RTDSP Preliminary Report

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

February 15, 2020

1 Introduction

In this preliminary report, we present a list of shortlisted Digital Signal Processing (DSP) methods for detecting in real-time specific sounds in an audio input taken under typical lift conditions, at a sampling frequency of 8kHz. These sounds are less than 1 second long, and belong to two categories - (A) prerecorded speech saying 'First/Second... floor' or (B) a beep from the user pressing a button. We then justify our choice of methods by considering their accuracy, robustness to background noise, speed and calculation burden.

2 Methods

2.1 Characterising input sound types

The method for detecting sound type (B) is simple. Each button beep contains a characteristic set of a few pure tones, which we can easily detect by doing FFT and checking if the amplitudes at the characteristic frequencies exceed a certain threshold. Alternatively, we can apply a filter with multiple passbands at the characteristic frequencies, and check if the amplitude of the output exceeds a certain threshold.
Sound type (A), a recording of a specific phrase spoken by a specific voice, is more difficult to detect. The speech recording contains a wide range of frequency components, and furthermore the environmental background noise will often contain human speech which will have frequency components overlapping with the lift speech. Therefore we turn to more sophisticated methods to detect sound type A.

2.2 Overview of methods (primarily to detect sound type (A))

We break down the task of detecting sound type(A) into three stages: (1) pre-processing (2) feature extraction and (3) matching.
Note that to execute each method in real-time, each audio input is processed in terms of overlapping windows and each Fast Fourier Transform (FFT) is executed over each window frame. The window size, extent of overlap and the number of frequency bins are to be empirically determined.

2.3 Pre-processing stage: Noise elimination

To improve matching results, we first suppress unwanted noise from the audio input and retain the desired signal. A basic method would be to set an amplitude threshold such that all frequencies below the threshold are suppressed (small gain) while frequencies above are left intact (gain = 1), in a process known as noise gating. Alternatively, we can estimate the background noise by assuming that every duration of 1s will contain periods of silence. This is a reasonable assumption because we know that both sound types A and B are less than 1s long. We then find in real-time the minimum amplitudes for each frequency bin in that 1s duration and subtract the estimated noise spectrum from the input audio using an oversubtraction factor.

2.4 Feature extraction stage

The choice of features to represent the audio sample is vital to the accuracy of matching. Furthermore, feature extraction allows us to represent the input audio in a more compact manner, thus reducing processing load.

2.4.1 Frequency spectrum

A simple set of features would be the magnitude of the frequency spectrum of the audio input. We compute the frequency spectrum of each frame over a duration of 1s by FFT and average the frequency spectra to get a feature vector. We then apply min-max normalisation to ensure that audio inputs of different volumes would have the same features. Each speaker has a characteristic frequency spectrum, thus making it a good feature vector.

2.4.2 MFCC

Instead of using the raw frequency spectrum, the raw spectrum can be transformed through frequency warping: $Mel(f) = 1125ln(1 + \frac{f}{700})$, log and cosine transforms to produce the Mel-Frequency Cepstrum Coefficients (MFCC) which models human auditory perception. [1] MFCC is a popular feature used in speech recognition, and might prove useful in our task.

2.4.3 Spectrogram characteristics

However, having just the frequency spectrum ignores temporal characteristics which are significant in sound type (A) as each syllable occurs in a sequence. Hence, extracting features based on the audio input's spectrogram which contains both time and frequency information would be more effective. This technique is used by the Shazam music search engine [2]. Shazam compresses the spectrogram of the audio input into a constellation of spectral peaks defined by their time and frequency coordinates (t, f) (see Figure 1). Each peak is then paired with neighbouring peaks, and then a hash is formed from each pair by taking $(f1, f2, t_2 - t_1)$. Therefore, each audio input is now represented by a series of hashes based on spectral peak pairs. This method is resistant to background noise and works very well at low SNR as only spectral peaks are considered. However, because sound (A) is very short, too few hashes might be captured to uniquely identify the specific sound.

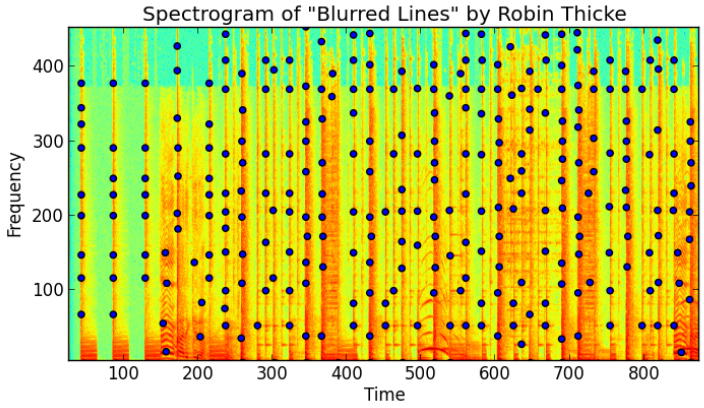

Figure 1: Constellation of spectral peaks on spectrogram for the song 'Blurred Lines' [3]

2.5 Matching stage

We identify the audio input by comparing its features with features of clean sounds in a database.

2.5.1 Distance/Similarity metric

The first approach requires careful choice of a similarity metric, which also depends on the features used. If the frequency spectrum or MFCC is used, a basic approach would be to calculate the Euclidean distance, $d(x, y) = \sqrt{\sum_{i=1}^{N}(x_i - y_i)^2}$ between the feature vector of the input x and the clean audio y in the database. We could also use Pearson's correlation coefficient, $\rho(x, y) = \frac{1}{N-1}\sum_{i=1}^{N}(\frac{x_i-\mu_x}{\sigma_x})(\frac{y_i-\mu_y}{\sigma_y})$ to assess the similarity of the two feature vectors.

2.5.2 Shazam hash matching

With Shazam's features of spectral peak pairs, we can use an efficient matching algorithm. The hashes $(f1, f2, t_2 - t_1)$ of the audio input are compared with those of the clean sound (A) sample in the database. For every exact hash match, t_1 of both the input and the database sample are stored in time pairs, the rationale being that if the clean sample sound (A) matches with the input, the time difference between the t_1 time pairs should be the same for all matching hashes. The similarity score is then the largest number of time pairs with the same time difference. This method is potentially very fast since each hash is small and easy to compare.

2.5.3 Machine Learning

A binary classifier to determine if the audio input contains sound (A) could be trained. This would require preparation of a training dataset with positive and negative samples - the former can be achieved by synthetically adding realistic environmental noise to the clean sound (A) sample and the latter by just having the environmental noises. Then, we extract features (e.g. frequency spectrum, MFCC) from these labelled synthetic training audio samples. For the learning model, we can use a Support Vector Machine (SVM), which is relatively simple to implement, or more complex neural networks which other researchers have proven to be useful for audio event detection and classification [4]. Training offline could potentially be time consuming, but once the model is trained and optimal parameters obtained, testing it with an audio input in real-time should be possible.

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] W. Drevo, "Audio fingerprinting with python and numpy." [Online]. Available: https://willdrevo.com/fingerprinting-and-audio-recognition-with-python/

[4] 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.