Separation of speech mixture using time-frequency masking implemented on a DSP

Javier Gaztelumendi and Yoganathan Sivakumar

March 13, 2017

1 Introduction

This report describes the implementation of a blind source separation algorithm, used for speech separation, on a DSP. The method used in this work is the time-frequency masking, which is a method well suited for human speech applications. The process consisted on implementing and refining the algorithm on Matlab first and later adapting it for an Analog Devices ADSP signal processor.

Blind source separation (BSS) consists on recovering two source signals from two or more mixed signals that contain both signals, often without knowing the mixture parameters.

The speech separation and, more generally, blind source separation algorithms, have many applications in areas such as hearing aids, simultaneous translation, communications, and many more.

2 Background

2.1 Fourier Transform

The Fourier transform is a mathematical function that allows toconvert signals from the time domain to the frequency domain, allowing different forms of treatment signal treatment. When implementing it in digital systems, the Fast Fourier Transform (FFT) algorithm is usually used, as it is less computationally expensive than a direct digital implementation of the Fourier transform. This algorithm imposes the limitation of only working with 2^N samples.

2.2 Inverse Fourier Transform

The inverse Fourier transform is the inverse operation of the Fourier transform, which allows to convert signals from the frequency domain to the time domain. As the original transform, there is also an Inverse Fast Fourier Transform (IFFT) that allows faster processing when working digitally.

2.3 Spectral Leakage

The Fourier transform algorithm assumes that the sampled signal is periodic, which is usually not the case when working with audio signals. This causes the so called spectral leakage, where false frequency peaks are created by this non-periodicity.

2.4 Hanning Window

To avoid spectral leakage, windowing functions can be used. A window function is a mathematical function that is zero-valued outside of a chosen interval. When another function/data is multiplied by this function, the result is also zero outside of the chosen interval, hence, like looking at it through a window.

Using this window functions reduces spectral leakage, because they will give more relative importance to the samples in the middle of each block rather than to the ones on the sides, where discontinuities happen. This can be seen as "smoothing" the transitions between consecutive blocks.

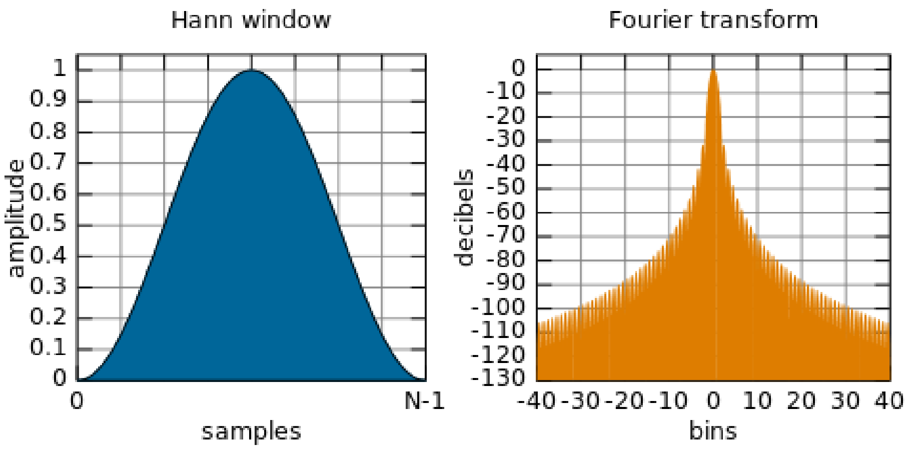

Figure 1: Hanning window and its Fourier transform (Wikimedia commons)

2.5 Overlap addition method

Using a Hanning window to reduce leakage introduces a new problem: loss of information. After applying the window function, the frequency bins at the sides of the spectrum will get reduced, which would cause information loss when applying the IFFT. Overlap addition consists on making some of the samples in consecutive blocks to overlap. For example, a 50% overlap would mean that the new block begins at the middle of the previous block, therefore overlapping with the second half of the previous block. Doing this after applying the windowing function to each block reduces the loss of information while maintaining a clean spectrum.

3 Algorithm

For demonstartion purposes, an algorithm to separate two speakers using two microphones is implemented.

3.1 Assumptions

The main assumption is that one signal will arrive to both microphones with almost the same magnitude, given that both microphones are closely spaced. However, each signal will arrive first to one microphone, getting picked by the other with some amount of delay. This would only work if the speakers are not perpendicular to the microphones, and will give better results when both speakers are facing each other, with the microphones in the middle.

The second assumption in order to apply time-frequency masking is that the sources are sparese. This means that most of the frequency bins have no energy during most of the time, which holds true for human voice: even if most of the time-frequency information were removed from a speech signal, it would still be possible for humans to distinguish the message.

Finally, it is assumed that the sources are disjoint, this is, their time-frequency bins do not overlap. Again, this is mostly true for human speech, where two different voices will rarely have the same time-frequency components.

Assuming this conditions, it is possible to separate the bins between mixed sources.

3.2 Algorithm

Let x and y be the mixed signals acquired by both microhpones, and s_1 and s_2 the original signals. The microhpone signals can be represented as:

$$x = s_1(t) + s_2(t+\tau) \tag{1}$$

$$y = s_1(t+\tau) + s_2(t) \tag{2}$$

The first step after capturing the signals, is to apply the windowing function and perform the Fourier transformation for both of them, getting their frequency domain representation:

$$X(\omega) = S_1(\omega) + S_2(\omega) \cdot e^{j\omega\tau} \tag{3}$$

$$Y(\omega) = S_1(\omega) \cdot e^{j\omega\tau} + S_2(\omega) \tag{4}$$

Now, it is possible to find the correlation between both signals in the frequency spectrum using cross spectrum analysis. In this case, we calculate the complex cross spectrum, defined as the multiplication of one spectrum by the complex conjugate of the other one:

$$H(\omega) = X(\omega) \cdot \overline{Y(\omega)} \tag{5}$$

$$H(\omega) = S_1(\omega)\cdot\overline{S_2(\omega)}\cdot e^{j\omega\tau} + S_1(\omega)\cdot\overline{S_1(\omega)}\cdot e^{j\omega\tau} + S_2(\omega)\cdot\overline{S_2(\omega)}\cdot e^{j\omega\tau} + S_2(\omega)\cdot\overline{S_1(\omega)}\cdot e^{j\omega\tau} \tag{6}$$

The result of this operation yields a complex spectrum, where the real part represents the co-spectrum, this is, the in-phase parts of both signals, and the imaginary part represents the out-of-phase signal. This imaginary part is what allows the algorithm to identify and classify each time-frequency point as belonging to either signal: when the delay is positive it will correspond to the first signal, and when it is negative, to the second signal. Taking into account the assumption that only one of the signals is considered to be active at each time-frequency point, the original signal can be reconstructed by classifying the input signal at any time-frequency point:

$$\begin{cases} S_1(\omega) = X(\omega) & im(H(\omega)) > 0 \\ S_1(\omega) = 0 & im(H(\omega)) < 0 \\ S_2(\omega) = X(\omega) & im(H(\omega)) < 0 \\ S_2(\omega) = 0 & im(H(\omega)) > 0 \end{cases}$$

Once the frequency components are classified between both signals, it is possible to reconstruct the originals using the IFFT algorithm. The algorithm here presented yields reasonable results, however, it is still very rudimentary, and some of the assumptions it relies on are not true in real world environments.

3.3 Matlab implementation

The algorithm was first implemented in Matlab, in order to prove its validity and to gain understanding on how it works. A mixing function provided by our teacher was used to simulate the mixture of two audio signals in two microphones.

As the goal was to teset the algorithm rather than making a final implementation, Matlab's built-in functions were used, as well as matrix processing, in order to simplify and accelerate the script. When the results were deemed acceptable, the algorithm was finally implemented on a DSP.

3.4 DSP implementation

The main difference between the Matlab and DSP implementations is that the DSP works with blocks of a fixed number of samples, in our case 256 sample blocks. In order to perform real time processing, one block is read, processed and sent to the output before the next block is processed. In order to process a block, each sample has to be processed iteratively, for example, inside of a for loop, which was not necessary in Matlab due to direct matrix multiplication notation.

In order to achieve a 50% overlap, 128 sample input blocks were used, combining them with the previous block, to form effective 256 sample blocks, were half of the samples were overlapped.

The DSP libraries already included most of the needed functions, suchas as FFT and windowing, making the implementation rather simple. The provided framework was modified to get a sampling frequency of 16 kHz and 128 sample input blocks

4 Problems found

No big problems where found during the process, however, some details had to be taken into consideration for a correct implementation on the DSP.

The first was that the FFT function returns the data corresponding to the mirrored spectrum for frequencies over Nyquist. This was solved by flipping the second half of the spectrum and substituting the values for their complex conjugates. This fixes the symmetry and ensures a correct output.

Another detail was that some overlap had to be applied in order to get smoother results. This was solved using a 50% overlap, meaning that from the 256 samples that form each block, the first 128 correspond to the last half of the previous block, and the last 128 are new samples.

When programming the DSP, some difficulties were found for flashing it with the program and making the buttons work. Finally it was deemed as a faulty DSP, so the program should work in a correctly working DSP.