enhancement 1 for car1.wav which has high energy noise at low frequencies. To better see the low frequency components, we here adjust the frequency axis to be completely logarithmic. We can see a noticeable suppression of noise under 70Hz in Figure 17 (b). And if we listen to the truncated output, there is no more unpleasant low-frequency noise residue as before.



(a) Enhancement 1,3,5 and 8, α = 4, λ = 0.02



(b) Enhancement 1,3,5,8 and Ex1, α = 4, λ = 0.02

Figure 17: Wave and spectrogram of "Boats are stronger … conditions" for car1.wav after different enhancements

5.2. Extra Enhancement 2 (Ex2)

In Enhancement 6, we adjust the α value according to different frequencies. Now we adjust the α value based on different SNR value. If SNR is low which means the noise is relatively loud, we want to suppress the noise more by increasing α, and vice versa. Berouti, M. [2] suggested the following relationship between α and SNR:

$$\alpha = \begin{cases} 4.75 & SNR < -5dB \\ \alpha_0 - \dfrac{SNR}{s} & -5dB < SNR < 20dB \\ 1 & SNR > 20dB \end{cases}$$

where α_0 is the y-intercept, 1/s is the gradient and α now is a variable which varies with different SNR. As suggested by Berouti, M. [2], we set α_0 to 4 and set 1/s to 0.15, as larger value of 1/s would make the dynamic range of the speech too large. We set SNR to be:

$$SNR = 20\log_{10}\left(\frac{|X(\omega)| - |N(\omega)|}{|N(\omega)|}\right)$$

then the SNR will be in the correct unit (dB).

The α and SNR would have a linear relationship for SNR between -5dB and 20dB, and SNR would be constant outside this range. The plot for the function of α is shown in Figure 18.

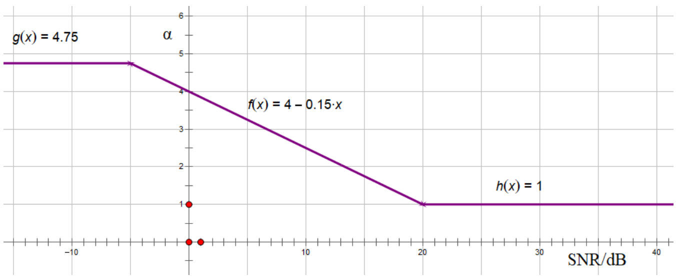


Figure 18: Function of α depending on SNR

And transform the function into C:

```
...
SNR = 20*log10f((X[i]-a*Ntmp)/(a*Ntmp));
// calculate signal to noise ratio

if(Ex2){
  if(SNR<-5) a = 4.75;
  else if (SNR>20) a = 1;
  else a = ALPHA0-SNR*ALPHA_gradient;
  // adaptive ALPHA value
}
...
```

Code 13: Extra Enhancement 2

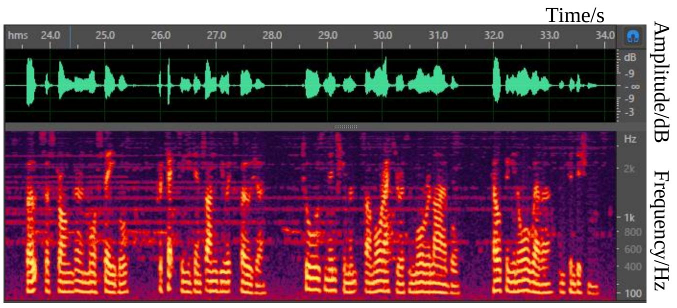


Figure 19: Wave and spectrogram of "Boats are stronger … conditions" for car1.wav after Enhancement 1, 3, 5, 8, Ex1 and Ex2

Enhancement Ex2 noticeably removes the noise residue and musical noise and leaves the voice undistorted.

5.3. Extra Enhancement 3 (Ex3)

According to S. Boll [3], in order to attenuate additional signal during non-speech activity, a measurement T in below has been defined to detect whether speech activity exists:

$$T = 20log_{10}\left[\frac{1}{2\pi}\int_{-\pi}^{\pi}\left|\frac{X(\omega)}{N(\omega)}\right|d\omega\right] \quad (4)$$

where X(ω) is the speech spectrum and N(ω) is the noise spectrum. However, the T in equation (4) is in continuous frequency domain and, we need to transform this equation into discrete frequency domain:

$$T = 20\log_{10}\frac{1}{2\pi}\sum_{\omega=0}^{256}\frac{|X(\omega)|}{|N(\omega)|}$$

and using the symmetric property:

$$T = 20\log_{10}\frac{1}{\pi}\sum_{\omega=0}^{128}\frac{|X(\omega)|}{|N(\omega)|}.$$

If T is less than Ex3_threshold, we classified as absence of speech activity with no output signal, so we define the following algorithm:

$$X(\omega) = \begin{cases} X(\omega) & T \geq Ex3\_threshold \\ 0 & T \leq Ex3\_threshold \end{cases}.$$

C implementation:

```
...
for(i=0;i<HALF_FFTLEN;i++){
...
  if(Ex3){
    Y = cabs(inframe[i]);// magnitude of output frame
    if(E8) T += Y/N1[i];
    // sum, N1 is the 1 frame delayed noise spectrum
    else T += Y/N[i];
    // sum, N is the current noise spectrum
  }
...
}
...
if(Ex3){
  if(T<Ex3_threshold){
    for(i=0;i<HALF_FFTLEN;i++){
      inframe[i] = cmplx(0,0);
      // truncate the output frame during non-speech
  interval
    }
  }
  T = 0;    // reset the sum
}
...
```

Code 14: Extra Enhancement 3

However, this enhancement results in cracking sound and it is what we expect. Each audio file has different SNR, and the Ex3_threshold cannot be unified, which means that the program would recognise the non-speech intervals differently. Thus, the Ex3_threshold in above cannot work perfectly to all the files.

5.4. Other Possible Enhancements

Even use all the best enhancements we implemented, there is still light noise residue and musical noise in the background. If the background noise is predominant (like phantom4.wav), the speech enhancement program is not able to generate a satisfying result. Voice Activity Detection (VAD) is able to discriminate the speech and non-speech frames [5], resulting in an easier but better noise cancelling algorithm. More importantly, an effective VAD can mute the channel during non-speech periods. Also, some researchers [6] suggested using Deep Neural Networks would achieve significant improvements over traditional noise cancelling techniques.

6. Final Choice

After compromising noise reduction level and speech quality, we derive the following enhancement settings and parameters that maximally meet both criteria in all audio files:

Enhancement	**ON/OFF**
E1	ON
E2	OFF
E3	ON
E4.1-E4.4	ALL OFF
E5.1-E5.5	E5.5 ON
E6	OFF
E7	OFF
E8	ON
E9	OFF
Ex1	ON
Ex2	ON
Ex3	OFF
α	4
λ	0.02

Table 2: Final choice of enhancements and parameters

And the spectrograms for all speech files are shown in the Appendix 8.2. The overall enhancement and noise elimination effect is optimistic. However, the speech enhancements for phantom4.wav still leaves quite a lot of noise. If we try to improve the speech enhancements to be more optimal for phantom4.wav, the adverse effect will show for the rest of audio files. It can also be seen that the musical noise has reduced since there is fewer random small part colour distribution. Overall, we choose the “best” combination to improve majority of audio files while keeping phantom4.wav tolerable.

7. Summary

In conclusion, in this project, we designed a noise cancelling program with speech enhancements. We discussed the basic noise cancelling mechanism and made trade-offs between noise suppression level and speech intelligibility in order to achieve equalizing results in all speech audio files.

8. Appendix

8.1. Reference List

[1] P. D. Mitcheson, “Real Time Digital Signal Processing Section 8 - Speech Enhancement”.

[2] Berouti,M., Schwartz,R. and Makhoul,J., “Enhancement of Speech Corrupted by Acoustic Noise,” pp. 208-211, 1979.

[3] S. Boll, “Suppression of Acoustic Noise in Speech using Spectral Subtraction,” *IEEE Trans ASSP 27(2),* pp. 113-120, 04 1979.

[4] J. Rodman, “The Effect of Bandwidth on Speech Intelligibility”.

[5] M. H. Moattar and M. M. Homayounpour , "A SIMPLE BUT EFFICIENT REAL-TIME VOICE ACTIVITY DETECTION," *17th European Signal Processing Conference (EUSIPCO 2009),* pp. 2549-2553, 24-28 08 2009.

[6] X. Yong, D. Jun, D. Li-Rong and L. Chin-Hui, "A Regression Approach to Speech Enhancement," *DOI10.1109/TASLP.2014.2364452, IEEE/ACM Transactions on Audio, Speech, and Language Processing,* 2013.

8.2. Spectrograms

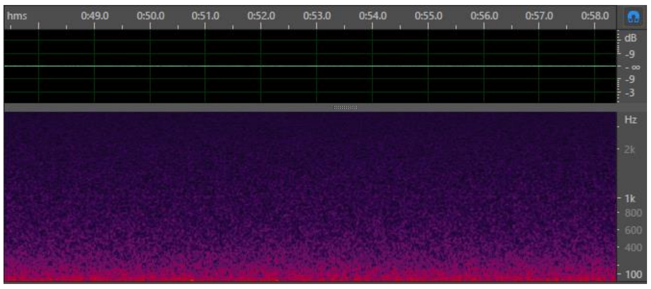
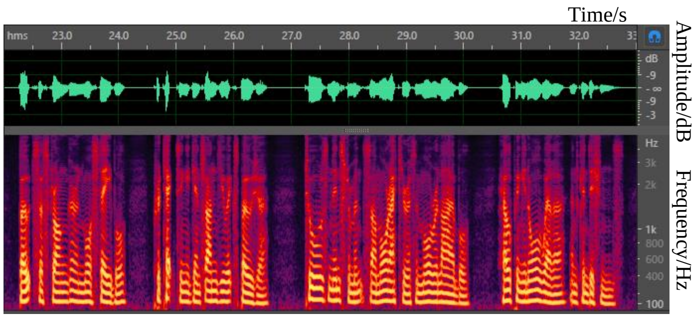
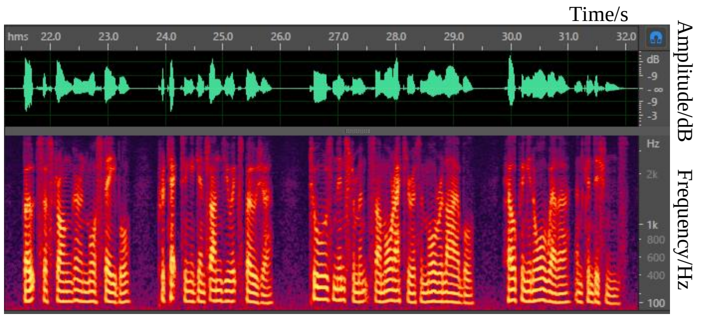


Figure 20 Systematic noise (imperfection of sound card)



(a) Original .wav file (Sampling frequency 11025Hz)



(b)Output from DSK board (Sampling frequency 8kHz)

Figure 21: Wave and spectrogram of "Boats are stronger … conditions" in original and from DSK board

No noise cancelling: **After noise cancelling and enhancements:**

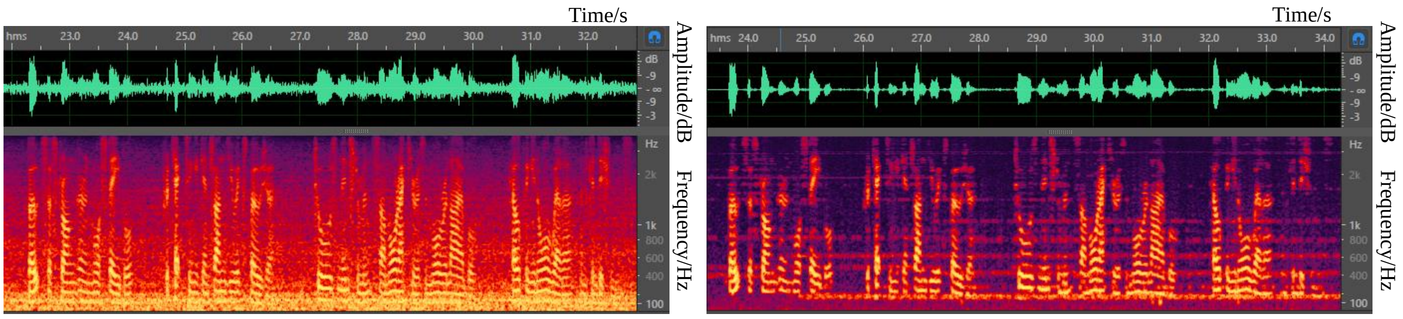


Figure 22: Wave and spectrogram of "Boats are stronger … conditions" for car1.wav

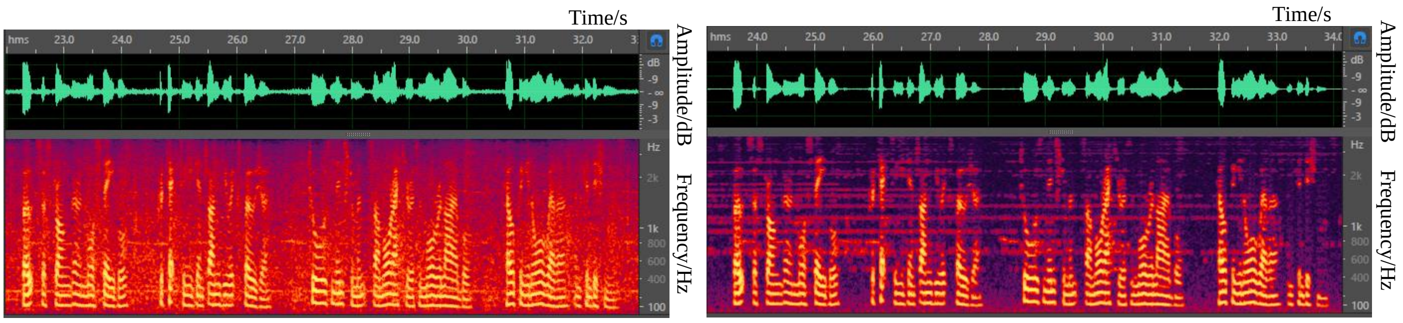


Figure 23: Waves and spectrograms of "Boats are stronger … conditions" for factory1.wav

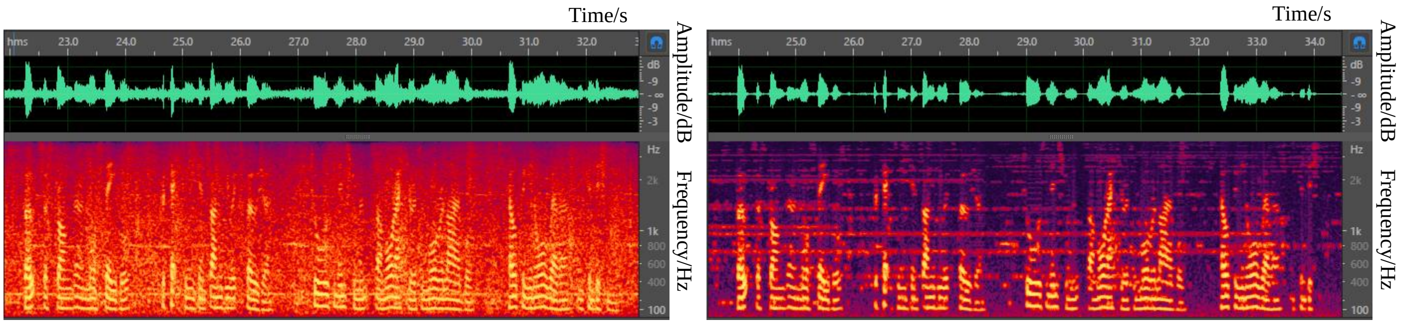


Figure 24: Waves and spectrograms of "Boats are stronger … conditions" for factory2.wav

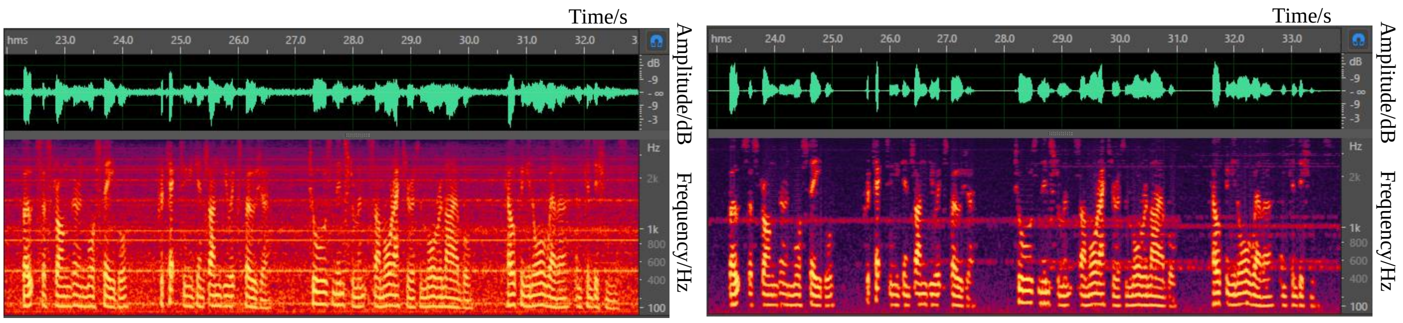


Figure 25: Waves and spectrograms of "Boats are stronger … conditions" for lynx1.wav

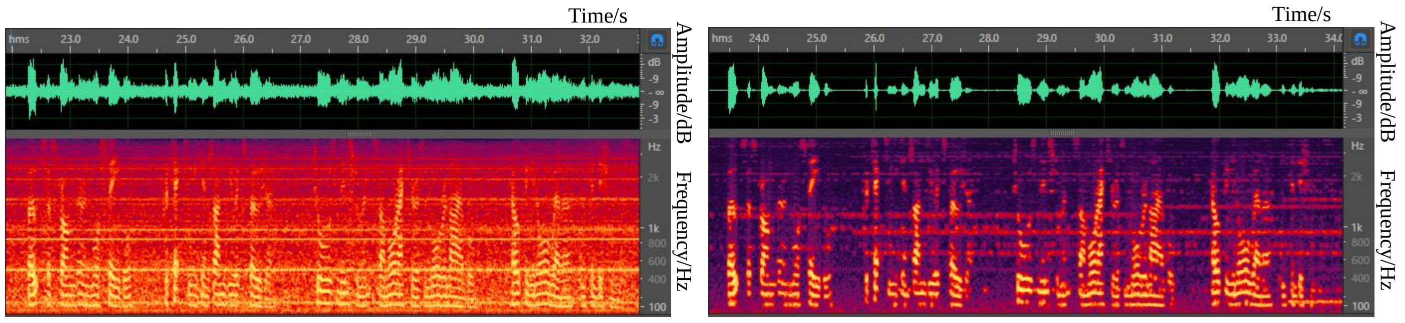


Figure 26: Waves and spectrograms of "Boats are stronger … conditions" for lynx2.wav

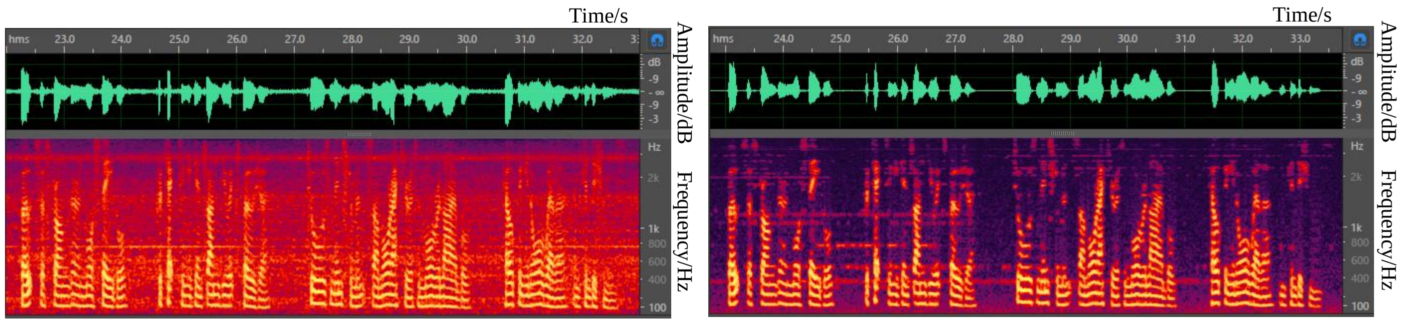


Figure 27: Waves and spectrograms of "Boats are stronger … conditions" for phantom1.wav

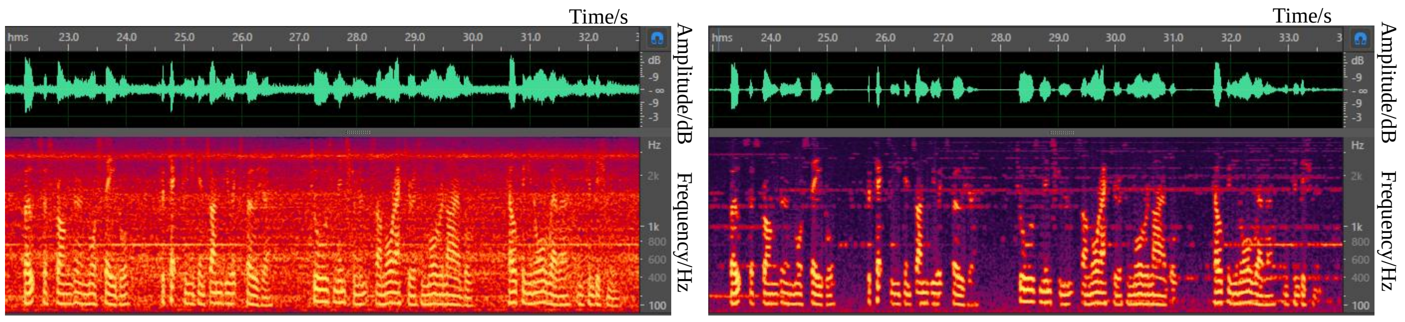


Figure 28: Waves and spectrograms of "Boats are stronger … conditions" for phantom2.wav

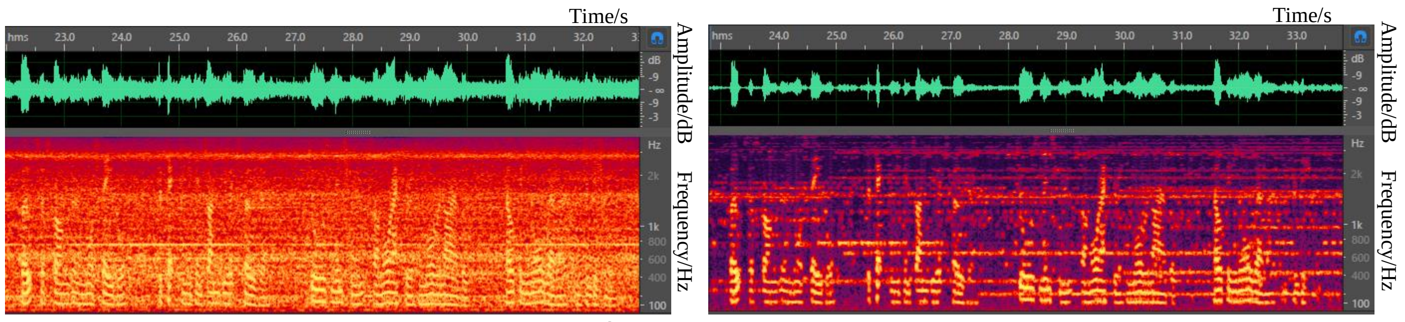


Figure 29: Waves and spectrograms of "Boats are stronger … conditions" for phantom4.wav

8.3. Full Version of Code

```
/*************************************************************************************
             DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING
                     IMPERIAL COLLEGE LONDON

              EE 3.19: Real Time Digital Signal Processing
                 Dr Paul Mitcheson and Daniel Harvey

                     PROJECT: Frame Processing

                    ********* ENHANCE. C **********
               Shell for speech enhancement

      Demonstrates overlap-add frame processing (interrupt driven) on the DSK.

 *************************************************************************************
```

```
                     By Danny Harvey: 21 July 2006
               Updated for use on CCS v4 Sept 2010
 *************************************************************************************/
/*
 *  You should modify the code so that a speech enhancement project is built
 *  on top of this template.
 */
/***************************** Pre-processor statements ******************************/
//  library required when using calloc
#include <stdlib.h>
//  Included so program can make use of DSP/BIOS configuration tool.
#include "dsp_bios_cfg.h"

/* The file dsk6713.h must be included in every program that uses the BSL.  This
   example also includes dsk6713_aic23.h because it uses the
   AIC23 codec module (audio interface). */
#include "dsk6713.h"
#include "dsk6713_aic23.h"

// math library (trig functions)
#include <math.h>

/* Some functions to help with Complex algebra and FFT. */
#include "cmplx.h"
#include "fft_functions.h"

// Some functions to help with writing/reading the audio ports when using interrupts.
#include <helper_functions_ISR.h>

#define WINCONST 0.85185     /* 0.46/0.54 for Hamming window */
#define FSAMP 8000.0   /* sample frequency, ensure this matches Config for AIC */
#define FFTLEN 256   /* fft length = frame length 256/8000 = 32 ms*/
#define NFREQ (1+FFTLEN/2)     /* number of frequency bins from a real FFT */
#define OVERSAMP 4         /* oversampling ratio (2 or 4) */
#define FRAMEINC (FFTLEN/OVERSAMP) /* Frame increment */
#define CIRCBUF (FFTLEN+FRAMEINC)  /* length of I/O buffers */

#define OUTGAIN 16000.0      /* Output gain for DAC */
#define INGAIN  (1.0/16000.0)    /* Input gain for ADC  */
// PI defined here for use in your code
#define PI 3.141592653589793
#define TFRAME FRAMEINC/FSAMP       /* time between calculation of each frame */

/**************************** Additional Declarations ******************************/
#define HALF_FFTLEN (FFTLEN/2)     // half of the FFTLEN
#define NOISETIME 10.0         // the time for choosing N (in second)
#define MSHIFTTIME (NOISETIME/4*FSAMP) // number of samples for every shift of M1-M4
#define TAU 0.08           // TAU used to calculate K
#define ALPHAMAX 10.0        // The maximum value of ALPHA (used in E6)
#define ALPHADECRATE 1.0       // The decrease rate of ALPHA (used in E6)

/******************************* Global declarations ********************************/
/* Audio port configuration settings: these values set registers in the AIC23 audio
   interface to configure it. See TI doc SLWS106D 3-3 to 3-10 for more info. */
DSK6713_AIC23_Config Config = { \
       /**********************************************************************/
       /*   REGISTER             FUNCTION           SETTINGS         */
       /**********************************************************************/\
    0x0017,  /* 0 LEFTINVOL  Left line input channel volume  0dB                   */\
    0x0017,  /* 1 RIGHTINVOL Right line input channel volume 0dB                   */\
    0x01f9,  /* 2 LEFTHPVOL  Left channel headphone volume   0dB                   */\
    0x01f9,  /* 3 RIGHTHPVOL Right channel headphone volume  0dB                   */\
    0x0011,  /* 4 ANAPATH    Analog audio path control       DAC on, Mic boost 20dB*/\
    0x0000,  /* 5 DIGPATH    Digital audio path control      All Filters off       */\
    0x0000,  /* 6 DPOWERDOWN Power down control              All Hardware on       */\
    0x0043,  /* 7 DIGIF      Digital audio interface format  16 bit                */\
    0x008d,  /* 8 SAMPLERATE Sample rate control        8 KHZ-ensure matches FSAMP */\
    0x0001   /* 9 DIGACT     Digital interface activation    On                    */\
       /**********************************************************************/
};

// Codec handle:- a variable used to identify audio interface
DSK6713_AIC23_CodecHandle H_Codec;

float *inbuffer, *outbuffer;       /* Input/output circular buffers */
complex *inframe;         /* Input frames */
float *outframe;                /* Output frames */
float *inwin, *outwin;               /* Input and output windows */
float ingain, outgain;       /* ADC and DAC gains */
float cpufrac;            /* Fraction of CPU time used */
volatile int io_ptr=0;               /* Input/ouput pointer for circular buffers */
volatile int frame_ptr=0;            /* Frame pointer */
```

```
/*************************** Additional Declarations ******************************/
float *X, *X1;      // X = amplitude, X1 = amplitude delayed by 1 frame
float *M1, *M2, *M3, *M4, *Mtmp; // M1-M4 store the minimum spectrums,
                 // need an extra Mtmp to shift pointers in ISR
float *N, *N1;      // N = Noise spectrum, N1 is the same but delayed by 1 frame
float *p;       // p = power spectrum in frequency domain

complex *fftframe, *fftframe1, *fftframe2;
// used in E8 to perform algorithm regarding current, last and next frames
// fftframe = next frame, fftframe1 = current frame, fftframe2 = last frame

int samp_count = 0;       // number of samples that have been read
int M_shifted = 1;        // a flag indicating whether M1-M4 are shifted
float K = 0;          // a constant used in E1 and E2
float psqr = 0;         // p square (p^2)
float a = 0;          // "a" is the calculated ALPHA value
float XMAX = 100;       // maximum value of amplitude in frequency domain
float ALPHA = 4;        // ALPHA value
float LAMBDA = 0.02;      // LAMBDA value
float E8_threshold = 0.8;  // a manually set threshold value used in E8
float HPFreq = 70;        // high-pass filter frequency (in Hz)
float HPF = 0;          // high-pass filter frequency (in frequency bin index)
float SNR = 0;          // signal to noise ration (in decimal value, not in dB)
float ALPHA_gradient = 0.15; // the slope of ALPHA decaying, used in Ex2
float ALPHA0 = 4;       // the y-intercept of ALPHA decaying, used in Ex2
float Ex3_threshold = 0.789; // a manually set threshold value used in Ex3
float T = 0;          // a sum used in Ex3
float Y = 0;          // magnitude of output frame (used in Ex3)

int E1 = 1;     // Enhancement 1, {0,1}
int E2 = 0;     // Enhancement 2, {0,1}
int E3 = 1;     // Enhancement 3, {0,1}
int E4 = 0;     // Enhancement 4, {0,1,2,3,4}
int E5 = 5;     // Enhancement 5, {0,1,2,3,4,5}
int E6 = 0;     // Enhancement 6, {0,1}
int E8 = 1;     // Enhancement 8, {0,1}
int Ex1 = 1;    // Extra Enhancement 1, {0,1}
int Ex2 = 1;    // Extra Enhancement 2, {0,1}
int Ex3 = 0;    // Extra Enhancement 3, {0,1}
// where 0 = off, 1 = on, {1,2,3,...} = on with different versions

/******************************* Function prototypes ******************************/
void init_hardware(void);      /* Initialize codec */
void init_HWI(void);            /* Initialize hardware interrupts */
void ISR_AIC(void);             /* Interrupt service routine for codec */
void process_frame(void);       /* Frame processing routine */

/*************************** Additional Declarations ******************************/
void noise_processing(void);   // processing in frequency domain as a seperate function
void M_initial(float *M);        // array initialization
float min(float a, float b);       // minimum value of a and b
complex min_cmplx(complex a, complex b); // minimun value of a and b (complex version)
float max(float a, float b);       // maximum value of a and b


/********************************** Main routine ************************************/
void main()
{
  int k; // used in various for loops

  K = exp(-TFRAME/TAU);        //a constant used in E1 and E2

  /*  Initialize and zero fill arrays */
  inbuffer  = (float *) calloc(CIRCBUF, sizeof(float)); /* Input array */
  outbuffer = (float *) calloc(CIRCBUF, sizeof(float)); /* Output array */
  inframe   = (complex *) calloc(FFTLEN, sizeof(complex)); /* Array for processing*/
  outframe  = (float *) calloc(FFTLEN, sizeof(float)); /* Array for processing*/
  inwin     = (float *) calloc(FFTLEN, sizeof(float)); /* Input window */
  outwin    = (float *) calloc(FFTLEN, sizeof(float)); /* Output window */

  /************** Additional assignment **************/
  // perform dynamic memory allocation
  X     = (float *) calloc(HALF_FFTLEN, sizeof(float));
  X1    = (float *) calloc(HALF_FFTLEN, sizeof(float));
  M1    = (float *) calloc(HALF_FFTLEN, sizeof(float));
  M2    = (float *) calloc(HALF_FFTLEN, sizeof(float));
  M3    = (float *) calloc(HALF_FFTLEN, sizeof(float));
  M4    = (float *) calloc(HALF_FFTLEN, sizeof(float));
  Mtmp  = (float *) calloc(HALF_FFTLEN, sizeof(float));
  N     = (float *) calloc(HALF_FFTLEN, sizeof(float));
  N1    = (float *) calloc(HALF_FFTLEN, sizeof(float));
  p    = (float *) calloc(HALF_FFTLEN, sizeof(float));

  fftframe  = (complex *) calloc(HALF_FFTLEN, sizeof(complex));
  fftframe1 = (complex *) calloc(HALF_FFTLEN, sizeof(complex));
  fftframe2 = (complex *) calloc(HALF_FFTLEN, sizeof(complex));

  // initialization of M1-M4, assign them to be a reasonably large value
  M_initial(M1);
  M_initial(M2);
  M_initial(M3);
  M_initial(M4);

  /* initialize board and the audio port */
  init_hardware();

  /* initialize hardware interrupts */
  init_HWI();

  /* initialize algorithm constants */
  for (k=0;k<FFTLEN;k++)
  {
  inwin[k] = sqrt((1.0-WINCONST*cos(PI*(2*k+1)/FFTLEN))/OVERSAMP);
  outwin[k] = inwin[k];
  }
  ingain=INGAIN;
  outgain=OUTGAIN;

  /* main loop, wait for interrupt */
  while(1)  process_frame();
}
/********************************** init_hardware() *********************************/
void init_hardware()
{
  // Initialize the board support library, must be called first
  DSK6713_init();

  // Start the AIC23 codec using the settings defined above in config
  H_Codec = DSK6713_AIC23_openCodec(0, &Config);

  /* Function below sets the number of bits in word used by MSBSP (serial port) for
  receives from AIC23 (audio port). We are using a 32 bit packet containing two
  16 bit numbers hence 32BIT is set for  receive */
  MCBSP_FSETS(RCR1, RWDLEN1, 32BIT);

  /* Configures interrupt to activate on each consecutive available 32 bits
  from Audio port hence an interrupt is generated for each L & R sample pair */
  MCBSP_FSETS(SPCR1, RINTM, FRM);

  /* These commands do the same thing as above but applied to data transfers to the
  audio port */
  MCBSP_FSETS(XCR1, XWDLEN1, 32BIT);
  MCBSP_FSETS(SPCR1, XINTM, FRM);
}

/********************************** init_HWI() **************************************/
void init_HWI(void)
{
  IRQ_globalDisable();     // Globally disables interrupts
  IRQ_nmiEnable();       // Enables the NMI interrupt (used by the debugger)
  IRQ_map(IRQ_EVT_RINT1,4);  // Maps an event to a physical interrupt
  IRQ_enable(IRQ_EVT_RINT1);   // Enables the event
  IRQ_globalEnable();      // Globally enables interrupts
}


/******************************** process_frame() ***********************************/
void process_frame(void)
{
  int k, m;
  int io_ptr0;

  // convert HPF and LPF in frequency into frequency bin index
  HPF = HPFreq*2/FSAMP*HALF_FFTLEN;

  /* work out fraction of available CPU time used by algorithm */
  cpufrac = ((float) (io_ptr & (FRAMEINC - 1)))/FRAMEINC;

  /* wait until io_ptr is at the start of the current frame */
  while((io_ptr/FRAMEINC) != frame_ptr);

  /* then increment the framecount (wrapping if required) */
  if (++frame_ptr >= (CIRCBUF/FRAMEINC)) frame_ptr=0;
  /* save a pointer to the position in the I/O buffers (inbuffer/outbuffer) where the
  data should be read (inbuffer) and saved (outbuffer) for the purpose of processing */
  io_ptr0=frame_ptr * FRAMEINC;

  /* copy input data from inbuffer into inframe (starting from the pointer position) */

  m=io_ptr0;
  for (k=0;k<FFTLEN;k++)
  {
    inframe[k] = cmplx(inbuffer[m]*inwin[k],0);
    // inframe points to a dynamic array with complex value, the imaginary part is 0

    if (++m >= CIRCBUF) m=0; /* wrap if required */
  }

  /************************* DO PROCESSING OF FRAME  HERE **************************/
  fft(FFTLEN, inframe);  // perform fast fourier transform

  noise_processing();    // do frequency domain processing
  M_shifted = 0;       // reset flag to be zero

  ifft(FFTLEN, inframe);   // perform inverse fast fourier transform

  for (k=0;k<FFTLEN;k++)
  {
    outframe[k] = inframe[k].r;/* copy input straight into output */
  }
  /*********************************************************************************/

  /* multiply outframe by output window and overlap-add into output buffer */

  m=io_ptr0;

  for (k=0;k<(FFTLEN-FRAMEINC);k++){    /* this loop adds into outbuffer */
    outbuffer[m] = outbuffer[m]+outframe[k]*outwin[k];
    if (++m >= CIRCBUF) m=0; /* wrap if required */
  }
  for (;k<FFTLEN;k++){
    outbuffer[m] = outframe[k]*outwin[k];   /* this loop over-writes outbuffer */
    m++;
  }
}

/****************************** Speech Enhancement ******************************/
// simple noise cancelling can be performed by disabling all enhancements
void noise_processing(void){
  int i = 0;      // index of frequency bins
  float g = 0;    // g(w) as presented in lab instruction
  float Ntmp = 0;  // temporary value of N[i]

  // do frequency processing only on half of the spectrum using symmetric property
  for(i=0;i<HALF_FFTLEN;i++){
    X[i] = cabs(inframe[i]); // calculate the amplitude of inframe
    XMAX = max(XMAX, X[i]);  // get the maximum amplitude of any frequency bin

    // calculate different p depending on different enhancement settings
    if(E1) p[i] = (1-K)*X[i]+K*p[i]; // low-pass filtered X in amplitude domain
    else if(E2){           // low-pass filtered X in power domain
      psqr = (1-K)*X[i]*X[i]+K*psqr;
      p[i] = sqrt(psqr);
    }
    else p[i] = X[i];        // do nothing for X

    // update M1
    if(M_shifted == 1) M1[i] = X[i]; // when M1-M4 are shifted
    else M1[i] = min(p[i], M1[i]);   // when M1-M4 are not shifted

    if(E6) a = ALPHA + (ALPHAMAX-ALPHA)/(ALPHADECRATE*i+1);
    // deliberately overestimate the noise level in low frequency bins

    else a = ALPHA;          // do nothing for ALPHA

    Ntmp = min(min(M1[i],M2[i]), min(M3[i],M4[i]));
    // take minimum value in M1-M4 for the current frequency bin

    SNR = 20*log10f((X[i]-a*Ntmp)/(a*Ntmp)); // calculate signal to noise ratio
    if(Ex2){
      if(SNR<-5) a = 4.75;
      else if (SNR>20) a = 1;
      else a = ALPHA0-SNR*ALPHA_gradient;    // adaptive ALPHA value
    }
    if(E3) N[i] = (1-K)*a*Ntmp+K*N[i]; // low pass filter the noise estimate
    else N[i] = a * Ntmp;      // Noise spectrum times ALPHA

    // E4 and E5 are mutually exclusive, they cannot be enabled at the same time
    if(E4!=0 && E5==0){
      if(E4==1)     g = max(LAMBDA*N[i]/X[i], 1-N[i]/X[i]);
      else if(E4==2)  g = max(LAMBDA*p[i]/X[i], 1-N[i]/X[i]);
      else if(E4==3)  g = max(LAMBDA*N[i]/p[i], 1-N[i]/p[i]);
      else if(E4==4)  g = max(LAMBDA, 1-N[i]/p[i]);
    }
    else if(E4==0 && E5!=0){
      if(E5==1)     g = max(LAMBDA*N[i]/X[i],sqrt(1-(N[i]*N[i])/(X[i]*X[i])));
      else if(E5==2)  g = max(LAMBDA*p[i]/X[i], sqrt(1-(N[i]*N[i])/(X[i]*X[i])));
      else if(E5==3)  g = max(LAMBDA*N[i]/p[i], sqrt(1-(N[i]*N[i])/(p[i]*p[i])));
      else if(E5==4)  g = max(LAMBDA, sqrt(1-(N[i]*N[i])/(p[i]*p[i])));
      else if(E5==5)  g = max(LAMBDA,sqrt(1-(N[i]*N[i])/(X[i]*X[i])));
    }
    else       g = max(LAMBDA, 1-N[i]/X[i]);  // do nothing for g(w)

    inframe[i] = rmul(g, inframe[i]);  // inframe multiplies by g(w)

    if(E8){
      // perfrom delay of frames
      fftframe2[i] = fftframe1[i];
      fftframe1[i] = fftframe[i];
      fftframe[i]  = inframe[i];

      if(N1[i]/X1[i] > E8_threshold){
        inframe[i] = min_cmplx(min_cmplx(fftframe[i], fftframe1[i]), fftframe2[i]);
        // take minimum value of frequency bin index i of 3 adjacent frames
      }
    }

    if(Ex1){
      if(i < HPF) inframe[i] = cmplx(0,0);
      // deliberately eliminate low frequency components
    }

    if(Ex3){
      Y = cabs(inframe[i]);  // magnitude of output frame
      if(E8) T += Y/N1[i];   // sum, N1 is the 1 frame delayed noise spectrum
      else T += Y/N[i];    // sum, N is the current noise spectrum
    }

    if(E8){
      N1[i] = N[i];  // perfrom 1 frame delay of noise spectrum
      X1[i] = X[i];  // perfrom 1 frame delay of noise spectrum
      //inframe[i] = fftframe1[i];
    }
  }

  if(Ex3){
    if(T<Ex3_threshold){
      for(i=0;i<HALF_FFTLEN;i++){
        inframe[i] = cmplx(0,0);
        // truncate the output frame during non-speech interval
      }
    }
    T = 0;    // reset the sum
  }

  // copy values to the other half of the spectrum
  for(i=HALF_FFTLEN;i<FFTLEN;i++){
    inframe[i].r = inframe[FFTLEN-i].r;
    inframe[i].i = -inframe[FFTLEN-i].i;
    // real parts are the same, imaginary part inversed because of complex conjugate
  }
}


/****************************** Additional functions ******************************/
// array initialization
void M_initial(float *M){
  int i = 0;
  for(i=0;i<HALF_FFTLEN;i++){
    M[i] = XMAX + 100;   // assign M[i] to be a reasonably large value
  }
}

// minimum value of a and b
float min(float a, float b){
  if(a<b) return a;
  else return b;
}

// minimum value of a and b (complex version)
complex min_cmplx(complex a, complex b){
  float aabs, babs;
  aabs = sqrt(a.r*a.r+a.i*a.i);
  babs = sqrt(b.r*b.r+b.i*b.i);

  if(aabs < babs) return a;
  else return b;
}

// maximum value of a and b
float max(float a, float b){
  if(a>b) return a;
  else return b;
}

/*************************** INTERRUPT SERVICE ROUTINE  *****************************/
// Map this to the appropriate interrupt in the CDB file
void ISR_AIC(void)
{
  short sample;
  /* Read and write the ADC and DAC using inbuffer and outbuffer */

  sample = mono_read_16Bit();
  inbuffer[io_ptr] = ((float)sample)*ingain;
    /* write new output data */
  mono_write_16Bit((int)(outbuffer[io_ptr]*outgain));

  /* update io_ptr and check for buffer wraparound */
  if (++io_ptr >= CIRCBUF) io_ptr=0;

  //in every 2.5 seconds
  if(++samp_count > MSHIFTTIME){

    // shift of pointers
    Mtmp = M4;
    M4 = M3;
    M3 = M2;
    M2 = M1;
    M1 = Mtmp;

    M_initial(M1);     // initialize M1
    samp_count = 0;    // reset samp_count every 2.5 seconds
    M_shifted = 1;     // set flag indicating M1-M4 shifts are performed
  }
}
/***********************************************************************************/

/************************* simple noise cancelling *****************************/
// this is the original version of noise cancelling without any enhancement
/*
void noise_processing(void){
  int i = 0;
  float g = 0;

  for(i=0;i<HALF_FFTLEN;i++){
    X[i] = cabs(inframe[i]);
    XMAX = max(XMAX, X[i]);

    if(frames_shifted == 1){
      M1[i] = X[i];
    }
    else{
      M1[i] = min(X[i], M1[i]);
    }
    N[i] = ALPHA * min(min(M1[i],M2[i]), min(M3[i],M4[i]));
    g = max(LAMBDA, 1-N[i]/X[i]);
    inframe[i] = rmul(g, inframe[i]);
  }

  for(i=HALF_FFTLEN;i<FFTLEN;i++){
    inframe[i].r = inframe[FFTLEN-i].r;
    inframe[i].i = -inframe[FFTLEN-i].i;
  }
}
*/
```