Real-Time Voice Changer Using MATLAB

2. Introduction

This project is about making a voice changer that works in real-time. It takes sound from the microphone, changes the pitch of the voice (makes it sound higher or lower), and then plays it through the speaker. MATLAB is used for this, and the sound is changed while you are speaking, so there is no delay.

3. Objectives of the Project

- To create a simple voice changer that works instantly.
- To change the pitch of the voice using a basic method.
- To show what the sound looks like before and after changing it.
- To learn how to handle live audio in MATLAB.

4. Methodology

First, we take sound from the microphone using MATLAB tools. Then we store the sound in a temporary memory known as buffer. The pitch is changed by reading the sound from this memory at a different speed. This makes the voice sound higher or lower. The changed sound is then played through the speakers.

Every 5 seconds, the project also makes two pictures:

- One shows the sound's frequency (called a **spectrogram**).
- Another shows how loud different parts of the sound are (called an **FFT plot**).

5. Code

```
% Real-time voice changer with pitch shifting and frequency analysis
% Uses MATLAB Audio Toolbox

% Initialize parameters
sampleRate = 44100;
bufferSize = 512;
pitchFactor = 2.0;

% Create audio device objects
mic = audioDeviceReader('SampleRate', sampleRate, 'SamplesPerFrame', bufferSize);
speaker = audioDeviceWriter('SampleRate', sampleRate);

% Delay buffer
delayBuffer = zeros(round(sampleRate * 0.1), 1); % 100 ms buffer
writePtr = 1;
readPtr = 1;

% Plotting buffers
plotInterval = 5; % seconds
plotBufferSize = plotInterval * sampleRate;
inputHistory = zeros(plotBufferSize, 1);
outputHistory = zeros(plotBufferSize, 1);
historyIndex = 1;
plotTimer = tic;

% Inform user
disp('Starting real-time voice changer. Press Ctrl+C to stop.');

% Main processing loop
try
    while true
        % Read from mic
        audioIn = mic();
        audioOut = zeros(bufferSize, 1);

        for i = 1:bufferSize
            delayBuffer(writePtr) = audioIn(i);

            % Interpolated read
            readPtrInt = floor(readPtr);
            frac = readPtr - readPtrInt;
            readPtrNext = mod(readPtrInt, length(delayBuffer)) + 1;
            readPtrPrev = mod(readPtrInt - 1, length(delayBuffer)) + 1;

            % Linear interpolation
            audioOut(i) = (1 - frac) * delayBuffer(readPtrPrev) + frac *
delayBuffer(readPtrNext);

            % Update pointers
            writePtr = mod(writePtr, length(delayBuffer)) + 1;
            readPtr = readPtr + 1 / pitchFactor;
            if readPtr > length(delayBuffer)
                readPtr = readPtr - length(delayBuffer);
            end
        end
```

```
% Play modified audio
speaker(audioOut);

% Save for plotting
idxRange = historyIndex:historyIndex + bufferSize - 1;
if idxRange(end) <= plotBufferSize
    inputHistory(idxRange) = audioIn;
    outputHistory(idxRange) = audioOut;
    historyIndex = historyIndex + bufferSize;
end

% Plot every 5 seconds
if toc(plotTimer) > plotInterval
    figure(1);
    clf;

    % ----- Spectrogram -----
    subplot(2, 2, 1);
    spectrogram(inputHistory, 256, 200, 512, sampleRate, 'yaxis');
    title('Input Voice Spectrogram');
    ylim([0 4]); % up to 4kHz

    subplot(2, 2, 2);
    spectrogram(outputHistory, 256, 200, 512, sampleRate, 'yaxis');
    title(['Output Voice Spectrogram (Pitch ×' num2str(pitchFactor) ')']);
    ylim([0 4]);

    % ----- Frequency Spectrum -----
    fftLen = 2048;
    if length(inputHistory) >= fftLen
        xInput = inputHistory(end - fftLen + 1:end) .* hann(fftLen);
        xOutput = outputHistory(end - fftLen + 1:end) .* hann(fftLen);

        f = (0:fftLen/2 - 1) * sampleRate / fftLen;
        X_in = abs(fft(xInput));
        X_out = abs(fft(xOutput));

        subplot(2, 2, 3);
        plot(f / 1000, 20*log10(X_in(1:fftLen/2)));
        title('Input Voice Spectrum');
        xlabel('Frequency (kHz)');
        ylabel('Magnitude (dB)');
        grid on;

        subplot(2, 2, 4);
        plot(f / 1000, 20*log10(X_out(1:fftLen/2)));
        title('Output Voice Spectrum');
        xlabel('Frequency (kHz)');
        ylabel('Magnitude (dB)');
        grid on;
    end

    drawnow;

    % Reset buffer
    historyIndex = 1;
    inputHistory(:) = 0;
    outputHistory(:) = 0;
    plotTimer = tic;
```

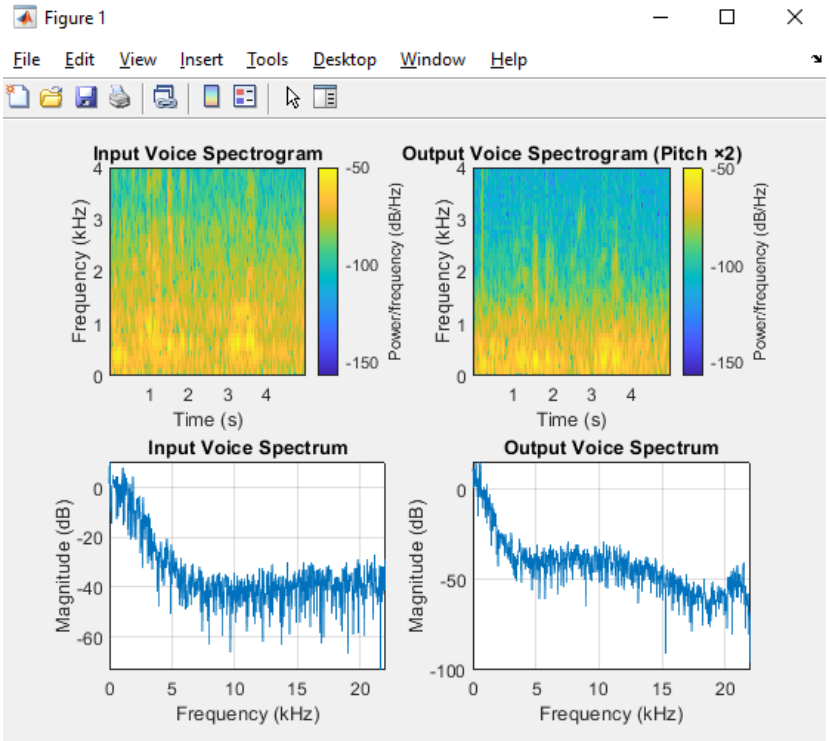
```
        end
    end

catch err
    release(mic);
    release(speaker);
    disp('Voice changer stopped.');
    rethrow(err);
end

% Cleanup
release(mic);
release(speaker);
```

6. Results

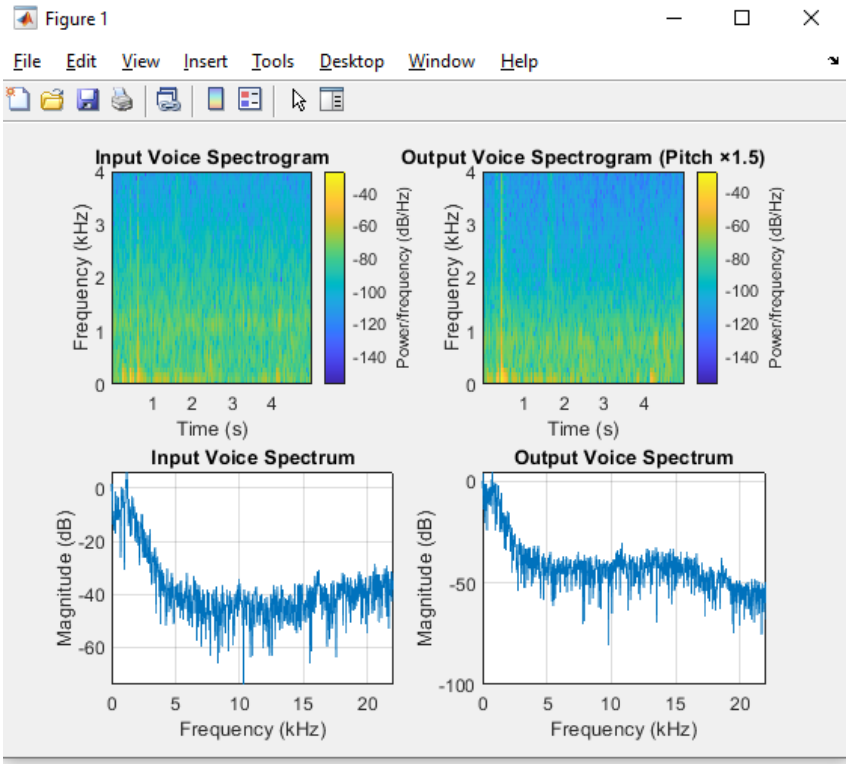
The voice changer worked successfully. When the pitch factor was set to 1.5, the voice sounded higher. The program also showed pictures (graphs) of the sound. These pictures showed that the sound had changed, especially in the frequencies.

There was almost no delay, and the voice was heard immediately after speaking.

OUTPUT When pitch is 2.0



OUTPUT When pitch is 1.5



7. Comparison

We compared the original voice and the changed voice using graphs.

- In the spectrogram, we could see the sound moved to higher frequencies after changing.
- In the FFT plot, we saw the sound peaks also moved up.
- In the waveform (sound shape), the difference was not big, so we needed these graphs to really see the change.

8. Future Direction / Limitations

Limitations:

- Sometimes the voice may sound a little robotic.
- It always uses the same pitch change (1.5x) — we cannot change it while speaking.
- It doesn’t remove background noise.

Future Ideas:

- Allow users to change pitch live with buttons or a slider.
- Make the voice sound more natural after pitch change.
- Add a feature to remove noise.
- Record and save the changed voice.

9. Conclusion

This project shows how to make a simple voice changer using MATLAB. It takes live voice input, changes the pitch, and plays it immediately. We also made graphs to see how the sound changed. The project is a good start and can be improved with more features in the future.