RTDSP Project Report (Option 1)

Kai Wen Yoke (CID 01409802), Shiyu Zheng (CID 01334805)

March 26, 2020

1 Abstract

We implemented beep detection using an adaptive frequency amplitude threshold and 'eleven' detection using correlation between frequency spectra. Informal testing showed that our algorithms work satisfactorily in non-noisy situations, but 'eleven' detection is unlikely to be robust to noise. For further improvement, we could combine a noise-elimination method with 'eleven' detection or implement a spectrogram-based method for more accurate 'eleven' detection in noisy conditions that has been proven to work effectively on Matlab.

2 Introduction

2.1 Description and analysis of sounds to be detected

We were given two sounds to be detected - a beep sound (A) and the sound of the lift saying 'eleven' (B). We analysed the time and frequency characteristics of the given clean sounds, and compared it with those of similar sounds.

2.1.1 Analysis of beep sound

The noisy beep sounds were extracted from the 4th to 5th seconds of the given recording 'journey_no_noise_8k.wav'. From Figure 1 showing the beep characteristics of clean and noisy samples, we conclude that the defining characteristic of a beep is a high amplitude at frequency 2585 to 2592Hz.

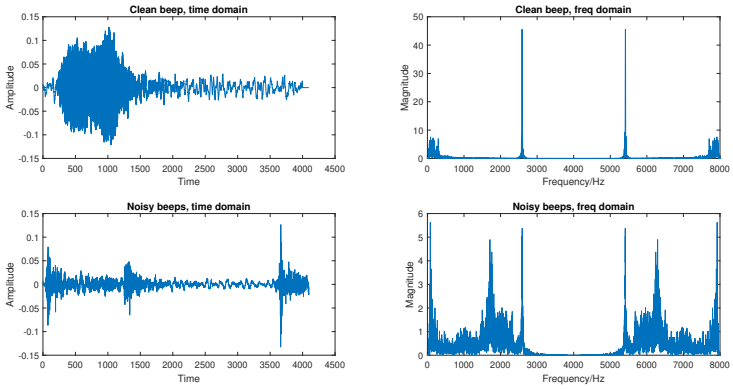

Figure 1: Time and frequency characteristics of the beep sound

2.1.2 Analysis of 'eleven'

However, the lift's 'eleven' is not as straightforward as it contains a wide range of frequency components. Some of the speech samples analysed in Figure 2 were taken at different points of the recording 'journey_no_noise_8k.wav' ('seven' and 'opening'), 'eleven_noisy_synthetic' was produced by adding talking noise to the 'eleven_clean', 'another_eleven' was our own recording of the lift saying 'eleven' and 'kw_eleven' was Kai Wen saying 'eleven'. Ideally, we expect only 'eleven_clean', 'another_eleven' and 'eleven_noisy_synthetic' to have similar characteristics. However, this is not very obvious at first sight from the frequency spectra shown in Figure 2. Instead, we notice that

different words spoken by the lift (first five rows of Figure 2) share similar frequency spectra since the lift's voice has its own characteristic frequencies, while it is clear that Kai Wen's voice saying 'eleven' exhibits quite different harmonics from the lift.

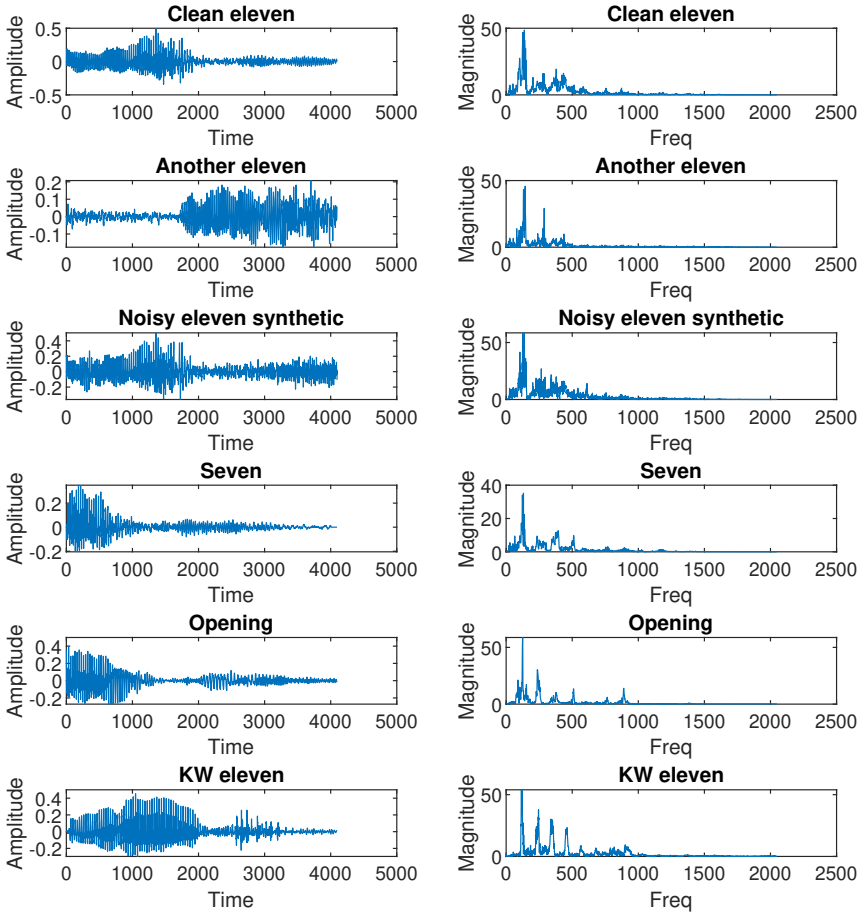

Figure 2: Time and frequency characteristics of different words spoken by the lift, and by Kai Wen

The spectrograms (Figure 3) combining both time and frequency characteristics are more revealing about the characteristics of different words, clearly showing the similarities in the high intensity pixels (two initially horizontal but subsequently downward sloping lines) for 'clean eleven', 'another eleven' and 'noisy eleven synthetic' spoken by the lifts (ignoring the time shift), which are markedly different from the rest of the words.

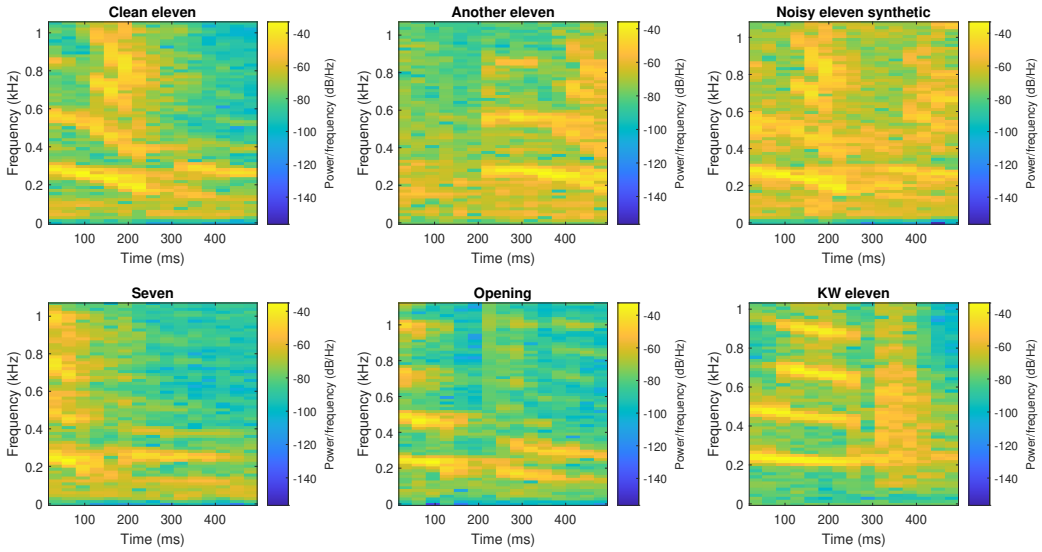

Figure 3: Spectrograms of different words spoken by the lift, and by Kai Wen

2.2 Survey of potential techniques

In our preliminary report, we considered the following methods: (1) setting a frequency amplitude threshold at the characteristic beep frequency, above which a beep would be considered present; (2) computing a similarity measure such as correlation or euclidean distance between the FFTs (Fast Fourier Transform) or Mel-Frequency Cepstrum Coefficients (MFCC) [1] of the clean and the input sample; (3) comparison between spectral peaks obtained from spectrograms inspired by music search engine Shazam [2]; (4) training a binary classifier using machine learning [3] with training dataset generated by ourselves.

2.3 Shortlisting and chosen solutions

We eventually implemented fully Method (1) and (2) on the DSK with mixed success (see results in section 5.1), while Method (3) was only implemented fully on Matlab but not yet working on the DSK. Below, we justify our thought process in shortlisting each proposed method.

2.3.1 Method (1): Frequency amplitude threshold

Since the beep sound contains just one characteristic frequency as expected, Method (1) is clearly a simple, feasible and effective technique for identifying beeps, and is hence chosen to tackle beep identification.

2.3.2 Method (2): Correlation between FFTs

While we were doubtful about the effectiveness of Method (2) given that from Figure 2, the FFTs do not appear to be very distinctive for the lift's 'eleven', we still tried implementing correlation to compare the FFT of the clean 'eleven' with the input sound, as a basic method to fall back on. Furthermore, method (2) is likely not robust to noise and should be implemented together with some noise suppression technique which we have yet to implement on the DSK.

2.3.3 Method (3): Comparing spectrogram characteristics

Method (3) seems promising from Figure 3 and is moreover resistant to noise. We tried implementing it on Matlab and were halfway through its implementation on the DSK before labs were terminated prematurely due to Covid-19.

2.3.4 Method (4): Training binary classifier with machine learning

We did not have time to try out machine learning, and we were also sceptical of its effectiveness since the range of negative training samples is large and could be anything from environmental noise to different speech samples spoken by different people.

3 Preliminary implementation of chosen solutions on Matlab

3.1 Method (1): Frequency amplitude threshold for beep detection

Method (1) involves checking if the amplitude of the frequency at 2586Hz (beep tone) is above a certain threshold to conclude if a beep is present. This works effectively if the threshold is chosen suitably. However, this threshold needs to be empirically determined, as will be further explained in the DSP implementation in section 4.

3.2 Method (2): Pearson correlation between FFT for 'eleven' detection

Method (2) involves computing the Pearson correlation $\rho(A, B)$ between the FFT of the clean 'eleven' (A) and that of the input (B) as shown in equation 1, where μ is the mean and σ is the standard deviation. We decided to use Pearson correlation as our similarity measure instead of euclidean distance because the Pearson correlation measures the linear relationship between two signals, and is insensitive to the exact amplitudes of the input signals (as can be seen from how each element in the input vectors are standardised by the mean and deviation in equation 1), meaning that two identical signals of different volumes would still return perfect correlation, but euclidean distance would be sensitive to signal amplitudes. This is good because we do not know how loud the volume of the input sound will be compared to the clean sample we have.

$$\rho(A, B) = \frac{1}{N-1} \sum_{i=1}^{N} \left(\frac{A_i - \mu_A}{\sigma_A} \right) \left(\frac{B_i - \mu_b}{\sigma_B} \right) \tag{1}$$

This is easily executed on Matlab in the following code snippet.

```
R = corrcoef(A, B);
correlation = R(1,2)
```

3.2.1 Preliminary testing

We initially used a window of 4096 samples to obtain a 4096-point FFT. This was because it takes roughly 0.5s for the lift to say 'eleven', so we wanted a frame just large enough to capture the entire 'eleven'. We calculated the correlation of the 4096-point FFTs of different sounds compared to that of the clean 'eleven' as shown in Table 1. Against our expectations when observing the FFT spectra in Figure 2, Table 1 actually shows that taking the correlation of frequency spectra works fairly well, probably because our eyes were not able to discern each component of the frequency spectra since speech spectra are quite complex. 'Noisy eleven synthetic' gives the

highest correlation as expected since it was the exact 'clean eleven' with noise added to it, while 'another eleven' and 'seven' both achieved scores above 0.7, much higher than 'opening' and 'KW eleven'. Since 'seven' sounds quite similar to 'eleven', it is a forgivable false positive given the crudeness of Method 2. Also as expected, words spoken by the lift correlates much better than words spoken by another speaker. However, for the most challenging test 'eleven_noisy' which was taken from the 47th second of the given noisy recording 'noisy_journey_8k.wav', the correlation score was only 0.532, thus showing that correlation does not work well in noisy conditions.

	eleven_ clean	another_ eleven	eleven_ noisy	eleven_ noisy synthetic	kw_eleven	seven	opening
Correlation	1	0.767	0.532	0.929	0.436	0.775	0.622

Table 1: Table showing correlation scores of different spoken words with 'clean eleven'

3.3 Method (3): Comparing spectral peaks from spectrogram

We leveraged on open source Matlab code ('Robust Landmark-Based Audio Fingerprinting') inspired by Shazam [4] (see zip file *project_matlab.zip* for full code).

3.3.1 Spectral peak identification

We identified local prominent spectral peaks in the spectrograms of 'clean eleven', 'another eleven', 'noisy eleven synthetic' and 'seven', and then formed pairs of nearby peaks, as shown in Figure 4. The number of spectral peaks found is dependent on a number of parameters associated with the calculation of the time and frequency-varying thresholds. Only pixels with energy above the thresholds will be considered, and out of these pixels, the local maxima, which are the spectral peaks, are found through differentiation. The spectral peaks are then paired with each other if they are sufficiently close in both time and frequency. As observed in Figure 4, it is clear that some of the spectral peak pairs are common, and furthermore they occur in the same relative sequence. Moreover, since spectral peaks are unobscured by noise unless the SNR is very low, this method is also resistant to noise and hence appears to be very promising.

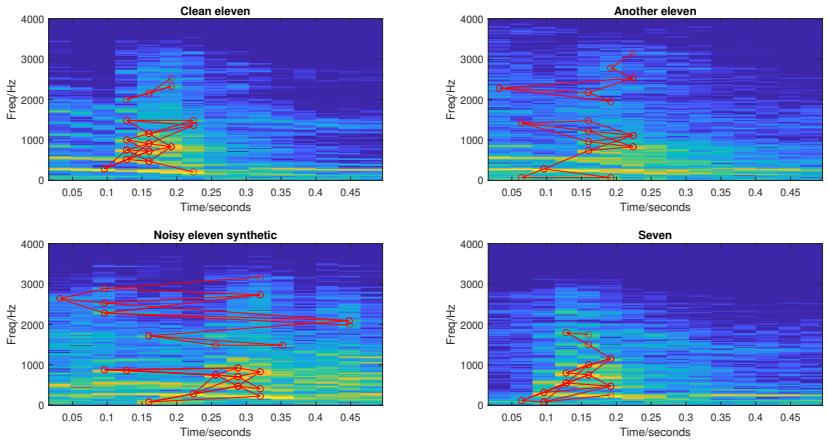

Figure 4: Identified spectral peak pairs on spectrograms of words spoken by the lift

3.3.2 Compressing spectral peak pairs to hashes

Each spectral peak pair is then represented in a 4 column vector $[t_1, f_1, f_2, \Delta t]$ and converted to a 20-bit hash value with 8 bits of f_1, 6 bits of Δf, 6 bits of Δt, each associated with the start time t_1. This allows each spectrogram of dimension 512×15 (512 frequency bins, 50% FFT overlap and 4096 time samples leads to 15 frames) to be represented with a number of 20-bits hashes and is thus memory efficient. The hashes of 'clean eleven' are then stored into a hash table.

3.3.3 Hash matching

To determine if an input window of 4096 samples is the lift saying 'eleven', we first count the number of matching hashes by accessing the hash table, and then out of the matching hashes, we count how many share the same $t_{1,input} - t_{1,database}$ since the time offset between hashes should match for an exact hit. The score is then the number of matching hashes corresponding to the most popular time offset.

3.3.4 Preliminary testing

We calculated the scores for the sounds 'eleven_clean', 'another_eleven', 'eleven_noisy', 'eleven_noisy_synthetic', 'seven', 'opening' and 'kw_eleven' in Table 2. It turns out that the algorithm is extremely sensitive to the exact sound sample, as it yields a high score only for 'eleven_clean' and 'eleven_noisy_synthetic' which was derived from the exact same 'clean eleven'. However, the algorithm does work but is less confident for 'another_eleven' and 'eleven_noisy' both of which are different recordings of the lift saying eleven, and unlike the correlation method, 'seven' is no longer falsely identified as 'eleven' and furthermore 'eleven_noisy' was identified despite the high level of noise. With further tuning of parameters related to thresholds for peak detection, this spectrogram based method is likely to accurately identify instances of the lift saying 'eleven' even at low SNR conditions.

	eleven_ clean	another_ eleven	eleven_ noisy	eleven_ noisy synthetic	kw_eleven	seven	opening
Score	64	4	1	21	0	0	0

Table 2: Table showing matching hashes score for different spoken words

Unfortunately, we did not have time to fully implement this method on the DSK, but while attempting to do so, we ran into problems of memory again. We were unable to process 15 frames of 512 frequency bins to generate the spectrogram, and hence we could not implement the sophisticated methods of dynamic threshold calculation used in Matlab which required global knowledge over the entire spectrogram. Perhaps it is also because the FFT is an expensive operation and doing it too many times stalls the DSK. In any case, we tried on Matlab to set a threshold which depended only on local knowledge of each frame, but it does not appear to work well. In the future, we could investigate further on how to optimise the code so as to fit within the memory and computing limitations of the DSK.

4 DSP implementation

4.1 Triple buffering scheme and 50% FFT frame overlap

We use a triple buffering scheme (input, intermediate and output buffers) to carry out frame processing as we do not need to calculate a 512-point FFT for every input sample, but instead only for every 256 samples. This is achieved by $while(index)$ that waits for $index$ of the input buffer to be reset to zero (see code snippet for function $ISR\_AIC()$)* before rotating buffers and doing frame processing. This reduces unnecessary computation. We used two 256-point buffers for each of input, intermediate and output which are initialised as shown in the function $init\_arrays()$. After every 256 samples, buffer rotation occurs (see code snippet of function $wait\_buffer()$) and a 512-point FFT is calculated for the two intermediate 256-point buffers $mid1, mid2$. This effectively achieves a 50% FFT frame overlap. We chose to do 50% frame overlap combined with a Hamming window (defined in $init\_arrays()$), because the envelope of overlapping windows always add to one and allows us to perform frequency domain processing on each frame without having spectral artefacts (which would have been introduced by the default rectangular window) and without losing power in the signal (because we overlap frames).

*$ISR\_AIC()$ code is different from the submitted C file because we realised we made a mistake, and should have only updated buffer in1[] only instead of switching between in1[] and in2[].

```
void ISR_AIC(void)
{
    float scale = 11585;
    sample = mono_read_16Bit();
    in1[index] = ((float)sample)/scale;
    //write new output data
    mono_write_16Bit((short)(out1[index]*scale));
    //update index and check for full buffer
    if (++index == BUFLEN/2){
        index=0;
  }
}

void init_arrays(void)
{

    in1         = (float *) calloc(BUFLEN/2, sizeof(float));
    ... /***repeat for in2, mid1, mid2, out1, out2***/

    //init hamming window
    for (i=0; i<BUFLEN; i++) {
        window[i] = 0.54 - 0.46*cos(2*PI*i/BUFLEN);
    }
}

void wait_buffer(void)
{
  float *p;
  /* wait for array index to be set to zero by ISR */
  while(index);
  /* rotate data arrays */
    p = in1; in1 = out2; out2 = out1;
    out1 = mid2; mid2 = mid1; mid1 = in2; in2 = p;

  /***** DO PROCESSING OF FRAME  HERE *****/
}
```

4.2 Beep detection

We determine if a beep is present by checking if the frequency amplitude at 2586 Hz exceeds a certain threshold.

4.2.1 Adaptive thresholding scheme

We designed an adaptive frequency amplitude threshold *thresh* instead of a fixed threshold. The problem with a fixed threshold (we initially used an empirically determined threshold of 0.12) is that we fail to identify a beep when the overall volume is low, and when the overall volume is loud, we pick up a lot of false positives, for example when someone is talking. To solve this problem, we set a small fixed threshold $thresh\_const = 0.05$ when there is little sound in the frame (dubbed 'silent frame'), and an adaptive threshold that is set to $\frac{max}{\alpha}$ where max is the maximum frequency amplitude in a non-silent frame and $alpha = 3$ is the divisor parameter. α was chosen experimentally to maintain an optimum balance between false positives and false negatives during beep detection. To determine if a frame is silent, we set another threshold $thresh\_sum = 0.5$ which is the value that the sum mag_sum of all frequency magnitudes in a 512-point* FFT frame mag_spec has to exceed for a frame to be considered non-silent. The reason why we could not directly implement just the adaptive threshold was because a lot of false positives were generated when the frame was silent, since the maximum magnitude of a frame would be very small and random noise around 2590 Hz would yield false positives, hence we had to set a constant threshold at silent frames to stop such false positives. The code snippet is shown below.
*Number of frequency bins arbitrarily chosen; beep detection works well with at least 128 bins.

```
void detect_beep(float mag_spec[]) {
    //check for silent frame
    mag_sum = 0;
    for (i=0; i<BUFLEN; i++) {mag_sum += mag_spec[i];}
    if (mag_sum > thresh_sum) { silence = 0;}
    else {silence = 1;}

    //if not silent, thresh is max/alpha
    max = 0;
    if (!silence) {
        for (i=0; i < BUFLEN; i++) {
            if (mag_spec[i] > max) {max = mag_spec[i];}
        }
        thresh = max/alpha;
    }
    else {thresh = thresh_const;}

    beep_amp = mag_spec[fbin];
    if (beep_amp > thresh) {present = 1;}
    else {present = 0;}
}
```

4.3 'Eleven' detection

We implemented Pearson's correlation between FFT frames of 'clean eleven' and the input according to equation 1. We initially planned to use a 4096-point FFT like what we did in Matlab (see section 3.2) but we ran into memory issues. After experimentation, we decided to use a 512-point FFT to represent each input 4096-samples-window as it fits within the memory limits and has high enough resolution for correlation to work well. There is also the problem of trying to

capture the entire input word 'eleven' in a 4096-window instead of only fragments of 'eleven' so that the accuracy of the correlation score would not be affected.

4.3.1 Memory limitation: Representing 'eleven' in 512 frequency bins

To represent each 4096-window in a 512-point FFT, we take the FFT of every 512 time samples (windowed by Hamming window) with a 50% (i.e. 256 samples) overlap until the FFTs cover the entire 4096-window. This means that each 4096-window requires computation of 15 FFT frames, which are all added together to finally produce a 512-point feature vector that represents all frequency components present in a 4096-window. We first compute on Matlab the 512-point feature vector for 'clean eleven' and store it in a txt file *elevenfft.txt* (together with its mean and standard deviation used for calculating correlation) using the code snippet shown below, and include the txt file in our C file *detect_sound.c* for comparison with the input 4096-window.

```
%% 512-sample frames, 50% overlap, span over 4096 samples
fftbins =512;
%frame overlap
clean = eleven;
N = length(clean);
window = hamming(fftbins);
spec = zeros(fftbins,1);
for w =1 :15
    frame = clean(fftbins/2*(w-1)+1:fftbins/2*(w+1));
    frame_w = window.*frame;
    spec = spec + abs(fft(frame_w, fftbins));
end

% save eleven into txt file for reading into c
handle = fopen('elevenfft.txt', 'wt');
fprintf(handle, 'float meanA = %e\n', mean(spec));
fprintf(handle, 'float stdA = %e\n', std(spec));
fprintf(handle, 'float elevenfft[] = {');

for i = 1: length(spec)-1
    fprintf(handle,'%e, ',spec(i));
end

fprintf(handle, '%e}; \n', spec(length(spec)));
fclose('all');
```

In our *detect_code.c* file, the feature vector is calculated in the function $find\_fft()$. We reset the 512-point buffer *spec*[] to zero after every 15 frames. Then for every frame, we apply a Hamming window, take the FFT and accumulate it in *spec*[].

```
//find fft of all four specs (only one spec shown here for illustration)
void find_fft() {
    //reset spec to zero after every 15 frames
    if (index2 == 0) {
        for (i=0; i<BUFLEN; i++) {spec[i] = 0;}
    }
    //windowing
    for (i=0; i<BUFLEN/2; i++) {
        m1[i] = window[i]*mid1[i];
        m2[i] = window[i+BUFLEN/2]*mid2[i];
    }
    //copy values to complex variable to do fft
    for (i=0;i<BUFLEN/2;i++)
```

```
    {
        C[i] = cmplx(m1[i],0);
        C[i + BUFLEN/2] = cmplx(m2[i],0);
    }
    fft(BUFLEN, C);
    for (i=0; i<BUFLEN;i++) {spec[i] += cabs(C[i]);}
}
```

4.3.2 Overlapping windows: Ensuring each entire 'eleven' is captured in one frame

We implemented a 75% window overlap so that when 'eleven' is being said, it would be mostly captured by at least one window. This means that we simultaneously keep four 512-point buffers $spec[], spec2[], spec3[], spec4[]$ that represent the features of four overlapped windows as illustrated in Figure 5. We keep four indices, $index2, index3, index4, index5$ to keep track of how many of the 15 FFT frames have been calculated for each 512-point buffers as shown in the code snippet of function $ISR\_AIC()$. Each index is reset to zero whenever 15 FFTs for the corresponding 512-point feature vector have been computed. We also ensure that each index is staggered by 4 frames ($\frac{15}{4}$) from the index of the preceding buffer to achieve 75% overlap.

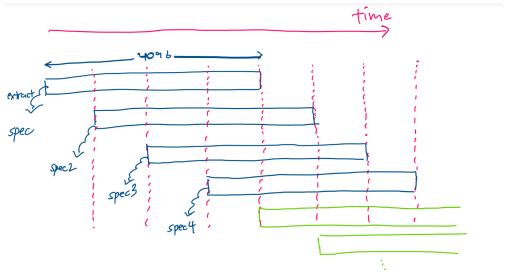

Figure 5: Diagram for 75% window overlap

```
void ISR_AIC(void)
{
    ...
  //update index2 to index5 to achieve 75% overlap for buffers spec to spec4
  //add up 15 FFT frames (50% overlap) to form one FFT to represent 4096 input
    window
  if (index2 == 14) {index2 = 0;}
  else {index2 ++;}

  if (index2 == 3 || index3 == 14) {index3 = 0;}
    else {index3 ++;}

  if (index3 == 3 || index4 == 14) {index4 = 0;}
    else {index4 ++;}

  if (index4 == 3 || index5 == 14) {index5 = 0;}
    else {index5 ++;}
}
```

4.3.3 Implementing correlation and determining correlation threshold

We compute the correlation of the 512-point feature vector of the input 4096-window (B) and the pre-stored feature vector of 'clean eleven' (A) (see *calc_corr_eleven*(*mag_spec*) in the code snippet below and refer to equation 1). We determine an "eleven" to be present if the correlation value *corr_eleven* is greater than a set threshold $corre\_thresh = 0.9$.

```
void calc_corr_eleven(float mag_spec[]) {
    float sumB= 0;
    for (i=0; i<BUFLEN; i++) {
        sumB += mag_spec[i];
    }
    meanB = sumB/BUFLEN;
    stdB=0;

    for(i=0;i<BUFLEN;i++){
        stdB += pow(mag_spec[i]-meanB,2);
    }
    stdB = sqrt(stdB/BUFLEN);

    SUM = 0;
    for(i=0;i<BUFLEN;i++){
        SUMB = mag_spec[i]-meanB;
        SUMA = elevenfft[i]-meanA;
        SUM += SUMA*SUMB;
    }

    corr_eleven = SUM/stdB/stdA/(BUFLEN-1);
}
```

4.4 Making LEDs light up for one second when beep, 'eleven' detected

We light up LED(3) or LED(2) for one second whenever 'eleven' or beep is detected respectively. To do this, we keep a counter each for beep and 'eleven'. If detected and the current counter is zero, the corresponding LED will light up and the counter will start counting up until 8000, after which the LED turns off and the counter resets to zero.

```
void ISR_AIC(void)
{
    ...
  //detect eleven and light LED for one second
  if (corr_eleven>corre_thresh && count11_1sec==0) {
        count11_1sec = 1;
    }

  if (count11_1sec > 0 && count11_1sec < 8000) {
      DSK6713_LED_on(3);
      count11_1sec++;
  }
  else if (count11_1sec == 8000) {
      DSK6713_LED_off(3);
      count11_1sec = 0;
  }
    .../***repeat for beep***/
}
```

5 Conclusion

5.1 Results

We were unable to do rigorous testing as labs were suddenly terminated due to Covid-19, but below we detail what we could remember from our informal testing.

5.1.1 Beep detection

We tested our beep detection on the given sample *journey_no_noise_8k.wav*, and all beeps, with and without background talking, can be detected if the volume of the computer's speaker is set to be larger than 35. However, if the speaker volume is below 35, the beep cannot be detected. There is also a screeching noise which is also around 2600Hz in the recording that triggers a false positive.

5.1.2 'Eleven' detection

Testing on *journey_no_noise_8k.wav* and other non-noisy self-recorded lift recordings, the results for detecting "eleven" were decent. Nearly all true "eleven" can be correctly determined but one was missed, and other people saying 'eleven' do not trigger false positives. However, there were also some false positives, such as "Seven", which has a frequency spectrum very close to "Eleven".

5.2 Future directions

Our beep detection is already quite robust but 'eleven' detection is still quite lacking. We have not tested 'eleven' detection in noisy situations, but in theory correlation is unlikely to work well, unless we implement an effective noise elimination algorithm. Given the opportunity, we would also continue working on our spectrogram based method for 'eleven' detection.

References

[1] J. Lyons, "Mel frequency cepstral coefficient (mfcc) tutorial." [Online]. Available: http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/

[2] A. L. chun Wang and T. F. B. F, "An industrial-strength audio search algorithm," in *Proceedings of the 4 th International Conference on Music Information Retrieval*, 2003.

[3] M. Baelde, C. Biernacki, and R. Greff, "Real-time monophonic and polyphonic audio classification from power spectra," *Pattern Recognition*, vol. 92, p. 82–92, 2019.

[4] D. Ellis, "Robust landmark-based audio fingerprinting," Nov 2009. [Online]. Available: https://uk.mathworks.com/matlabcentral/fileexchange/23332-robust-landmark-based-audio-fingerprinting