Digital Signal Processing

Pitch shifting

Erika Do, Jakub Július Šmýkal

January 31, 2025

Contents

1 Introduction

1.1 Background

Pitch-shifting is a fundamental technique in Digital Signal Processing (DSP) that modifies the pitch of an audio signal without altering its duration. This method has undergone substantial development, beginning with analog systems in the 1960s, such as those famously utilized in Beatles recordings, before transitioning to digital hardware innovations in the 1970s. Today, it is integral to music production for pitch correction and creative effects. Despite superior quality in commercial tools often outperforming academic algorithms, their proprietary nature poses challenges for reverse engineering [12].

Auto-Tune, a highly recognized pitch correction tool, stands as one of the most prominent applications of pitch-shifting algorithms in the music industry. Developed by Antares Audio Technologies in the late 1990s, it was initially designed to correct off-pitch vocals during recordings or live performances. This technology enables singers to stay in tune in real-time, making it indispensable for both studio and live audio production where instant pitch correction is required. Its distinctive robotic or synthetic effect later gained prominence in genres such as pop, hip hop, and electronic music. Utilizing pitch-shifting techniques, it adjusts vocal pitch without altering the timing or other inherent characteristics of the voice.

Auto-Tune (as shown in Figure 1) works by detecting a vocal signal's pitch and adjusting it to the closest semitone. It can be used subtly for minor pitch corrections or more dramatically to create noticeable effects, such as a "robotic" or synthesized voice. In these extreme cases, it results in the voice sharply leaping from one note to another, similar to a synthesizer, rather than smoothly transitioning between notes [16].

Figure 1: Auto-Tune Pro 11: The professional standard for pitch correction. Source: [14].

In the realm of DSP, pitch-shifting algorithms like those in Auto-Tune rely on sophisticated methods such as time-domain resampling, Fourier transforms and phase vocoder techniques. These methods manipulate the harmonic structure of the signal and its temporal properties to achieve the desired pitch shift while minimizing unwanted artifacts, like distortion or unnatural warbling. Unlike simple pitch transposition, which changes both pitch and length through sample rate conversion, pitch-shifting preserves harmonic relationships, ensuring that the transformation sounds natural. However, challenges remains in maintaining the formants and avoiding inharmonic effects, which can lead to metallic or unnatural tones [2].

1.2 Motivation

The motivation for this project lies in the practical applications of pitch shifting across various fields of audio processing. In music production, artists, producers, and sound engineers frequently rely on pitch manipulation to fine-tune vocals or instruments, achieving specific sound effects or harmonizing multiple layers. Pitch shifting also plays a critical role in the post-production process, enabling audio engineers to match different musical elements in a composition. This technique is rooted in the broader context of DSP, which has revolutionized music reproduction, voice telecommunications and synthetic speech, all of which are adapted to the auditory perception of the human ear [13].

From an academic and technical perspective, this project offers an opportunity to explore core DSP concepts such as Fourier transforms, filtering techniques and interpolation methods. By applying these to pitch shifting, the project offers a deeper understanding of signal manipulation and its practical uses. Additionally, it imparts insights into real-world applications like music production, speech processing and audio effects, providing valuable practical experience with DSP techniques. The ultimate goal is to implement a basic pitch-shifting algorithm that maintains audio quality and is computationally efficient.

2 Theoretical Framework and Methodology

2.1 Fundamental Theory

The underlying theory of pitch-shifting is based on manipulating the frequency content of a signal. Several mathematical and computational methods are used to achieve pitch-shifting in a way that minimizes distortion and preserves the natural qualities of the sound. The key elements of this theory include:

1. **The Fourier Transform (FT)**:

   Before delving into the Fourier transform, it is essential to first understand the concept of sinusoids. Sinusoids represent sound waves as periodic fluctuations in air pressure, with the frequency determining the pitch and the amplitude affecting loudness. However, humans cannot hear pure tones at very low frequencies, which are often components of sound signals. Despite this limitation, all sounds can be decomposed into a combination of pure tones, and similarly, any time-domain signal can be analysed by breaking it down into constituent sinusoids. This process, and its consequences, is the subject of frequency domain analysis and Fourier transforms [8].

   The Fourier transform is a fundamental mathematical tool in DSP that maps a time-domain signal $x(t)$ into its frequency-domain representation $X(\omega)$, where ω represents the angular frequency. This is accomplished through the integral formula in Equation 1.

$$X(\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}\,dt \tag{1}$$

   One key property of the Fourier transform is its linearity, meaning no cross-frequency interactions occur in linear time-invariant systems. Cross-spectral interactions, therefore, indicate non-linear behaviour. In practice, Fourier transforms are implemented as discrete transforms, incorporating windowing functions to limit bounds and reduce high-frequency leakage. However, the fixed width of these windows restricts time-domain resolution below this limit, making Fourier transforms less suitable for time-frequency analysis [5].

   The Fourier transform offers the ability to isolate and manipulate individual frequency components of a signal, enabling modifications to characteristics such as pitch. After making frequency adjustments, the inverse Fourier transform (IFT) reconstructs the signal in the time domain. Additionally, while the Fourier Transform provides a powerful tool for analysing frequency content, practical applications often require the use of a discrete version of the transform (DFT), especially when dealing with digital signals.

   The Discrete Fourier Transform (DFT) represents a signal as a sum of complex sinusoids, forming its spectrum. For a discrete signal $x[n]$, where $0 \leq n \leq N-1$, the DFT is expressed as in Equation 2.

$$X[k] = \sum_{n=0}^{N-1} x[n]e^{-j2\pi nk/N} \tag{2}$$

   Both the input signal and its DFT share the same length N. Each coefficient $X[k]$ or frequency bin, corresponds to a sinusoid with a normalized frequency k/N. The bin's magnitude indicates the sinusoid's amplitude, while its phase defines the time offset. Since the DFT is invertible, the original signal can be reconstructed using the inverse DFT if the frequency coefficients remain unaltered [12].

2. **Time-domain Resampling**:

   It refers to altering the playback rate of a signal. Slowing down the playback rate lowers the pitch, while speeding it up raises the pitch. However, this process also impacts the signal's duration, which can create challenges in time-sensitive contexts such as music or speech applications. To address this, advanced techniques like time-stretching are often combined with pitch-shifting. Time-stretching enables the modification of the signal's duration independently of its pitch, preserving its temporal characteristics while adjusting the pitch. This is particularly critical for music or speech signals, where maintaining natural rhythm and tempo is essential. As highlighted in [13], when resampling is performed in the time domain, interpolation is typically used to ensure the pitch-shifted signal remains smooth and free from artifacts or discontinuities. These resampling methods underpin many audio processing algorithms, including those for pitch-shifting and time-stretching.

3. **The Phase Vocoder**:

   The Phase Vocoder is a well-known tool used for processing speech and audio signals in the frequency domain, with applications such as time compression, expansion, pitch modification and noise reduction. In the context

of time-scale or pitch-scale adjustments, the phase vocoder is recognized for producing high-quality results, particularly when large modifications are made to polyphonic or non-pitched signals. The process begins by dividing the signal into overlapping frames or windows, followed by applying a Fourier Transform (usually Short-Time Fourier Transform or STFT) to each segment. This results in a time-frequency representation that preserves both temporal and spectral information. The phase vocoder allows for independent manipulation of the time and frequency components, making it well-suited for tasks such as time-stretching and pitch-shifting. However, the phase vocoder is also associated with certain artifacts, often described as "phasiness", "reverberation" or "loss of presence". Recent studies have focused on addressing these issues, with some solutions offering significant quality improvements at the cost of high computational demands, while others provide more affordable, though less effective, alternatives [9].

4. **Pitch Detection and Correction**:

   A critical step in pitch-shifting is the precise identification of the pitch in the original audio signal. This process typically involves identifying the fundamental frequency (F0), which corresponds to the perceived pitch. In this work, methods like linear interpolation and SOLA (Synchronized Overlap-Add) are utilized to achieve accurate pitch detection and manipulation [4].

   - **SOLA** is a widely used algorithm in time-domain audio processing, employed for both time-stretching and pitch-shifting. Introduced by Lawrence Rabiner and Ronald Schafer in 1978, SOLA works by segmenting the audio signal into overlapping frames, as shown in Figure 2. The amount of overlap between these frames is adjusted to achieve the desired pitch shift. Increasing the overlap results in a lower pitch, while decreasing the overlap raises the pitch. One of the main advantages of SOLA is its ability to perform real-time processing with relatively low computational cost, making it ideal for applications that require fast audio transformations. However, SOLA can encounter limitations when handling complex harmonic structures, such as in orchestral music, where inaccuracies in pitch detection may arise. Despite these drawbacks, SOLA continues to be an efficient and popular technique for pitch and time manipulation [17].

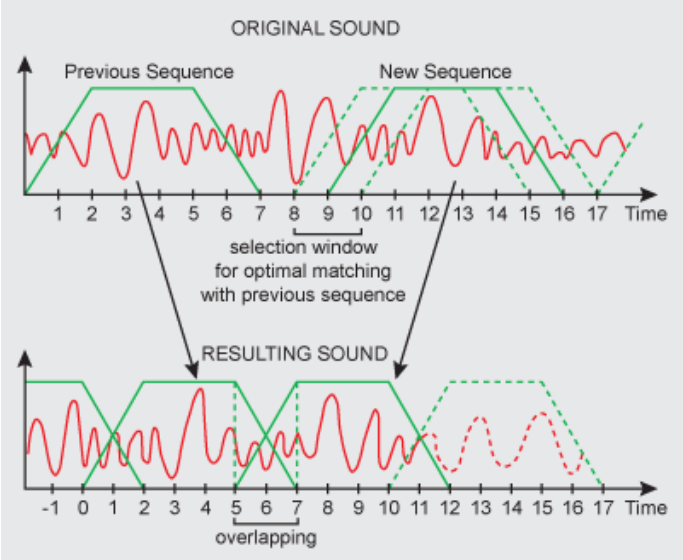

Figure 2: Illustration of SOLA and its overlapping segments. Source: [11].

   - **Linear Interpolation** is a basic method in DSP used to estimate unknown values between two known data points. In the context of audio signal processing, linear interpolation involves drawing a straight line between two adjacent samples and calculating intermediate values along that line. This technique is particularly useful for tasks such as resampling, where the goal is to change the sampling rate of an audio signal. By inserting new samples between existing ones, linear interpolation offers a straightforward approach to modify the signal's temporal characteristics. However, while it is computationally efficient, it may not effectively suppress high-frequency aliasing, potentially leading to audible artifacts in the processed audio. Therefore, in applications where audio fidelity is critical, more sophisticated interpolation methods, such as spline or sinc interpolation, are often preferred to achieve higher-quality results [1]. In Figure 3, figures (a) and (c) each consist of 50 samples. By zero-padding the frequency domain, these are expanded to 400 samples, resulting in figures (b) and (d), respectively.

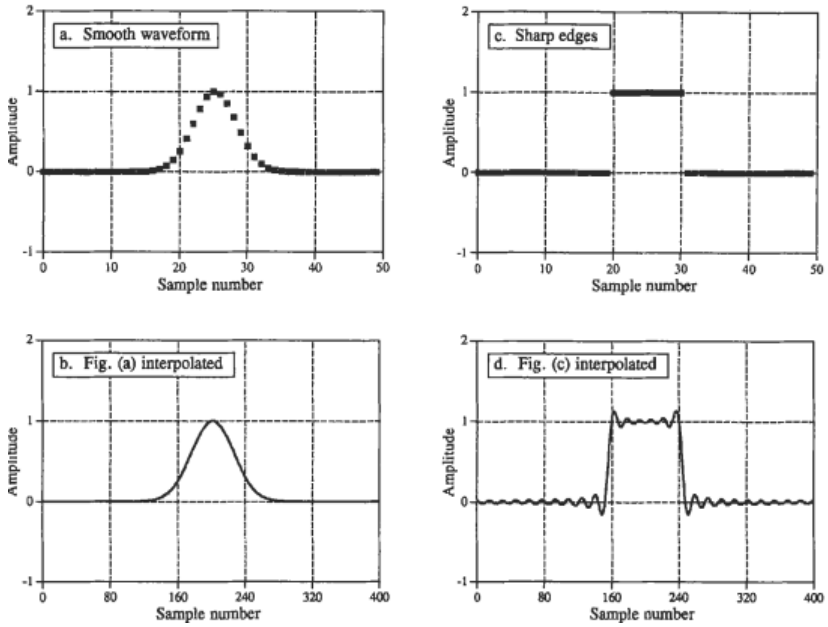

Figure 3: Interpolation by padding the frequency domain. Source: [13].

- **Librosa** is a Python package widely used for music and audio signal analysis. It provides essential tools for tasks such as feature extraction, onset detection and beat tracking, making it a valuable resource in music information retrieval systems. The library simplifies the implementation of various signal processing tasks, including time-stretching and pitch-shifting, and is recognized for its efficient management of audio data. With its comprehensive functionality, librosa supports both beginner-level and advanced audio analysis workflows. The library is actively maintained, with official documentation available for reference and advanced use cases [10].

2.2 Approach and Techniques

The methodology for implementing pitch-shifting in this project is primarily based on the work outlined in [7, 6].

2.2.1 Musical Theory and Timbre

Musical pitch is structured into octaves, each containing twelve semitones, with each semitone corresponding to a specific note. Pure notes are generally defined by their fundamental frequency, as shown in Figure 4. However, when different instruments play the same note, the sound differs due to the harmonics, which are integer multiples of the fundamental frequency. These harmonics contribute to the instrument's timbre. Pitch shifting modifies the pitch by a certain number of semitones, affecting both the fundamental frequency and harmonics, described by the relationship between the original and target frequencies.

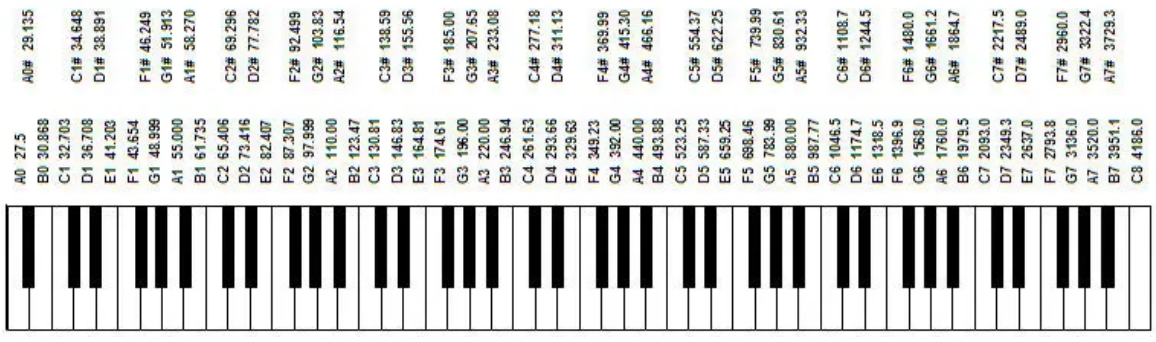

Figure 4: Fundamental frequency chart of a piano. Source: [15].

2.2.2 Proper Solution

One might assume that playing a recording at double speed would raise the pitch by an octave, but it also shortens the audio's duration, causing a mismatch between the desired pitch change and timing. A more effective method is to first time-stretch the signal to retain its original length and then adjust the pitch. For a simpler illustration of this process, refer to Figure 5. This approach ensures pitch shifting while preserving both timing and pitch. The details of this method will be further discussed in the following sections.

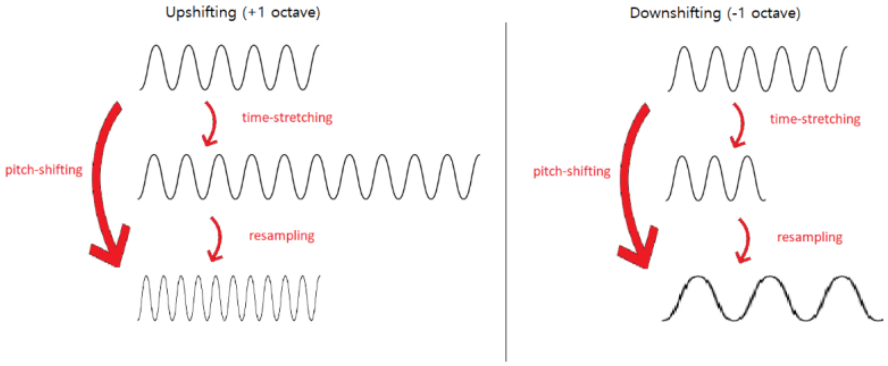

Figure 5: Pitch-shifting achieved through the combination of time-stretching and resampling. Source: [12].

Frame Overlap is essential for maintaining continuity in audio processing. Audio signals are divided into overlapping frames, usually with a 75 % overlap between consecutive frames. The spacing of these frames is then adjusted to either stretch or compress the signal. However, altering frame spacing can cause discontinuities, leading to audible glitches. To avoid this, advanced techniques such as the phase vocoder come into play to ensure smooth transitions between frames and eliminate such artifacts.

Phase Vocoder algorithm breaks the process into three stages:

1. Analysis: The signal is divided into small segments or frames and a Hanning window is applied to each to minimize spectral distortion from the windowing process. A rectangular window has more energy in the side lobes, while the Hanning window focuses most of its energy near DC, leading to a cleaner frequency representation. Each frame undergoes a Fast Fourier Transform (FFT) to convert it into the frequency domain, yielding a high-resolution spectrum with overlapping frames, enhancing the accuracy of the analysis. Figure 6 allows for a direct comparison between the Hanning and rectangular windows

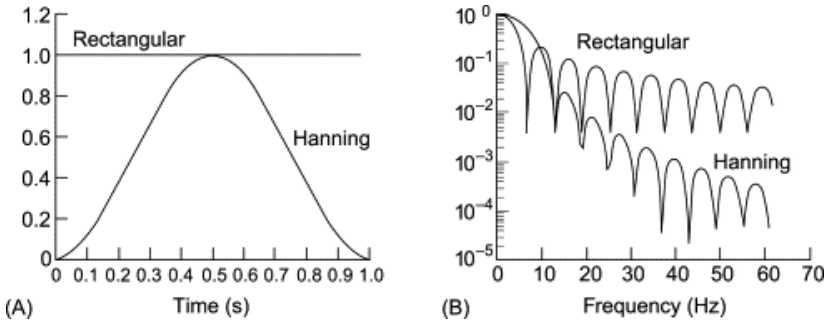

Figure 6: Rectangular and Hanning windows: (A) time domain shapes and (B) frequency domain magnitudes. Source: [3].

2. Processing: Frequencies are adjusted by examining phase differences between consecutive frames. When applying the FFT, the signal is divided into frequency bins. If a signal's frequency falls between bins, its energy is spread across neighbouring bins. Phase information is key for accurately determining the true frequency, particularly when it lies between bins. Analysing phase shifts between frames ensures smooth transitions and continuity, preventing artifacts and distortion in the frequency domain.

3. Synthesis: After phase adjustments in the frequency domain, the signal is converted back to the time domain using the inverse discrete Fourier transform (IDFT). A Hanning window is applied to each frame to smooth the transitions, ensuring a seamless reconstruction of the signal.

Resampling the signal is the final step to adjust playback speed and achieve the desired pitch shift. For integer scaling factors, this is straightforward, as samples can simply be skipped or repeated. When dealing with non-integer scaling factors, linear interpolation is used to calculate intermediate sample values. This method effectively reduces aliasing and preserves the audio's integrity.

By combining these stages, an effective and cohesive framework is established to achieve smooth and natural pitch shifts. This chapter concludes with a detailed review of the methods employed, enhanced by visual aids that reinforce the main concepts discussed.

3 Experimental Design and Setup

All algorithms and related experiments were implemented in `Python`, more precisely in `Jupyter Notebook` file format. As the development environment `Google Colab` was selected, due to its ability to quickly experiment with different libraries without the need to download or install them and the ability to create an online notebook accessible to multiple people.

We implemented 2 different algorithms, which are commonly used for pitch-shifting and also tried using the implementation from `librosa` library. The results of all three methods were then compared in our experiment Section 4.

3.1 Libraries

These are libraries that were used for the implementation of pitch-shifting algorithms. All of them are present in a separate cell in `Jupyter Notebook`. Default versions of all libraries available in the `Google Colab` environment were used. Therefore, it is recommended to run this code from said service, as then no issues regarding wrong versions of used libraries should occur.

```
import IPython.display as idisplay
import librosa
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d
```

3.2 Sped-up Audio

One of the cells is used to display a sped-up version of the original audio. This means that the audio is played at a modified sampling frequency. Of course, the audio is shortened/lengthened depending on which way the audio is shifted. The amount of pitch shift is defined in semitones. The purpose of this cell was mainly to quickly gather an understanding of what a selected audio would sound like after the change.

3.3 SOLA

We implemented the `Synchronized Overlap-Add (SOLA)` algorithm as described in the Section 2.1. This algorithm is used to increase the number of samples present in the signal, effectively time-stretching the audio. When this modified audio is then replayed at appropriate sampling rate, the audio will have the same length as the original, but the pitch will be changed according to selected amount. Same as with the sped-up version, the amount of shift is defined in semitones. Our implementation of `SOLA` uses `Hanning-window` to reduce the sharp transitions between windows, thus reducing artifacts in resulting audio.

3.3.1 How to use

This algorithm is implemented in the function `sola_pitch_shift`. Function signature:

```
def sola_pitch_shift(input_sgn: np.array, window_length: int,
            hop_length: int, semitone_shift: int
            ) -> tuple[np.array, float]:
```

As can be seen from the function signature, it has 4 required parameters. The parameters can be described as follows:

- `input_sgn` - original signal that we intend to modify
- `window_length` - length of overlapping windows used in `SOLA`
- `hop_length` - length of the window before overlapping with next window starts
- `semitone_shift` - the amount of semitones we want to change the pitch

The function returns `tuple` of two values, which are the modified signal (`np.array`) and the modified sampling frequency (`float`).

3.4 Phase Vocoder

The implementation of this algorithm was based on [6]. This algorithm works by modifying the signal in the frequency domain and then reconstructing the signal. By manipulating the phase information of the signal , the function can shift the pitch without significantly altering the timing of the audio. Therefore, this modified signal can be ran with the original sampling frequency. To improve the sound clarity and decrease the amount of aritfacts in the resulting audio, it also uses the `Hanning-window` to smooth out transitions between windows. The implementation is capable of processing stereo audio (or multi-channel audio), where each channel is treated separately.

3.4.1 How to use

This algorithm is implemented in the function `pitch_shift`. Function signature:

```
def pitch_shift(input_sgn: np.array, window_length: int,
        hop_length: int, semitone_shift: int
        ) -> np.array:
```

As can be seen from the function signature, it has 4 required parameters. The parameters can be described as follows:

- `input_sgn` - original signal that we intend to modify
- `window_length` - length of windows used for analyzing and processing the signal
- `hop_length` - step size between consecutive frames
- `semitone_shift` - the amount of semitones we want to change the pitch

The function returns only the modified signal (`np.array`), as its length is the same as the length of the original one and no changes to the sampling frequency are needed.

3.5 Librosa

From the `Librosa` library we used function `librosa.effects.pitch_shift`[1]. It allows us to change the pitch of an audio signal by a specified number of semitones, without altering its duration. The function uses a phase vocoder technique, which works by resampling the signal in the frequency domain, adjusting the pitch, and then converting it back to the time domain. This enables us to compare the quality of vocoder from Section 3.4 with the one used in `Librosa`.

[1] https://librosa.org/doc/main/generated/librosa.effects.pitch_shift.html

4 Analysis of Results

4.1 Original Audio

Through out this project we used only one audio sample[2]. Its original length is 13 seconds (588 000 samples), but to make all visualizations clearer, they show approximately only the first 4.5 seconds of the audio (200 000 samples).

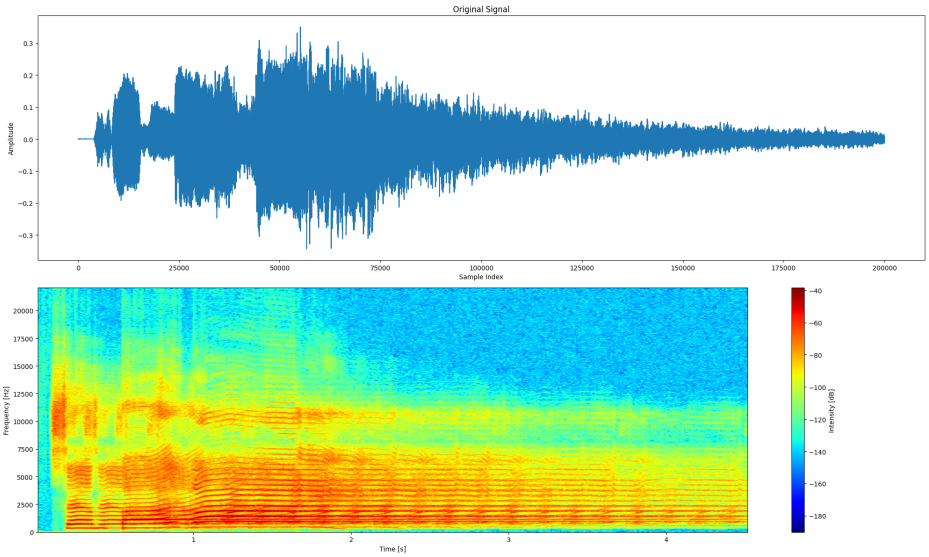

Figure 7: Original Signal

4.2 SOLA Pitch-shifted Audio

As can be seen on both graphs, the audio is shifted compared to the original signal. The reason for this is that our implementation of `SOLA` modifies the length of the signal. With proper sampling frequency, the audio length is the same as the original. From the waveform graph, we can see more noise at the beginning of the audio, mainly in places where the magnitude changes more sharply. On the spectrogram the introduced noise is harder to spot. Upon listening, the resulting audio does not contain any ear-piercing artifacts.

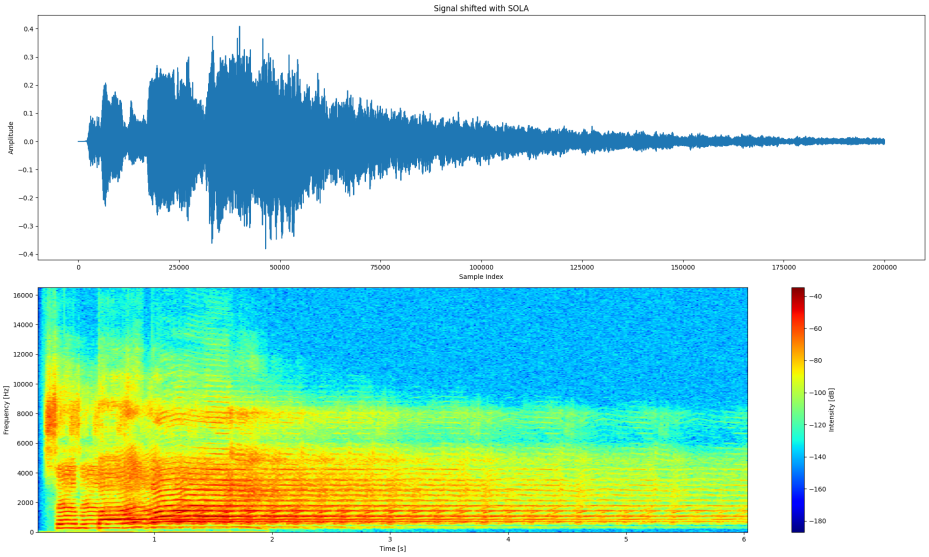

Figure 8: SOLA Pitch-shifted Signal

[2]https://samplefocus.com/samples/voice-fx-shot-anime-style-decay-long

4.3 Vocoder Pitch-shifted Audio

The output of this vocoder implementation is fairly distorted. The noise introduced during pitch-shifting can be clearly seen on both the waveform graph and the spectrogram. This is probably caused by non-ideal fusion of overlapping windows. Higher frequencies in the spectrogram have a much higher power than in the original signal and this newly introduced level of intensity seems to continue along the whole length of the signal. Upon listening, the audio is pitch-shifted but has a poor quality and noticeable artifacts.

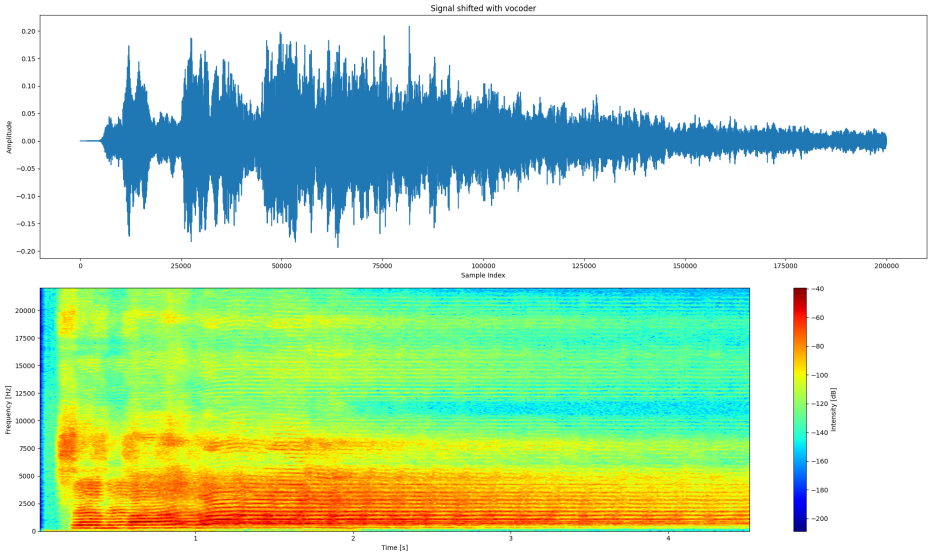

Figure 9: Vocoder Pitch-shifted Signal

4.4 Librosa Pitch-shifted Audio

Waveform graph shows us that the vocoder implemented in `Librosa` library does not introduce any significant noise. The spectrogram shows us that this was partially achieved by removing high frequencies. This could potentially cause some distortion at higher frequencies. Lower frequencies also have much larger power compared to the spectrogram of the original signal. Upon listening, the audio has good sound quality and does not have any noticeable artifacts.

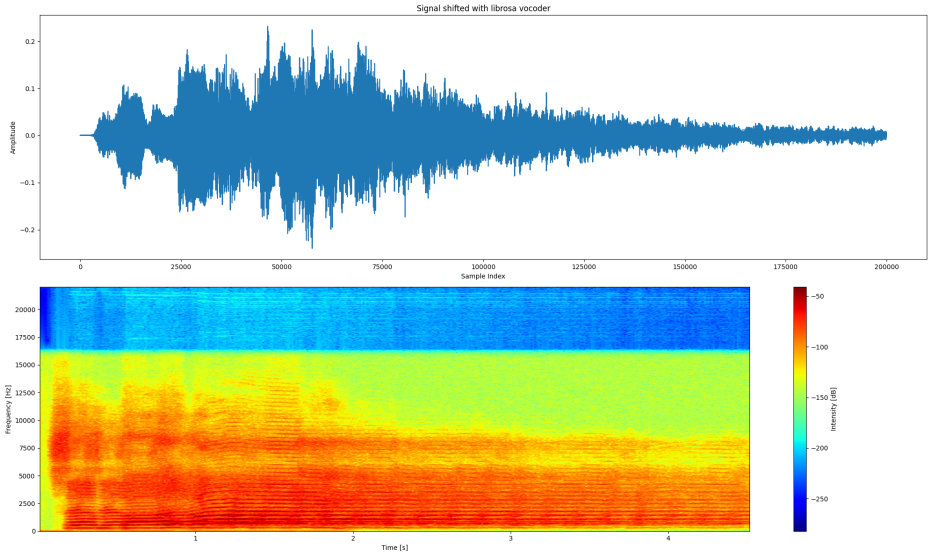

Figure 10: Librosa Vocoder Pitch-shifted Signal

5 Conclusion

All implementations of the pitch-shifting algorithms introduce noise or distortion to some level, but it is most noticeable in the case of the vocoder. The `SOLA` algorithm is computationally the easiest and provides fairly good results, but at least this version changes the number of samples present, which can be inconvenient in some use cases. The best result was created using the `Librosa` library implementation. This implementation of vocoder probably uses much more complex techniques than other tested algorithms, as the execution of the `Librosa` function takes much more time to finish, 9 seconds compared to 2 seconds in the case of `SOLA` or 5 seconds for the vocoder. All of these times were taken from `Google Colab` where execution time is shown for each ran cell.

Overall, the `SOLA` and `Librosa` methods appear to perform better in preserving the natural characteristics of the signal, while the vocoder introduces more noticeable artifacts. Combining this with the computational complexity of all of these algorithms, the `Librosa` seems to be the best, when used in non-real-time applications. The `SOLA` is useful when used in real-time applications or with limited hardware capabilities, as it still provides good results.

Bibliography

[1] Appleton, N. *Real-Time Re-Sampling and Linear Interpolation*. 1. November 2015. Accessed: January 11, 2025. Available at: https://www.appletonaudio.com/blog/2015/real-time-re-sampling-and-linear-interpolation.

[2] Bernsee, S. M. *Time Stretching and Pitch Shifting of Audio Signals – An Overview*. 18. August 1999. Accessed: January 11, 2025. Available at: https://blogs.zynaptiq.com/bernsee/time-pitch-overview.

[3] Braun, S. *Hanning Window*. 2001. Accessed: January 15, 2025. Available at: https://www.sciencedirect.com/topics/engineering/hanning-window.

[4] Gearspace. *Time/Pitch-Shifting: Formant Preservation?* 2017. Accessed: January 11, 2025. Available at: https://gearspace.com/board/music-computers/1161465-time-pitch-shifting-formant-preservation.html.

[5] Greenblatt, R. E., Pflieger, M. E. and Ossadtchi, A. E. Connectivity measures applied to human brain electrophysiological data. *Journal of Neuroscience Methods*. May 2012, vol. 1, p. 1–16. Accessed: January 11, 2025. Available at: https://www.sciencedirect.com/science/article/abs/pii/S0165027012000817.

[6] Grondin, F. *Algorithm*. 2025. Accessed: January 11, 2025. Available at: https://www.guitarpitchshifter.com/algorithm.html.

[7] Grondin, F. *Pitch Shifting*. 2025. Accessed: January 11, 2025. Available at: https://www.guitarpitchshifter.com/pitchshifting.html.

[8] Kulkarni, A. *Frequency Domain and Fourier Transforms*. Accessed: January 12, 2025. Available at: https://www.princeton.edu/~cuff/ele201/kulkarni_text/frequency.pdf.

[9] Laroche, J. and Dolson, M. Phase-vocoder: about this phasiness business. In: *Proceedings of 1997 Workshop on Applications of Signal Processing to Audio and Acoustics*. 1997, p. 4 pp.–. DOI: 10.1109/ASPAA.1997.625603. Accessed: January 11, 2025.

[10] Librosa. *Librosa: Audio and Music Signal Analysis in Python*. 2025. Accessed: January 11, 2025. Available at: https://librosa.org/doc/latest/index.html.

[11] Parviainen, O. *Time and pitch scaling in audio processing*. 2006. Accessed: January 15, 2025. Available at: https://www.surina.net/article/time-and-pitch-scaling.html.

[12] Royer, T. *Pitch-shifting algorithm design and applications in music*. 2019. Accessed: January 11, 2025. Available at: https://www.diva-portal.org/smash/get/diva2%3A1381398/FULLTEXT01.pdf.

[13] Smith, S. W. *Digital Signal Processing: A Practical Guide for Engineers and Scientists*. 2003. Accessed: January 11, 2025. Available at: https://pub.deadnet.se/Books_and_manuals_on_various_stuff/Electronics%20and%20Communications/Electronics%20ebook%20collection%20II/SMITH%2C%20S.%20W.%20%282003%29%20Digital%20Signal%20Processing%20-%20A%20Practical%20Guide%20for%20Engineers%20and%20Scientists.pdf.

[14] Technologies, A. A. *Tutorial: Auto-Tune Pro 11*. 9. April 2024. Accessed: January 12, 2025. Available at: https://www.antarestech.com/community/tutorial-video-getting-started-with-auto-tune-pro-11.

[15] timbravo. *Fundamental Frequency Chart: Indispensible EQ Tool*. 17. November 2010. Accessed: January 15, 2025. Available at: https://mediapronerdzone.wordpress.com/2010/11/17/fundamental-frequency-chart-indispensible-eq-tool/.

[16] Wikipedia. *Auto-Tune*. 2025. Accessed: January 11, 2025. Available at: https://en.wikipedia.org/wiki/Auto-Tune.

[17] Wong, P. H. W., Au, O. C., Wong, J. W. C. and Lau, W. H. B. On improving the intelligibility of synchronized over-lap-and-add (SOLA) at low TSM factor. In: *TENCON '97 Brisbane - Australia. Proceedings of IEEE TENCON '97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications (Cat. No.97CH36162)*. 1997, vol. 2, p. 487–490 vol.2. DOI: 10.1109/TENCON.1997.648251. Accessed: January 11, 2025.