Thomas Gornet

tgornet2

ECE 210H

12/9/2024

DSP MIDI Synthesizer Project Report

Introduction

This project explores real-time MIDI audio synthesizing using a MIDI Keyboard, a Raspberry Pi 4, and Python. The implementation also integrates several audio signal processing effects such as pitch shifting, vibrato, distortion, chorus, and reverb to simulate effects on a synthesizer. This project connects concepts in ECE210 and 210H such as Fourier Transforms, in particular FFTs, time modulation, frequency modulation, and LFOs. The project was chosen because of my interest in music. As a piano player, I chose this project because it combines my passion for music with waveform manipulation, real-time signal processing, and embedded systems—concepts aligned with ECE 210H coursework.

Theory

Key Features and Their Theory:

1. MIDI Input Handling:

MIDI is a protocol developed to standardize communication between electronic musical instruments, computers, and other audio devices. MIDI signals do not carry audio directly; instead, they send control messages representing musical information like: note on/off Messages, pitch wheel data, and control change messages. These MIDI messages can be interpreted and mapped to different frequencies, control signals, and other parameters for use in waveform generation and signal processing.

2. Waveform Synthesis:

Once the MIDI messages are decoded, they must be translated into audible waveforms. In the case of this project, the frequencies of each note were simply used to form sine or saw waves depending on the input mode, however more complicated waves can also be created from the MIDI messages. These waveforms are digital which means they have a finite number of sampled points that represent the "continuous signal" based on a sampling frequency.

3. Signal Processing:

After turning the MIDI messages into time-domain waveforms, the system alters the waveforms using several different signal processing techniques listed below.

Pitch Shifting – The system adjusts pitch by performing a FFT. This turns the time-domain waveform generated previously into a frequency domain signal. Since the relationship between pitch and frequency is exponential, in order to increase the pitch, instead of simply modulating the frequencies, you must modulate each frequency by multiple of that particular frequency. After shifting the frequencies, the time-domain signal can be reobtained by performing an inverse FFT.

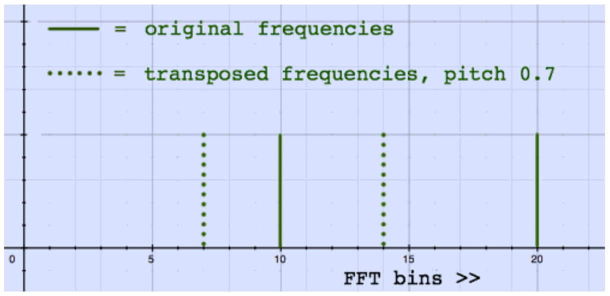

Image 1: Diagram showing FFT pitch shifting – Note how the frequencies are not shifted by the same value, but rather are shifted by a ratio of each frequency.

Vibrato – Vibrato is a periodic modulation of the pitch at a low frequency. Rather than using an FFT, in order to shift the pitch for vibrato I used interpolation to stretch or shrink the time domain signal, which has the same pitch change effect as the FFT approach.

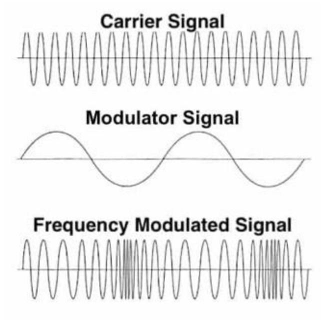

Image 2: Diagram showing how vibrato modulation works – the base waveform changes frequency based on a low frequency, modulator signal, often called a LFO. Although this diagram shows frequency modulation, the same idea applies forpitch modulation.

Distortion – Distortion adds sound to the waveform by raising the amplitude and chopping the peak amplitude. This emulates distortion caused in speakers from the signals being limited by an op-amps biased terminal. Adds a “crunchy” sound to the waveform.

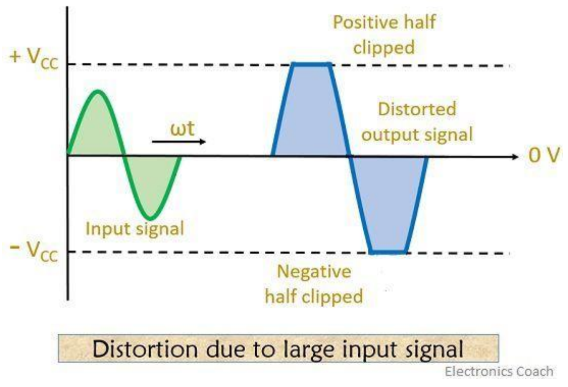

Image 3: Diagram showing distortion – the signals amplitude is increased and then chopped, leading the waveform to take on a more jagged, harsher sounding shape.

Chorus – The chorus effect combines delay and pitch modulated copies of the waveform. Thus, the effect is made up of two copies of an identical signal, time shifted and interpolated in order to delay and pitch modulate the signal. This makes the waveform sound fuller like the natural delay and pitch modulation in a choir or band.

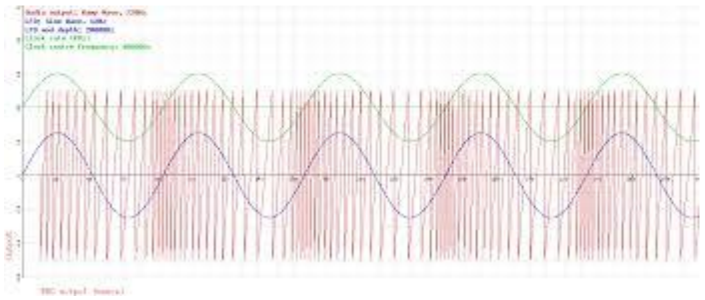

Image 4: Diagram showing how the chorus effect works – since the frequency and phase are different, there are imperfect beat frequencies in the sound due destructive and constructive interference.

Bit-Crush – A bit crusher functions by changing the sampling rate of the sound. It does this by quantizing the signal. This means that it multiplies the signal by a certain value, rounds the number off, and then divides the signal to make smaller differences in amplitude. This makes the sound crunchier and lower quality.

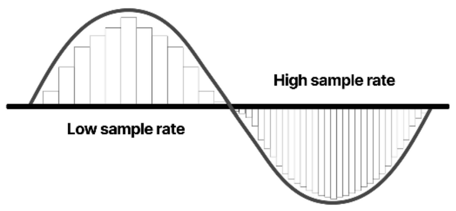

Image 5: Diagram showing the basis of a bit-crusher – Via quantization it changes the signals sampling rate, which adds more jagged, sharper values to the waveform.

Development

I started the implementation of my project by setting up the Raspberry Pi. This was one of the most time-consuming parts of the project. I had trouble with SSH and installing Python libraries. However, I was able to solve the SSH problem by connecting it directly through ethernet, and I was able to solve my Python problems by using a virtual environment on the Raspberry Pi. The first part of code that I wrote was the MIDI input system. This was fairly simply through the use of the Mido Python library. The Mido Library transforms the MIDI messages from the keyboard to a format that makes it easy to manipulate in Python. Therefore, all it took to set up the MIDI input was just to for-loop through the MIDI messages and use a dictionary to turn the note messages into frequencies.

The next part of the project that I implemented was audio output. This part of the project was also simple. Since I just used the pyaudio library to turn waveforms into an audio stream to be output by the Raspberry Pi.

After coding the audio output, I worked on the waveform generation to make the synthesizer functional. This is where I ran into my first major bug. I had an issue where when I held down a note it would cause a periodic popping noise from the speaker. After debugging, I realized that the issue was from phase misalignments of the waveform. Since the audio streams are only 0.06 seconds in duration, if the phase between the waves do not line up between iterations, the waveform would have to rapidly change in amplitude due to discontinuities, causing the speaker to make a popping noise. To fix this, I tracked each wave's phase shift and then added it to the waveform of every cycle to keep the signal continuous.

Afterwards I started to work on the signal processing effects. The first effect I made was distortion. This was very simple because it just required me to scale the amplitude of the wave and then clip it with NumPy. After distortion, I worked on pitch shifting using FFTs. Originally, I tried to shift the bins on my own after performing an FFT. However, this caused weird artifacts in the signal, such as random high and low frequency sounds. So, to improve the phase shifting, I used the interpolation function of NumPy which estimates how the bins should shift if they are out of range or do not line up correctly. However, this did not solve all my bugs. I again faced a phase difference issue where when I performed the FFT I lost phase information that caused discontinuities in the time domain. So, I again solved this by keeping track of the phase throughout the FFT and applying it again in the end. Since pitch shifting with FFTs did not work as well, when I started working on vibrato, I used only interpolation. This worked much better and making the vibrato function was simple. After vibrato, I implemented the chorus effect. This was also fairly simple, but I had some bugs related to circular shifting for the time delay. However, I was not able to figure out what caused the sound related bugs with the chorus effect, but the noise was not too bad, so I simply mixed it with the original waveform at different ratios, making the effect sound much better. Finally, the last effect I implemented was a bit-crusher. The bit-crusher was not hard to implement since it only used rounding to artificially lower the sample rate.

As for hyperparameters that I changed throughout the process, I changed the duration of each audio stream and the number of bins. I increased the duration of the audio stream in order to get minimize artifacts caused by phase offsets, but I made sure not to increase it too high in order to keep the synthesizer responsive. I decreased the number of bins for the audio stream because I realized that it reduced the latency.

Results

I think that my project's performance met the expectations that I set. I was able to make a playable keyboard with a variety of waveforms as well as filters to come up with unique sounds to play with. However, there are some things I am unhappy about. For instance, I worked a long time trying to implement convolution-based reverb. However, I ran into a lot of problems with convolution since it increased the length of a sample. So, it made it very difficult to create a long echoey reverb effect. Furthermore, some effects such as chorus are not as clean sounding as I would want them to be. At the end of the day however, all the signal processing effects work as intended, yet a bit noisy, and the MIDI input and audio output work perfectly.

Conclusion

In conclusion, this project successfully demonstrated the integration of MIDI input, waveform synthesis, and real-time signal processing on a Raspberry Pi 4 using Python. The synthesizer implemented MIDI input handling, waveform generation, and several audio effects such as pitch shifting, vibrato, distortion, and chorus. These effects provided an interactive and dynamic audio experience that aligns with principles from ECE 210H, such as Fourier Transforms, time modulation, and frequency modulation.

Although the project met most of its goals, challenges such as phase alignment, latency management, and implementing convolution-based reverb presented difficulties. Despite these, the synthesizer performs well and offers a range of sounds and effects that are functional, even if not entirely noise-free. The hands-on experience with real-time signal processing, debugging, and waveform manipulation strengthened my understanding of concepts from ECE 210H and their practical applications.