Playback Rate Modification of Audio Signals using MATLAB

Challenge 3

Pablo Martin Garcia
ECE 160 Multimedia Systems
UCSB
Santa Barbara, CA
martingarcia@umail.ucsb.edu

Antonio Javier Samaniego Jurado
ECE 160 Multimedia Systems
UCSB
Santa Barbara, CA
samaniegojurado@umail.ucsb.edu

I. Introduction

In this project we will write a program to modify the rate of playback of an audio signal, speeding it up or slowing it down, without affecting the frequency content, pitch, or intelligibility of the signal. In other words, we will extend the length of the audio signal without making it sound lower in pitch. If the audio is speech, for example, it should sound like the same person is speaking more slowly or quickly.

We are going to create a function in MATLAB that takes three inputs: the audio signal to be modified, the playback speed or rate, and the sample frequency. With these parameters, the function will give the modified signal as output which would have a different playback speed without changing the original spectrum of the input signal.

II. Methods

A. Initialize Variables

Firstly, the function takes three inputs as arguments, those being: the audio signal to be modified, the playback speed or rate of modification and the sample frequency. After taking these arguments, some variables need to be initialized to perform the transformation and modification of the playback rate.

We have standardized the window length of our hamming window to be of 1000 samples. After this, the hop length of our code is 0.2 times the length of the hamming window, which means that there is an overlap of 80 percent.

B. Perform the STFT

The Short-Time Fourier Transform (STFT) is a Fourier-transform used to determine the frequency and phase content of local sections of a signal as it changes over time. In order to compute this, we have to divide a time signal into shorter segments of equal length and then compute the Fourier transform separately on each segment. This calculates the spectrum of each segment.

The STFT of a discrete signal is calculated with the following equation:

$$STFT[k] = \sum_{n=0}^{M} x[n]\, w[n - mR]\, e^{2\pi jkn/N}$$

where R = is the block length. Also, summing over N_f frames, indexed by m we can reformulate the equation like:

$$\sum_{m=0}^{N_f - 1} X_m[k] = DFT(x[n])\, W[n]$$

where W[n] is the sum of shifted windows given by:

$$W[n] = \sum_{m=0}^{N_f} w[n - mR]$$

To ensure that the time-averaged STFT is proportional to the DFT, W[n] = W_0 = constant. With this property we can recover the original signal performing inverse DFT of each block and then combining the results by summing time-shifted blocks.

C. Calculate the Interpolated Version of the STFT

To calculate the new version of the audio with a different playback rate we need to abstract in some way new versions of the spectrum that are estimated with the original spectrum of the input signal.

The process of interpolation consists of extracting estimated samples in between samples that we already have from the original signal. We have to consider that we would want to interpolate in case we want to add new blocks of the spectrum so as to have more samples in the STFT, meaning that the audio file would increase in time in accordance with the playback rate. If the desired result is to extend the length of the audio signal, the playback rate should be between 0 and 1. Then the process of interpolation takes place.

In the other hand, if the desired outcome is to reduce the length of the audio signal, the playback rate should be greater than 1. In this case, samples or blocks of the original STFT are eliminated so as to have fewer samples in time of the spectrum, and then reconstruct a signal which is shorter in time. This

process is actually called decimation (the inverse process of interpolation).

To clarify both methods, an image is shown of a discrete signal that can be interpolated (if the operation is done from the lower signal to the upper signal) or decimated (if the operation is done from the upper signal to the lower signal). The image is shown below in Fig. 1.

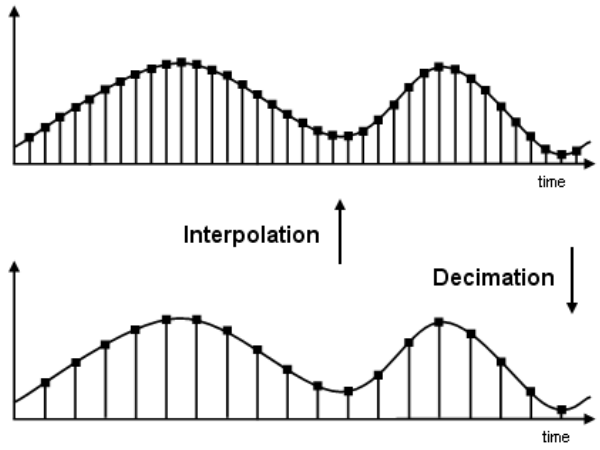

Fig. 1. Process of interpolation (operation done to the lower signal and has an output of the upper signal) and decimation (operation done to the upper signal and has an output of the lower signal). X axis is time while Y axis is amplitude.

To simplify our report, we refer to both of these processes (both interpolation and decimation) as interpolation, since the playback rate is an unknown function input by the user.

Using the STFT of the original signal and depending on the rate parameter that we specify as input for the function, either if it is a speedup or a slowdown, a vector is created to be used as the new time-base samples, which has into consideration this rate of audio modification. It would have more samples if the rate is between 0 and 1 and more samples if the rate is greater than 1.

For each value of the new 2D array that is created to interpolate the signal in accordance with the new time-base samples, the spectral magnitudes of STFT are taken from the two neighbor STFT columns, which are added up, as well as the difference in phase between the successive spectrums of the STFT.

Those two new magnitude and phase values represent the new complex number array that is inserted on each column of the interpolated version of the STFT ('stft_interp'). A graphic version of the process can be seen in Fig. 2.

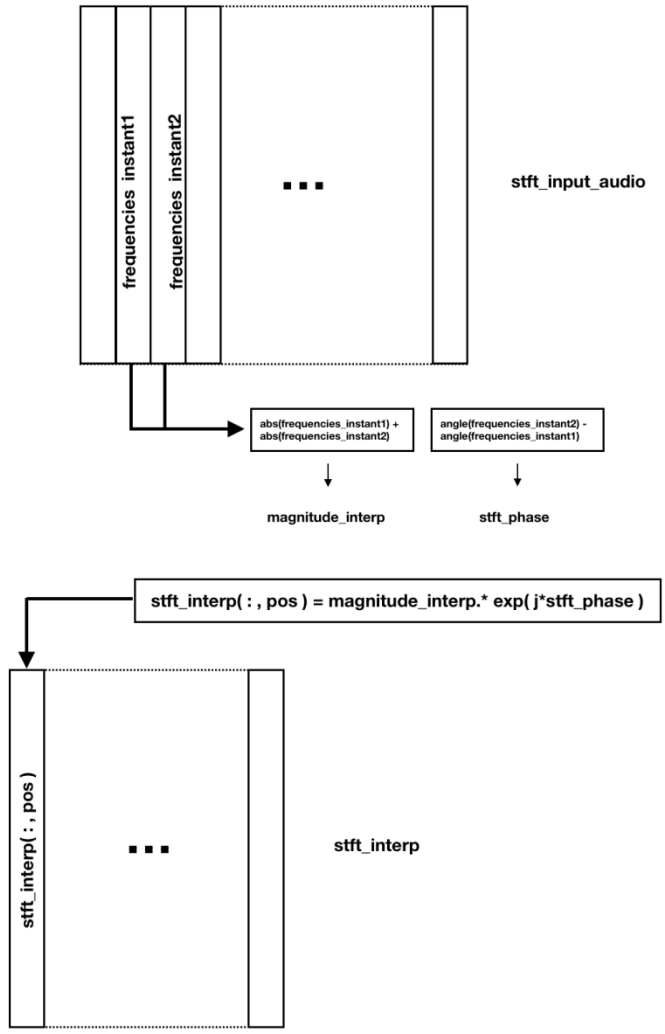

Fig. 2. STFT Interpolation process.

D. Perform the ISTFT

To transform the modified signal to the time domain so that it can be played and be heard, we need to perform an Inverse Short-Time Fourier Transform (ISTFT). For this the Inverse Discrete Fourier Transform (IDFT) has to be applied in every block of the STFT of the modified signal that we calculated interpolating the original version of the STFT of the input signal.

$$y[n,m] = \mathcal{F}^{-1}(X[k,m])$$

With this we get a two dimensional variable, the audio signal in the time domain that is calculated through the IDFT of each block for each of the blocks that the STFT has. Then, a combination of these results by summing time-shifted blocks will result in the desired output signal.

$$x[n] = W_0 \sum_m y[n - mR, m]$$

This is how we can transform the modified signal from its STFT back to the discrete time domain. If we want to hear the

output version we can use the audioread function from MATLAB specifying its sample frequency.

It should be pointed out that before calculating the Inverse Discrete Fourier Transform of each block, the algorithm calculates and concatenates the N/2 complex conjugate points at each spectrogram column, as we need those values to reconstruct the initial signal (the initial signal spectrum is symmetric and contains those points).

E. Overall Alogirthm

```
Input: audio file, rate, fs
Output: output audio, with modified rate

Set Parameters
windowLength ← set window length
hopLength ← windowLength*0.2
hopLength ← windowLength
w ← hamming(windowLength)

Perform STFT to Input Signal
preallocate stft_input_audio

for all stft_input_audio columns
   windowPosition ← set start/stop window index
   input_at_window ← input_audio(windowPosition)
   fft_window ← fft(input_at_window)
   stft_input_audio(: , column) ← fft_window
end

Perform Interpolation Depending on Rate
time_jumps ← set time positions to perform interpolation
                (every (stft_input_audio columns)/rate positions)

stft_phase ← angle(stft_input_audio(: , 1)
for all time_jumps positions
    frequencies_instant1 ← neighbor column 1
    frequencies_instant2 ← neighbor column 2

    magnitude_interp ← abs(frequencies_instant1) + abs(frequencies_instant2)
stft_interp(: , time_jumps position) ← magnitude_interp.*exp(j* stft_phase)
    stft_phase ← angle(frequencies_instant2) - angle(frequencies_instant1)
end

Perform STFT to Input Signal
preallocate output_audio

hops ← set positions that will adjust output signal based on hops
          and window overlap

for all time_jumps positions
    m ← hops(time_jumps position)
    inverse_window ← stft_interp(: , k)'

    for (windowLength/2), decreasing until starting pos
        conj_inverse_window ← conj(inverse_window)
    end
    conc_inverse_window ← [inverse_window, conj_inverse_window]

   window_pos ← (m+1):(m+windowLength)
    inverse_fft_window ← real(ifft(conc_inverse_window))
    output_audio(window_pos) ← output_audio(window_pos) + w.* inverse_fft_window
end

Plot result
plot spectrogram of input audio
plot input/output audio in time and frequency domain
```

III. Results

After having programmed our algorithm for the modification of the playback rate of an audio signal, we have analyzed the accuracy of the algorithm with different signals and different playback rates.

We have plotted the spectrogram of the input audio to show that our algorithm calculates correctly this parameter.

After this, to compare both input and output signals, we have plotted both the input and output signals in both a time domain and a frequency domain. This illustrates perfectly the differences between input and output signals, although if you want to go further, a hearing of both signals is recommended.

A. Performing the STFT

To illustrate the correct implementation of the STFT we have represented it into a plot, which can be seen below in Fig. 3.

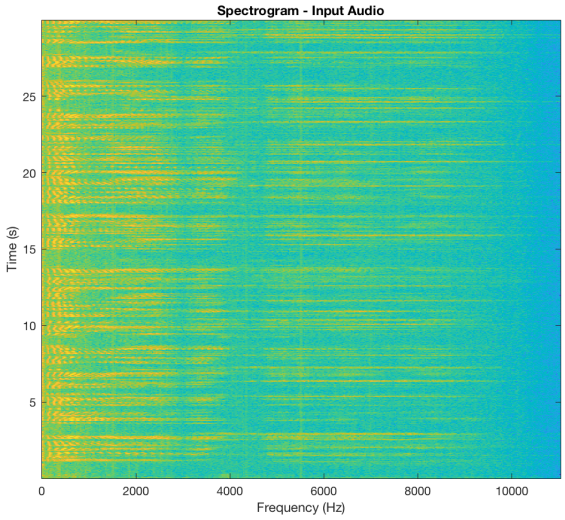

Fig. 3. Spectrogram of the input audio. Colors closer to yellow represent a higher power/frequency. X axis is frequency in hertz and Y axis is time in seconds.

B. Input/Output Comparison Analysis

We have compared input and output signals both in frequency and time. In a frequency analysis, both input and output signals should have the same spectrum so as not to affect the pitch of the output signal in comparison to the input signal. On the contrary, in a time analysis, the output signal should have the length of the input signal multiplied by the playback rate.

To analyze the behavior of the algorithm we have processed an audio signal with a sample frequency and a playback rate of 1.5. Particularly, this will decrease the length of the audio signal in time, while having the same output frequency response or spectrum.

To illustrate the comparison in both frequency and time between an input signal and the output signal we have plotted both of them in the frequency domain as well as in the time domain. You can see the plot in the image below in Fig. 4.

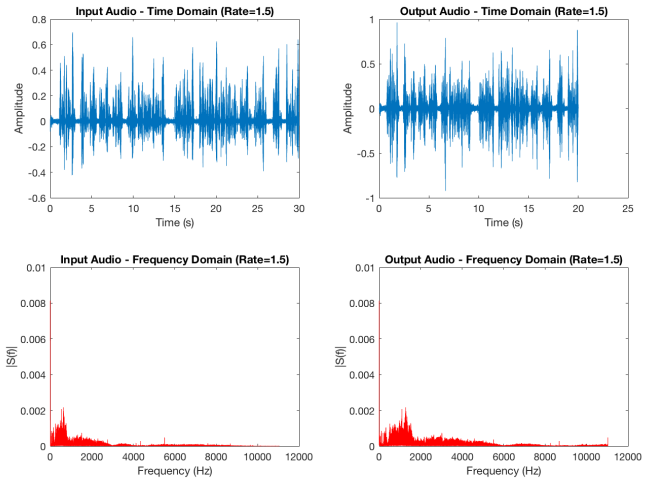

Fig. 4. Comparison between the input signal and the output signal both in the time and the frequency domains with a playback rate of 1.5. Left to right and top to bottom: input audio in the time domain, output audio in the time domain, input audio in the frequency domain, output audio in the frequency domain.

As you can see, the output audio length in time has decreased from 30 seconds to 20 seconds. However, the spectrum of the signal is almost the same. It is not exactly the same because there is an error due to the process of interpolation and the blocking size effect.

To better cover the analysis of the behavior of the algorithm, we have also processed an audio signal with certain sample frequency and a playback rate of 0.5. Particularly, this will increase the length of the audio signal in time, while having the same output frequency response or spectrum.

To illustrate the comparison in both frequency and time between this input signal and the output signal we have plotted both of them in the frequency domain as well as in the time domain. You can see the plot in the image below in Fig. 5.

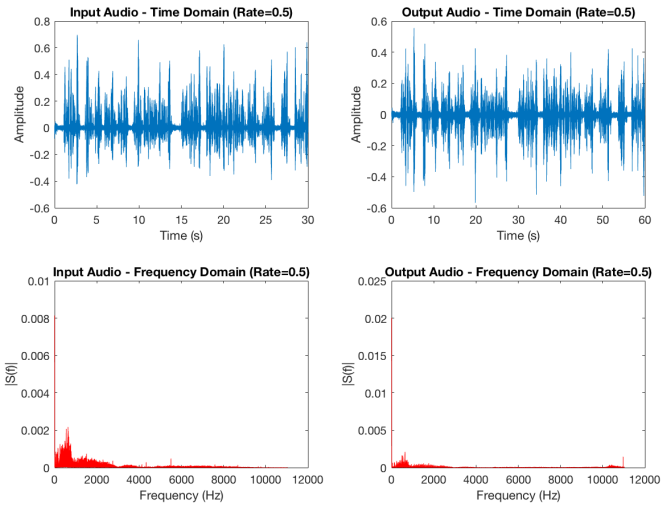

Fig. 5. Comparison between the input signal and the output signal both in the time and the frequency domains with a playback rate of 0.5. Left to right and top to bottom: input audio in the time domain, output audio in the time domain, input audio in the frequency domain, output audio in the frequency domain.

As you can see, the output audio length in time has increased from 30 seconds to 60 seconds (doubled by the rate of 0.5). However, the spectrum of the signal is almost the same, not entirely though because there is an error due to the process of interpolation and the selected blocking size.

In both cases, there is a small distortion of the frequencies, both for the 1.5 playback rate and for the 0.5 playback rate. This will distort some of the frequencies, but the overall pitch is not heavily affected. In fact, it is very accurate and similar to the original one.

C. Eligibility of Output Audio

Eligibility of the audios should also be a parameter we should have into consideration when analyzing the accuracy of the algorithm. There is a distortion due to the abstraction of new blocks for the STFT, and we get a slightly "metalized" audio, but the overall pitch is very accurate and similar to the original one, which is the goal of this project.

IV. Conclusion

To conclude, we have to say that this challenge has been an interesting project to work on. When you think of interpolation (or decimation) of a discrete signal, you also have in mind that the spectrum of the signal is modified proportionally to the ratio of interpolation (or decimation) that it is chosen. If we interpolate (or decimate) a signal, we are abstracting (or eliminating) samples in time and so, the spectrum is reduced (or amplified). If the requirement is to not modify the pitch of the input signal, then the problem becomes much more challenging, because in this case the spectrum should stay and remain the same.

We have been able to come up with a really decent solution and algorithm that does not modify the pitch or spectrum of the original signal while achieving the modification on the playback rate. The results are solid and the algorithm is efficient.

.