HARMONIZER EXTENSION OF MIDI KEYBOARD

Gavin Baker

University of Rochester

ABSTRACT

LPC analysis allows us to create a filter based on any input signal, and while that signal is many times used to filter a signal of similar character, there is no requirement that this be true. I have applied a LPC filter to a voice signal, and then used that filter to modulate the spectra of a MIDI-input pitched collection of sawtooth waves, resulting in a voice-like output with only the desired frequency content. This is similar to the classic vocoder, and differs by changing the carrier frequency to be user-definable. While this works in principle, the wide range of harmonic amplitudes in a sawtooth waveform complicates this process, leaving a spectrum that, while certainly between voice and sawtooth wave, definitely trends toward the sawtooth wave.

1. INTRODUCTION

This project largely relies on the vocoder algorithm for baseline design [4], and modernizes that design by using LPC analysis [5] instead of the classic vocoder filter bank. The classic vocoder algorithm reads in a voice signal, passes it through a bank of band pass filters with envelope followers, and then uses those envelopes to modulate the spectrum of a carrier signal on the output (usually wide band white noise). While not perfect, this concept can produce intelligible speech as output, which is very useful for the creation of a harmonizer.

The classic vocoder design uses a bank of band-pass filters to extract envelopes for various spectral components. With more modern tools, however, we can instead use the Fourier Transform of a signal to approximate a filter bank far larger than is reasonable to construct in a classical setting. This allows for much finer grained tuning of the output signal, and is a critical digital update for modern implementations and extensions of the classic vocoder.

While the Fourier Transform of a signal is directly useful for modulating the output noise of a vocoder, we can improve this design by using Linear Predictive Coding to create a filter based on the envelope of the spectrum of a signal. While this destroys the harmonic nature of voice input, it creates a much smoother overall contour of the signal in question, allowing for more of the work to be done by the carrier signal (i.e. the wide band noise in the classical vocoder). This is desirable in a harmonizer context because we want to do just that- provide a much more complex carrier signal to modulate over.

Additionally, it is important to note that in order to modulate the carrier by the input, all we need to do is multiply their spectra together. This will scale the two spectra by each other, causing the product to be a direct combination of the two spectra.

1.1. MIDI Input

A critical part of this project is a user-defined carrier signal. I achieve this by taking input from a MIDI keyboard, and using that input to create a set of sawtooth waves at pitches defined by the frequency associated with each MIDI number. These waves are then summed together to create one composite waveform, which acts as the carrier signal in my implementation.

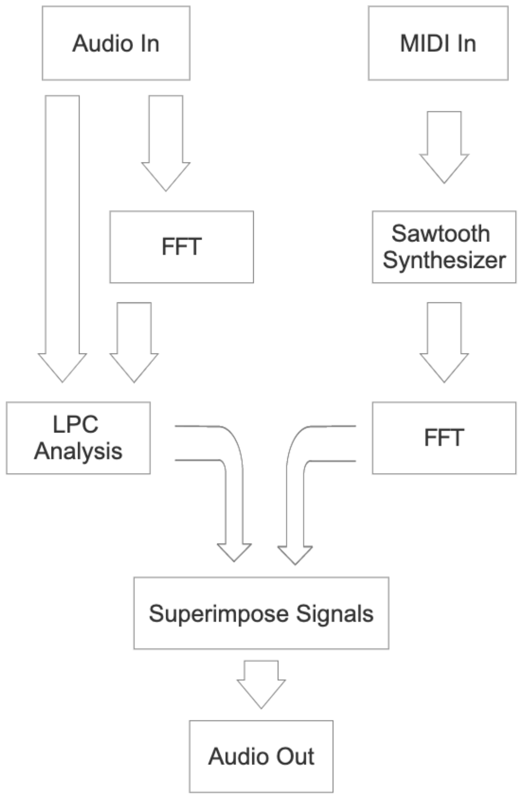
Sawtooth waves are close to the ideal waveform for this implementation, as they contain all harmonic frequencies of the fundamental pitch. Since specific harmonics of the fundamental are accentuated by speaking different vowels, a carrier spectrum that does not ignore any harmonics is critical for intelligibility of speech on the output.

1.2. LPC Analysis

As previously discussed, I performed a Linear Predictive Coding analysis on the inputted voice audio, in order to find the overall envelope of its spectrum. This coding allows much greater freedom for the carrier signal to not exactly match the frequency content of the voice, and negates the need for accurate (and thus computationally expensive) input signal pitch tracking. My LPC analysis constructs a filter based on the spectrum of the input signal, which can then be applied elsewhere. While many applications of LPC based filters are used on signals similar to the signal used to generate the filter, this is not necessary, and it is possible, and even convenient, to use this filter on other signals, such as in my implementation of this project.

2. METHODS

2.1. Block Diagram



2.2. Implementation

I implemented the final version of this project in Python, using PyAudio [1], PyGame [2], and LibRosa [3]. I also implemented several mockups and test runs in MATLAB, in order to develop the algorithms that I would use in the real time processing version.

In order to implement the LPC analysis, I used the LibRosa lpc() function, which uses Burg's method CITE to find the LPC coefficients of a signal. This is not the same as the MATLAB signal processing toolbox lpc() function, but while MATLAB computes coefficients for the denominator of a signal, LibRosa computes the same coefficients, however modified so as to be suitable for a numerator term instead. I also computed the error (epsilon) term in order to normalize the filter, but I was unsuccessful in doing so.

The MIDI input for this project was implemented with PyGame, which has a MIDI message interpreter and device finder ready to go out of the box. To synthesize the output, PyGame polls the MIDI stream for either NoteOn (code 144) or NoteOff (code 128) events, and, upon finding one, updates a list of frequencies with the frequency equivalent to the inputted MIDI note. This list of frequencies is subsequently polled, and a sawtooth wave is generated for each element in the list.

If each sawtooth wave were generated raw at the beginning of each callback, there would be a noticeable presence of artefacts in the sound where one callback ended and the next began, due to the fact that we cannot ensure that the waves begin and end at the same level. To fix this, we retain the position of the wave at the end of the most recent callback, and begin the next wave at the same position. This ensures that the audio plays back smoothly and without artefacts at the beginnings and endings of callbacks.

Once this step is completed, these waves are summed together, and the result is the carrier signal to be modulated by the input audio.

Combining these ideas, we can assemble a vocoder design with the LPC analyzed input signal modulating a MIDI generated carrier. While this will work well assuming the input audio and the carrier signal are in similar (i.e. within approximately 80 hz of each other), it completely fails when they differ. To combat this, we can pitch shift the input audio to within the desired frequency range, to re-achieve this similarity.

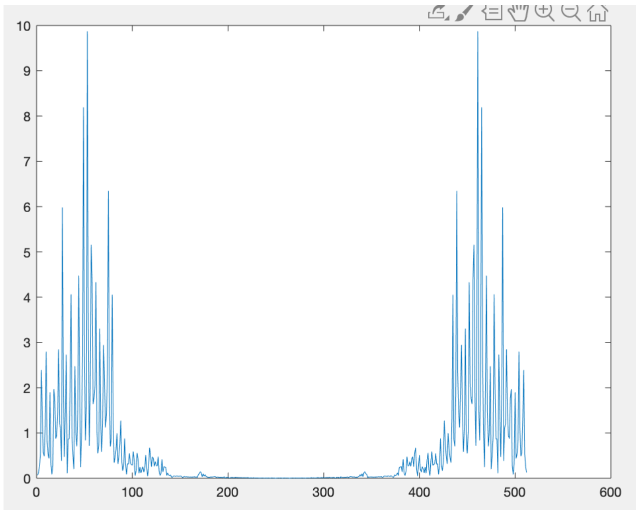
Since the band of input audio frequencies that will work for any given carrier pitch is so wide, it is not necessary to use a computationally heavy pitch detection algorithm- rather it is sufficient to use one that will predict the pitch to within a range of a few Hz of the true fundamental frequency. To this goal, I apply a frequency domain based algorithm that finds the first local maximum frequency bin above a certain threshold on the dB scale. This is the same pitch detector that was employed in class Homework 3. While it is not perfect, and often produces bad results on non-vowel based sounds, it is sufficient for my purposes. Indeed, performance on non-vowel sounds is not relevant, as the harmonic spectrum of non-vowel sounds is much less sensitive to specific frequencies than voiced sounds.

Now that we can achieve better performance across pitch classes, we can multiply the magnitude spectra of both the carrier and the (pitch shifted) modulator together to create a combination spectrum that is representative of both. In particular, since the modulator is normalized to have a maximum value of 1, the sawtooth waves will be scaled down by a factor of the modulator.

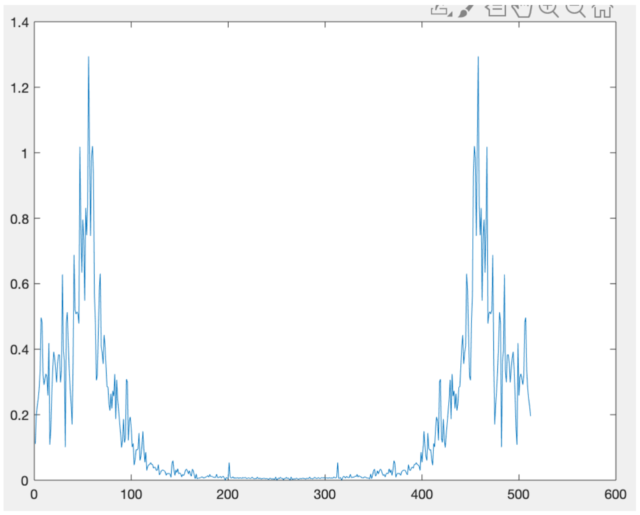
3. CONCLUSION

3.1. Signal Comparisons

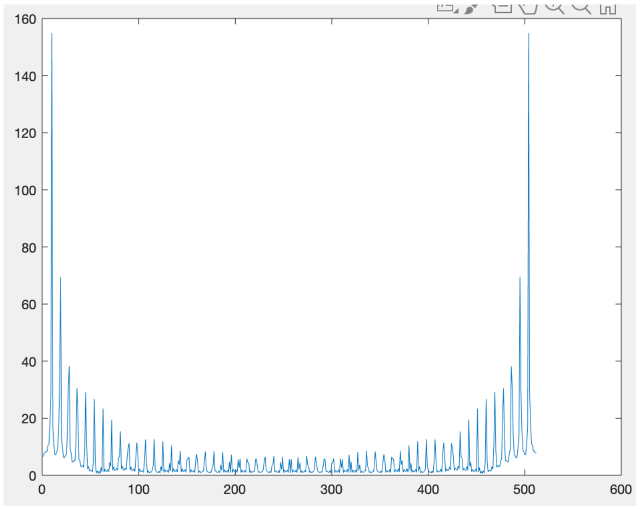
Given an input frame FFT with clear harmonic intervals:



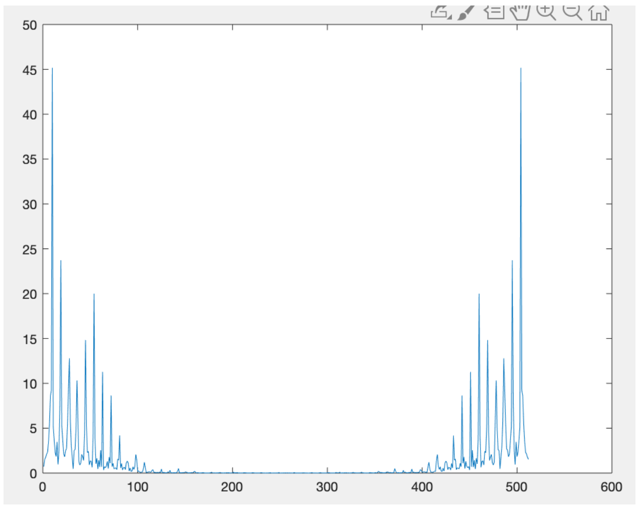
We can find the LPC filtered spectrum:



We then find the MIDI input spectrum (in this case, 1 note at 277 Hz):



The superimposition of the LPC filtered spectrum and the MIDI input spectrum are then:



Which is clearly a combination of the two input spectra.

When applied over many frames subsequently, this produces a synthesis of the two signals.

3.2. Further Work

While I am happy with the design of my algorithm and signal processing system, there are several shortcomings that have appeared over the course of the development of this system.

Firstly, there is a huge audio input latency issue that will prevent this system from working in real time. There is approximately 3/4 of a second of delay between audio being read in to the input buffer and audio being processed by the algorithm on my computer. I believe that this is due to the overhead of computing my callback function, which, through the sheer number of calls, has turned out to be a heavy load. I believe that rewriting this application in a statically typed language, such as C, would speed up computation by freeing up time used up by the Python type checker.

Secondly, despite my best efforts, the synthesizer still has some issues with artefacts, likely to do with latency as well. I believe that this problem has a similar solution, which is not within the scope of this project.

Besides porting this project over to C, I plan to continue working on this project by changing my algorithm slightly. To combat the shortcoming of more of the sawtooth wave showing up than the vocal spectrum, I will simply label all the harmonics of all of the notes played by the MIDI input on the LPC filtered input audio, and then zero out any frequency bins that are unlabeled. Thus, I will preserve the exact ratios of the frequency bins present in the voice spectrum, but also maintain intonation.

4. REFERENCES

[1] H. Pham. PyAudio: Python bindings for PortAudio; [online] Available at:

https://people.csail.mit.edu/hubert/pyaudio/ Accessed 4/25/2020

[2] P. Shinners. PyGame.midi: pygame module for interacting with midi input and output. [online] Available at: https://www.pygame.org/docs/ref/midi.html Accessed 4/18/2020

[3] LibRosa Dev. Team. LibRosa: a python package for music and audio analysis; [online] Available at: https://librosa.github.io/librosa/ Accessed 5/2/2020

[4] H. Dudley (October 1940). "The Carrier Nature of Speech". *Bell System Technical Journal.* XIX (4).

[5] R.M. Gray (2010). "A History of Realtime Digital Speech on Packet Networks: Part II of Linear Predictive Coding and the Internet Protocol" (PDF).*Trends Signal Process.* 3 (4): 203–303. doi:10.1561/2000000036. ISSN 1932-8346.