Military Institute of Science and Technology

DEPARTMENT OF ELECTRICAL ELECTRONIC AND COMMUNICATION ENGINEERING

Course Title: Digital Signal Processing

Course Code: EECE 312

Group No: 09

Project Name: Replicating and Extending the Traditional Phase Vocoder: A Comprehensive MATLAB GUI Implementation

Course Instructor:

1. Maj Kazi Newaj Faisal
2. Maj Marzuka Ahmed Jumana
3. Asst. Prof. A. A. M. Shah Sadman
4. Lec Reefah Tasnia
5. Lec Tareque Bashar Ovi
6. Lec Zia Md Galib Ul Alim

Group Members:

Tashnova Elahi	202116170
Pretom Das	202216072
Sheikh Iffat Ara Prianka	202216091
Zakia Sultana Simin	202216112
Abdullah Ahnaf Karim	202216303

Table of Contents (TOC)

Table of Contents (TOC) **1**
Theoretical Analysis **3**
Short-Time Fourier Transform (STFT) 3
Time-Stretching (α) 3
Phase Adjustment 3
Pitch-Shifting (β) 4
Inverse Short-Time Fourier Transform (ISTFT) 4
Phase Unwrapping 4
Window Function 5
Overlap-Add (OLA) Method 5
Echo Effect 5
Equation: 5
Reverb Effect 5
Distortion Effect 5
Resampling 6
Block Diagram **6**
Project Procedure **7**
MATLAB code of different blocks **9**
Input Block : 9
1. Load Audio block : 10
2. Time scaling : 11
3. Pitch Scaling Block : 12
4. Effects Block : 13
4.a Echo : 13
4.b Reverb : 14
4.c Distortion : 14
5. Choose Presets Block : 14
6. Process Audio Block : 15
7. Save and Play Block : 15
Process block : 16
1. PV algorithm : 18
2. Effect Processing : 19
3. Resampling : 19
4. Undo : 20
Output block : 20
1. Waveform display : 21
2. FFT : 21
3. Audio info display : 22
4. Audio output and wave : 23
Total code : 24
Simulation Results **26**
GUI: 27
Voice loaded: 27

Slow motion(alpha = 1.5) :.....28
Fast Forward(alpha = 0.5) :.....28
Revered:.....29
Echo:.....29
Discussion.....30
Improvement Ideas.....31
1. Functionality Enhancements.....31
2. User Experience Improvements.....31
3. Technical Optimizations.....32
4. Advanced Features.....32
Reference.....33

Theoretical Analysis

The Phase Vocoder is a frequency-domain technique used for time-stretching and pitch-shifting audio signals. It operates by modifying the Short-Time Fourier Transform (STFT) of the signal while preserving the phase relationships between frequency components.

Short-Time Fourier Transform (STFT)

The STFT is used to analyze the frequency content of a signal over time. It divides the signal into overlapping frames, applies a window function, and computes the Fourier Transform for each frame.

Equation:

$$X(m,k) = \sum_{n=0}^{N-1} x(n+mH) \cdot w(n) \cdot e^{-j2\pi kn/N}$$

- $x(n)$ = Input signal.
- $w(n)$ = Window function (e.g., Hann window).
- N = Window size.
- H = Hop size (number of samples between consecutive frames).
- m = Frame index.
- k = Frequency bin index.

Time-Stretching (α)

Time-stretching changes the duration of the signal without altering its pitch. This is achieved by modifying the synthesis hop size (Hs) relative to the analysis hop size (Ha).

Synthesis Hop Size equation:

$$Hs = \alpha \cdot Ha$$

- α: Time-stretching factor

$\alpha > 1$ for slowing down	$\alpha < 1$ for speeding up

Phase Adjustment

To maintain phase continuity, the phase of each frequency bin is adjusted based on the time-stretching factor.

Phase Difference equation:

$$\Delta\phi(m,k) = \phi(m,k) - \phi(m-1,k)$$

- $\phi(m,k)$ = Phase of bin k in frame m

Instantaneous Frequency equation:

$$\omega(m,k)=\Delta\phi(m,k)/Ha$$

Adjusted Phase equation:

$$\phi synth(m,k)=\phi synth(m-1,k)+\omega(m,k)\cdot Hs$$

Pitch-Shifting (β)

Pitch-shifting changes the pitch of the signal without altering its duration. This is achieved by resampling the time-stretched signal.

Resampling equation:

$$y(n)=resample(x(n),P,Q)$$

- P/Q = Resampling ratio (P/Q=β for pitch-shifting).
- β = Pitch-shifting factor

$\beta>1$ for higher pitch	$\beta<1$ for lower pitch

Inverse Short-Time Fourier Transform (ISTFT)

The Inverse Short-Time Fourier Transform (ISTFT) is the process of reconstructing the original signal from its STFT representation by summing the inverse Fourier transforms of the segmented windows. Both techniques are commonly used in time-frequency analysis and signal processing.

Equation:

$$y(n) = \sum_{m=-\infty}^{\infty} \left[\frac{1}{N} \sum_{k=0}^{N-1} Y(m,k) \cdot e^{j2\pi kn/N} \right] \cdot w(n - mH_s)$$

- Y(m,k) = Modified STFT.
- w(n) = Synthesis window function.

Phase Unwrapping

Phase unwrapping ensures that the phase differences between consecutive frames are correctly interpreted. This avoids phase discontinuities, which can cause artifacts.

Equation :

$$\Delta\phi_{\text{unwrap}}(m,k) = \Delta\phi(m,k) + 2\pi \cdot \text{round}\left(\frac{\Delta\phi(m-1,k) - \Delta\phi(m,k)}{2\pi} \right)$$

Window Function

A window function is applied to each frame to reduce spectral leakage. Common choices include the Hann window and Hamming window.

Hann Window:

$$w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N-1}\right)\right), \quad 0 \leq n \leq N-1$$

Overlap-Add (OLA) Method

The OLA method is used to reconstruct the time-domain signal from the modified STFT. It ensures smooth transitions between frames.

Equation:

$$y(n) = \sum_{m=-\infty}^{\infty} y_m(n - mH_s) \cdot w(n - mH_s)$$

- ym(n) = Time-domain signal for frame

Echo Effect

The echo effect adds delayed and attenuated copies of the signal to the original.

Equation:

$$y(n)=x(n)+\alpha \cdot x(n-D)$$

- D = Delay in samples.
- α =Attenuation factor.

Reverb Effect

Reverb simulates the natural reflections of sound in a room. It is typically implemented using a feedback delay network (FDN).

Equation:

$$y(n) = x(n) + \sum_{i=1}^{M} \alpha_i \cdot y(n - D_i)$$

- Di = Delay lengths.
- αi =+ Feedback gains

Distortion Effect

Distortion introduces non-linearities into the signal, often using a hyperbolic tangent function.

Equation:

$$y(n)=\tanh(\alpha \cdot x(n))$$

- α: Distortion gain

Resampling

Resampling changes the sampling rate of the signal, which is used for pitch-shifting.

Equation:

$$y(n) = \sum_{k=-\infty}^{\infty} x(k) \cdot \text{sinc}\left(\frac{n \cdot Q - k \cdot P}{P}\right)$$

- sinc(x)=sin(πx)/πx ;Sinc function.

Block Diagram

The block diagram below illustrates the key components and workflow of the Phase Vocoder system, which enables time-stretching and pitch-shifting of audio signals.

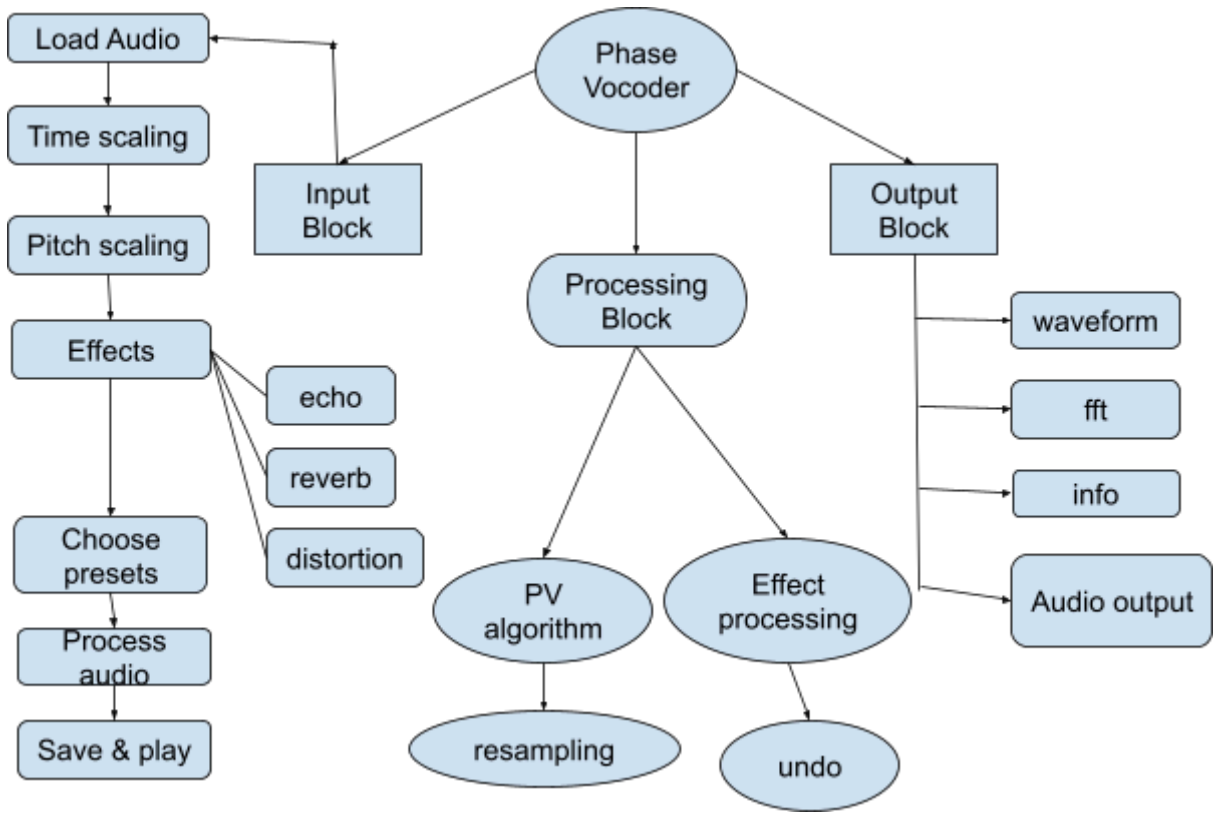

Figure 1 : Block Diagram of the Code

Project Procedure

This segment outlines the step-by-step procedure for developing this project. Each step describes the task to be performed and the expected outcome.

1. Initialize the GUI :

Task	Outcome
Set up the main figure, panels, buttons, sliders, and axes for the Graphical User Interface (GUI).	A user-friendly interface is created, providing controls for loading audio, adjusting parameters, and displaying results.

2. Load Audio File :

Task	Outcome
Implement a function to load an audio file using a file dialog, convert it to mono, and store it in the `original_audio` variable.	The audio file is successfully loaded, and its waveform and metadata (e.g., file name, duration, sampling rate) are displayed in the GUI.

3. Set Up Controls for Time and Pitch Scaling :

Task	Outcome
Add sliders and edit boxes for adjusting time scaling (α) and pitch scaling (β) factors. Link the sliders and edit boxes for real-time updates.	Users can interactively adjust time and pitch scaling parameters, and the changes are reflected in real time.

4. Implement Phase Vocoder Algorithm :

Task	Outcome
Develop the `phase_vocoder` function to perform time scaling without altering the pitch of the audio signal.	The audio signal is time-stretched or compressed based on the selected scaling factor, while preserving its pitch.

5. Implement Resampling for Pitch Scaling :

Task	Outcome
Use the `resample` function to adjust the pitch of the audio signal based on the selected scaling factor.	The audio signal's pitch is shifted up or down without changing its duration.

6. Add Audio Effects (Reverb, Echo, Distortion) :

Task	Outcome
Implement functions to apply reverb, echo, and distortion effects to the audio signal.	Users can enhance the audio with effects such as reverb (simulating space), echo (delayed repetitions), or distortion (gritty sound).

7. Process Audio :

Task	Outcome
Combine time scaling, pitch scaling, and effects into a single `process_audio` function. Save the current state of the audio to an undo stack before processing.	The audio signal is processed based on the selected parameters, and the modified signal is displayed in the GUI.

8. Implement Undo Functionality :

Task	Outcome
Add an `undo_processing` function to revert to the previous state of the audio signal using the undo stack.	Users can undo the last processing step and restore the previous version of the audio signal.

9. Display Waveform and FFT :

Task	Outcome
Implement functions (`plot_audio` and `plot_fft`) to display the waveform and frequency spectrum (FFT) of the original and modified audio signals.	Users can visualize the time-domain waveform and frequency-domain spectrum of the audio signals.

10. Display Audio Information :

Task	Outcome
Implement the `update_info_panel` function to display metadata about the audio file (e.g., file name, sampling rate, duration, file size).	Users can view detailed information about the loaded audio file in the GUI.

11. Add Playback Controls :

Task	Outcome
Implement functions (`play_original`, `play_modified`, and `stop_audio`) to play the original or modified audio and stop playback.	Users can listen to the original or modified audio and stop playback as needed.

12. Save Modified Audio :

Task	Outcome
Implement a `save_audio` function to save the modified audio signal as a WAV file using a file dialog.	Users can save the processed audio signal to their desired location.

13. Test and Debug :

Task	Outcome
Test the GUI thoroughly to ensure all functionalities (loading, processing, playback, saving) work as expected. Debug any issues.	A fully functional and bug-free Phase Vocoder GUI is ready for use.

14. Final Deliverables :

Task	Outcome
Package the project with all necessary files and documentation.	A complete Phase Vocoder GUI application is delivered, allowing users to load, process, and save audio files with time scaling, pitch scaling, and effects.

This procedure ensures the project is developed systematically, with clear goals and outcomes at each step.

MATLAB code of different blocks

Input Block :

This block initializes the GUI and sets up the main variables and UI components.

```
function phase_vocoder_gui()
    % Initialize GUI figure
    fig = figure('Name', 'Phase Vocoder', 'NumberTitle', 'off', ...
        'Units', 'normalized', 'Position', [0, 0, 1, 1], 'MenuBar',
'none', 'ToolBar', 'none');

    % Initialize variables
    original_audio = [];
    modified_audio = [];
    fs = 44100; % Default sampling rate
    undo_stack = {}; % Stack for undo functionality
    audio_file_info = struct('FileName', '', 'FileSize', 0, 'NumChannels',
```

```
0, 'BitDepth', 0); % Audio file metadata

    % Create main panel
    main_panel = uipanel('Parent', fig, 'Title', 'Phase Vocoder', ...
        'Units', 'normalized', 'Position', [0.02, 0.02, 0.96, 0.96], ...
        'BackgroundColor', [0.95, 0.95, 0.95]);
```

Code Explanation :

- phase_vocoder_gui(): Initializes the Phase Vocoder GUI.
- fig = figure(...): Creates a full-screen figure window with the title "Phase Vocoder."
- Initialize variables:
  - original_audio and modified_audio: Placeholder variables for storing audio data.
  - fs = 44100: Sets the default sampling rate to 44.1kHz.
  - undo_stack = {}: Initializes an empty undo stack for audio processing actions.
  - audio_file_info: Stores metadata of the audio file (filename, size, channels, bit depth).
- Create main panel: Adds a panel to the GUI with a title "Phase Vocoder" and a light gray background.

1. **Load Audio block :**

This block allows the user to load an audio file.

```
% Load Audio button
    uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String', 'Load
Audio', ...
        'Units', 'normalized', 'Position', [0.02, 0.9, 0.1, 0.05],
'Callback', @load_audio);

    function load_audio(~, ~)
        [file, path] = uigetfile({'*.wav;*.mp3;*.ogg;*.flac;*.au', 'Audio
Files'}); % Open file dialog
        if file == 0
            return; % User canceled
        end
        [x, fs] = audioread(fullfile(path, file)); % Read audio file
        original_audio = mean(x, 2); % Convert to mono
        modified_audio = original_audio; % Initialize modified audio
        undo_stack = {}; % Clear undo stack

        % Update audio file info
        audio_file_info.FileName = file;
        audio_file_info.FileSize = dir(fullfile(path, file)).bytes / 1024;
% Size in KB
```

```
        audio_file_info.NumChannels = size(x, 2);
        info = audioinfo(fullfile(path, file));
        % audio_file_info.BitDepth = info.BitsPerSample;

        % Update plots and info panel
        plot_audio(ax_original, original_audio, 'Original Signal');
        plot_audio(ax_modified, modified_audio, 'Modified Signal');
        plot_fft(ax_fft, original_audio, fs);
        update_info_panel(ax_info, original_audio, fs, audio_file_info);
        progress_bar.String = 'Audio Loaded';
    end
```

Code Explanation :

- ➢ Creates a push button labeled "Load Audio" in main_panel.
- ➢ Clicking it calls load_audio().
- ➢ Opens a file dialog (uigetfile) for .wav, .mp3, .ogg, .flac, .au files.
- ➢ Reads the selected audio (audioread), converts stereo to mono (mean(x,2)).
- ➢ Stores file name, size (KB), and channel count.
- ➢ Uses audioinfo() for metadata (BitDepth is commented out).
- ➢ Plots waveform for original & modified audio.
- ➢ Plots FFT of the original signal.
- ➢ Updates info panel and progress bar ("Audio Loaded").

2. **Time scaling :**

This block allows the user to adjust the time scaling factor (α).

```
   % Time scaling controls
    uicontrol('Parent', main_panel, 'Style', 'text', 'String', 'Time
Scaling (α)', ...
        'Units', 'normalized', 'Position', [0.02, 0.8, 0.1, 0.03],
'HorizontalAlignment', 'left', ...
        'BackgroundColor', [0.95, 0.95, 0.95]);
    alpha_slider = uicontrol('Parent', main_panel, 'Style', 'slider', ...
        'Units', 'normalized', 'Position', [0.02, 0.77, 0.1, 0.02], 'Min',
0.5, 'Max', 2, 'Value', 1);
    alpha_edit = uicontrol('Parent', main_panel, 'Style', 'edit', ...
        'String', '1.0', 'Units', 'normalized', 'Position', [0.02, 0.74,
0.1, 0.03]);

    % Link slider and edit box
```

```
    alpha_slider.Callback = {@update_slider, alpha_edit};
    alpha_edit.Callback = {@update_edit, alpha_slider};
```

Code Explanation :

Time Scaling Controls

- ➢ Label: Displays "Time Scaling (α)" in main_panel.
- ➢ Slider (alpha_slider): Adjusts time scaling between 0.5x (slower) and 2x (faster), default is 1.0x.
- ➢ Edit Box (alpha_edit): Shows the current α value (editable).

Slider & Edit Box Synchronization :

- ➢ alpha_slider.Callback = {@update_slider, alpha_edit};
  - ○ Updates the edit box when the slider moves.
- ➢ alpha_edit.Callback = {@update_edit, alpha_slider};
  - ○ Updates the slider when a new value is entered.

3. **Pitch Scaling Block :**

This block allows the user to adjust the pitch scaling factor (β).

```
    % Pitch scaling controls
    uicontrol('Parent', main_panel, 'Style', 'text', 'String', 'Pitch
Scaling (β)', ...
        'Units', 'normalized', 'Position', [0.02, 0.68, 0.1, 0.03],
'HorizontalAlignment', 'left', ...
        'BackgroundColor', [0.95, 0.95, 0.95]);
    beta_slider = uicontrol('Parent', main_panel, 'Style', 'slider', ...
        'Units', 'normalized', 'Position', [0.02, 0.65, 0.1, 0.02], 'Min',
0.5, 'Max', 2, 'Value', 1);
    beta_edit = uicontrol('Parent', main_panel, 'Style', 'edit', ...
        'String', '1.0', 'Units', 'normalized', 'Position', [0.02, 0.62,
0.1, 0.03]);

    % Link slider and edit box
    beta_slider.Callback = {@update_slider, beta_edit};
    beta_edit.Callback = {@update_edit, beta_slider};
```

Code Explanation :

Pitch Scaling Controls

- ➢ Label: Displays "Pitch Scaling (β)" in main_panel.

- ➢ Slider (beta_slider): Adjusts pitch between 0.5x (lower) and 2x (higher), default is 1.0x.
- ➢ Edit Box (beta_edit): Displays the current β value (editable).

Slider & Edit Box Synchronization

- ➢ beta_slider.Callback = {@update_slider, beta_edit}; → Updates edit box when slider moves.
- ➢ beta_edit.Callback = {@update_edit, beta_slider}; → Updates slider when a new value is entered.

4. **Effects Block :**

This block allows the user to apply effects like reverb, echo, and distortion.

```
   % Effect options
    uicontrol('Parent', main_panel, 'Style', 'text', 'String', 'Effects',
...
        'Units', 'normalized', 'Position', [0.02, 0.47, 0.1, 0.03],
'HorizontalAlignment', 'left', ...
        'BackgroundColor', [0.95, 0.95, 0.95]);
    effect_menu = uicontrol('Parent', main_panel, 'Style', 'popupmenu',
...
        'String', {'None', 'Reverb', 'Echo', 'Distortion'}, ...
        'Units', 'normalized', 'Position', [0.02, 0.44, 0.1, 0.03]);
```

Code Explanation :

Effect Options

- ➢ Label: Displays "Effects" in main_panel.
- ➢ Dropdown Menu (effect_menu): Allows users to select an effect from:
  - ○ None (default)
  - ○ Reverb
  - ○ Echo
  - ○ Distortion

4.a Echo :

Echo creates a delayed repetition of the audio signal, simulating the effect of sound reflecting off distant surfaces.

```
case 'Echo'
    delay = 0.3; % 300 ms delay
    modified_audio = echoEffect(modified_audio, fs, delay);
```

Code Explanation :

- ➢ Case 'Echo': This part is likely part of a switch-case structure, handling the echo effect.

- delay = 0.3;: Sets the echo delay to 300 milliseconds.
- modified_audio = echoEffect(modified_audio, fs, delay);: Applies the echo effect to the modified_audio signal using the specified delay and the sampling frequency (fs), and stores the result in modified_audio.

4.b **Reverb :**

Reverb simulates the natural reflections of sound in a physical space, such as a room or hall. It adds a sense of space and depth to the audio.

```
case 'Reverb'
    h = reverberator('PreDelay', 0.05, 'WetDryMix', 0.3); % Fixed
PreDelay value
    modified_audio = h(modified_audio);
```

Code Explanation :

- Case 'Reverb': Applies reverb effect.
- h = reverberator('PreDelay', 0.05, 'WetDryMix', 0.3);: Creates reverb with 50ms pre-delay and 30% wet signal.
- modified_audio = h(modified_audio);: Applies reverb to audio.

4.c Distortion :

Distortion alters the waveform of the audio signal by clipping or saturating it, creating a gritty or harsh sound.

```
case 'Distortion'
    modified_audio = tanh(5 * modified_audio); % Non-linear distortion
```

Code Explanation :

- Case 'Distortion': Applies distortion effect.
- modified_audio = tanh(5 * modified_audio);: Applies non-linear distortion using the hyperbolic tangent function.

5. **Choose Presets Block :**

This block allows the user to choose from predefined presets.

```
    % Preset options
    uicontrol('Parent', main_panel, 'Style', 'text', 'String', 'Presets',
...
```

```
        'Units', 'normalized', 'Position', [0.02, 0.56, 0.1, 0.03],
'HorizontalAlignment', 'left', ...
        'BackgroundColor', [0.95, 0.95, 0.95]);
    preset_menu = uicontrol('Parent', main_panel, 'Style', 'popupmenu',
...
        'String', {'Custom', 'Chipmunk (β=1.5)', 'Slow Motion (α=1.5)',
'Fast Forward (α=0.5)', 'Robot (α=1, β=0.8)'}, ...
        'Units', 'normalized', 'Position', [0.02, 0.53, 0.1, 0.03],
'Callback', @apply_preset);
```

Code Explanation :

- Preset options: Defines UI controls for preset audio effects.
- uicontrol('Parent', main_panel, 'Style', 'text', 'String', 'Presets', ...): Adds a text label for the presets section.
- preset_menu = uicontrol('Parent', main_panel, 'Style', 'popupmenu', ...): Creates a dropdown menu with preset options like "Chipmunk," "Slow Motion," "Fast Forward," and "Robot."
- Callback, @apply_preset: Calls the apply_preset function when a selection is made from the dropdown.

6. **Process Audio Block :**

This block processes the audio based on the selected parameters.

```
   % Process button
    uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String',
'Process', ...
        'Units', 'normalized', 'Position', [0.02, 0.38, 0.1, 0.05],
'Callback', @process_audio);
```

Code Explanation :

- ➢ Process button: Creates a button for processing audio.
- ➢ uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String', 'Process', ...): Adds a button labeled "Process."
- ➢ Callback, @process_audio: Triggers the process_audio function when the button is clicked.

7. **Save and Play Block :**

This block allows the user to save the modified audio and play the original or modified audio.

```
% Save button
    uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String',
```

```
'Save Output', ...
       'Units', 'normalized', 'Position', [0.02, 0.26, 0.1, 0.05],
'Callback', @save_audio);

    % Play buttons
    uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String',
'Play Original', ...
       'Units', 'normalized', 'Position', [0.02, 0.2, 0.1, 0.05],
'Callback', @play_original);
    uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String',
'Play Modified', ...
       'Units', 'normalized', 'Position', [0.02, 0.14, 0.1, 0.05],
'Callback', @play_modified);

    % Stop button
    uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String',
'Stop', ...
       'Units', 'normalized', 'Position', [0.02, 0.08, 0.1, 0.05],
'Callback', @stop_audio);
```

Code Explanation :

- ➢ Save button: Adds a button to save the output audio.
  - Callback, @save_audio: Triggers the save_audio function.
- ➢ Play buttons: Creates buttons to play original and modified audio.
  - Callbacks, @play_original and @play_modified: Play respective audio versions when clicked.
- ➢ Stop button: Adds a button to stop audio playback.
  - Callback, @stop_audio: Stops audio when clicked.

Process block :

This block handles the processing of the audio signal based on the selected parameters (time scaling, pitch scaling, and effects).

```
function process_audio(~, ~)
       if isempty(original_audio)
           errordlg('Please load an audio file first.');
           return;
       end

       % Push current state to undo stack
       undo_stack{end+1} = modified_audio;

       % Get scaling factors
```

```
        alpha = alpha_slider.Value; % Time scaling factor
        beta = beta_slider.Value;   % Pitch scaling factor

        % Phase Vocoder Algorithm
        N = 1024;       % Window size
        Ha = N/4;       % Analysis hop size (25% overlap)
        progress_bar.String = 'Processing...';
        drawnow;
        y_time_scaled = phase_vocoder(modified_audio, alpha, N, Ha);

        % Resampling for Pitch Scaling
        [P, Q] = rat(beta, 0.0001); % Rational approximation of beta
        modified_audio = resample(y_time_scaled, P, Q);

        % Effect Processing
        effect = effect_menu.String{effect_menu.Value};
        switch effect
            case 'Reverb'
                h = reverberator('PreDelay', 0.05, 'WetDryMix', 0.3); %
Fixed PreDelay value
                modified_audio = h(modified_audio);
            case 'Echo'
                delay = 0.3; % 300 ms delay
                modified_audio = echoEffect(modified_audio, fs, delay);
            case 'Distortion'
                modified_audio = tanh(5 * modified_audio); % Non-linear
distortion
        end

        % Update plots and info panel
        plot_audio(ax_modified, modified_audio, 'Modified Signal');
        plot_fft(ax_fft, modified_audio, fs);
        update_info_panel(ax_info, modified_audio, fs, audio_file_info);
        progress_bar.String = 'Processing Complete';
    end
```

Code Explanation :

- ➢ Check Audio File: If no audio is loaded (original_audio is empty), show an error message.
- ➢ Undo Stack: Save the current modified_audio state to the undo stack.
- ➢ Scaling Factors: Retrieve time (alpha) and pitch (beta) scaling factors from sliders.
- ➢ Phase Vocoder: Apply time scaling using phase vocoder with a window size of 1024 and 25% overlap.
- ➢ Pitch Scaling: Approximate pitch scaling (beta) using rat and apply it with resample.
- ➢ Effect Processing: Apply selected effect from the dropdown menu:

- Reverb: Adds reverb with a fixed pre-delay and wet/dry mix.
- Echo: Adds echo with 300 ms delay.
- Distortion: Applies non-linear distortion using the tanh function.

➢ Update Plots: Plot the modified audio in time domain and frequency domain, and update the info panel.

➢ Completion: Update the progress bar to "Processing Complete".

1. **PV algorithm :**

The phase vocoder is used for time scaling without affecting the pitch of the audio.

```
function y = phase_vocoder(x, alpha, N, Ha)
    Hs = round(Ha * alpha); % Synthesis hop size
    w = hann(N, 'periodic'); % Window function

    % STFT parameters
    L = length(x);
    num_frames = floor((L - N) / Ha) + 1;

    % Pad input signal
    x_padded = [x; zeros(N + Ha*num_frames - L, 1)];

    % Initialize STFT
    X = zeros(N, num_frames);
    for t = 1:num_frames
```

Code Explanation :

➢ Inputs: Takes the audio signal x, time scaling factor alpha, window size N, and hop size Ha.

➢ Windowing: Uses a Hann window to process the signal in frames.

➢ STFT: Applies FFT to the windowed frames to get the frequency representation (X).

➢ Phase Processing: Adjusts phase to account for time scaling, smoothing transitions between frames.

➢ ISTFT: Inverse FFT is used to reconstruct the time-domain signal from the modified frequency data.

➢ Normalization: The output is normalized to prevent clipping.

➢ Output: Returns the time-scaled audio signal y.

2. **Effect Processing :**

This block applies the selected effect (reverb, echo, or distortion) to the audio signal.

```
        % Effect Processing
        effect = effect_menu.String{effect_menu.Value};
        switch effect
            case 'Reverb'
                h = reverberator('PreDelay', 0.05, 'WetDryMix', 0.3); %
Fixed PreDelay value
                modified_audio = h(modified_audio);
            case 'Echo'
                delay = 0.3; % 300 ms delay
                modified_audio = echoEffect(modified_audio, fs, delay);
            case 'Distortion'
                modified_audio = tanh(5 * modified_audio); % Non-linear
distortion
        end
```

Code Explanation :

- Effect Selection: The selected effect is retrieved from the effect_menu dropdown.
- Reverb: Applies a reverb effect with a 50 ms pre-delay and 30% wet/dry mix using a built-in reverberator object.
- Echo: Adds an echo effect with a 300 ms delay using a custom echoEffect function.
- Distortion: Applies nonlinear distortion by passing the audio through a tanh function with a scaling factor of 5.

3. **Resampling :**

Resampling is used for pitch scaling by changing the sampling rate of the signal.

```
        % Resampling for Pitch Scaling
        [P, Q] = rat(beta, 0.0001); % Rational approximation of beta
        modified_audio = resample(y_time_scaled, P, Q);
```

Code Explanation :

- Rational Approximation: rat(beta, 0.0001) approximates the pitch scaling factor beta as a rational number P/Q.
- Resampling: The audio (y_time_scaled) is resampled using the calculated P and Q to adjust the pitch by the factor beta.

4. **Undo :**

The undo option allows the user to revert to the previous state of the audio signal.

```
    function undo_processing(~, ~)
        if isempty(undo_stack)
            errordlg('No previous state to undo.');
            return;
        end
        modified_audio = undo_stack{end}; % Restore previous state
        undo_stack(end) = []; % Remove last state from stack
        plot_audio(ax_modified, modified_audio, 'Modified Signal');
        plot_fft(ax_fft, modified_audio, fs);
        update_info_panel(ax_info, modified_audio, fs,
audio_file_info);
        progress_bar.String = 'Undo Complete';
    end
```

Code Explanation :

- ➢ Check Undo Stack: If the undo stack is empty, show an error message.
- ➢ Restore State: The last modified audio state is restored from the undo stack and removed from the stack.
- ➢ Update Display: The modified audio is plotted in the time and frequency domains, and the info panel is updated.
- ➢ Completion: The progress bar text is updated to "Undo Complete".

Output block :

This block handles the display of waveforms, FFT, audio information, and audio playback.

```
 % Create axes for plots and additional information
    ax_original = axes('Parent', main_panel, 'Units', 'normalized',
'Position', [0.15, 0.55, 0.4, 0.4]);
    ax_modified = axes('Parent', main_panel, 'Units', 'normalized',
'Position', [0.6, 0.55, 0.4, 0.4]);
```

Code Explanation :

Axes for Plots:

- ○ ax_original: Displays the original audio waveform.
- ○ ax_modified: Displays the modified audio waveform.
- ○ ax_fft: Displays the frequency domain representation (FFT).
- ○ ax_info: A panel to show additional audio-related details.

Play and Stop Buttons:

- "Play Original": Calls play_original function to play the original audio.
- "Play Modified": Calls play_modified function to play the modified audio.
- "Stop": Calls stop_audio function to stop any playing sound.

1. **Waveform display :**

Plots the waveform of the audio signal and sets the title and x-axis limits.

```
function plot_audio(ax, audio, title_str)
    plot(ax, audio);
    ax.Title.String = title_str;
    ax.XLim = [0, length(audio)];
end
```

Code Explanation :

Input Parameters:

- `ax`: The axes where the waveform will be plotted.
- `audio`: The audio signal to plot.
- `title_str`: The title of the plot.

Plot the Signal:

- `plot(ax, audio)`: Plots the audio signal on the specified axes.

Set Title:

- `ax.Title.String = title_str`: Adds a title to the plot.

Set X-Axis Limits:

- `ax.XLim = [0, length(audio)]`: Adjusts the x-axis limits to match the length of the audio signal.

2. **FFT :**

Computes and plots the frequency spectrum of the audio signal using FFT.

```
function plot_fft(ax, audio, fs)
    L = length(audio);
    Y = fft(audio);
    P2 = abs(Y/L);
    P1 = P2(1:L/2+1);
    P1(2:end-1) = 2*P1(2:end-1);
    f = fs*(0:(L/2))/L;
    plot(ax, f, P1);
    ax.Title.String = 'FFT of Signal';
    ax.XLabel.String = 'Frequency (Hz)';
```

```
        ax.YLabel.String = 'Magnitude';
    end
```

Code Explanation :

Inputs:

- ➢ `ax`: Axes for plotting.
- ➢ `audio`: Audio signal.
- ➢ `fs`: Sampling rate.

Steps:

- ➢ Compute FFT of the audio signal.
- ➢ Normalize and extract a single-sided magnitude spectrum.
- ➢ Create frequency axis in Hz.

Plot:

- ➢ Plot magnitude vs frequency.
- ➢ Add title and axis labels.

3. **Audio info display :**

Updates the panel with metadata about the audio file, including file name, sampling rate, duration, channels, bit depth, and file size.

```
    function update_info_panel(panel, audio, fs, audio_info)
        % Clear previous content
        delete(get(panel, 'Children'));

        % Display audio information
        info_str = { ...
            sprintf('File Name: %s', audio_info.FileName), ...
            sprintf('Sampling Rate: %d Hz', fs), ...
            sprintf('Duration: %.2f s', length(audio)/fs), ...
            sprintf('Number of Channels: %d', audio_info.NumChannels),
...
            sprintf('Bit Depth: %d bits', audio_info.BitDepth), ...
            sprintf('File Size: %.2f KB', audio_info.FileSize) ...
        };

        % Add text controls for each piece of information
        for i = 1:length(info_str)
            uicontrol('Parent', panel, 'Style', 'text', 'String',
info_str{i}, ...
```

Code Explanation :

Input Parameters:

- ➢ `panel`: The panel where audio information will be displayed.

- ➢ `audio`: The audio signal (used to calculate duration).
- ➢ `fs`: The sampling rate of the audio signal.
- ➢ `audio_info`: A structure containing metadata about the audio file.

Clear Previous Content:

- ➢ `delete(get(panel, 'Children'))`: Clears any existing content in the panel.

Prepare Audio Information:

- ➢ `info_str`: A cell array containing formatted strings with audio metadata:
  - File name.
  - Sampling rate.
  - Duration (calculated as `length(audio)/fs`).
  - Number of channels.
  - Bit depth.
  - File size.

Display Information:

- ➢ A loop creates text controls (`uicontrol`) for each piece of information.
- ➢ Each text control is positioned vertically in the panel.

4. **Audio output and wave :**

These functions play the original audio, play the modified audio, and stop any ongoing audio playback, respectively.

```
function play_original(~, ~)
    if ~isempty(original_audio)
        sound(original_audio, fs);
    end
end

function play_modified(~, ~)
    if ~isempty(modified_audio)
        sound(modified_audio, fs);
    end
end

function stop_audio(~, ~)
    % Stop audio playback
    clear sound;
    progress_bar.String = 'Playback Stopped';
end
```

Code Explanation :

play_original` Function :

- ➢ Plays the original audio signal.
- ➢ Checks if `original_audio` is not empty.
- ➢ Uses `sound(original_audio, fs)` to play the audio at the sampling rate `fs`.

`play_modified` Function :

- ➢ Plays the modified audio signal.
- ➢ Checks if `modified_audio` is not empty.
- ➢ Uses `sound(modified_audio, fs)` to play the audio at the sampling rate `fs`.

`stop_audio` Function :

- ➢ Stops any ongoing audio playback.
- ➢ Uses `clear sound` to stop playback.
- ➢ Updates the `progress_bar` text to indicate playback has stopped.

Total code :

Part 1

```
unction phase_vocoder_gui()
    fig = figure('Name', 'Phase
Vocoder', 'NumberTitle', 'off', ...
        'Units', 'normalized',
'Position', [0, 0, 1, 1], 'MenuBar',
'none', 'ToolBar', 'none');

    original_audio = [];
    modified_audio = [];
    fs = 44100;
    undo_stack = {};
    audio_file_info =
struct('FileName', '', 'FileSize', 0,
'NumChannels', 0, 'BitDepth', 0);

    main_panel = uipanel('Parent', fig,
'Title', 'Phase Vocoder', ...
        'Units', 'normalized',
'Position', [0.02, 0.02, 0.96, 0.96],
...
        'BackgroundColor', [0.95, 0.95,
0.95]);

    uicontrol('Parent', main_panel,
'Style', 'pushbutton', 'String', 'Load
Audio', ...
        'Units', 'normalized',
'Position', [0.02, 0.9, 0.1, 0.05],
'Callback', @load_audio);

    % Time scaling controls
    uicontrol('Parent', main_panel,
'Style', 'text', 'String', 'Time
Scaling (α)', ...
```

Part 2

```
 'Units', 'normalized', 'Position',
[0.02, 0.8, 0.1, 0.03],
'HorizontalAlignment', 'left', ...
        'BackgroundColor', [0.95, 0.95,
0.95]);
    alpha_slider = uicontrol('Parent',
main_panel, 'Style', 'slider', ...
        'Units', 'normalized',
'Position', [0.02, 0.77, 0.1, 0.02],
'Min', 0.5, 'Max', 2, 'Value', 1);
    alpha_edit = uicontrol('Parent',
main_panel, 'Style', 'edit', ...
        'String', '1.0', 'Units',
'normalized', 'Position', [0.02, 0.74,
0.1, 0.03]);
 % Pitch scaling controls
    uicontrol('Parent', main_panel,
'Style', 'text', 'String', 'Pitch
Scaling (β)', ...
        'Units', 'normalized',
'Position', [0.02, 0.68, 0.1, 0.03],
'HorizontalAlignment', 'left', ...
        'BackgroundColor', [0.95, 0.95,
0.95]);
    beta_slider = uicontrol('Parent',
main_panel, 'Style', 'slider', ...
        'Units', 'normalized',
'Position', [0.02, 0.65, 0.1, 0.02],
'Min', 0.5, 'Max', 2, 'Value', 1);
    beta_edit = uicontrol('Parent',
main_panel, 'Style', 'edit', ...
        'String', '1.0', 'Units',
'normalized', 'Position', [0.02, 0.62,
0.1, 0.03]);
```

Part 3	Part 4

% Preset options
uicontrol('Parent', main_panel, 'Style', 'text', 'String', 'Presets', ...
'Units', 'normalized', 'Position', [0.02, 0.56, 0.1, 0.03], 'HorizontalAlignment', 'left', ...
'BackgroundColor', [0.95, 0.95, 0.95]);
preset_menu = uicontrol('Parent', main_panel, 'Style', 'popupmenu', ...
'String', {'Custom', 'Chipmunk (β=1.5)', 'Slow Motion (α=1.5)', 'Fast Forward (α=0.5)', 'Robot (α=1, β=0.8)'}, ...
'Units', 'normalized', 'Position', [0.02, 0.53, 0.1, 0.03], 'Callback', @apply_preset);
 	
% Effect options
uicontrol('Parent', main_panel, 'Style', 'text', 'String', 'Effects', ...
'Units', 'normalized', 'Position', [0.02, 0.47, 0.1, 0.03], 'HorizontalAlignment', 'left', ...
'BackgroundColor', [0.95, 0.95, 0.95]);
effect_menu = uicontrol('Parent', main_panel, 'Style', 'popupmenu', ...
'String', {'None', 'Reverb', 'Echo', 'Distortion'}, ...
'Units', 'normalized', 'Position', [0.02, 0.44, 0.1, 0.03]);% Buttons for processing and saving
uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String', 'Process', ...
'Units', 'normalized', 'Position', [0.02, 0.38, 0.1, 0.05], 'Callback', @process_audio);
uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String', 'Undo', ...
'Units', 'normalized', 'Position', [0.02, 0.32, 0.1, 0.05],
	'Callback', @undo_processing);
	uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String', 'Save Output', ...
	'Units', 'normalized', 'Position', [0.02, 0.26, 0.1, 0.05], 'Callback', @save_audio);
	
	% Play buttons
	uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String', 'Play Original', ...
	'Units', 'normalized', 'Position', [0.02, 0.2, 0.1, 0.05], 'Callback', @play_original);
	uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String', 'Play Modified', ...
	'Units', 'normalized', 'Position', [0.02, 0.14, 0.1, 0.05], 'Callback', @play_modified);
	uicontrol('Parent', main_panel, 'Style', 'pushbutton', 'String', 'Stop', ...
	'Units', 'normalized', 'Position', [0.02, 0.08, 0.1, 0.05], 'Callback', @stop_audio);
	
	% Progress bar
	progress_bar = uicontrol('Parent', main_panel, 'Style', 'text', 'String', 'Ready', ...
	'Units', 'normalized', 'Position', [0.02, 0.02, 0.1, 0.03], 'HorizontalAlignment', 'left', ...
	'BackgroundColor', [0.95, 0.95, 0.95]);
	
	% Create axes for plots and information
	ax_original = axes('Parent', main_panel, 'Units', 'normalized', 'Position', [0.15, 0.55, 0.4, 0.4]);
	ax_modified = axes('Parent', main_panel, 'Units', 'normalized', 'Position', [0.6, 0.55, 0.4, 0.4]);

Part 5	Part 6

ax_fft = axes('Parent', main_panel, 'Units', 'normalized', 'Position', [0.15, 0.1, 0.4, 0.35]);function load_audio(~, ~)
[file, path] = uigetfile({'.wav;.mp3;.ogg;.flac;*.au', 'Audio Files'});
if file == 0
return;
end
[x, fs] = audioread(fullfile(path, file));
original_audio = mean(x, 2);
modified_audio = original_audio;
undo_stack = {};
 	
% Update plots and info panel
plot_audio(ax_original, original_audio, 'Original Signal');
plot_audio(ax_modified, modified_audio, 'Modified Signal');
plot_fft(ax_fft, original_audio, fs);
update_info_panel(ax_info, original_audio, fs, audio_file_info);
progress_bar.String = 'Audio Loaded';
end
	ax_info = uipanel('Parent', main_panel, 'Title', 'Audio Information', ...
	'Units', 'normalized', 'Position', [0.6, 0.1, 0.4, 0.35], ...
	'BackgroundColor', [0.95, 0.95, 0.95]);
	
	function process_audio(~, ~)
	if isempty(original_audio)
	errordlg('Please load an audio file first.');
	return;
	end
	undo_stack{end+1} = modified_audio;
	
	% Get scaling factors
	alpha = alpha_slider.Value;
	beta = beta_slider.Value;
	modified_audio = phase_vocoder(modified

Simulation Results

GUI:

The blank GUI represents the initial state of the Phase Vocoder application before any audio file is loaded. It showcases the layout of controls, buttons, and display panels, ready for user interaction.

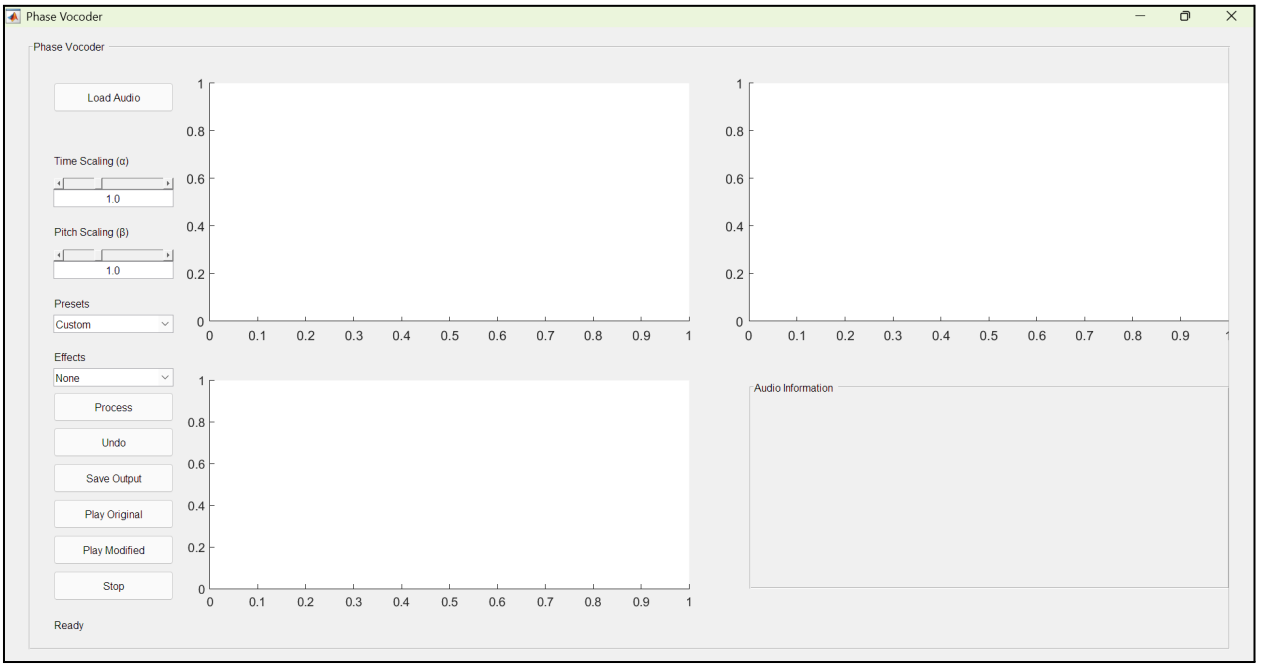

Figure 2 : GUI Output of MATLAB

Voice loaded:

The voice-loaded GUI displays the interface after an audio file has been successfully imported. The original and modified waveform of the loaded audio is shown with its fft diagram, and metadata such as file name, duration, and sampling rate is displayed in the information panel.

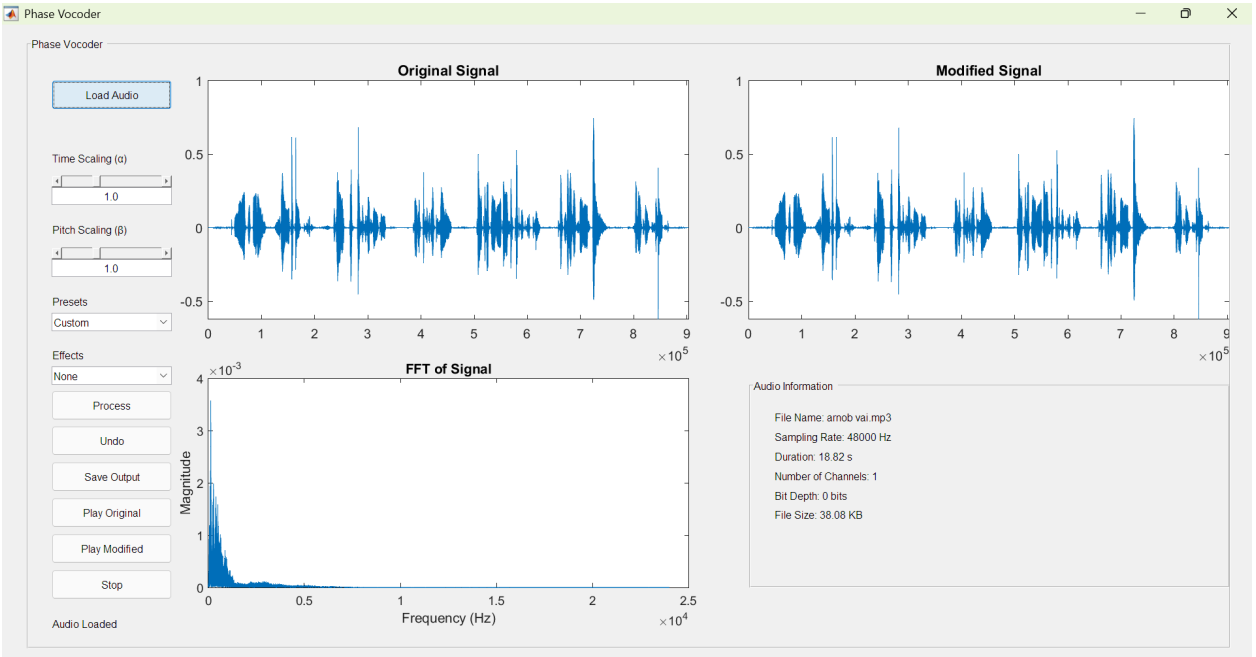

Figure 3 : Voice Loaded Output of MATLAB

Slow motion(alpha = 1.5) :

The slow motion GUI demonstrates the application of time scaling to slow down the audio. The waveform is stretched, and the modified signal is displayed alongside the original for comparison.

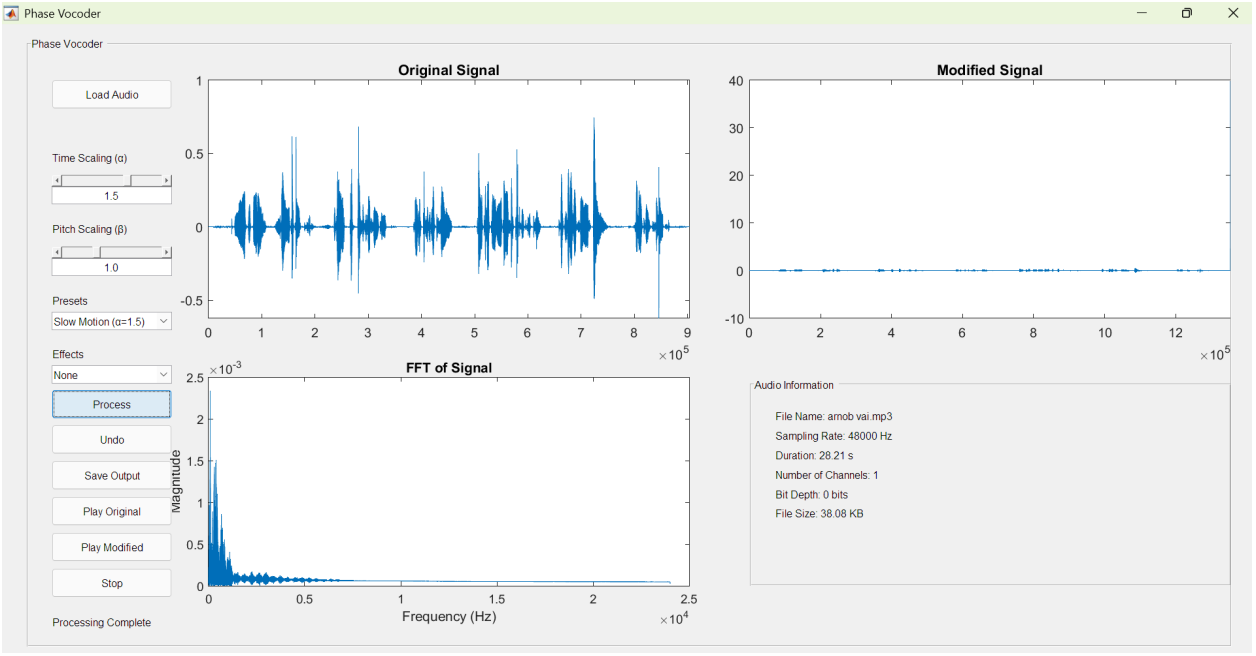

Figure 4 : Slow Motion Voice Output of MATLAB

Fast Forward(alpha = 0.5) :

The fast forward GUI illustrates the effect of time scaling to speed up the audio. The waveform is compressed, and the modified signal is shown, highlighting the reduced duration of the audio.

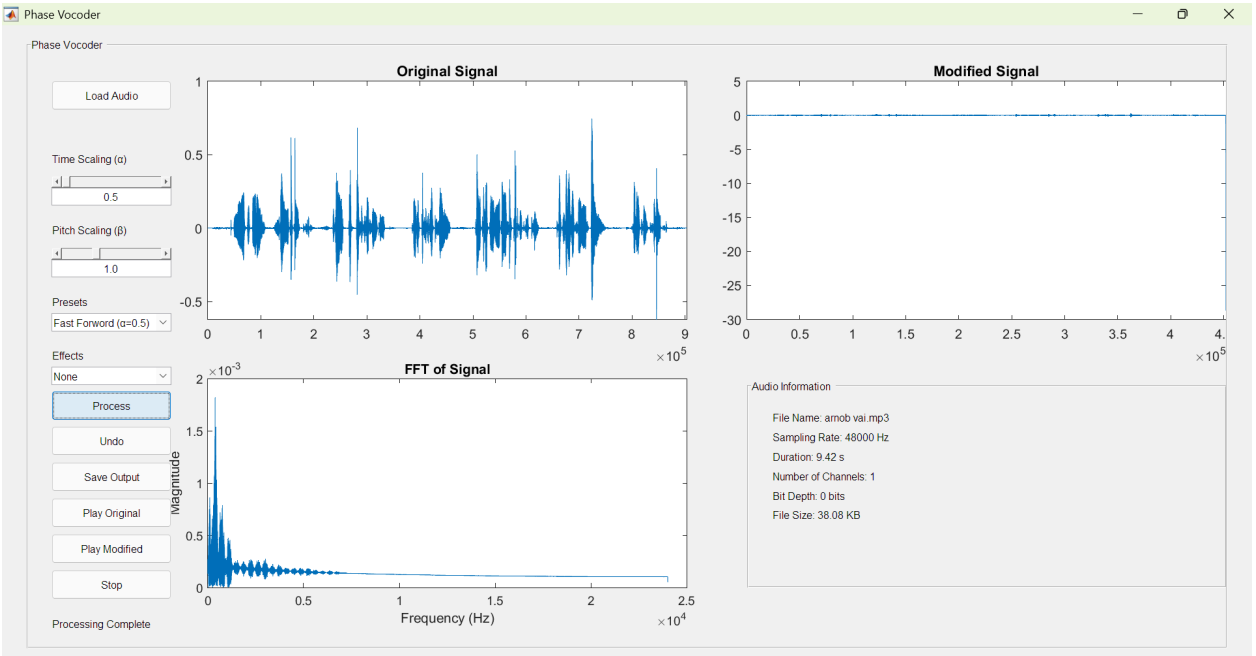

Figure 5 : Fast Forward Voice Output of MATLAB

Revered:

The reverberation GUI shows the result of applying a reverb effect to the audio. The waveform reflects the added spatial characteristics, simulating the sound of a room or hall.

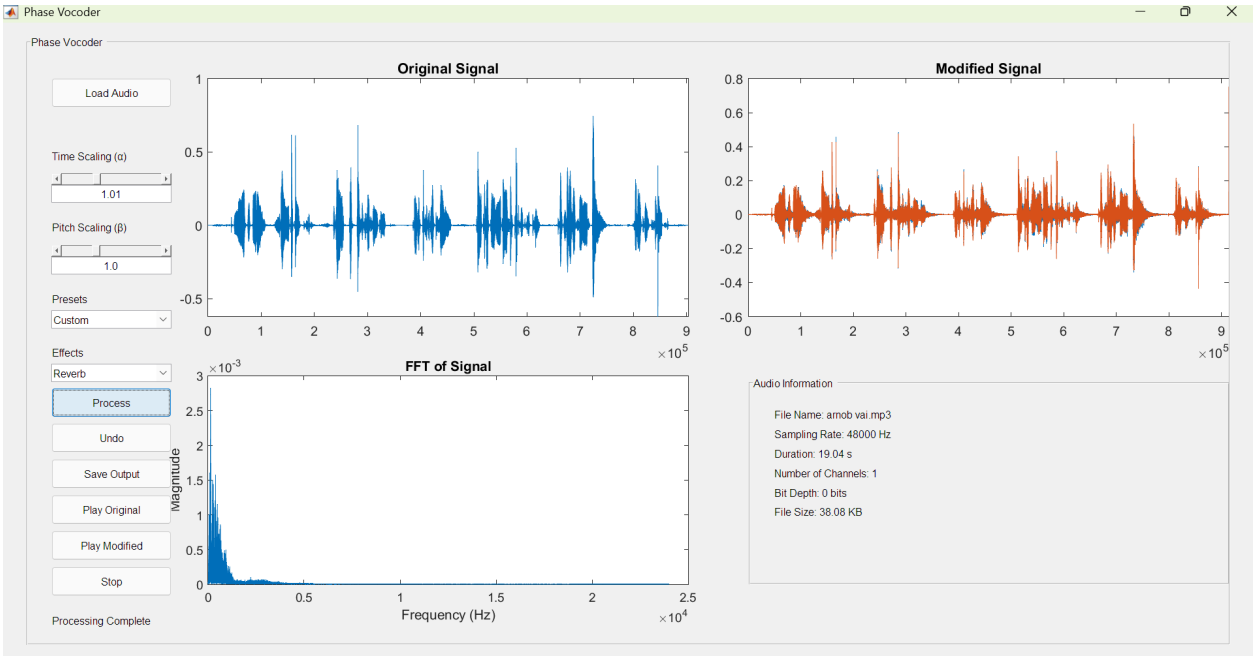

Figure 6 : Revered Voice Output of MATLAB

Echo:

The echo GUI displays the outcome of adding an echo effect to the audio. The waveform reveals repeated, decaying repetitions of the original signal, creating a sense of depth and space.

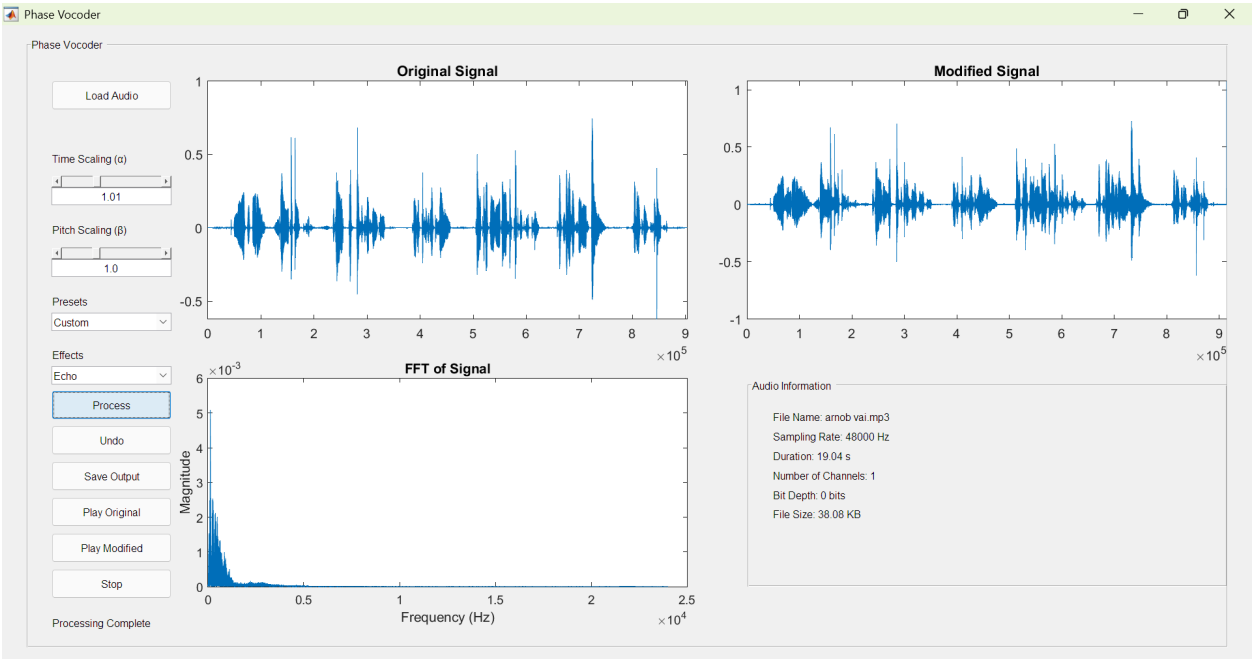

Figure 7 : Echo Voice Output of MATLAB

Discussion

The Phase Vocoder GUI project was designed to create a comprehensive and user-friendly tool for audio processing, enabling users to perform time scaling, pitch shifting, and apply various audio effects such as reverb, echo, and distortion. The project successfully achieved its primary objectives by implementing a phase vocoder algorithm for time scaling, resampling for pitch shifting, and integrating these features into an intuitive graphical user interface (GUI). The GUI allowed users to visualize audio waveforms and frequency spectra, providing a clear understanding of the changes being applied to the audio signal.

One of the key accomplishments of the project was the seamless integration of advanced audio processing techniques into a simple and accessible interface. Users could load audio files, adjust parameters using sliders and dropdown menus, and immediately observe the results through visualizations and playback. The ability to undo processing steps and save modified audio files added to the tool's practicality and usability.

However, the project also faced several challenges. A significant issue was the computational complexity of the phase vocoder algorithm, which made real-time processing difficult, especially for longer audio files. Additionally, synchronizing the GUI with audio processing tasks required careful handling to avoid delays or crashes. These challenges emphasized the importance of optimizing algorithms and managing system resources effectively in audio applications.

Through this project, we gained valuable insights into digital signal processing, particularly in the areas of time-frequency transformations and audio effects. We also learned the importance of designing user interfaces that balance functionality with simplicity, ensuring that even non-technical users can operate the tool effectively. The experience highlighted the need for iterative testing and user feedback to refine the application.

Looking ahead, there are several avenues for further improvement. Real-time processing capabilities could be added to enhance interactivity, while support for additional audio formats would increase the tool's versatility. Advanced features such as MIDI integration, formant preservation, and a parametric equalizer could be incorporated to expand the tool's functionality. Additionally, optimizing the application for performance and memory efficiency would ensure smooth operation even with large or complex audio files. Overall, this project serves as a strong foundation for developing more advanced and versatile audio processing tools in the future.

Improvement Ideas

The Phase Vocoder GUI project can be further enhanced to improve its functionality, usability, and performance. Below are some improvement ideas, categorized into functionality enhancements, user experience improvements, and technical optimizations.

1. Functionality Enhancements

A. Additional Audio Effects: More audio effects, such as chorus, flanger or parametric equalization, can be added to provide users with a wider range of creative options. This would allow for more advanced audio manipulation and customization.

B. Multi-Track Support: Support for loading and processing multiple audio tracks simultaneously can be added. This would enable users to work on more complex audio projects, such as mixing or layering sounds.

C. Advanced Phase Vocoder Features : Features like formant preservation during pitch shifting and transient preservation during time scaling can be implemented. These would help maintain the natural characteristics of voices and instruments, especially for complex audio signals.

D. Batch Processing: A batch processing feature can be added to allow users to apply the same effects and scaling parameters to multiple audio files at once. This would save time when working with large datasets or multiple files.

2. User Experience Improvements

A. Interactive Visualizations: The waveform and FFT plots can be made interactive, allowing users to zoom, pan, and select specific regions of the audio for detailed analysis. This would enhance the user's ability to inspect and edit audio signals.

B. Preset Management: Users can be given the ability to save and load custom presets for time scaling, pitch scaling, and effects. This would make it easier to reuse settings across different projects and streamline the workflow.

C. Undo/Redo Stack: The undo functionality can be expanded to include multiple levels of undo/redo. This would allow users to revert or reapply several changes, providing greater flexibility during editing.

D. Progress Indicators: A progress bar or spinner can be added during processing to give users feedback on the status of time-consuming operations. This would improve the user experience by providing transparency during long processing tasks.

E. Tooltips and Help Documentation: Tooltips can be added to buttons and controls to explain their functionality. Additionally, a help menu or documentation section can be included to guide users on how to use the tool effectively.

3. Technical Optimizations

A. Efficient Algorithms: The phase vocoder algorithm can be optimized for faster processing, especially for long audio files. Techniques like GPU acceleration (e.g., using MATLAB's Parallel Computing Toolbox) can be explored to speed up computationally intensive tasks like FFT and resampling.

B. Support for More Audio Formats: Support for additional audio formats, such as AAC or WMA, can be added. This would make the tool more versatile and compatible with a wider range of audio files.

C. Error Handling: Error handling can be improved to provide meaningful messages for issues like unsupported file formats, insufficient memory, or processing errors. This would make the tool more robust and user-friendly.

D. Memory Management: Efficient memory management techniques can be implemented to handle large audio files without crashing or slowing down the application. This would ensure smooth performance even with resource-intensive tasks.

4. Advanced Features

A. MIDI(Musical Instrument Digital Interface) Integration: MIDI file support can be added, allowing users to import MIDI files and synchronize them with the processed audio. This would be particularly useful for music production and composition.

B. Spectrogram Display: A spectrogram visualization can be added to provide a detailed time-frequency representation of the audio signal. This would help users analyze and edit audio with greater precision.

C. Export Settings: Users can be given the option to export processing settings (e.g., time scaling factor, effects parameters) as a configuration file. This would allow for easy sharing and reuse of settings.

D. Customizable Interface: The GUI layout can be made customizable, allowing users to resize panels, rearrange controls, or choose between light and dark themes. This would improve accessibility and user satisfaction.

By implementing these enhancements, the Phase Vocoder GUI can be transformed into a more powerful, user-friendly, and versatile tool for audio processing. These improvements would expand its functionality, improve the user experience, and optimize its performance, making it suitable for both casual users and professionals in music production, sound design, and audio research.

Reference

1. Hill, P. (2018). Audio and Speech Processing with MATLAB (1st ed.). CRC Press. https://doi.org/10.1201/9780429444067

2. Z. Průša and N. Holighaus, "Phase vocoder done right," 2017 25th European Signal Processing Conference (EUSIPCO), Kos, Greece, 2017, pp. 976-980, doi: 10.23919/EUSIPCO.2017.8081353. keywords: {Time-frequency analysis;Transient analysis;Signal processing algorithms;Vocoders;Frequency estimation;Coherence;Europe},

3. Hammer, F., 2001. Time-scale modification using the phase vocoder. *Diploma Thesis*.