Contents

Jaafar Rammal 01576194 (jr4918@ic.ac.uk)
Karl-Kareem Melaimi 01355022 (kkm51@ic.ac.uk)

Introduction

The following report aims to present considered solutions and implementations to tackle a real-time DSP task: removing vocal components from a music track. The problem constraints can be broken down into performance and implementation constraints. For performance, it is crucial to achieve an acceptable vocal removal in real-time within a margin of a 2-seconds delay. For the implementation, the solution is bounded by the TMS320F28379D board [10] hardware specs.

The following sections will first cover the general solution proposal, where the problem is broken down mathematically and the different solution components are identified, before describing the general implementation pipeline in a Simulink context. Then, different mask generation methods are explored from a theoretical and practical point of view, before concluding about the outcome of the project overall.

1 General solution

This section will cover the proposed solution pipeline in general, through defining the problem mathematically and in a DSP context, before breaking it down into different manageable components that can potentially be implemented in Simulink. The problem can be generalised as a single channel blind source separation problem (SCBSS), where we are decoupling the voice (or speech) from the rest of the music with no prior information of the music.

1.1 Mathematical definition

Referring to [2], the problem can be defined as follows: assume an input signal $x(t)$ is observed as the sum of two sources $s(t)$ and $m(t)$ (respectively source and music). The goal is to identify frequencies and classify them as belonging to $s(t)$ or $m(t)$ in order to suppress the signal $s(t)$. Therefore, the next step is to represent the input signal $x(t)$ in the frequency domain. The Short-Time-Fourier-Transform (STFT) can be used to visualize the time-frequency representation of the input signal. The function, available in Matlab and Simulink, has customizable Hann window, Fast Fourier Transform (FFT), and overlap lengths. Let $X(t, f)$ be the STFT of $x(t)$, where t represents the frame index and f is the frequency-index. Due to linearity, this translates into

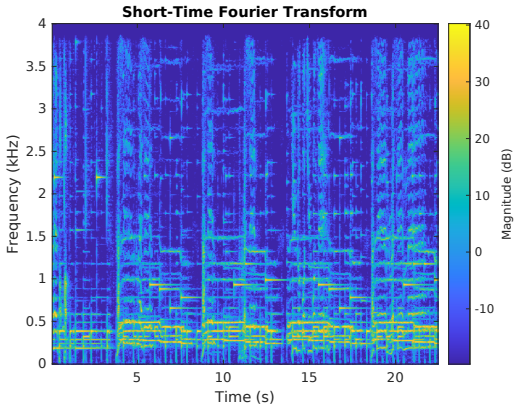
$$X(t, f) = S(t, f) + M(t, f) \tag{1}$$

$$|X(t, f)|e^{j\phi_{X(t,f)}} = |S(t, f)|e^{j\phi_{S(t,f)}} + |M(t, f)|e^{j\phi_{M(t,f)}} \tag{2}$$

It is reasonable to assume that the input signal and its sources are in phase, with $\phi_{X(t,f)} = \phi_{S(t,f)} = \phi_{M(t,f)}$. This yields to the following equality

$$X = S + M \tag{3}$$

where X, S and M are the respective magnitude spectograms of the input, speech, and music signals. S and M are unknowns and need to be estimated through $\widehat{S}$ and $\widehat{M}$, leading to an approximation $X \approx \widehat{S} + \widehat{M}$. Once the estimation is completed, $\widehat{S}$ can be discarded and therefore vocal removal from a song is achieved.



Assume, through an established method, a mask H is generated from a training set $X_i = S_i + M_i$, such as the approximation error of $X \approx \widehat{S} + \widehat{M}$ is minimized. The mask is related to the estimations such as:

$$\widehat{S} = H \otimes X \tag{4}$$

$$\widehat{M} = (1 - H) \otimes X \tag{5}$$

It is therefore enough to compute the mask matrix H to almost remove the speech signal S, through an element-wise

multiplication, from the input X to obtain the final output Y. This is equivalent to setting $Y = \widehat{M}$:

$$Y = \widehat{M} = (1 - H) \otimes X = X - (H \otimes X) \tag{6}$$

1.2 Implementation pipeline

The proposed mask-based solution can be implemented in Simulink using a wrapper of transforms around a mask generator to correctly deliver the input and output. Two different stages are involved: (1) the training stage where the generator learns to estimate a mask as seen in Fig. 6, and (2) the execution stage (running on the TMS320F28379D board) which will estimate a mask with minimal approximation error as seen in Fig. 2. Stage (1) is not bounded by a timing deadline and can run asynchronously, while stage (2) will be running following the task constraints and generating an output in real-time. The playable output $y(t)$ can be simply obtained by applying an Inverse-STFT on Y.

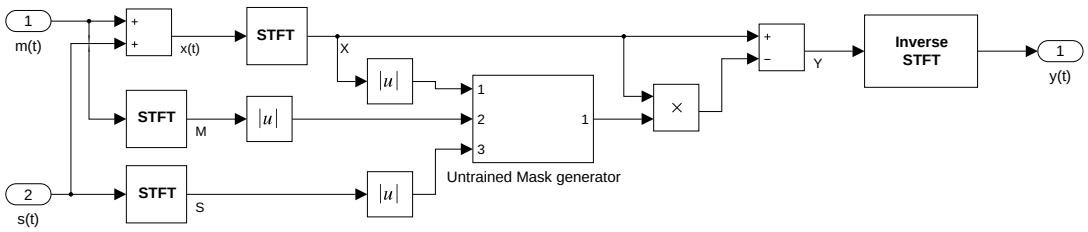


Figure 1: Mask generator asynchronous training stage

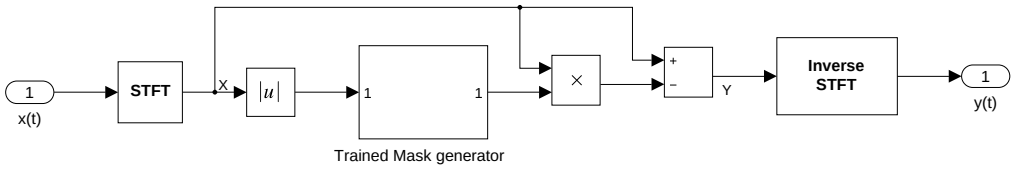


Figure 2: Mask generator real-time execution stage

1.3 Performance evaluation

To evaluate the performance of an algorithm, it is essential to have a baseline to compare the results to. In the case of the problem, the perfect solution would be the original version of the song without the vocals, or the output of a perfect software vocal removal. Therefore, it was decided to generated a large database of perfect samples (songs with vocals removed) and compare any algorithm output to the perfect samples. The database presents 5 different files for each song: the original track, the vocals track, the drums track, the bass track, and the "others" track (containing any leftover sounds). Overall, about 60 songs were used from the provided coursework database [1].

Provided the different subtracks of a song, it is now easy to evaluate the PC performance of an algorithm by checking the MSE (Mean Squared Error) between a generated output and an expected output. The MSE is applied on the STFT of both tracks (generated and expected), making it possible to also tune different parameters in the algorithm by picking combinations that result in a lower MSE. Given an expected output vector $\hat{Y}$ and a generated output vector Y, it is possible to compute the MSE using the mathematical formula 7:

$$MSE = \frac{1}{n}\Sigma(Y_i - \hat{Y}_i)^2 \tag{7}$$

With two tracks *filtered_output* (generated from *original_input*) and *perfect_vocals*, this corresponds to the Matlab code:

```
computed_vocals = original_input - filtered_output;
D = abs(perfect_vocals - computed).^2;
MSE = sum(D(:))/numel(perfect_vocals);
```

The DSP performance evaluation is a bit trickier since the provided speaker already generates some noise and does not offer a high quality output. Two evaluations are used: the first one is manual and biased, where the user listens to the output and judges the performance of the DSP. The second one is slightly more robust and makes use of a spectral analyser. The spectral output of the DSP is video-recorded (using screen-recording on a mobile phone) and the corresponding spectral output of the perfectly filtered track is also recorded. The two videos now contain the FFT of the DSP

output at any point in time and can be compared.

It is worth noting that a more mathematical approach was tried at first for the DSP performance evaluation. Just like the PC performance, it would have been possible to calculate the MSE provided the DSP could output in a debugging console the STFT of the generated output before converting it back to the time-domain and play the samples. A lot of time has even been dedicated to trying the CCS (Code Composer Studio) environment to print results to a debugging console from the DSP. The experience unfortunately proved unsuccessful mainly due to memory-map issues on the DSP.

2 Low-pass filter

The following section covers a very basic approach to the problem using an FIR filter (Finite Impulse Response) which implements an LPF (Low Pass Filter).

2.1 Theory

The easiest vocal removal method would be to simply suppress signals in the frequency range of the human voice. While the human voice ranges from 20 Hz to 20 KHz, the narrow band of the human voice, which is the usable voice frequency band, ranges from approximately 300 Hz to 3400 Hz. It is therefore possible to implement an FIR that suppresses this frequency range to remove most of the human voice while keeping some music samples that fall outside this range (bass, drums, and some of the piano / guitar).

2.2 PC Implementation & Validation

It is possible to test different frequency ranges using the *bandpass* command in Matlab and supplying different frequencies. It was quickly established that using a low-pass filter instead would produce better results for the 3 songs in question for this project. The code used to generate the output is as follows:

```
lpf_frequency = 300;
[y, Fs] = audioread('[Path To Audio File]');
lpf_y = bandpass(y, [0,lpf_frequency], Fs);
sound(lpf_y, Fs)
```

It is then possible to either listen to the filtered output *lpf_y* or take the STFT and compare it with the STFT of the perfect filtered output to get the MSE value. Fig. 3 shows the average MSE obtained over different songs for different frequencies. The best MSE achieve is 0.201 at a cutoff frequency of 500 Hz. While 0.201 is not an amazing result, it is a pretty good achieved filtering given that it only uses a simple filter and could be easy to implement on the DSP.

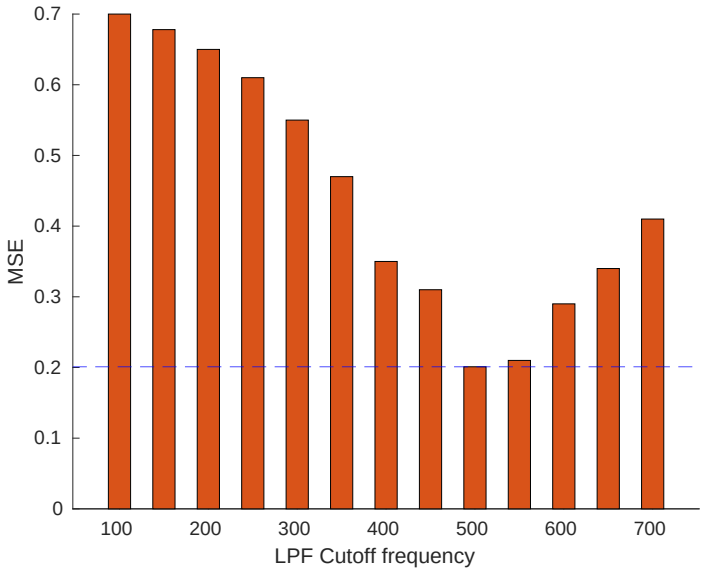


Figure 3: Average MSE between perfect filtered output and LPF output for different LPF cutoff frequencies

2.3 DSP Implementation & Validation

Implementing an LPF on the DSP is pretty straightforward and follows a process similar to that of Lab 2 in the module for implementing an FIR. For the filter block in Simulink, the following parameters are used:

- Filter type: FIR
- Passband edge frequency: 500 Hz
- Stopband edge frequency: 600 Hz
- Maximum passband ripple: 5 dB
- Minimum stopband attenuation: 80 dB
- Sample rate: 8000 Hz

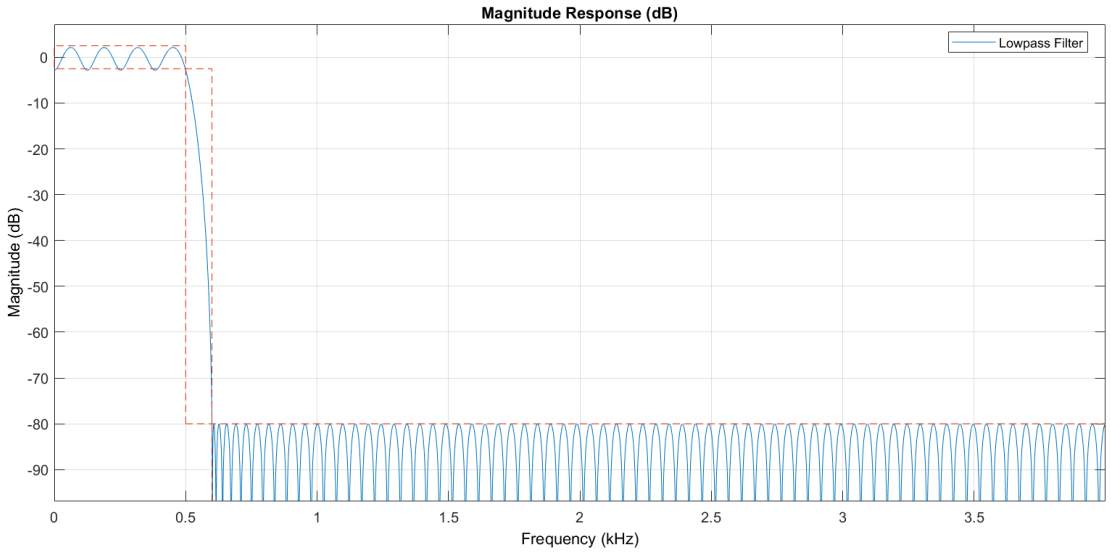


Figure 4: FIR filter response for the given parameters

Fig. 4 shows the filter response using the "Low-pass filter" block in Simulink to design and FIR filter. The block is then used within the provided player file as seen in Fig. 5.

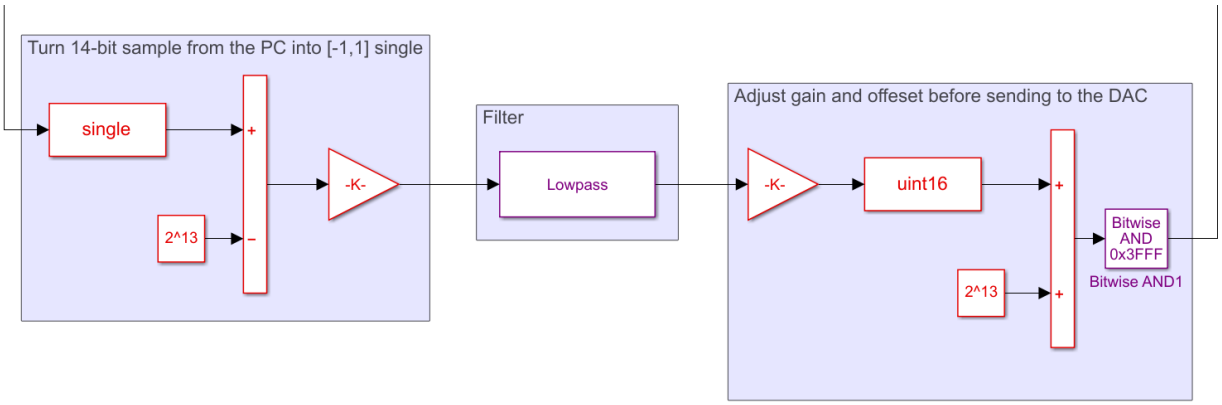


Figure 5: Including the filter in the DSP pipeline

The model successfully compiles and runs on the DSP. The output, as expected, does not fully filter out human voices (the MSE was computed during the PC evaluation to 0.201), but a pretty good chunk if filtered out while most of the instrument sounds remain and can still be heard. A quick use of the spectogram mobile application shows that most of the spectrum lays in the frequency range under 570 Hz, with some noise in higher frequencies, which validates the functionality of the DSP.

3 Non-Negative Matrix Factorisation

This section explores the different ways Nonnegative Matrix Factorisation (NMF) can be used to build a mask generator in the proposed solution pipeline.

3.1 NMF based SCBSS

NMF can be used in the magnitude spectral domain to decompose the original spectrogram of a mixed signal into a set of basis vectors which represent each source in the magnitude spectral domain. There are two methods which each have two stages: a training stage and a separation stage. The training stage for both uses the NMF to decompose the training data into a set of bases vectors for each source.

For the first Spectral Masks method, the separation stage uses the NMF to decompose the incoming mixed signal as a linear combination of the trained basis vectors from each source. The initial spectrogram estimate for each source is found by combining its corresponding components from the decomposed matrices. It then uses the initial estimates to build various masks, which can be used to find the contribution of every source in the mixed signal. [2]

In the second Sliding Window method, rather than using NMF to directly decompose the spectrogram of the signals, the matrices are formed to be decomposed as follows: the spectrogram frames are stacked in one vector. A window with length equal is passed to multiple spectral frames size to select the first column of the matrix, then is shifted or slid by one frame to choose the next column. By using a sliding window each frame is decomposed multiple times with different neighbour frames; the average of the different decomposition results for each frame is taken to find an accurate decomposition of the spectrograms.[3]

3.2 The Mathematics

An overview of the basic concept of NMF:
$\mathbf{V} \approx \mathbf{BW}$
Where: [11]
$\mathbf{V} \in R^{F \times T}$ - Original non-negative music data
- Each column is an F-dimensional data sample
- Each row represents a data feature
- The audio spectogram data is V

$\mathbf{B} \in R^{F \times K}$ - Matrix of basis vectors
- A column is a basis vector
- Not orthonomal, but can be normalised to be

$\mathbf{W} \in R^{K \times T}$ - Matrix of activations/weights/gains
- Rows are the gain of corresponding basis vectors
- Not orthonomal, but can be normalised to be

Usually,$K < F < T$ which leads to a compressed representation of the data and a low rank approximation of V. For the sliding window method and spectral mask method, the goal is to find optimal $\mathbf{B}, \mathbf{W}$ by solving the minimization problem:

$$\begin{gathered} \min_{\mathbf{B},\mathbf{W}} C(\mathbf{V}, \mathbf{BW}) \\ \text{Subject to: } \mathbf{B}, \mathbf{W} \geq 0 \end{gathered} \tag{8}$$

Both Euclidean distance and the divergence between V and BW have been analysed as cost functions within [2]. The outcome was that the Divergence worked well for audio source separation. Thus, the plan is to implement the Divergence:

$$\sum_{i,j} \left(\mathbf{V}_{i,j} \log \frac{\mathbf{V}_{i,j}}{(\mathbf{BW})_{i,j})} - \mathbf{V}_{i,j} + (\mathbf{BW})_{i,j} \right) \tag{9}$$

Which can be computed by alternating the updates of B & W like so:

$$\mathbf{B} \leftarrow \mathbf{B} \otimes \frac{\frac{\mathbf{V}}{\mathbf{BW}} \mathbf{W}^T}{\mathbf{1W^T}} \tag{10}$$

$$\mathbf{W} \leftarrow \mathbf{W} \otimes \frac{\frac{\mathbf{V}}{\mathbf{BW}} \mathbf{B}^T}{\mathbf{1B^T}} \tag{11}$$

Where $\mathbf{1}$ is a matrix of ones with the same size as V and all operations are element wise.

3.3 Training the Bases

For each method, two sets of training data are used, one for speech and one for music. Given this set of training data, the STFT is computed for each signal, and the magnitude spectrogram $\mathbf{S}_{train}$ and $\mathbf{M}_{train}$ of speech and music signals are calculated respectively.

For the Sliding Window method, instead of using the NMF to directly decompose the spectograms, the matrices $\mathbf{S}_{train}$ and $\mathbf{M}_{train}$ are built with colums that contain 2L + 1 frames of the speech and music spectograms respectively. This

produces matrices with each column containing 2L + 1 consequent frames from the spectogram stacked in one column. For example, the column number l in the training speech matrix $\mathbf{S}_{train}$ is:

$$\mathbf{s}(l) = \left[\mathbf{f}_s^T(l-L), ..., \mathbf{f}_s^T(l), ..., \mathbf{f}_s^T(l+L)\right]^T \tag{12}$$

Where $\mathbf{f}_s(l)$ is the frame number L of the training speech signal spectogram. From here, both methods are the same; the goal now is to use NMF to decompose these spectrograms into bases and weights matrices as follows:

$$\mathbf{S}_{train} \approx \mathbf{B}_{speech}\mathbf{W}_{speech} \tag{13}$$

$$\mathbf{M}_{train} \approx \mathbf{B}_{music}\mathbf{W}_{music} \tag{14}$$

The method in Eq.(10) & Eq.(11) is used to solve for $\mathbf{S}_{train}$ $\mathbf{M}_{train}$, as well as this, it is required to normalise the columns of $\mathbf{B}_{speech}$ & $\mathbf{B}_{music}$ in every iteration to ensure $\mathbf{S}_{train}$ and $\mathbf{M}_{train}$ have normalised columns. The matrix $\mathbf{B}$ & $\mathbf{W}$ is initialized with positive random noise and choose the number of basis vectors depending on the signal. Too many basis vectors and may result in redundant sets of basis which would not be computationally efficient, something that needs to be kept in mind of due to the limited nature of the implementation. However, too little basis vectors and the algorithm will not fully capture the components of the voice and music. Thus, there is likely an optimal number of basis vectors for each song. Furthermore, in the Sliding Window method, each column has 2L + 1 times the dimension of the spectogram frames, thus, more basis vectors than the single frame case will be used to be compatible with the dimension of the columns in $\mathbf{S}_{train}$ $\mathbf{M}_{train}$, a potential computational strain.

3.4 Signal separation and masking

3.4.1 Decomposition

Initially, the mixed signal x(t) is observed and a magnitude spectrogram $\mathbf{X}$ is created using the STFT, however, the goal is to use the NMF to decompose this into a set of basis vectors and weights such that:

$$\mathbf{X} \approx [\mathbf{B}_{speech}\, \mathbf{B}_{music}]\, \mathbf{W} \tag{15}$$

Where $\mathbf{B}_{speech}$ & $\mathbf{B}_{music}$ are calculated using Eq.(10) & Eq.(11) as in the previous section to arrive at Eq.(13) & Eq.(14) where $\mathbf{W}_{speech}$ & $\mathbf{W}_{music}$ are submatricies in matrix $\mathbf{W}$ that correspond to the speech and music components.

3.4.2 Reconstruction

Assuming noise is negligible, the sum of our estimated Speech and Music spectrogram should equal the mixed spectrogram. To ensure a zero error the estimated spectograms are used to build a mask as follows:

$$\mathbf{H} = \frac{\mathbf{S}^P}{\mathbf{S}^P + \mathbf{M}^P} \tag{16}$$

Where P is a parameter, $P > 0$, and using different values of P leads to different masks. For example, when P = 2 the mask is a Wiener filter, but when P = ∞ it is a hard mask (binary mask). Furthermore, elements of $H \in (0, 1)$.

The mask H scales the frequency components of the mixed signal with a ratio that explains how much each source contributes, such that it results in Eq.(4) & Eq.(5) as the final estimates of the speech and music spectrograms.

Finally, by using the inverse STFT, it is possible to recover the estimated Music spectogram with the phase angle of the mixed signal.

3.5 PC Implementation & Validation

It was possible to implement a successful NMF algorithm in Matlab that generates the different vectors. Following that, by selecting the correct vectors, it is possible to generate the correct filtered output with an acceptable average MSE of 0.104. Neverthless, as noticed during the NMF work and as highlighted by [14], the NMF has two flows. First, a classifier is required as well for the selection of the vectors, and second, the MSE goes up to 0.4 for very complicated musical pieces, which shows that the NMF is only effective on simple songs. Therefore, this method's implementation was not explored further on the DSP.

4 Median filter

The following section covers the most advanced and successful approach to the problem using a two-pass median filter (a pitched pass followed by a drums pass).

4.1 Theory

The algorithm for vocal separation uses a multipass median filtering-based approach.

The First pass separates a signal into it's harmonic-percussive counterparts. This is based on the idea that broad-band impulsive noises (percussion instruments) form stable vertical ridges in a magnitude or power spectrogram, whereas harmonics from pitches instruments form table horizontal ridges in a magnitude or power spectrogram.

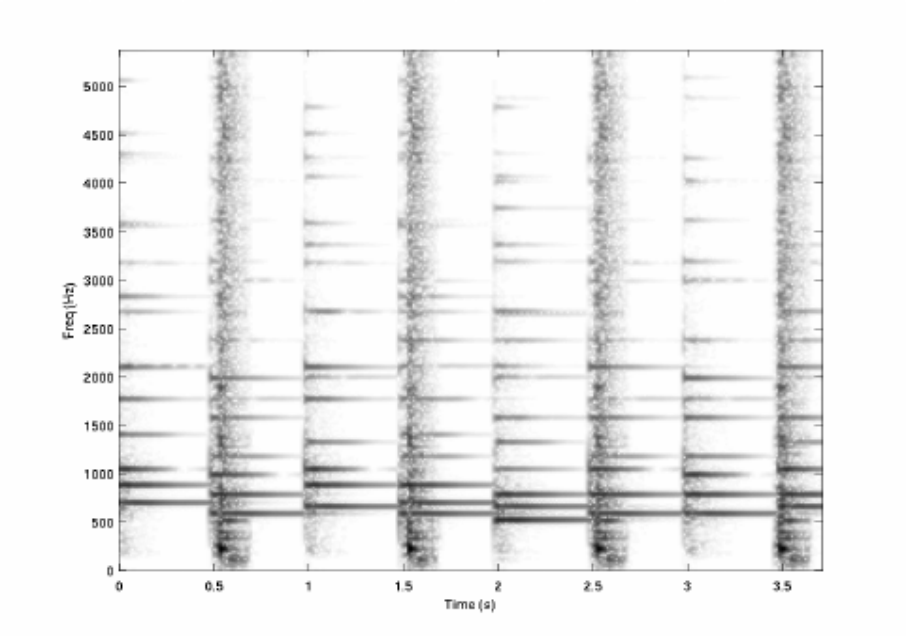


Figure 6: Spectrogram of a Drum and Piano

Referring to Fig. 6 from [14], one can see that the drums form the stable vertical ridges whereas the harmonics from the piano form horizontal ridges.

Median filters are often used in image processing to overcome the problems with local averaging operations, namely the blur of sharp discontinuities in intensity values in an image. This makes it quite effective at removing impulse noise (outliers) whilst maintaining image details.

Similarly, when applied to a musical signal, the harmonic and percussive events (impulses) are suppressed. In the event of a snare drum being played, each hit is a sharp jump in energy, making it hard to isolate. However, after median filtering, the energy signal is limited, making it easier to isolate.

The Median filter can be defined as such: given aninput vector $x(n)$, then $y(n)$ is the output of a median filter of length l, where l defines the number of samples over which median filtering takes place. When l is odd, the median filter can be defined as:

$$y(n) = median\{x(n-k:n+k), k = (l-1)/2\} \tag{17}$$

Like mentioned in section 1.1 Eq.4, the general Idea is to generate a mask which can be applied to the original spectrogram before inversion to the time domain.

To create a mask, it is necessary to generate the spectrogram frames which represent the pitched signal $\mathbf{P_i}$ vs the harmonic & vocal one $\mathbf{H_h}$.
Denoting the input magnitude spectrogram $\mathbf{S}$, the ith time frame as S_i, and the hth frequency slice as S_h, then the harmonic spectrogram frame H_h can be obtained from:

$$H_h = \mathcal{M}\{S_h, l_{harm}\} \tag{18}$$

Where l is the number of samples over which median filtering takes place. These filter slices can then be aggregated to yield a complete $\mathbf{H}$.

The effect of the Median Filter on a piece containing snare drums and piano can be seen in Fig:7 (from [14])

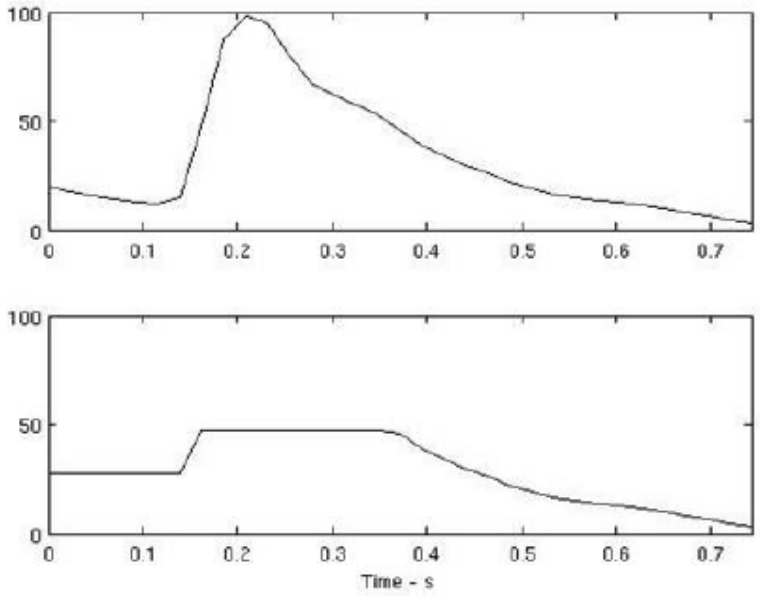


Figure 7: Spectrogram frequency slice from a spectrogram containing a mixture of snare drum and piano before and after the filter; a large portion of the energy has been removed

The pitched spectrogram form P_i can be generated by performing median filtering on S_i :

$$P_i = \mathcal{M}\{S_i, l_{pitch}\} \tag{19}$$

Where $\mathcal{M}$ is the median filtering operation, and l_{pitch} is the length of the pitched median filter. Repeating this for each spectogram frame will produce the spectogram $\mathbf{P}$.
After the median filters have been obtained, inverse STFT could be performed, however, median filtering introduces many artifacts into the spectrogram. Thus, to ensure a high quality re-synthesis, $\mathbf{H}$ & $\mathbf{P}$ are used to generate a mask to be applied to the original spectrogram before the inverse STFT. In this particular case, masks based on Wiener filtering are used:

$$\mathbf{M}_{\mathbf{H_{h,i}}} = \frac{\mathbf{H}_{h,i}^2}{\mathbf{H}_{h,i}^2 + \mathbf{P}_{h,i}^2} \tag{20}$$

$$\mathbf{M}_{\mathbf{P_{h,i}}} = \frac{\mathbf{P}_{h,i}^2}{\mathbf{H}_{h,i}^2 + \mathbf{P}_{h,i}^2} \tag{21}$$

Follows an element wise multiplication between the original complex valued spectrogram ($\hat{\mathbf{S}}$) and the masks as such:

$$\hat{\mathbf{H}} = \hat{\mathbf{S}} \otimes \mathbf{M}_{\mathbf{H}} \tag{22}$$

$$\hat{\mathbf{P}} = \hat{\mathbf{S}} \otimes \mathbf{M}_{\mathbf{P}} \tag{23}$$

The Second pass In the literature [14], depending on the parameters of the FFT length, window length and overlap length, the vocals would separated into different either with the pitched instruments or the percussive instruments. At a low FFT frequency resolution, the voice tended to be separated with the pitched instruments, while at high frequency resolution, such as an FFR size of 16384 samples, the majority of the voice tended to be separated with the percussion instruments. [14]
To overcome this, a hyperparameter search is employed that yielded the following:

- WindowLength = 512;
- FFTLength = 512;
- OverlapLength = 256;

4.2 PC Implementation

The Matlab implementation of the median filter is pretty straightforward and makes use of the Matlab *medfilt1* command, and implement the two passes with two separate for-loops after taking the input STFT:

```
% STFT parameters found using hyperparameter search
WindowLength = 512;
FFTLength = 512;
OverlapLength = 256;
Window = blackman(WindowLength);

% Generate STFT
input_stft = stft(input, 'Window', Window, 'OverlapLength',OverlapLength, 'FFTLength', FFTLength, 'FrequencyRange',
    'onesided');

% Median filter parameters
size = size(input_stft);
abs_input = abs(input_stft);

% Windows found using hyperparameter search
median_window_1 = 100;
median_window_2 = 25;

% Preallocate passes
pitch_pass = zeros(size);
drum_pass = zeros(size);

% Pitch pass for first mask
for i=1:size(1)
    pitch_pass(i,:) = medfilt1(abs_input(i,:),median_window_1);
end
pitch_mask = (pitch_pass./abs_input)>0.5;

% Drum pass for second mask
abs_input = abs_input.*(~pitch_mask);
for i=1:size(2)
    drum_pass(:,i) = medfilt1(abs_input(:,i),median_window_2);
end
drum_mask = (drum_pass./(abs_input + eps))>0.5;

% Combine and apply masks to get the output
final_mask = pitch_mask | drum_mask;
output_stft = input_stft.*(final_mask);
```

The *input* variable is assumed to already contain the samples of the target song at a sampling rate of 8KHz. The *output_stft* can then be feeded into an inverse STFT, as seen in Section 1.2, and generate the output.
Once the median passes are completed, two different masks are obtained. Due to independence these can be simply OR-ed together to generate a final mask that includes all instruments and excludes vocals. The mask is then applied to the original input STFT.

4.3 PC Validation

As can be seen in the previous subsection code, some parameters are needed for the STFT and the median filter. The parameters in question are:

- STFT window length
- STFT FFT length
- STFT overlap length
- STFT window
- Median filter window size 1
- Median filter window size 2

Since training data is provided with the perfect vocal and instrumental sources, it is possible to apply an MSE evaluation to validate the performance of the algorithm. Following the same approach used with the low-pass filter cutoff frequency finding, a brute-force on the first three parameters sets the STFT window length and FFT length to 512, and the overlap length to 256 (the table has been omitted as it includes more than 600 different values). The STFT window used is Blackman with the results of all different windows presented in Fig. 8. As can be seen in the graph, the median filter algorithm provides almost 99% validation of the vocal track regardless of the window. What matters therefore is the sound quality generated output. While both Blackman and Hamming provide the lowest MSE (of 0.0099), it was judged from an SNR and listening perspective that the Hamming window was noisier, and therefore the Blackman window was selected.

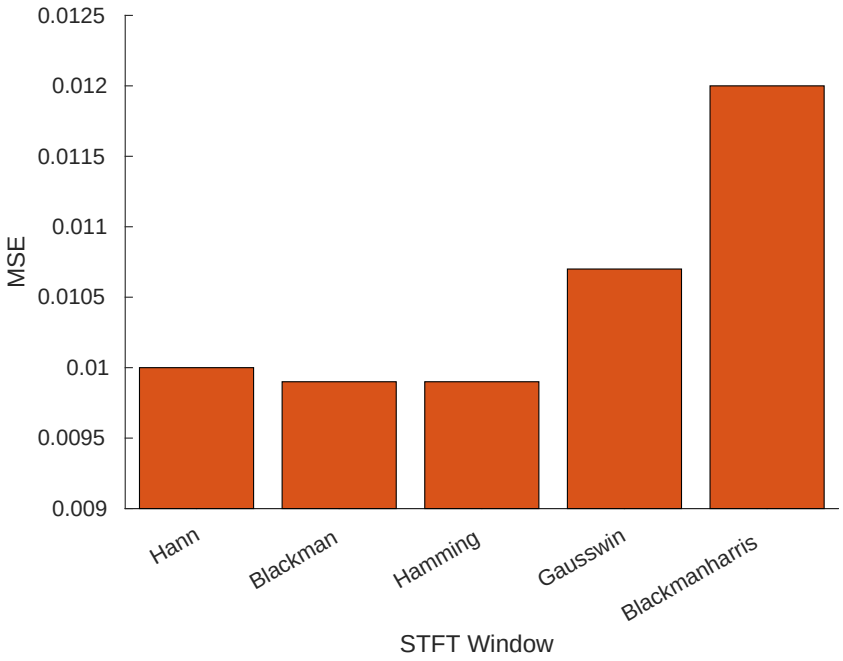
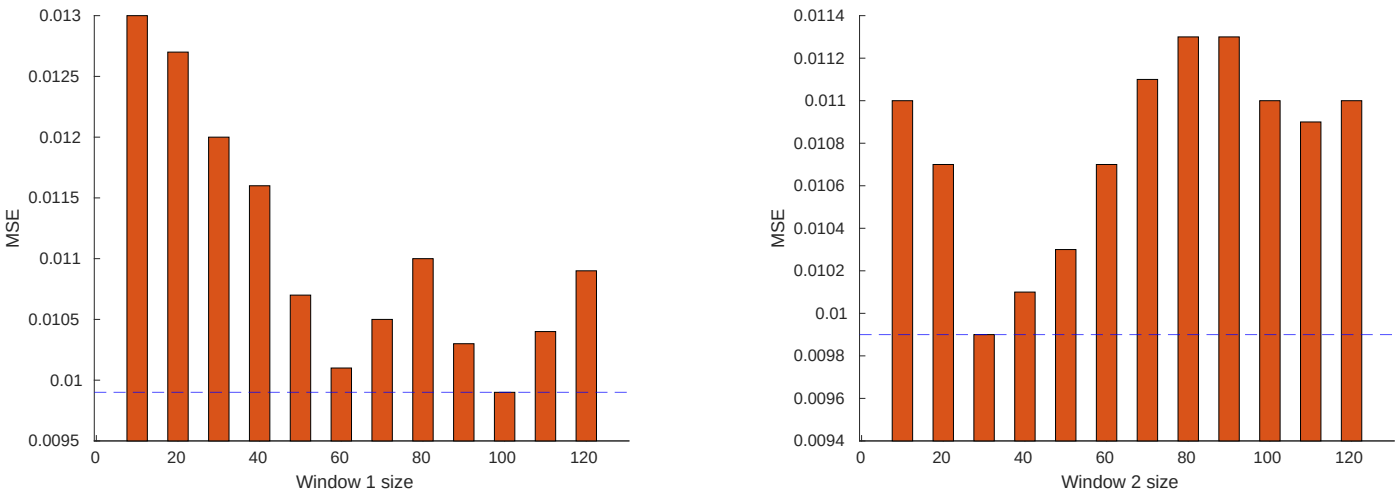


Figure 8: Hyperparameter search on the different STFT windows with the average MSE

Following the STFT parameters, it is possible to inspect the median filter windows. The results can be seen in Fig. 9, where the optimal window sizes are set to 100 and 30.



(a) Window 1 MSE validation

(b) Window 2 MSE validation

Figure 9: MSE varying with different window sizes for the median filter

4.4 DSP Implementation

The median filter algorithm could be theoretically easily implemented in Simulink using the *Median Filter* block from the DSP toolbox and using the corresponding window sizes. Nevertheless, what proved to be the bottleneck of the design was implementing the STFT itself on the DSP. The first method was to try and use the STFT block from within Simulink, but that one is unfortunately implemented for continuous and not discrete time, which does not compile for the DSP. The second approach was to implement a custom STFT block using FFT blocks and discrete time by triggering the window loop using an ISR. Yet, it was not possible to get this solution to work, and therefore the median filter algorithm was not fully implemented in Simulink to run on the DSP.

4.5 DSP Validation

While it was not possible to implement the STFT on the DSP, it was still possible to try and run the model in a Simulink simulation. The functional Simulink diagram can be seen in Fig. 10 and has been evaluated by getting the MSE compated to the expected vocals from the training dataset.

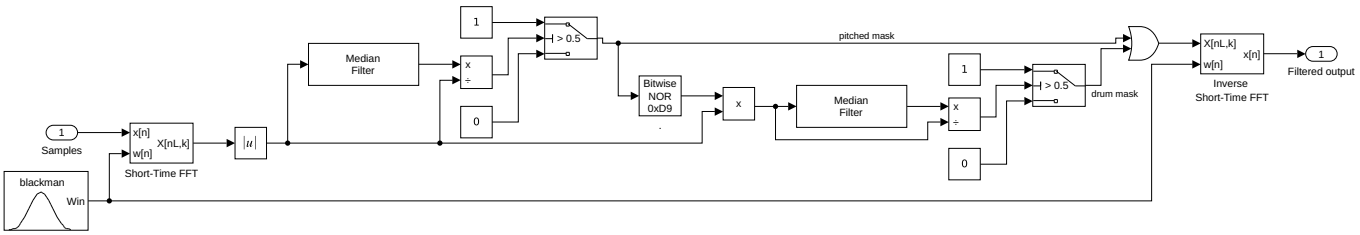


Figure 10: Simulink implementation of the median filter

5 Conclusion

In conclusion, this report evaluates the effectiveness and implementation of multiple methods for single channel source separation including: Low-pass filters, which were simple, implementable but not very accurate or effective; Non-Negative Matrix Factorisation, which was mathematically tractable, but was not very effective with complex musical pieces and was slightly too complex to implement on the DSP; and finally Median filters which were the least sophisticated, however, produced the best results and would be our method of choice for a slightly less hardware constrained project.

The main limitation of getting a more complex idea working was the computational constraints; effective implementation of an STFT was the first step of many of these techniques and would have required the board to do many parallelised computations before even starting to implement a technique.

6 References

[1] Music archive provided for the project
https://freemusicarchive.org

[2] Emad M. Grais & Hakan Erdogan "Single channel speech music separation using nonnegative matrix factorization and spectral masks" 2011
https://ieeexplore.ieee.org/document/6004924

[3] Emad M. Grais & Hakan Erdogan "Single channel speech music separation using nonnegative matrix factorization with sliding windows and spectral masks"
research.sabanciuniv.edu/17516/1/sliding_windows.pdf

[4] Dr Andy Thomas "Neural Networks introduction"
https://adventuresinmachinelearning.com/wp-content/uploads/2020/02/A-beginners-introduction-to-neural-networks-V3.pdf

[5] Jianxin Wu "Convolutional Neural Networks introduction" 2017
https://cs.nju.edu.cn/wujx/paper/CNN.pdf

[6] Example of voice separation with deep learning using Matlab
https://uk.mathworks.com/help/deeplearning/ug/cocktail-party-source-separation-using-deep-learning-networks.html

[7] Ale Koretzky "Isolating vocals from stereo music using Convolutional Neural Networks" 2019
https://towardsdatascience.com/audio-ai-isolating-vocals-from-stereo-music-using-convolutional-neural-networks-210532383785

[8] Aapo Hyvarinen "A Short Introduction to Independent Component Analysis with Some Recent Advances"
http://wwwf.imperial.ac.uk/ nsjones/TalkSlides/HyvarinenSlides.pdf

[9] Bin Gao "Single channel Blind Source Separation A thesis submitted to the university of Newcastle for the degree of Doctor of Philosophy" 2011 https://core.ac.uk/download/pdf/40007882.pdf

[10] Texas instruments Micro-Controller used
https://www.ti.com/lit/gpn/tms320f28379s

[11] Nicholas Bryan & Dennis Sun "Source Separation Tutorial Mini-Series II: Introduction to Non-Negative Matrix Factorization" 2013
https://ccrma.stanford.edu/ njb/teaching/sstutorial/part2.pdf

[12] Nicholas Bryan & Dennis Sun "Source Separation Tutorial Mini-Series III: Extensions and Interpretations to Non-Negative Matrix Factorization" 2013
https://ccrma.stanford.edu/ njb/teaching/sstutorial/part3.pdf

[13] Nicholas Bryan & Dennis Sun "Source Separation Tutorial Mini-Series III: Extensions and Interpretations to Non-Negative Matrix Factorization" 2013
https://ccrma.stanford.edu/ njb/teaching/sstutorial/part3.pdf

[14] Derry Fitzgerald & Mikel Gainza "Single Channel Vocal Separation using Median Filtering and Factorisation Techniques" 2010
https://www.researchgate.net/publication/254583302$_{S}ingle_{C}hannel_{V}ocal_{S}eparation_{u}sing_{M}edian_{F}iltering_{a}nd_{F}actorisatio$