**Project: Phase Vocoder**

**Group members:**

1. Jeevtesh Singh (js10359)
2. Muniba Bashir (mb6496)

**Introduction**:

Phase Vocoder is described as a group of techniques that are used to transform audio signals in frequency domain instead of transforming them in time domain. While most of the transformation on the audio signal can be performed in the time domain, there are limitations on it. These include the ability to only perform transformations on monophonic audio signals, and for pitch shifting and time-scaling in time domain, there is constraint on the modification factor (should not be larger than ± 20% to ±30%)[1]. Hence, we use phase vocoder techniques in our project to implement the following transformations on audio signals:

1. Robotization
2. Whisperization
3. Pitch-Shifting
4. Denoising

To perform the phase vocoder techniques in the frequency domain, we first have to perform Short Time Fourier Transform (STFT) on the input audio signal. A basic implementation of phase vocoder is as follows:

The STFT of a signal can be written as Discrete Fourier Transform of the windowed input signal and is as follows:

[2]

It is important to see that now we have domain index *m*, and frequency domain index *k*. Hence *X[m,k]* can be visualized as follows:

A screenshot of text

Description automatically generated

[2]

Figure 1

This visualization is important in writing the algorithm for phase vocoder. *X[m,k]* is DFT of a point ,with frequency index *m* and time index *k*, can be written as or . Hence *X[m,k]*, in python, is a matrix of size (ndarray in python of shape ) and all the computations on it will be dealing with each and every distinct *m* and *k*. One more important factor to notice is the window type and length for STFT, which plays a very important role in both STFT and inverse STFT.

**Project:**

The basic implementation of the project can be shown in a flow chart in figure 2.

Goal 6:

Goal 5:

Goal 4:

Goal 3:

Goal 2:

Goal 1:

Figure 2

* Basic Algorithm for STFT:

Our project used the library SciPy for the implementation of STFT and inverse STFT. This was done using:

[*f, t, Zxx*] = **scipy.signal.stft(***x***,***sampling rate***,***window='hann'***,***nperseg=256***,***noverlap=None****)***[5]

Where *f* and *t* are frequency and time indices, and *Zxx* is the STFT corresponding to each *f* and *t*.

And inverse STFT[5] was computed using:

[*t, xmod*] = **scipy.signal.istft(***Zxx***,***sampling rate***,***window='hann'***,***nperseg=256,**noverlap=None***)**

Where *t* is the time and *xmod* is the modified time domain signal.

For these two functions, we have used Hanning window and a sampling rate of 44.1 kHz for computer speaker output. Additionally, our program uses block length of 4096 when output is sent through computer speakers. If using headphones, the recommended sampling rate for our program is 32 kHz. “*nperseg”* is the length of window used and is adjusted based on each phase vocoder technique that is implemented, same goes for *noverlap* which is the overlap between windows in the STFT computation.

* Phase Vocoder Techniques:

1. *Robotization*

Robotization is implemented by using the magnitude of *Zxx* while computing the inverse STFT. Hence the output is as follows:

For this feature, we keep our window size *nperseg* = 512. This was implemented in python using the library *numpy*, specifically by *np.absolute()*. Robotization works best with a window size between 256 to 1024 samples[3]. The effect of robotization is also observed in the frequency of the output.

[3]

Where *fs* is the sampling rate and *H* is the hop length of the window.

1. *Whisperization*

Whisperization is implemented using magnitude if *Zxx* and multiplying it with a random phase ϕk. This can be implemented using the following equation, where ϕk is random.

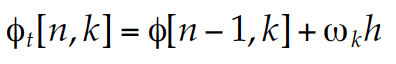
The implementation in python included generation of random (ndarray) for each value (both time and frequency) of *Zxx* and then multiplication with the magnitude of *Zxx*. Then we take the inverse STFT of this modified frequency domain signal. Whisperization works best with shorter window size, roughly between 64 to 256 samples[3]. Our program implements it with the window size of 256.

1. *Pitch-Shifting*

Pitch-shifting is implemented by first stretching the signal in time and then sampling at a higher rate to preserve the time-scale and only cause the pitch to change[3]. However, change sampling rate every instant is not very practical. Hence, an interpolation is done between the output samples to increase the number of samples between two time instants.

In our program we have implemented time stretching by assuming that the input signal can be written as a combination of sinosoids *xsine*, the derivation of the angle increments is then performed on the frequency domain equivalent of *xsine*. It is then assumed (under certain conditions discussed in [1]) that these increments replicate the increments in angle of the STFT of input signal. Under this condition, we can modify the phase as follows:

Target Phase = sum of previous unwrapped phase + expected phase increment

 [3]

*ωk* = frequency of that index, *ϕt[n,k]* = target phase, h = hop size

A close up of a sign

Description automatically generated [3]

*ϕd[n,k]* = phase deviation

A close up of a logo

Description automatically generated [3]

Where arg*ϕd[n,k]* is the principle argument which stays between ±π

The important distinguishing feature between pitch-shifting from other vocoder techniques is the hop ratio which is defined as where R is the stretch factor, synthesis hop size is the size of window overlap for the inverse STFT and analysis hop size is the size of window overlap for STFT of the input signal. In all the other cases, the synthesis and analysis hop are equal hence R = 1. For pitch-shifting and time-scaling, we define synthesis hop size *hs* and analysis hop size *ha*. Now to compute the phase in input signal, we use *ha* as follows:

A close up of a sign

Description automatically generated [3]

Since we have obtained *ϕi[n,k]* and *Ai[n,k]*, we can then compute the output phase and magnitude as follows using *hs*:

A picture containing object, clock, watch

Description automatically generated [3]

A picture containing object

Description automatically generated [3]

A picture containing object

Description automatically generated [3]

Our output *y(n)* is just a summation of products of magnitude and phase that have been calculated and is as follows:

A close up of a logo

Description automatically generated [3]

In our program, we have used the default *ha=* 80 and *hs* = 80. User can however change these by changing hop ratio (or stretch factor) *R* by using the GUI slider. In doing so, the value of *ha* and *hs* change through the relation . Once these values are obtained, they are used in the STFT and inverse STFT computation using *scipy.signal.stft()* (and *scipy.signal.istft()*) in the *noverlap* field. Previously, this field was set to ‘none’ signifying that there is no over-lap in the STFT adjacent windows. However, in this technique we set this field to be *nperseg – ha* for STFT and *nperseg – hs* for inverse STFT. For pitch-shifting we set the value of *nperseg* to be 512. This combined with the iterations of the phase angle creates the desired pitch-shift result as we expect. However, the computation requires time and is usually accompanied by reverberation. That turns out to be an artifact of phase vocoder method for pitch-shifting and some research papers have been devoted to solve this issue.

1. *Denoise*

The denoise feature is implemented in the frequency domain using threshold function. The threshold function can be described as follows:

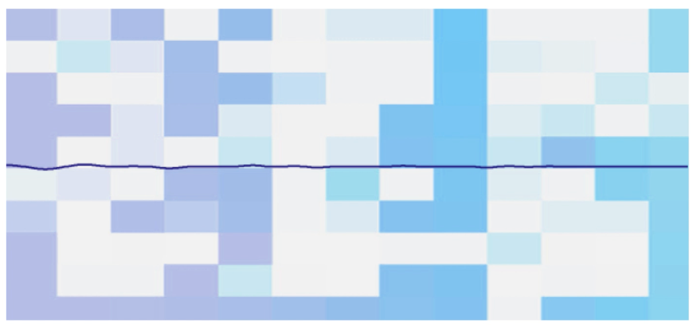
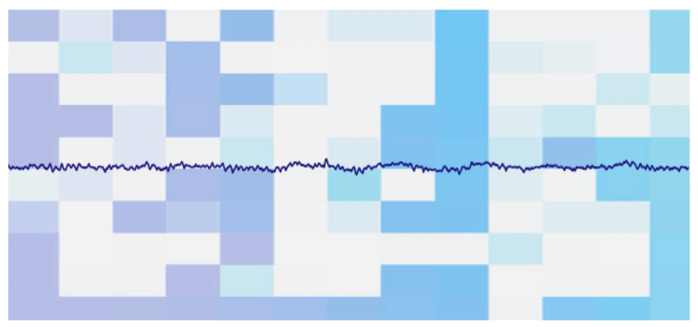
*A close up of a clock

Description automatically generatedA picture containing table, air

Description automatically generated*[4]

Figure 3

We selected value of *T* empirically by observing the input and output wave. For our program, we selected the value of *T* = 0.5 and applied this threshold function on the STFT of input signal. We selected *nperseg* = block length i–e 1024 for this feature. The output of this feature can be seen as follows:



Output Audio signal (denoised)

Input Audio signal (noisy)

Result of the denoise feature can be seen in these figures above.

1. *Graphical User Interface (GUI)*

We have also included the following features in GUI:

1. Pyplot embedded in GUI

* This is implemented using *FigureCanvasTkAgg* from *matplotlib.backends.backend\_tkagg*
* Background of pyplot is modified using pyplot attributes

1. Scale to change hop size

* Increases/decreases *R* which in turn changes *hs* and *ha*

1. Scale to change volume

* Increases/decreases gain

1. Buttons to switch modes:

* Normal – No processing
* Robotization
* Whisperization
* Pitch Shift
* De-Noise
* Quit

In some of these Phase Vocoder techniques, we use *signal.resample(x, BLOCKLEN)* to readjust the length of output signal since after inverse STFT the length of output signal can vary owing to window size and overlap between adjacent windows.

**Summary:**

Following is a glimpse of our program:

A screenshot of a cell phone

Description automatically generated

All the features of this program work as expected. There are some glitches that show up in the pitch-shift which are universal problems encountered by a phase vocoder. Some research papers are published regarding this issue.

**Conclusion:**

There are many phase vocoder techniques, this project encompasses robotization, whisperization, one way of implementing pitch shifting and denoising. Phase vocoder techniques use processes which are generally not iterative, compared to the time domain processes which are mostly iterative. This reduces error propagation; however, it increases the computational cost which is less for the time domain systems. There is yet more research to be done on phase vocoder methods to make them more efficient to use.

**References:**

[1]Laroche, J., and M. Dolson. “Improved Phase Vocoder Time-Scale Modification of Audio.” *IEEE Transactions on Speech and Audio Processing*, vol. 7, no. 3, 1999, pp. 323–332., doi:10.1109/89.759041.

[2] E85.2607: Lecture 7 Phase Vocoder. [www.ee.columbia.edu/~ronw/adstspring2010/lectures/lecture7.pdf](http://www.ee.columbia.edu/~ronw/adstspring2010/lectures/lecture7.pdf)

[3] Reiss, Joshua D., and Andrew McPherson. “The Phase Vocoder.” Audio Effects : Theory, Implementation and Application, CRC Press LLC, 2014, pp. 189–211.

[4] NYU Classes: Digital Signal Processing Lab (ECE 4163 / ECE 6183), Demo 24 Exercises: Real-time STFT, denoising, and robotization

[5] “Scipy.signal.stft.” Scipy.signal.stft - SciPy v1.4.1 Reference Guide, <https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.stft.html>

[6] Pitch Shifting and Time Dilation Using a Phase Vocoder in MATLAB - MATLAB & Simulink, <https://www.mathworks.com/help/audio/examples/pitch-shifting-and-time-dilation-using-a-phase-vocoder-in-matlab.html>