Frequency Dependent Time-Scale Modification

Timothy Roberts
Signal Processing Laboratory
Griffith University
Brisbane, Australia
timothy.roberts@griffithuni.edu.au

Kuldip K. Paliwal
Signal Processing Laboratory
Griffith University
Brisbane, Australia
k.paliwal@griffith.edu.au

***Abstract*—Current Time-Scale Modification algorithms scale all frequencies by the same amount. This paper presents an efficient method and implementation for time scaling of arbitrary frequency regions, called Frequency Dependent Time-Scale Modification. This is achieved by creating a composite frequency spectrum frame before using traditional frequency domain time-scaling methods. Testing was undertaken with results presented from varied processing of 3 files. Links to and description of a MATLAB implementation are provided.**

Availability: A MATLAB software implementation can be found on Github at github.com/zygurt/TSM.

***Index Terms*—audio, effect, frequency, FX, phase vocoder, time-frequency, time-scale modification, TSM**

I. Introduction

Time-Scale Modification (TSM) is a well-researched area, with the main processing methods making use of either the frequency domain [1]–[4] or time domain [5], [6]. These methods manipulate the speed of playback by adjusting the relationship between the analysis and synthesis shifts. Each method attempts to maintain phase coherency in different ways. For example, the phase vocoder and variants, update the phase spectrum for each frame based on the known phase progression for the original time base, and the required progression for the new time base [3], ensuring phase coherence for each bin. The phase vocoder also allows for various musical effects such as mutating between two sounds, robotization, and whisperization [7].

An area that has yet to be explored however, is the application of TSM to different frequency ranges of the input signal. Frequency domain methods are prime candidates for this application, with signal frames transformed to the frequency domain as part of the algorithm. Additionally, the same ratio between analysis and synthesis shifts is used for all bins, providing a platform for augmentation. Time-scaling could be applied to each signal after a filterbank, however this is computationally complex and will be discussed further later in the paper.

The proposed method is presented in Section II, testing methodology in Section III, results are presented in Section IV, the supplied implementation is presented in Section V, and spectrograms of example processed files are presented in Section VI.

II. Method

The proposed method is a modification of the Analysis-Modification-Synthesis (AMS) framework used by frequency domain TSM methods.

A. Analysis

During Analysis, a composite frequency domain frame is constructed using multiple time-domain frames and Fourier transforms. This composite frame is constructed by first extracting r time domain frames from the signal, where $0 \leq r \leq R$ and R is the total number of frequency regions being used. Shown in (1) where m is the frame number, S_a^r is the analysis shift for the current region and N is the frame length. Due to using a vector of analysis shifts, the time scaling parameter α becomes α_r and is calculated as per (2). Regions, rather than bins, are used to give a scalable algorithm and will be discussed later. The end of the signal should be adaptively zero-padded to allow for all regions to be processed.

$$\mathbf{x}_r(n) = x(n) \quad , mS_a^r \leq n \leq m + N - 1 \tag{1}$$

$$\alpha_r = \frac{S_s^r}{S_a^r} \tag{2}$$

These region frames, for the current analysis time instance t_a^m, are windowed using $h(n)$ and transformed to the frequency domain, as seen in (3). This results in an array of frequency domain frames $\hat{\mathbf{X}}_r(t_a^m, k)$.

$$\hat{\mathbf{X}}_r(t_a^m, k) = \sum_{n=0}^{N-1} h(n)\mathbf{x}_r(n)e^{\frac{-j2\pi kn}{N}} \tag{3}$$

The composite frame is constructed by concatinating the bin values for each region, as in (4). Each region has an upper (r_u) and a lower (r_l) bound.

$$X(t_a^m, k) = \begin{bmatrix} \hat{\mathbf{X}}_0(t_a^m, k) \\ \hat{\mathbf{X}}_1(t_a^m, k) \\ \hat{\mathbf{X}}_2(t_a^m, k) \\ \vdots \\ \hat{\mathbf{X}}_{R-1}(t_a^m, k) \end{bmatrix} \quad , r_l \leq k \leq r_u \tag{4}$$

Finally, the magnitude and phase responses for the current time instance are calculated before modification using (5) and (6).

$$|X(t_a^m, k)| = \sqrt{[\Re(X(t_a^m, k))]^2 + [\Im(X(t_a^m, k))]^2} \quad (5)$$

$$\angle X(t_a^m, k) = tan^{-1}\frac{\Im(X(t_a^m, k))}{\Re(X(t_a^m, k))} \quad (6)$$

B. Modification

As frequency domain methods maintain horizontal phase coherence within each bin [3], each bin can be arbitrarily time scaled, given a time-scale ratio per bin, and by extension region. This change requires minor changes to the phase vocoder and is shown below. During modification the instantaneous frequency, $\widehat{\omega}_k(t_a^u)$ is calculated by first calculating the heterodyned phase increment, using (7), where $\Omega_k = \frac{2\pi k}{N}$. The phase increment Φ_k^u is reduced to $\pm\pi$ by taking its principle determinate, (8). Finally, the instantaneous frequency of the closest sinusoid to the centre frequency of the bin is calculated using (9).

$$\Delta\Phi_k^m = \angle X(t_a^m, k) - \angle X(t_a^{m-1}, k) - S_a^r\Omega_k \quad (7)$$

$$\Delta_p\Phi_k^m = \Delta\Phi_k^m - 2\pi(round(\frac{\Delta\Phi_k^m}{2\pi})) \quad (8)$$

$$\widehat{\omega}_k(t_a^m) = \Omega_k + \frac{1}{S_a^r}\Delta_p\Phi_k^m \quad (9)$$

After calculating the instantaneous frequency, the synthesis phase can be calculated using the phase propagation formula in (10), where S_s^r is the synthesis shift for each region.

$$\angle Y(t_s^m, k) = \angle Y(t_s^{m-1}, \Omega_k) + S_s^r\widehat{\omega}_k(t_a^m) \quad (10)$$

C. Synthesis

During synthesis, the modified frame is reconstructed, according to (11), by using the original magnitude and the synthesised phase spectra. Finally, the frame is transformed back to the time domain, with a single inverse Fourier transform before the overlap-add process is used to combine the resulting frame and the output signal.

$$Y(t_s^m, \Omega_k) = |X(t_a^m, k)|e^{\angle Y(t_s^m, k)} \quad (11)$$

D. Optimisation

When considering implementation of the proposed method, optimisation can be made through the use of a constant synthesis shift size, regions and a comparison of direct DFT implementation and the use of an FFT. These optimisations are discussed here, with testing discussed later.

By using an appropriate window that sums to unity during overlap adding, the resulting signal does not require window normalisation. In this case, the S_s^r vector may be reduced to a single value S_s and applied to all bins. The use of regions allows for a reduction in processing time when using an FFT for the frequency transformation, as multiple bins can be used from each FFT, reducing the total number of FFTs required for each time instance. This process also allows for the computational complexity to be scaled. By using $\frac{N}{2}$ regions, very smooth transitions can be generated, however processing is much slower than real-time. Conversely, by using a small number of regions, real-time suitable applications become possible.

The computational complexity of a standard direct implementation of the DFT is $\mathcal{O}(N^2)$, while the computational complexity of an FFT reduces this to $\mathcal{O}(N \log N)$. Within the proposed method however, up to N frames are calculated at each time instance resulting in $\mathcal{O}(N^3)$ and $\mathcal{O}(N^2 \log N)$ complexity for DFT and FFT respectively. In the case of a single bin per region, the complexity for the DFT is reduced to $\mathcal{O}(N^2)$ as only a single value needs calculating from each transform. The frequency transform can be further optimised by calculating $0 \leq k \leq \frac{N}{2}$, removing half of the calculations from both the DFT and FFT methods. Finally, further optimisation can be found through pre-computing the exponential factors within the DFT.

III. Testing

Testing of the proposed method was conducted in a number of stages, with comparisons between methods quantitatively assessed. As the proposed method is analogous to applying TSM to R length filterbank filtered signals, this was explored. The direct DFT method was compared to both FFTW [8] and Radix-2 Cooley-Tuckey FFT implementations. A variety of signals were processed with spectrograms of the resulting files, shown in Section VI.

The proposed method was implemented within MATLAB using a frame length of approximately 50ms, Hann windowing and a synthesis shift of$N/4$. The DFT method was implemented to make use of the fast matrix calculations available within MATLAB. The FFT method uses the FFTW library used by MATLAB. The filterbank method implementation creates R band-pass filtered signals using triangular filterbanks to ensure unity reconstruction. Each of these signals are then processed using a standard Phase Vocoder before being recombined to create the final signal. More details on using each implementation can be found in Section V.

Systematic batch processing of **Male_Speech.wav** was undertaken to compare the computation time of the proposed method and the filterbank method. The number of regions was increased from 2 to 500 with a frame size of 50ms. 750 and 1025 regions were also tested. The time-scale ratio was a linearly spaced vector from 50% to 200%. Due to the large amount of time required for the filterbank method, only a single trial was used for each number of regions.

For comparing the direct DFT implementation and the FFT implementation, three tests were used. Initially, processing was restricted to computing the composite frequency frame using a range of frame lengths of white noise, where $R = \frac{N}{2}$. DFT, FFTW and Radix-2 Cooley-Tukey FFT implementations were used in this initial testing. Secondary testing using DFT and FFTW implementations was conducted in which frame length was held at 50ms, while the number of regions was varied from 2 to $\frac{N}{2}$. The time taken to process a short speech file 50 times, with no change in speed, was averaged. The implementations have not been optimised to avoid processing with no change in speed. The break even point for the number of regions for each frame length was also tested. The DFT and FFT methods processed a frame of white noise 10 times subsequently increasing the number of regions until the DFT method was faster than the FFT.

A white noise burst was used during testing, as it gives a clear indication of the temporal manipulations from the proposed method. Speech and complex music were also tested, with figures included, however the reader is advised to listen to the resulting files for greater insight to the capabilities of the proposed method. The reversibility of the method was also examined through the application of inverse time-scale ratios.

Testing was done using a 6-core 1.6Ghz Xeon processor and MATLAB 2017a, with all code and results available online.

IV. Results

The proposed FDTSM method is significantly faster than the filterbank method. As can be seen in Figure 1, processing time is initially similar, but increases as the number of filters and regions increases. To process a 2.5 second file, at a sample rate of 44.1kHz and a frame length of 2048, the proposed method took 0.099 seconds while the filterbank method took 0.299 seconds for 2 regions. For the same file the proposed method took 7.82 seconds while the filterbank method took 38.23 minutes for 1025 regions. The ratio for computation time between the two methods linearly increases for the number of regions and fits the linear equation $\frac{t_{fbank}}{t_{fft}} = 0.287R + 5.006$ with $R^2 = 0.9977$. Subjectively, the bandpass nature of the filterbank method combined with a lack of phase coherency between the filtered signals produces a thin and tinny sound. This occured for all numbers of regions tested.

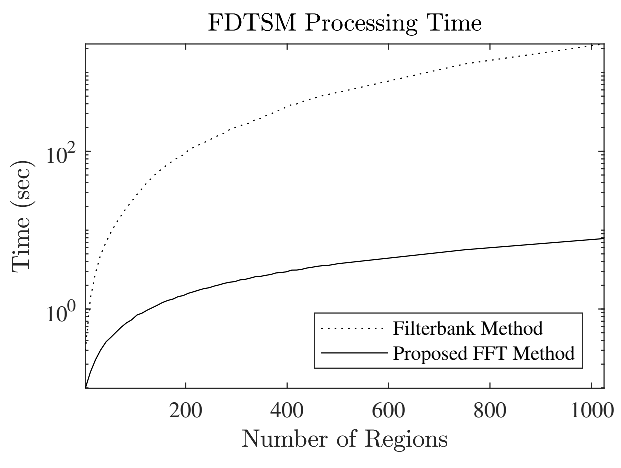


Fig. 1. *Processing time for Proposed and Filterbank FDTSM methods.*

Initial comparison of the speed of the DFT and FFT showed the high efficiency of the FFTW algorithm, beating both the Radix-2 Cooley-Tuckey FFT and DFT. After optimising the DFT implementation, results were found to confirm the theoretical computational complexity. This can be seen in Figure 2, where the frame size was increased and the number of regions was held at $\frac{N}{2}$.

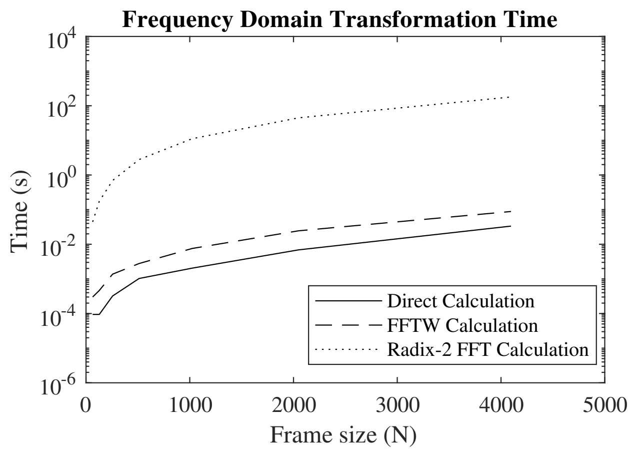


Fig. 2. *Frequency domain transform processing time for Direct DFT, FFTW and Radix-2 Cooley-Tuckey FFT.*

Following from this testing, the frame size was held constant while the number of regions was increased. It can be seen in Figure 3 that the efficiency of the FFT method is inversely proportional to the number of regions used for processing. The same is true for the DFT method, however it is not linear in nature and approaches a limit as the number of regions approaches $\frac{N}{2}$. This results in a break even point, where it is faster to process using the DFT method, even when using the highly optimised FFTW implementation. Further testing of the break even point was conducted, and it was found that the results fit the linear equation $R = 0.203N - 25.61$ with $R^2 = 0.9981$, for the computer used for testing.

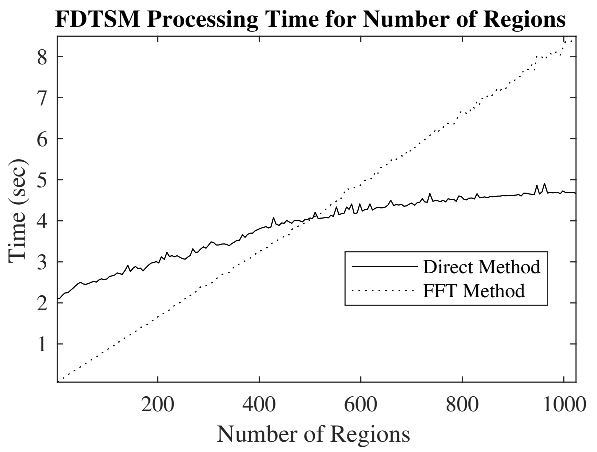


Fig. 3. *Processing time for Proposed DFT and FFTW FDTSM methods as the number of regions increases.*

V. Software Implementation

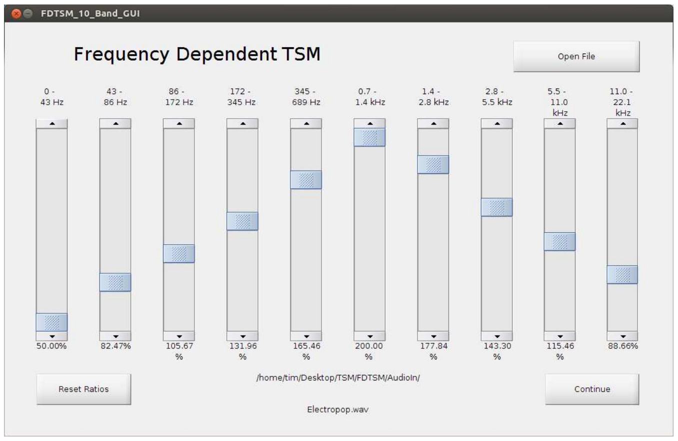
A MATLAB software implementation can be found at github.com/zygurt/TSM/ and is utilised in the following way.

1) **FDTSM.m**, **FDTSM_Direct.m** and **FDTSM_FFT.m** take the input signal (x), the frame length (N) and a region information structure as input arguments. The region information structure should contain 2 fields, region.TSM and region.upper, containing the TSM ratio for each region and the upper bound of each region, respectively. TSM region parameters are set such that 1 is 100%, 0.5 is 50% and 2 is 200% and must be greater than 0. These vectors must be the same length, with the final value in the upper bound vector equal to $\frac{N}{2}+1$. The function accepts multi-channel signals, however these signals are summed before processing. The time scaled signal is returned from the function.
2) **FDTSM_script.m** is a script that has the minimum code required to load an audio file, set region parameters to use the FDTSM function and write the resulting file to disk with a useful filename. Output files are saved into the **AudioOut** folder as Filename_TSM_ratios_FDTSM.wav.
3) **FDTSM_GUI_example.m** gives an example of a graphic equalizer style interface for FDTSM. The interface, shown in Figure 4, allows the user to set the speed of 10 bands, each an octave in width, to between 50% and 200%. The current TSM ratio is displayed below each slider. The user is also able to load a file and reset all of the bands to 100%. The frequency range for each band is displayed above each slider and is set automatically by the sample rate of the input audio file. Each region width, in bins, is a successive power of 2. The filename, path, and TSM ratios for each region are returned to the script, which applies time scaling before saving the resulting file to disk. This script makes use of both **FDTSM_10_Band_GUI.m** and **.fig** files.
4) The **AudioIn** folder contains the 3 source files used to generate the examples. **Electropop.wav** is synthetic polyphonic music, **Male_speech.wav** is an utterance of *"I am sitting in a room"*, and **White.wav** is a quarter second white noise burst starting at one second. Caution should be used during playback of **White.wav**. The files have a sample rate of 44.1 kHz and a bit depth of 16 bits.

VI. Processing Examples

Intended as an audio effect and for sound design, the three audio files included with the software implementation, were used to generate a range of sample results, using a frame length of 50ms. By processing **White.wav** with linearly spaced TSM values of 50-200% across $\frac{N}{2}+1$ regions the sweep signal in Figure 5 can be generated. Figures 6 and 7 shows the result of processing **White.wav** with bounded random TSM ratios.

When considering musical signals, setting the time scales for frequency ranges to multiples of each other yields interesting results, such as Figure 8, where **Electropop.wav** from 0-323Hz was scaled to 50% with the remainder of the signal left untouched. This results in a half time feel. FDTSM processing of speech also gives interesting results such as the subtle



0 - 43 Hz	43 - 86 Hz	86 - 172 Hz	172 - 345 Hz	345 - 689 Hz	0.7 - 1.4 kHz	1.4 - 2.8 kHz	2.8 - 5.5 kHz	5.5 - 11.0 kHz	11.0 - 22.1 kHz
50.00%	82.47%	105.67 %	131.96 %	165.46 %	200.00 %	177.84 %	143.30 %	115.46 %	88.66%

/home/tim/Desktop/TSM/FDTSM/AudioIn/

Electropop.wav

Fig. 4. *A 10-band graphic equalizer style GUI for FDTSM.*

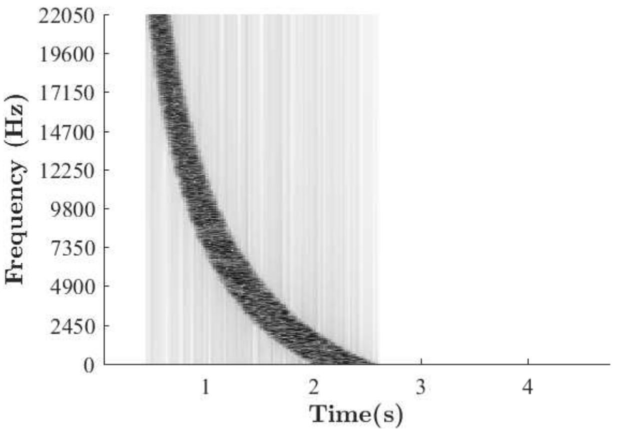


Fig. 5. *Spectrogram of* ***White.wav*** *after FDTSM with linearly spaced TSM ratios between 50% and 200% applied in 1025 regions.*

processing shown in Figure 9. This file was processed such that the middle frequencies of the speech were slowed, giving a slowly increasing delay to the band-passed signal. Finally, the FDTSM is reversible when the inverse TSM ratios are applied. Figure 10 shows the result of applying inverse time scaling ratios to the file shown in Figure 5. The resulting signal is similar, but not identical, to the original due to artifacts in the phase vocoder, but allows for interesting applications beyond sound design.

VII. Conclusion

This paper has presented an efficient method and implementation for the arbitary time-scaling frequency ranges of a signal. This was achieved through the use of a composite frequency frame for each time instance followed by standard frequency domain TSM. An example user interface, comparison of frequency domain transformation and processing examples were presented. Future work will improve the implementation to allow for the modification of time scales for each region during processing, phase locking within regions and adaptively choosing the frequency transform method based on the number

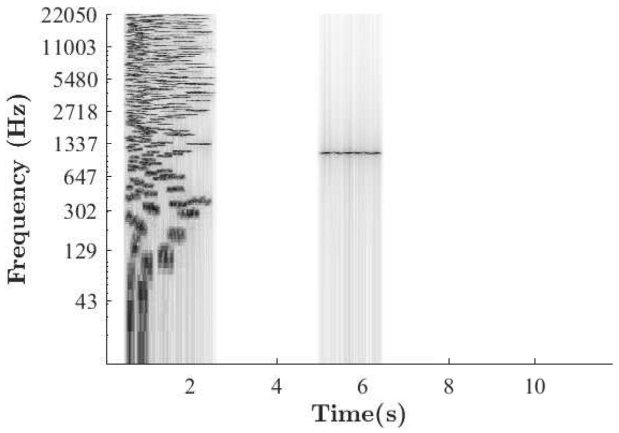


Fig. 6. *Spectrogram of* ***White.wav*** *after FDTSM with random TSM ratios between 50% and 200% applied in 1025 regions and 20% TSM applied to region 50.*

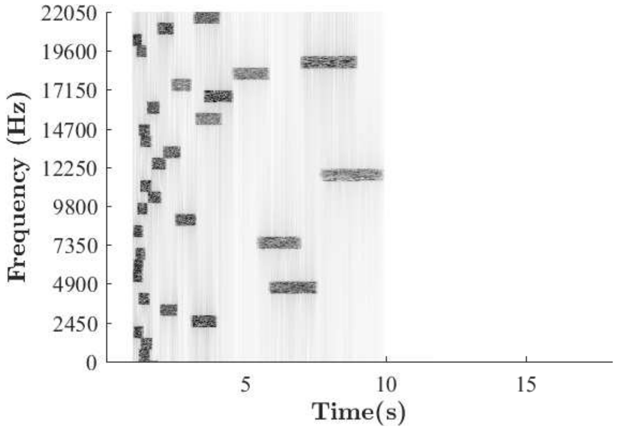


Fig. 7. *Spectrogram of* ***White.wav*** *after FDTSM with random TSM ratios between 0% and 100% applied in 32 regions.*

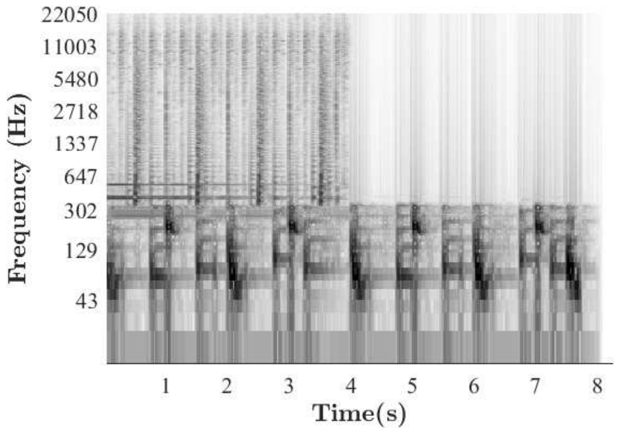


Fig. 8. *Spectrogram of* ***Electropop.wav*** *after FDTSM with 10-band GUI set to* $[50, 50, 50, 50, 100, 100, 100, 100, 100, 100]\%$.

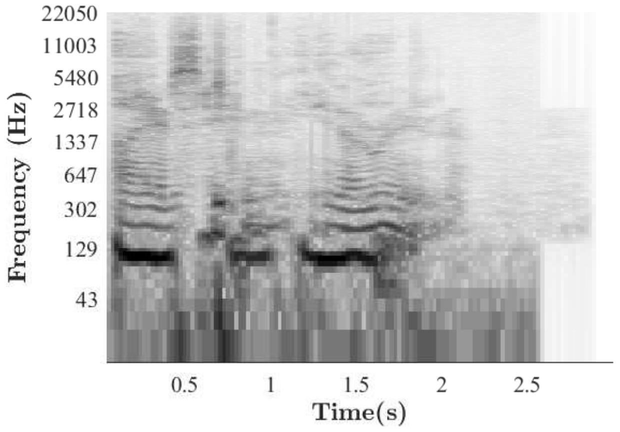


Fig. 9. *Spectrogram of* ***Male_Speech.wav*** *after FDTSM with 3 regions* $[N/64, N/16, N/2]$ *set to* $[100, 90, 100]\%$.

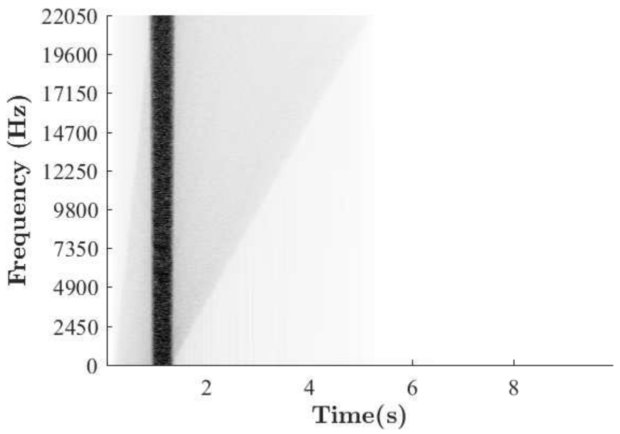


Fig. 10. *Spectrogram of Figure 5 after FDTSM recovery using inverse TSM ratios.*

of regions. Real time implementations for streams of audio will also be considered.

REFERENCES

[1] J. Flanagan and R. Golden, "Phase vocoder," *Bell System Technical Journal*, vol. 45, no. 9, pp. 1493–1509, 1966.

[2] M. Portnoff, "Implementation of the digital phase vocoder using the fast fourier transform," *IEEE Transactions on Acoustics, Speech, And Signal Processing*, vol. 24, no. 3, pp. 243–248, 1976.

[3] J. Laroche and M. Dolson, "Improved phase vocoder time-scale modification of audio," *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 3, pp. 323–332, 1999.

[4] J. Driedger and M. Muller, "A review of time-scale modification of music signals," *Applied Sciences*, vol. 6, no. 2, pp. 57–83, 2016.

[5] W. Verhelst and M. Roelands, "An overlap-add technique based on waveform similarity (wsola) for high quality time-scale modification of speech," *Proceedings of ICASSP '93*, vol. 2, pp. 554–557, 1993.

[6] S. Rudresh, A. Vasisht, K. Vijayan, and C. S. Seelamantula, "Epoch-synchronous overlap-add (esola) for time-and pitch-scale modification of speech signals," *arXiv preprint arXiv:1801.06492*, 2018, unpublished.

[7] U. Zölzer, X. Amatriain, J. Arfib, J. Bonada, G. De Poli, P. Dutilleux, et al., *DAFX - Digital Audio Effects*, John Wiley & Sons, 2002.

[8] M. Frigo and S. G. Johnson, "The design and implementation of fftw3," *Proceedings of the IEEE*, vol. 93, no. 2, pp. 216–231, 2005.