



Saketh Aripirala

# Effects of Bass Guitar Pickups on Pitch Detection and Pitch Shifting

Metropolia University of Applied Sciences

Bachelor of Engineering

Electronics

Bachelor's Thesis

24 May 2023

## Abstract

Author: Saketh Aripirala  
Title: Effects of Bass Guitar Pickups on Pitch Detection and Pitch Shifting  
Number of Pages: 51 pages + 2 appendices  
Date: 24 May 2023

Degree: Bachelor of Engineering  
Degree Program: Electronics  
Professional Major:  
Supervisors: Heikki Valmu, Principal Lecturer  
Juha Kivekäs, Embedded Systems Engineer

The thesis work aimed to study the effects bass guitar pickups produce when the signal is pitch-shifted or detected for synthesis. Due to the significant role a pickup plays in the sonic qualities of a stringed electric instrument, it is vital to understand the effects it yields when the signal is subject to pitch tracking or alteration algorithms. The main goal of the study was to aid Darkglass Electronics, a Finnish bass accessory manufacturer, in developing embedded effects for bass guitars without any compromise in sonic quality.

Using data analytics programming languages such as Python, a correlation and analysis tool was developed. Data points were collected from using an audio signal with the algorithms and compared to the ideal application. The script determined and flagged deviations and errors, which led to further investigation for the cause and suggested potential improvements. To perform tests on the pickup types, a bass guitar was modified to contain a humbucker pickup and a piezo bridge pickup. A debugging pre-amplifier was designed for the bass guitar to encompass signals from both pickups using Altium - an ECAD software.

The results of the tests concluded that the piezo pickup produced effects that were undesirable for the Darkglass DSP application of the pitch detection and shifting algorithms. Furthermore, the harmonic content analysis determined the fundamental difference between the pickup types. Although the piezo produced the most errors, various applications where the piezo would be a viable choice were also discovered.

---

Keywords: Guitar Pickups, Digital Signal Processing, Python, Bass Guitars

---

# Contents

## List of Abbreviations

1	Introduction	1
2	Fundamental Theories and Concepts	2
2.1	Digital Signal Processing	2
2.2	YIN Algorithm	7
2.2.1	Autocorrelation Function	8
2.2.2	Difference Function	10
2.2.3	Cumulative Mean Normalized Difference Function	11
2.3	Octaver Algorithm and Model	13
2.3.1	Analog Octaver	14
2.3.2	Octave Error	18
2.3.3	Phase Error	19
2.4	Pickup Fundamentals	20
2.4.1	Magnetic Pickups	20
2.4.2	Piezo-electric Pickups	26
3	Testing Methods	27
3.1	Debugging Pre-Amplifier and Bass Modifications	27
3.2	Test Data and Considerations	37
3.3	Python Testing Script and Sonic Visualizer	38
4	Results	44
5	Discussion	49
6	Conclusion	51
	References	52
	Appendices	
	Appendix 1: Datasheets	
	Appendix 2: Source Code	

## List of Abbreviations

AC:	Alternating Current
ACF:	Autocorrelation Function
CMNDF:	Cumulative Mean Normalized Difference Function
DC:	Direct Current
DFT:	Discrete Fourier Transform
DSP:	Digital Signal Processing
F0:	Fundamental Frequency
FFT:	Fast Fourier Transform
GPIO:	General Purpose Input Output
IC:	Integrated Circuit
JFET:	Junction Field Effect Transistor
MIDI:	Musical Instrument Digital Interface
Op Amp:	Operational Amplifier
PCB:	Printed Circuit Board
PYSPTK:	Python Signal Processing Tool Kit
SMD:	Surface Mount Device
THD+N:	Total Harmonic Distortion and Noise

## 1 Introduction

In the world of digital audio processing, pitch manipulation effects and sound synthesis are commonly researched subjects and are widely used by musicians to alter and produce new sounds. The origins of sound synthesizers trace back to the early 20<sup>th</sup> century, where analog oscillators were mainly utilized to produce pure tone sounds such as sine, square, and sawtooth waves. In more modern applications, synthesis uses digital signal processing or hybrid systems to produce more complex musical tones. Similarly, pitch manipulation is a very popularly used tool to modify the perceived pitch of an instrument or speech. Most common styles of pitch manipulation are often used to shift the signal to different musical intervals. An octaver is a common pitch manipulation tool widely used on instruments to shift the signal down an interval of an octave, essentially halving the frequency of the signal. This results in the addition of subharmonic bass frequencies.

With emerging audio technologies, the signal of a stringed instrument can be used to synthesize pure or complex tones by tracking the pitch of the note played. Although it may seem trivial to track the pitch or fundamental frequency of an instrument; in reality, there are complexities stemming from the timbre (tonal quality of a sound [1]) and the nature of the instrument that cause tracking errors or inconsistencies. Comparable issues occur when the pitch is shifted and worsened with certain cases where an error causes perceivable differences.

By understanding the fundamentals of guitar pickup technology, a much wider comprehension of the role pickups plays in the harmonic contents of the signal, and their effects on the algorithms can be achieved. Moreover, methods to mitigate errors in these algorithms can also be investigated.

To test the function of pickup types in errors and the overall functionality of the algorithms, a bass guitar was modified to contain two specific types of pickups: a generic humbucker pickup in a split-coil configuration and a piezo bridge pickup.

An on-board debugging pre-amplifier was designed to encompass the signals of the bass guitar utilizing Altium Designer - an ECAD software. Using python, a programming language widely used for data analytics, correlation and analysis functions were implemented to study the changes in the fundamental frequency tracking stability, errors, and deviations. Lastly, the analysis of the harmonic contents of the signal and the testing method was validated using Sonic Visualizer.

The findings of the study aid Darkglass Electronics, a Finnish bass guitar accessory manufacturer, in pursuing technology and methods to implement bass guitar effects embedded into an instrument. The algorithms used to acquire the test data were effects made in-house by Darkglass Electronics, which include a faithful modelling of an analog octaver, a digital hybrid octaver, and a bass guitar synthesizer.

## **2 Fundamental Theories and Concepts**

To understand the errors conditions and study goals, it is essential to have a solid comprehension on the fundamental implementation of the algorithms, guitar pickup technology, digital signal processing, and spectral analysis. The following section covers the necessary prerequisites.

### **2.1 Digital Signal Processing**

Digital signal processing is a commonly used technique to analyze and modify real world signals such as sounds, measurements, and other data. Analog signals are discretized digitally using Analog-to-Digital converters and using fundamental mathematical functions, the data is manipulated. [2.] For discretizing signals, the data is sampled at recurring instances. The rate at which these instances are captured is known as the sampling frequency ( $F_s$ ), measured in Hertz. [3.] According to the Nyquist-Shannon sampling theorem, an analog signal can be accurately reconstructed only if the sampling frequency is more than twice the

maximum frequency of the sample [4]. Failing to satisfy the Nyquist-Shannon theorem leads to an effect called signal aliasing.

An essential concept in Digital Signal Processing is windowing and hop size. Windowing divides a signal into smaller intervals of signal for which the processing is performed. Typically, windows are overlapped after each other; the number of samples in non-overlapping regions of the window is called the hop size. [5.] Figure 1 depicts windowing and hop size for an audio sample.

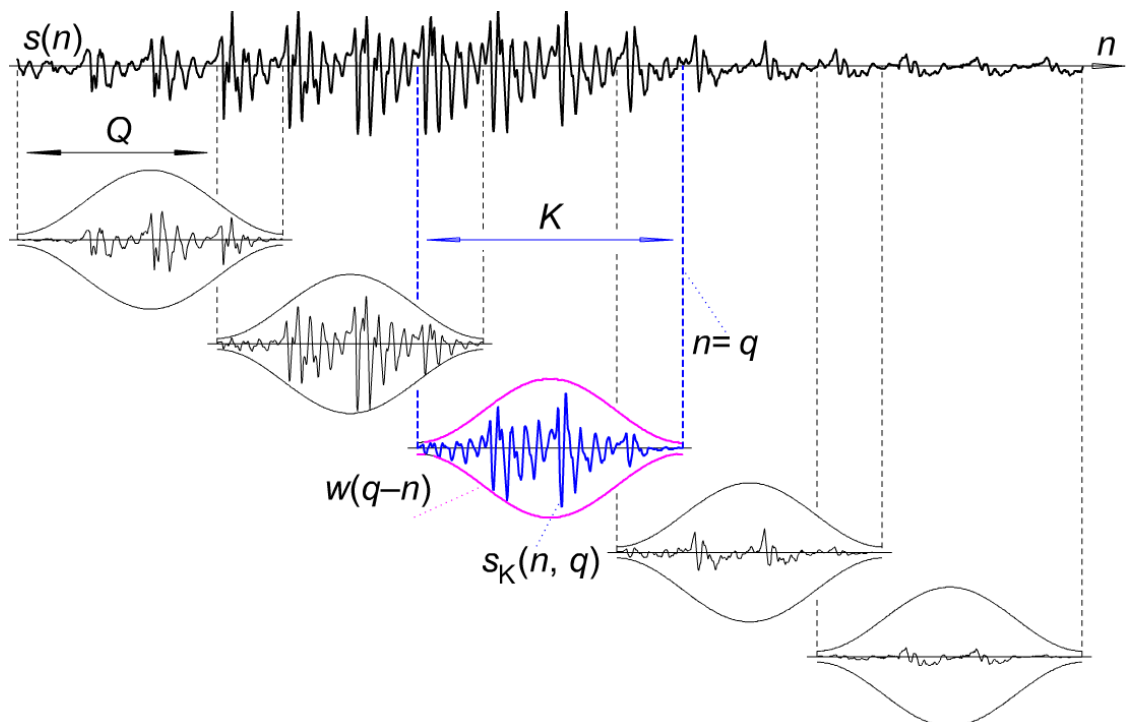


Figure 1. Windowing and Hop size.  $Q$  denotes the hop size and  $K$  represents the window length [6].

A signal's contents can be analyzed using methods like spectral analysis. In spectral analysis, the Fourier Transform of a signal is performed to calculate the magnitude of each frequency component present in a signal. The Fourier Transform is translated into DSP via the discretized and sample based Discrete Fourier Transform (DFT). The mathematical implementation of the Fourier Transform and DFT is shown in Equation (1) and (2):

Let  $f(t)$  be a function. Then the Fourier Transform of the function is given as follows:

$$F(k) = \int_{-\infty}^{\infty} f(t)e^{-2\pi jkt} dt \quad (1)$$

Where  $j$  is the complex imaginary unit and  $k$  is the frequency.

Then the DFT for  $N$  number of samples is given by:

$$F(k) = \sum_{k=0}^{N-1} f(k)e^{-2\pi jk/N} \quad (2)$$

Each value of  $k$  denotes a frequency bin. The magnitude and phase of the frequency bin is calculated by using the magnitude and angle formulas for a complex number  $a + jb$  where  $j$  is a complex number. The Fast Fourier Transform (FFT) algorithm improves the implementation of the DFT. It requires fewer computational steps to perform the calculations.

The spectral information calculated using DFT can be represented by graphing the magnitude for each frequency bin or a spectrogram. The graphing method provides the harmonic contents of the signal as a function of its magnitude, whereas a spectrogram provides the magnitude of the harmonic contents, or frequency bins, as a function of time. Figure 2 and Figure 3 contain the graphing and spectrogram representations.



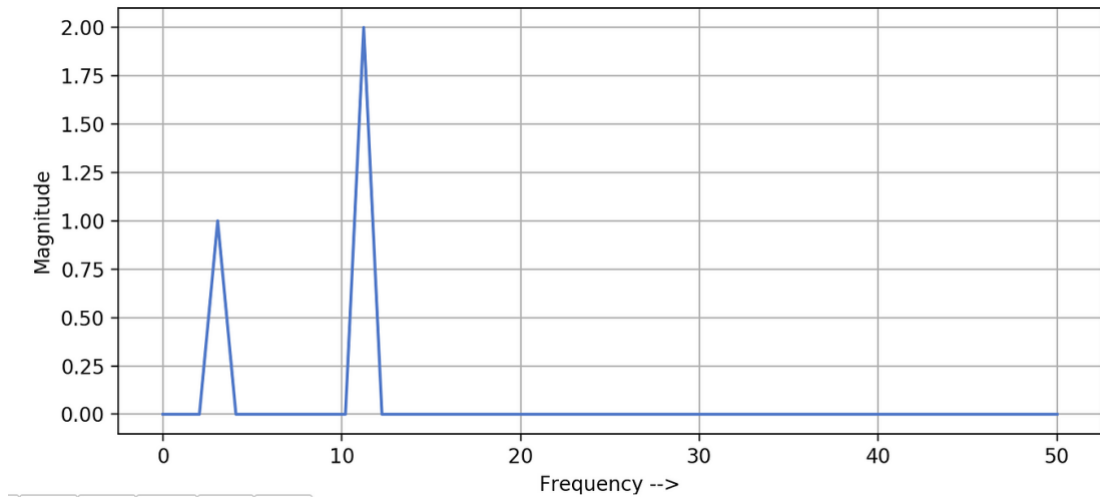


Figure 2. Graphing Frequency as a Function of Magnitude [7].

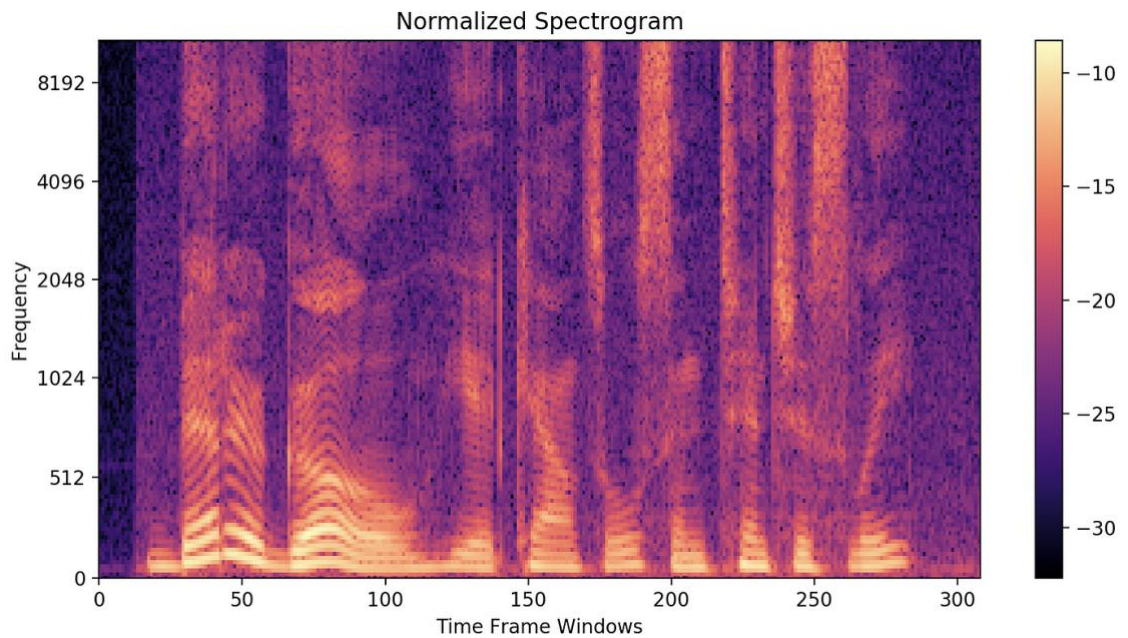


Figure 3. Spectrogram [7].

The lowest frequency component present in the signal is known as the fundamental frequency. In a periodic signal, multiples of the fundamental frequency are known as the harmonics or overtones. [8.] The relationship between the fundamental frequency and subsequent harmonics is shown by Equation (3).

$$f_n = (n + 1) * f_0 \quad (3)$$

Where  $f_n$  is the  $n^{\text{th}}$  harmonic. In practice however, the fundamental and subsequent harmonics do not necessarily maintain a perfect 1:  $n$  relationship with the harmonics - these deviated harmonics are commonly referred to as inharmonic overtones or partials. Partial may be slightly higher or lower from the ratio and possibly contain differing phases. The timbre of the sound is unique for different sounds due to the varying magnitudes of the harmonics and partials.

Figure 4 describes the relationship between the fundamental frequency and the subsequent harmonics. The period of the wave doubles for each harmonic, i.e., frequency is twice. In music, it is generally accepted that the perceived pitch is the fundamental frequency of the signal.

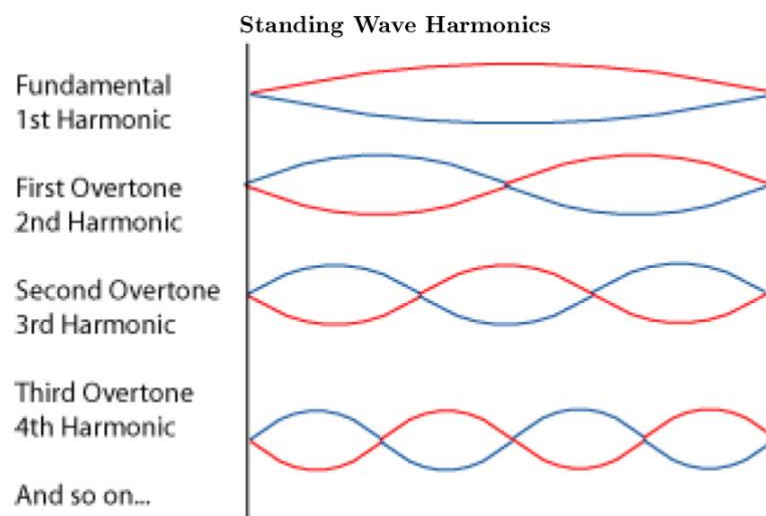


Figure 4. Relationship between Fundamental Frequency and Subsequent Harmonics [9].

During spectral analysis, it is quite valuable to apply the windowing when calculating the DFT of data to optimize for accuracy and performance. The windows are often truncated by applying various window functions such that the samples taper to zero at the start and the end of the window. Another benefit of applying windowing functions when performing spectral analysis is for avoiding

the effect of spectral leakage. The phenomenon of spectral leakage causes the magnitude information of a frequency bin to affect other bins [10]. This is often caused due to the overlapping windows causing discontinuities in the signal, as shown in Figure 5.

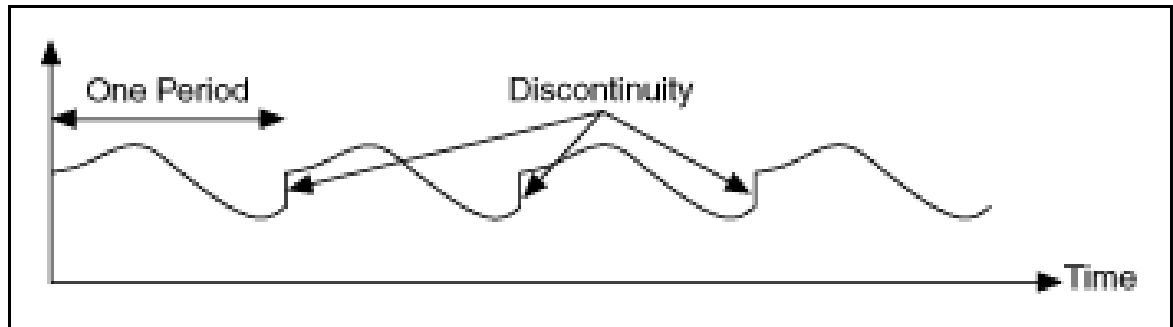


Figure 5. Discontinuities Produced when No Window Function is Applied [10].

## 2.2 YIN Algorithm

As initially established, estimating the fundamental frequency is a non-trivial subject due to the timbre of a signal. The harmonic contents play a large role in the transient changes in the time domain of the signal. Most fundamental frequency estimations fail to account for these changes. The YIN algorithm by Alain de Cheveigné and Hideki Kawhara is a robust method that improves existing implementations for fundamental frequency estimations. The three stages of the YIN algorithm are as follows:

1. Autocorrelation
2. Difference Function
3. Cumulative Mean Normalized Difference Function

The YIN algorithm is used in Darkglass' in-house bass guitar synthesizer, which produces a synthesized bass signal. Moreover, it is also implemented as a plugin in Sonic Visualizer to help validate the accuracy of the pitch correlation tool created for the research.

### 2.2.1 Autocorrelation Function

The Autocorrelation Function (ACF) is commonly used in statistics and signal processing for measuring the correlation between the signal and its time delayed variant [11]. The ACF of a periodic signal always returns a perfect correlation and the smallest time delay denotes the period of the signal [12]. The inverse of the time delay is an estimate of the frequency of the signal. Mathematical representation of the ACF function is presented in Equation (4):

Let  $s(t)$  be a periodic signal, then its ACF  $R(\tau)$  with delay  $\tau$  is:

$$R(\tau) = \frac{1}{(t_{max} - t_{min})} \int_{t_{min}}^{t_{max}} s(t)s(t + \tau)dt \quad (4)$$

The equation can be applied for a discrete signal with a window width of  $W$  and time delay  $\tau$ , hence Equation (4) can be modified; as shown in below in Equation (5):

$$r(\tau) = \sum_{j=t+1}^{t+W-\tau} x_j x_{j+\tau} \quad (5)$$

The ACF holds for estimating the fundamental frequency of pure tones; but for varying signals of electro-acoustic instruments, the ACF fails and produces errors. In Figure 6 the ACF values over a lag range for a sample signal is illustrated.

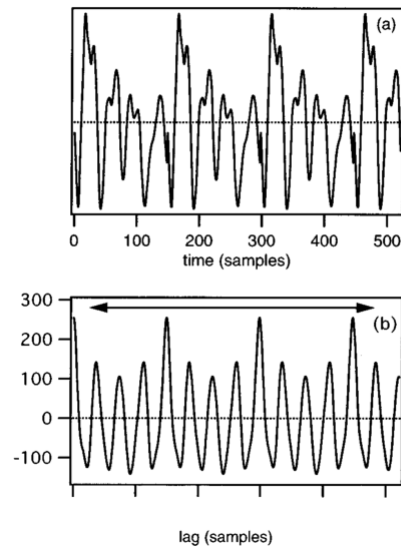


Figure 6. (a) Sample Signal. (b) Autocorrelation of the Sample Signal [13].

Listing 1 contains the python script to find the frequency of a sine wave using the ACF. The script utilizes a window width of 200 samples and generates a 5-second-long sine wave with a sampling frequency of 500.

```
def f(x):
    f_0 = 1
    return np.sin(x * np.pi * 2 * f_0)
#Generates a sine wave with frequency of 1 Hz

def ACF(f, W, t, lag):
    return np.sum(f[t : t + W] * f[lag + t : lag + t + W])

def returnACF(f, W, t, fs, bounds):
    ACFv = [ACF(f, W, t, i) for i in range(*bounds)]
    sample = np.argmax(ACFv) + bounds[0]
    return fs / sample
```

Listing 1. Python Script to Create a Sine Wave and Find its Frequency Using the ACF.

As expected, the ACF holds and returns a result of 0.9823 Hz, which can be rounded up to 1 Hz. Applying an exponentially decaying envelope to the sine signal as shown in Listing 2 proves that the ACF does not hold for decaying

signals with varying amplitude or fluctuating periods. Using the same parameters, the ACF returns a frequency estimate of 25 Hz.

```
def f(x):
    f_0 = 1
    envelope = lambda x: np.exp(-x)
    return np.sin(x * np.pi * 2 * f_0) * envelope(x)
```

Listing 2. Modified Sine Wave with a Decaying Envelope.

The sine wave with as decaying envelope is shown in Figure 7 below.

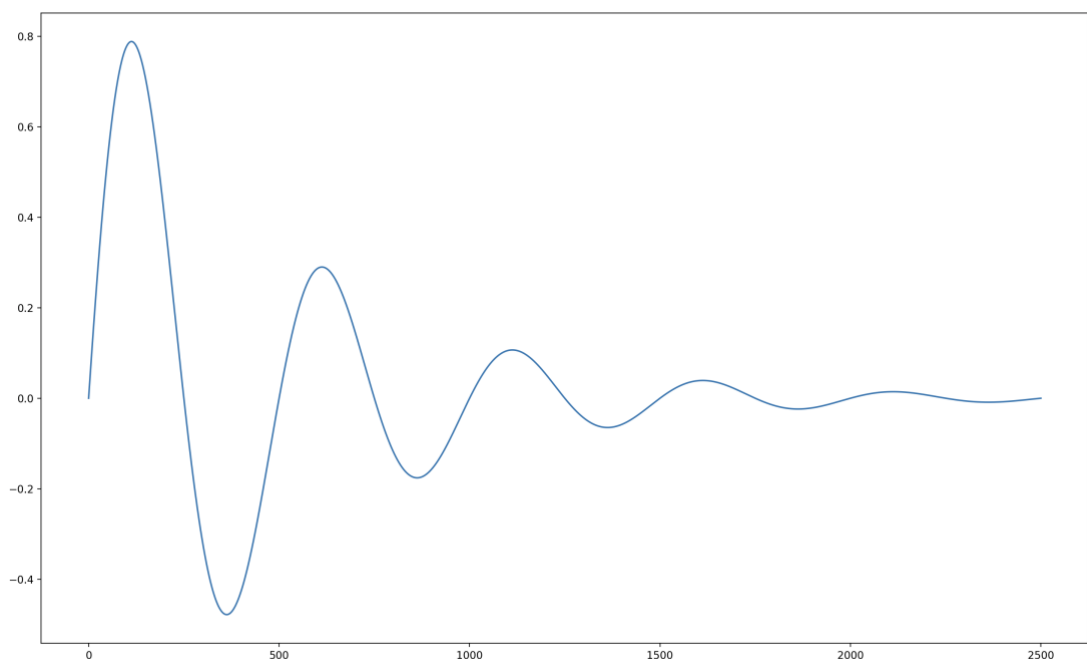


Figure 7. Sine Wave with an Exponentially Decaying Envelope.

### 2.2.2 Difference Function

For a shift invariant system, it is true that when a discrete signal  $x_t$  with period  $T$  is shifted by a time constant the output is equally shifted. Using this property, the following Equation (6) also holds true. [13.]

$$x_t - x_{t+T} = 0 \quad (6)$$

Similarly, by squaring the sums of Equation, the smallest time shift where the difference is zero gives the period of the signal. Equation (7) below describes the difference function for a discrete signal  $x_t$  using a sampling window of  $W$  and time shift  $\tau$ . [13.]

$$d_t(\tau) = \sum_{j=1}^W (x_j - x_{j+\tau}) \quad (7)$$

The difference function does not improve over the ACF in terms of estimating the fundamental frequency, but rather reduces the overall errors produced by the ACF. An elucidation for this is that the ACF is much more sensitive to amplitude. Moreover, the difference function can be described in terms of the ACF as shown by Equation (8) [13.]

$$d_t(\tau) = r_t(0) + r_{t+\tau}(0) - 2r_t(\tau) \quad (8)$$

The same is implemented in a python function as shown in Listing 3.

```
def DF(f, W, t, lag):
    return ACF(f, W, t, 0) + ACF(f, W, t + lag, 0) - (2 * ACF(f, W,
t, lag))
```

Listing 3. The Difference Function Implemented in terms of the ACF using Python.

Due to the difference function being an intermediate step of the YIN algorithm, the values of the difference function being used for estimating the fundamental frequency bears no value despite producing lower errors than the ACF. Hence, it was not tested using python.

### 2.2.3 Cumulative Mean Normalized Difference Function

The cumulative mean normalized difference function (CMNDF) is the final stage of the YIN algorithm. The presence of the 2<sup>nd</sup> harmonic causes the difference

function to produce zero lag regions, thus inducing errors [13]. The CMNDF avoids these zero lag regions and improves upon the difference function. Equation (9) below shows the CMNDF:

$$d'(\tau) = \begin{cases} 1, & \text{if } \tau = 0 \\ \frac{d_t(\tau)}{\left[\binom{1}{\tau} \sum_{j=1}^{\tau} d_t(j)\right]}, & \text{for all other } \tau \end{cases} \quad (9)$$

The function accomplishes lower errors by dividing the preceding difference function value over its average of shorter lag values. To further reduce errors, using an absolute minimum threshold for the lag values is useful since if no lag values are found, the function can default to the absolute threshold. [13.]

Using the parameters from Listing 1 and the decaying sine wave, the CMNDF implementation and F0 detection implementation in python is presented in Listing 4 below.

```
def CMNDF(f, W, t, lag):
    if lag == 0:
        return 1
    return DF(f, W, t, lag) / np.sum([DF(f, W, t, j+1) for j in
range(lag)]) * lag

def detect_pitch(f, W, t, fs, bounds, thresh = 0.1):
    CMNDF_vals = [CMNDF(f, W, t, i) for i in range(*bounds)]
    sample = None
    for i, val in enumerate(CMNDF_vals):
        if val < thresh:
            sample = i + bounds[0]
            break
    if sample is None:
        sample = np.argmin(CMNDF_vals) + bounds[0]
    return fs / sample
```

Listing 4. CMNDF and Fundamental Frequency Estimation.



Applying the detect pitch function on the decaying sine wave returns an F0 estimation of 1.002 Hz, which is 0.2% above the exact F0 of 1 Hz. Furthermore, the recommended threshold is 0.1 [13]; increasing the minimum threshold to 1 returns an estimate of 1.36 Hz, consequently increasing the error. Introducing bounds to the search range of the lag value is also beneficial to the processing time and improves the accuracy of the algorithm [13]. The knowledge pertaining to the source of the sound is also valuable. For example, in the Darkglass bass synthesizer, the frequency range of the bass guitar helps optimize the algorithm.

### 2.3 Octaver Algorithm and Model

The octaver is a signal processing effect that shifts a signal down a musical interval of an octave, which leads to the signal's frequency to be halved. The measure for the shift in musical intervals from a reference frequency is defined as a semitone [14]. For two frequencies  $f_1$  and  $f_2$ , the intervallic distance is calculated by using Equation (10):

$$\text{Semitones} = 12 * \log \left( \frac{f_2}{f_1} \right) \quad (10)$$

An octave is 12 semitones apart from a reference pitch.

Octavers are commonly used with guitars and basses to produce sub-bass frequencies. Furthermore, the octave down signal is also typically mixed with other types of effects such as distortion. Octavers are popularly used in the pedal format using analog or hybrid processing, but purely digital versions are also available and largely utilized. The digital formats provide the advantage of polyphonic processing; allowing a signal containing multiple frequency intervals to be shifted. Frequency domain-based processing is a common prerequisite for polyphonic processing. Whereas for analog octavers, the processing is monophonic, where only one frequency is processed at a time. Analog octavers often use time-domain processing.

### 2.3.1 Analog Octaver

The working principle of an octaver is to generate a square wave at half the frequency of a periodic signal. The generated square wave is then used to manipulate the input signal. A common method to manipulate the signal is by muting the signal with the sub-octave square wave. To generate the muting square wave, the peak of the signal needs to be detected and every other peak of the signal is considered as a candidate for the mute control circuit. As previously established, audio signals do not have perfect periodicity and can have varying amplitudes. This effect leads to muting in varied periods and can cause discrepancies in the resulting audio signal because of partials and their decay. Due to the confidentiality and the legality of sharing schematics of popular octavers, the block diagram of an analog octaver is only presented, as shown in Figure 8.

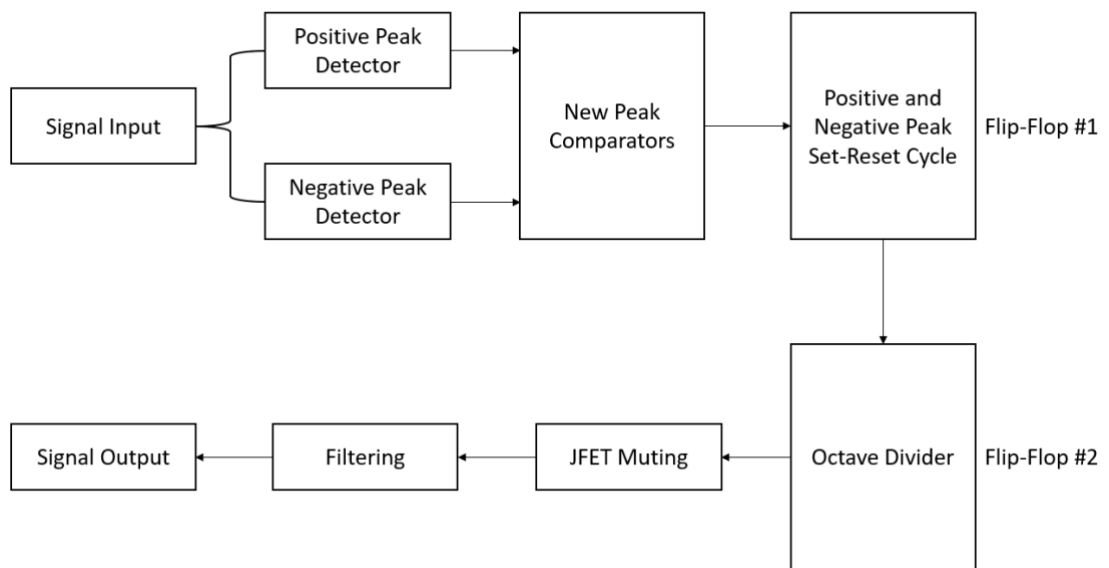


Figure 8. Analog Octaver Signal Block Diagram.

The most important section of the octaver is the generation of periodic square wave signals that alternate every second cycle of the signal. To achieve this, an analog peak detector is implemented using operational amplifiers (Op Amps). The signal is first clipped such that it converts the signal to an almost-square

wave, this is achieved using diodes and a gain stage to amplify the signal, like distortion circuits.

Second, the positive and negative half segments of the signal are individually peak detected using an op amp based peak detector. It outputs an exponentially decaying envelope of the peak. Third, the output of the signal is compared with a reference voltage to detect new peaks using op amp comparators, which produce a signal between HIGH and LOW every time the signal crosses or falls below the threshold. The comparators produce a pulse every time a new positive or negative peak is detected. The output of each stage is shown in Figure 9 for a 100 Hz sine wave signal.

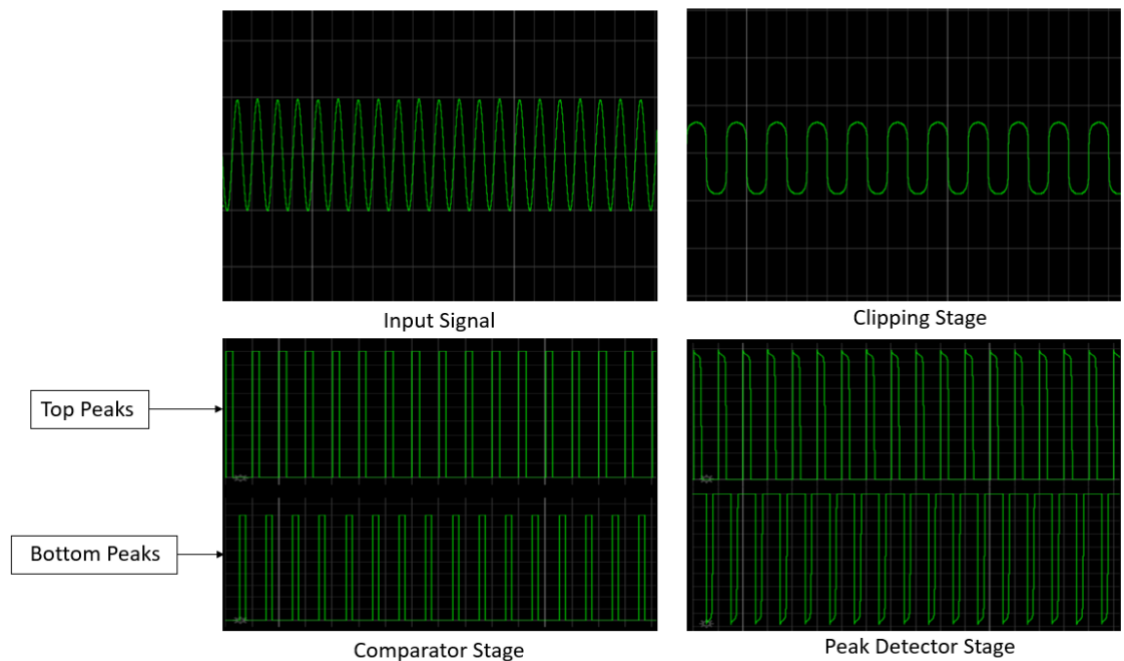
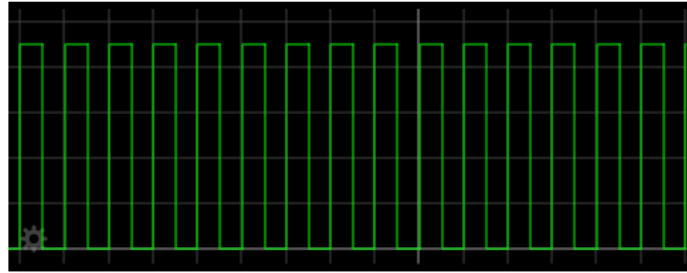
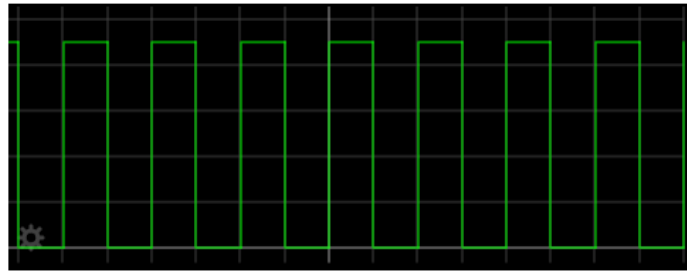


Figure 9. Stages of Square Wave Signal Generation.

Finally, the peak detected pulses are fed to two D-Flip Flop circuits. The set and reset signals are produced in a manner that the circuit sets for every new positive peak and reset for a negative peak in the first flip flop. With the second flip flop, the period is doubled by enabling set every rising edge of the previous flip flop stage. The output of the first and second flip flop stages are presented in Figure 10 below.



Flip Flop #1: Set-Reset Cycle Generator



Flip Flop #2: Period Doubler

Figure 10. Flip Flop Circuit Output.

The output of the period doubling flip flop circuit is used as a JFET muting circuit's control signal; the JFET conducts to ground when the signal goes LOW and disconnects the ground on HIGH. Abrupt muting in the signal causes high frequency contents and harmonics which degrade the audio quality. A low pass filter is implemented to remove these unnecessary frequencies and further acts as a mask for minor glitches that may occur. The final octave signal is achieved at this stage of the processing. It is quite common to add a signal mixer between the processed sub frequencies and the original clean signal for more control over the sound of the octaver. The final output signal of the octaver is shown Figure 11 below.



Figure 11. Octaver Output.

The analog octaver can be modelled using digital signal processing by emulating the analog component logic. Furthermore, the octaver's errors can be reduced by implementing the ACF, or a hybrid of the analog and ACF. Although the ACF reduces overall occurrences of errors, it is still prone to producing octave errors and inconsistencies in special cases. Lastly, methods other than muting can be implemented to produce the octaver's effect, such as flipping the phase of the signal. The output of an analog octaver model implemented using Darkglass' DSP methods in python is presented in Figure 12 below. The test signal is a 100 Hz sine wave.

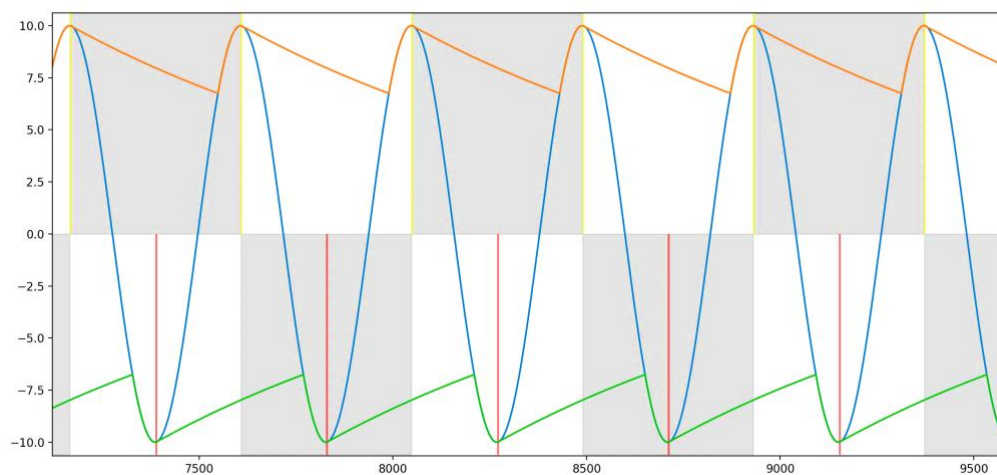


Figure 12. Darkglass' Digital Model of an Analog Octaver. Orange and Green represent the envelope. Yellow and Red are the new Peak Detections.

Typically, the analog octaver implementation may produce errors in certain cases. The two significant errors are the following:

1. Octave Error
2. Phase Error

### 2.3.2 Octave Error

The octave error typically occurs during sustained audio signals. During the decay of the audio signal, the fundamental of signal may not show dominant presence throughout the duration of the decay. Other harmonics usually display more presence during decay of the envelope of the signal. In a spectral analysis point of view, the fundamental frequency is still dominant. Due to this effect, the peak detector accounts for the new harmonics hence leading to the frequency of the flip flop circuit doubling.

In Figure 13, the first two cycles of the analog flip flop circuit (vertical red marking) produce perfect muting at alternating cycles of the signal. Whereas the advancing cycles have twice the period due to the envelope containing two dominant peaks. The SUB\_ANALOG square wave contains the set-reset cycle error.

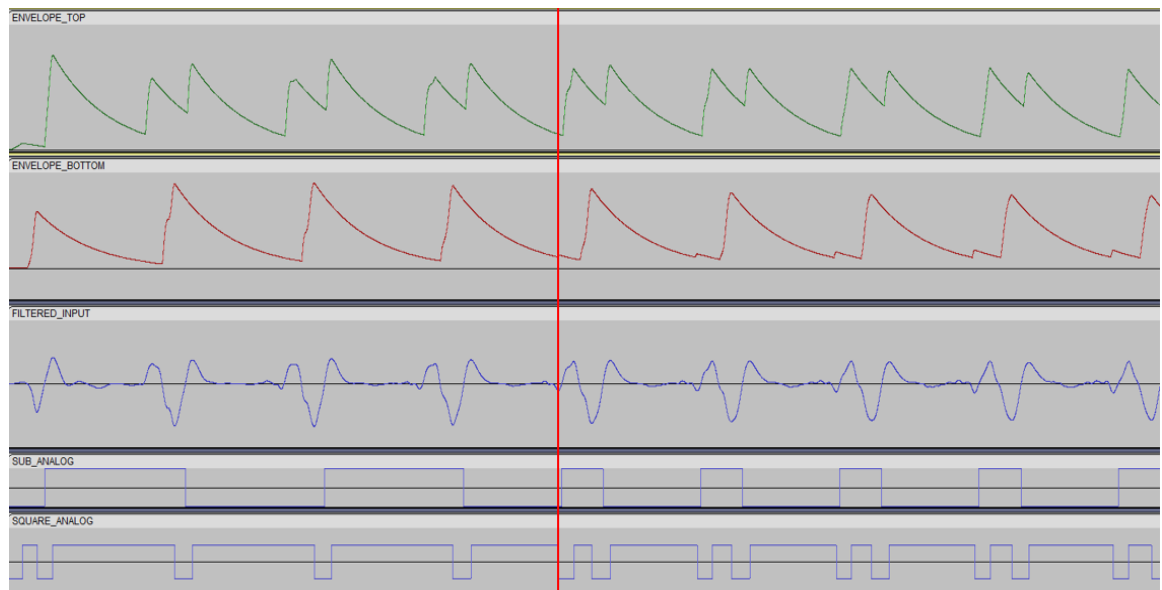


Figure 13. Octave Error. Signal After the Red Line Marks the Doubling of Period in the square wave (SUB\_ANALOG).

The octave error causes shifts in the signal's frequency. Furthermore, muting at varied cycles of the signal causes the output to shift into other frequencies; producing a slight detune effect.

### 2.3.3 Phase Error

Phase error produces subtle to very audible effects in the signal. Furthermore, it also affects the mixing of signals. The error occurs when the flip flop circuit polarity flips state or contains different periods in muting. This causes the muting to take place at points with large phase differences. In certain cases, when two signals are mixed for 180-degree phase changes, opposing phases cause destructive interference and reduce the overall loudness of the signal [15]. Moreover, the phase error worsens when the signal contains shifts above the zero region (DC offsets), since the periodic detection of zero-crossings in the ACF model no longer holds consistency. Figure 14 marks the signal contents changing over time and changes in the SQUARE\_ANALOG and SUB\_ANALOG signals, consequently causing a phase error.

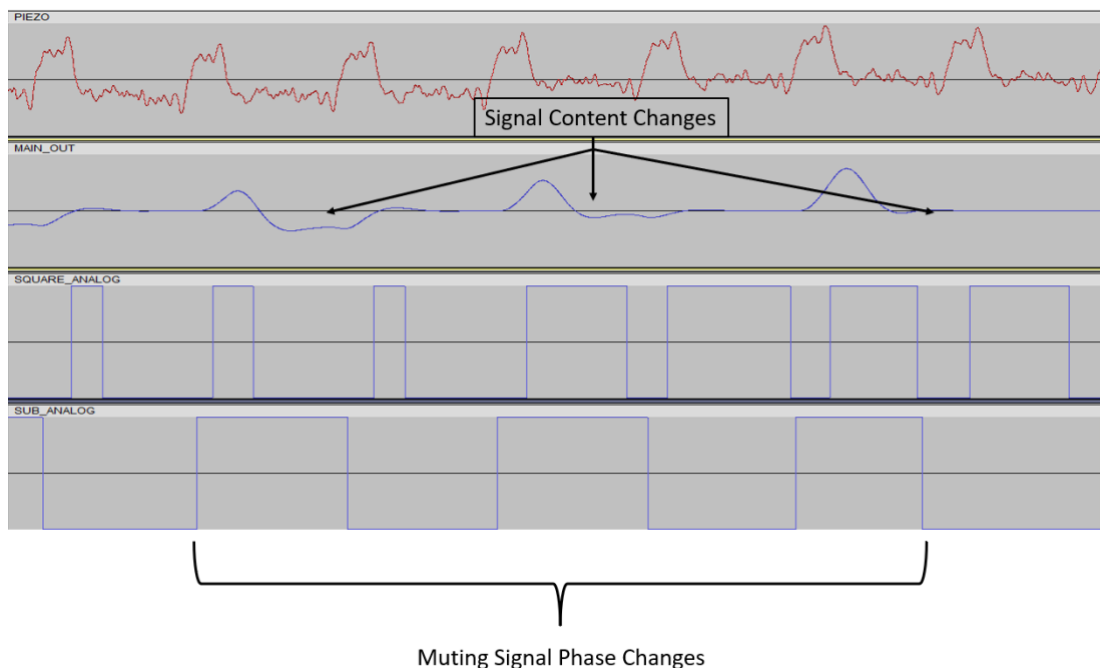


Figure 14. Phase Error. The Signal Contents Change and Sub Analog Changes Phase.

## 2.4 Pickup Fundamentals

In acoustic instruments, the sound of the strings is amplified through the sound hole [16]. Development of solid bodies for acoustic instruments (such as guitars and basses) simultaneously led to the electrification of them. The sound of the instrument is captured through a transducer called pickup [17]. Pickups are essential for a guitar and bass as they predominantly define the overall sonic qualities of the instrument and facilitate the usage of effects to amplify and alter the sound of the instrument.

The placement of the pickup plays a large role in the overall dynamic range and frequency response of the signal. Moreover, the timbre also greatly varies [18]. Common placements of the pickup in a guitar are the neck, middle, and bridge positions. The output voltage increases closer to the neck of the guitar while the presence of higher order harmonics decreases [18]. Pickups are classified into two types:

1. Magnetic Pickups
2. Piezo-electric Pickups

The primary difference between the two types of pickups is based on their construct and output.

### 2.4.1 Magnetic Pickups

Magnetic pickups are constructed by winding wire to form a coil around a permanent magnet [19]. The working principle of a magnetic pickup is based on the Faraday's Law of Induction, which states that changes in the magnetic field induces current into the coil [20]. Figure 15 shows the inner construction of the pickup.





Figure 15. Construction of Magnetic Coil Pickups [19].

Electrically, an ideal magnetic pickup can be simplified to an LCR circuit. Furthermore, the pickup can also be treated as an AC voltage source to the LCR circuit [21]. In Figure 16, the equivalent circuit can be observed.

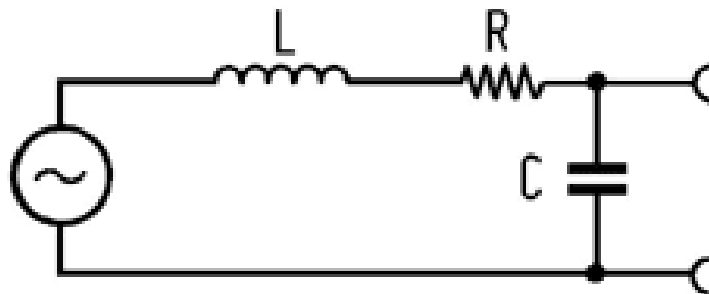


Figure 16. Electrical Equivalent Circuit of a Pickup [21].

The ideal pickup circuit behaves as a resonance circuit; therefore, a single frequency contains the highest amplitude. The resistive and capacitive elements form a low-pass filter, leading to frequencies above the cut-off frequency having lower amplitudes [21]. The frequency response and resonance of an ideal pickup circuit is presented in Figure 17.

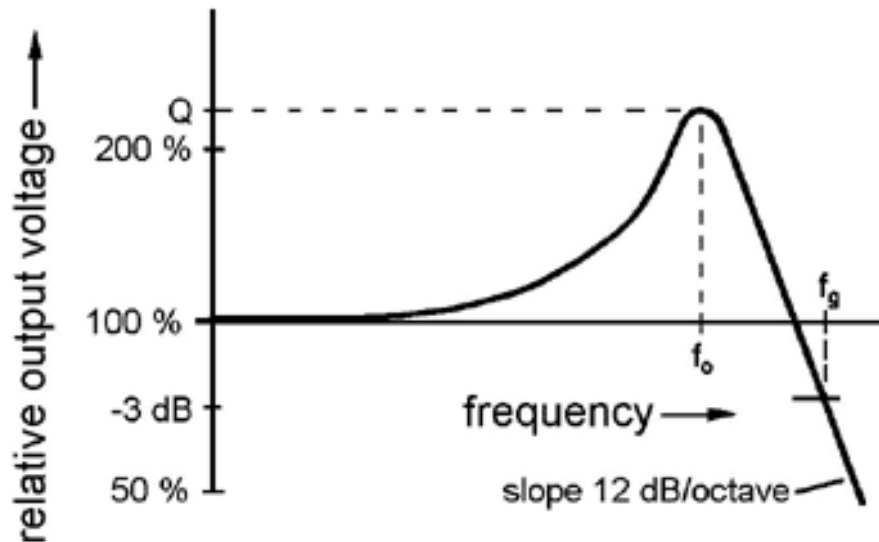


Figure 17. Frequency Response and Resonance of a Magnetic Pickup [21].

Due to external factors such as capacitance in cables, control potentiometers, and eddy currents in conductive parts, the overall frequency response and the resonance frequency can change [21]. Furthermore, the pickup only detects changes through the magnetic field, hence it is independent of the displacement of the string and much more sensitive to the velocity of the strings. Strings also tend to produce lesser displacement at higher frequencies. The two effects essentially cancel each other out. [19.]

In addition, the displacement of a string along a particular axis and the distance between the string and the pickup ( $d_0$ ) produces varied harmonic contents in the signal. The axis definitions and distances are shown in Figure 18. The output signal produces non-linear characteristics when the distance ( $d_0$ ) between the string and the pickup is small, whereas the output has a much more linear nature when placed farther away. The displacement along the Z-axis contains most of the fundamental frequency and movement along the Y-axis produces higher order harmonics. Moreover, the distortion also worsens for Y-axis motion. [22.]

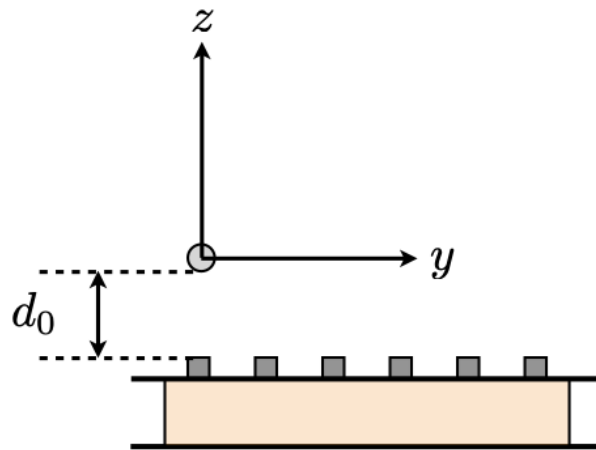


Figure 18. Axis Definitions [22].

The signal output and temporal evolutions of the string excitation along the Z-axis and Y-axis are presented in Figure 19.

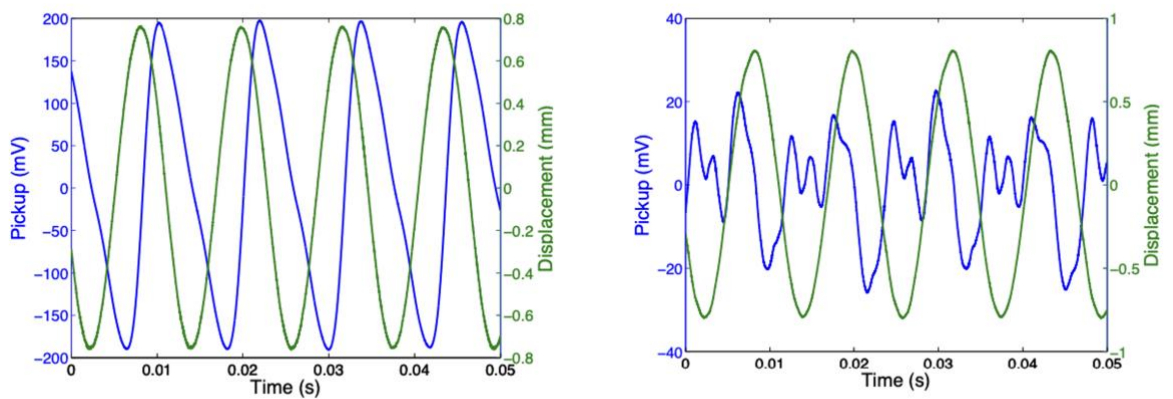


Figure 19. Left: Output for Displacement Along Z-axis. Right: Output for Displacement along Y-Axis [22].

Magnetic pickups have two common configurations:

1. Single Coil
2. Humbucker

The single coil configuration consists of an array of permanent magnets with a single coil. Single coils are susceptible to external magnetic fields, therefore, produce a hum-like sound and buzz. [23.] Additionally, single coil pickups contain a considerable amount of high frequency contents, commonly characterized as “bright” sounding [24]. Figure 20 illustrates a single coil pickup.

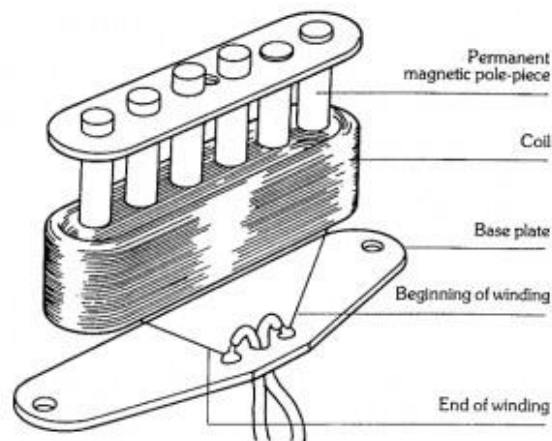


Figure 20. Single Coil Pickup [25].

Humbuckers inherently overcome the susceptibility of external magnetic fields. Humbucking pickups are constructed using two coils around magnetic pole pieces with alternating polarity. The output of the two coils are 180 degrees out of phase with each other, hence eliminate noise when mixed. [24.] The overall output of the humbucker configuration is much greater than a single coil. Figure 21 depicts the humbucking effect.

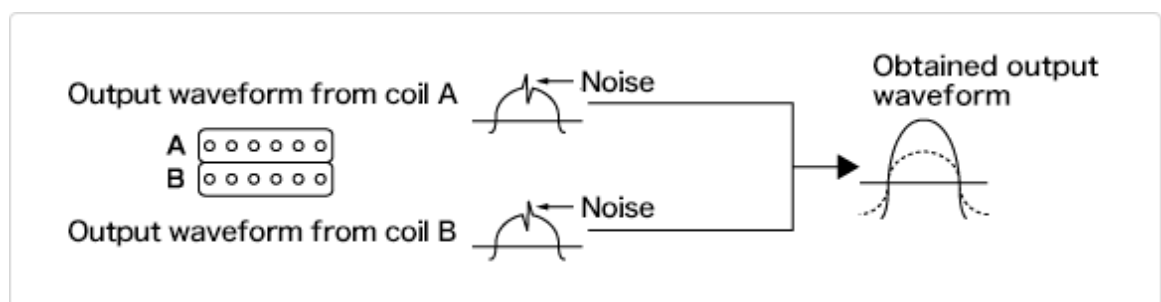


Figure 21. Humbucking Effect [19].

The internal construction of a humbucker is equivalent to two single coil pickups connected in series. Figure 22 shows the humbucker pickup construction.

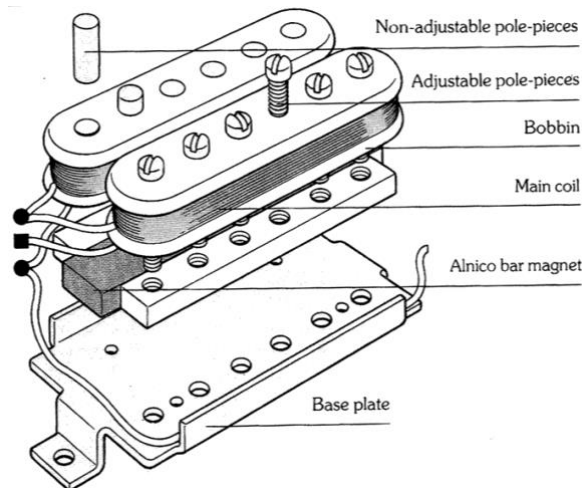


Figure 22. Humbucker Pickup [26].

Humbucker coils are also connected in several ways. Namely, the split-coil configuration and series-parallel configuration. The split-coil configuration effectively shorts one side of the humbucker coils to ground, rendering it to operate like a single coil pickup. This produces high frequency contents, but it also introduces the drawbacks of excess noise and lowers the overall output. The series-parallel configuration maintains the humbucking nature of the pickup, while preserving the high frequency contents. [24.] The two configurations for connecting a humbucker pickup are presented in Figure 23.

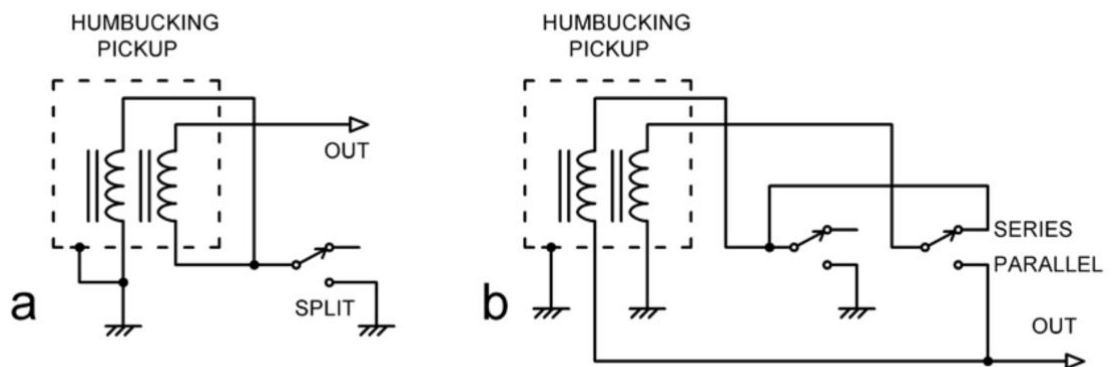


Figure 23. Humbucker Configurations. (a) Split Coil Configuration. (b) Series-Parallel Configuration [24].

### 2.4.2 Piezo-electric Pickups

Piezo-electric pickups operate based on the principle of piezo-electricity. The effect is produced when a crystal is subject to mechanical stress (or pressure), thus causing a voltage drop across. Crystals are molecular structures that are arranged in an orderly manner. The piezo-electric effect is widely used in medical ultrasound equipment, microphones, and time-keeping devices [27].

Similarly, in pickups, the crystal is placed in the bridge of the guitar. The pressure of the string motion across the bridge produces a current at the frequency of the string's vibration. Basic internal structure of the piezo-electric pickup is illustrated in Figure 24.

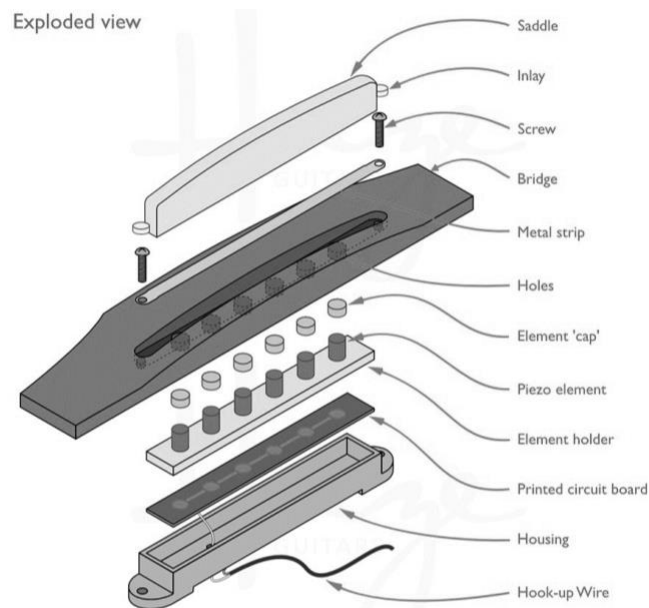


Figure 24. Basic Piezo-electric Pickup Structure [28].

The voltage output across the crystal is typically very low for direct usage, therefore, requires a pre-amplifier. Furthermore, the output of the piezo-electric pickup is non-linear. The output voltage does not scale very well with the dynamics of the string, hence causing distortion or abundance of quietness to occur in the audio. To prevent this, the signal is compressed in the pre-amp to maintain uniform signal levels throughout. [29.] In addition, piezo-electric pickups

have high output impedance and capacitance. Pre-amplifiers are commonly designed using discrete JFET amplifier circuits or JFET based Op Amp ICs (Integrated Circuits), since JFETs offer high input impedance and low output impedance. [30.] This prevents excessive loading in other stages. Component selection is quite imperative as pre-amplifiers need to maintain low Total Harmonic Distortion and Noise (THD+N). A basic piezo-electric pre-amplifier is presented in Figure 25.

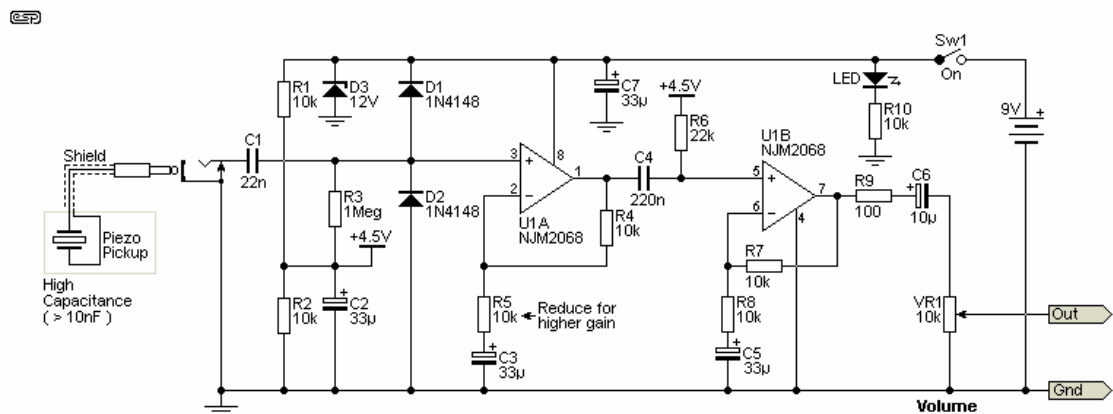


Figure 25. Basic Piezo-electric Pre-amplifier [30].

### 3 Testing Methods

The ideal behavior of the pickup is to produce the least number of errors with the octaver models and the YIN algorithm-based bass guitar synthesizer. It is necessary to understand the harmonic contents of the signals produced by the two pickups and compare the types of errors produced. The subsequent section covers the testing methods for the pickups and testing requirements.

#### 3.1 Debugging Pre-Amplifier and Bass Modifications

To test the piezo and humbucker pickups, a generic bass guitar was modified to contain both pickups. Data for testing both the pickups were routed through a debugging pre-amplifier Printed Circuit Board (PCB); designed using Altium Designer ECAD software. The objectives for the debugging pre-amplifier were to

provide an ease of accessing audio signals from both pickups and ensuring low noise and distortion. The bass guitar is pictured in Figure 26.



Figure 26. Bass Guitar.

The piezo pickup under test was placed in the bridge of the guitar and had a discrete pre-amplifier. Furthermore, the humbuckers were connected in a split-coil configuration with volume controls for each coil. The debugging PCB block diagram is shown in Figure 27.

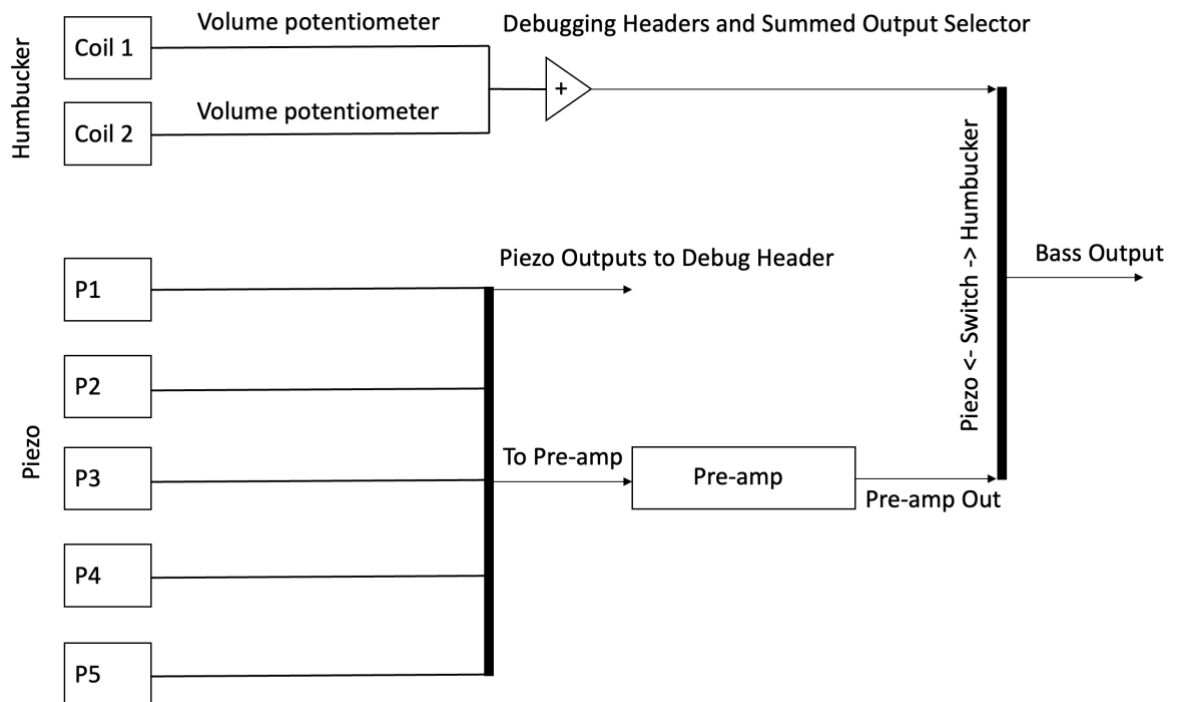


Figure 27. Debugging Pre-Amplifier Block Diagram.



Due to the piezo pickup's location being on the bridge, the original bridge on the bass was replaced. The string tension and the overall scale length of the guitar had to be considered during the replacement, since the consequences of inadequate string tension would lead to difficulty in maintaining intonation and tuning. To achieve this, the original placement and string positions on the saddles were marked on the bass. Furthermore, the original cavity for the electronics on the bass guitar was deemed to be insufficient for the addition of a PCB and the Fishman pre-amp. Therefore, the cavity was expanded. Similarly, an additional cavity was made to route the leads from the piezo bridge pickup. Figure 28 contains the bridge position markings and the cavities.

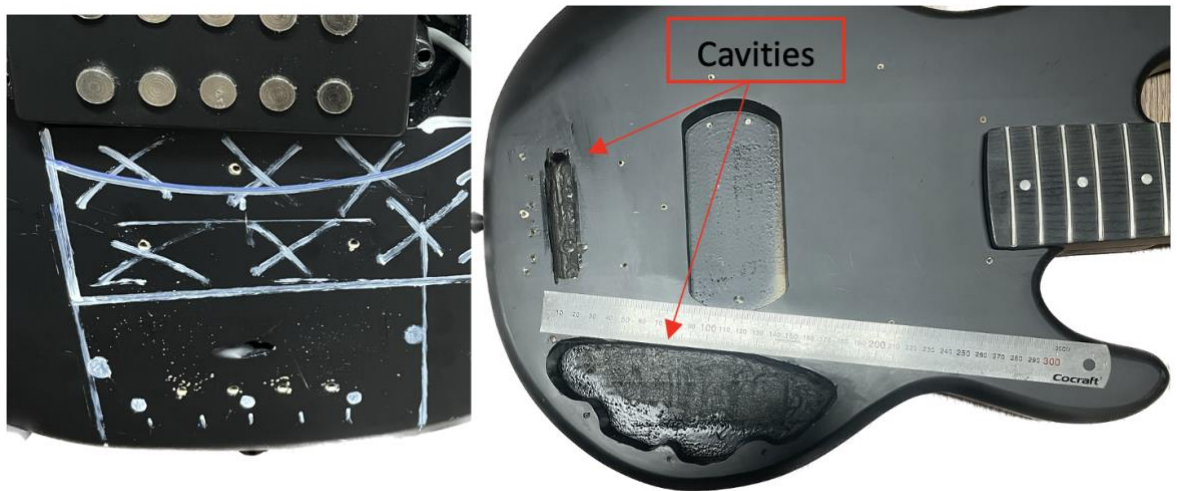


Figure 28. Left: Bridge Position Markings. Right: Expanded Cavities.

Subsequently, the electronics for debugging PCB were designed. The PCB was designed to ensure it was modular, consequently making debugging easier if issues were to occur due to design flaws. To accomplish modularity, Surface Mount Device (SMD) pads were utilized extensively.

Taking power supply noise into consideration was vital to the overall design of the PCB. Most generic power DC (Direct Current) supplies often contain a rectifier, or a regulator based designed. While they both have benefits, the most common susceptibility is RF (Radio Frequency) and ground noise. Although RF frequencies are imperceptible by human ears, ground noise produces a buzz like

effect at 50 Hz. For audio signals, this causes considerable disturbances in a signal. The effects of bad power supply sources can be mitigated by using proper precautions such as shielding, filtering, and grounding [31]. Figure 29 presents the power supply design schematics.

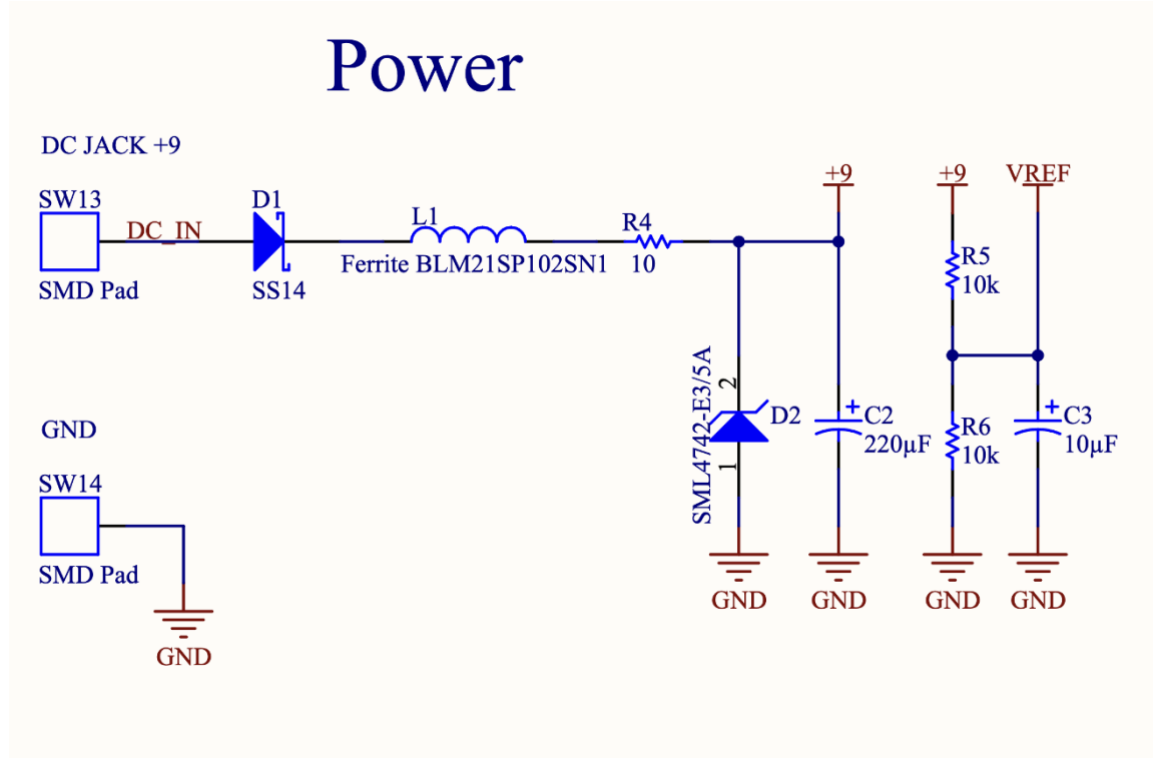


Figure 29. Power Supply Design Schematics.

The input DC voltage was connected to a center negative DC barrel power jack using wire leads; consequently, connecting the tip to ground and the sleeve to the positive supply voltage. D1 in the schematics is a Schottky diode to prevent reverse polarity connections, thus preventing accidental usage of negative supply voltage. Furthermore, ferrite bead L1 was implemented to filter power supply voltage. The specifications of the ferrite bead (Appendix 1) were chosen such that it offers low DC resistance, current rating, and a high resonant frequency. Then, the R4 and C2 form a low pass filter. R4 also acts as an overcurrent protection when excess current is drawn by the ICs, and C2 is a bypass capacitor that stores current and provides it when demanded. The cut-off frequency for the low pass filter is determined using Equation (11) and subsequently, the cut-off

frequency is calculated. The frequency response of the circuit is presented in Figure 30.

$$F_c = \frac{1}{2\pi RC} \quad (11)$$

$$F_c = \frac{1}{2\pi * 10 * 220 * 10^{-6}} \rightarrow F_c = 72.3 \text{ Hz}$$

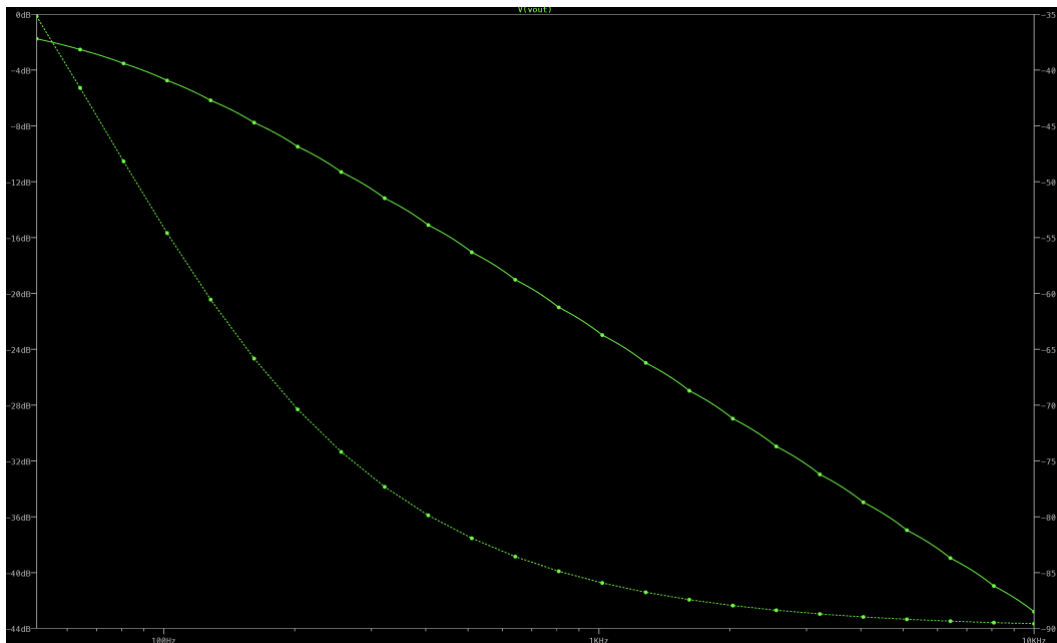


Figure 30. Power Supply RC Low Pass Filter Response.

Lastly, Zener diode D2 prevents shorts circuits to ground. Since active components require a bias point, the bias voltage (VREF) is generated by the voltage divider formed by R5 and R6; and C3 is a bypass capacitor. The output of the voltage divider is calculated using Equation (12).

$$V_{out} = V \left( \frac{R_2}{R_1 + R_2} \right) \quad (12)$$

$$V_{out} = 9 * \left( \frac{10 * 10^3}{10 * 10^3 + 10 * 10^3} \right) = 9 * \left( \frac{1}{2} \right) \rightarrow V_{out} = 4.5V$$

Following the power supply design, the humbucker input stage was designed. The primary consideration was high impedance of the pickup and the volume potentiometers for the individual coils. An ideal op amp in theory can handle the high input impedance without excessive loading, but this does not imply the same in practice. The utilization of Bipolar Junction Transistors based op amps in high impedance applications comes with a noise penalty and possibility of loading, which degrades the quality of the signal or drives the op amp to non-linearity. Furthermore, JFET based op amps are typically known for their high input impedance and low output impedance; therefore, a much more suitable option. [24.] The TL072C op amp is a cost effective and readily available JFET op amp that satisfies the needs for this application. Figure 31 illustrates the humbucker input circuit schematics.

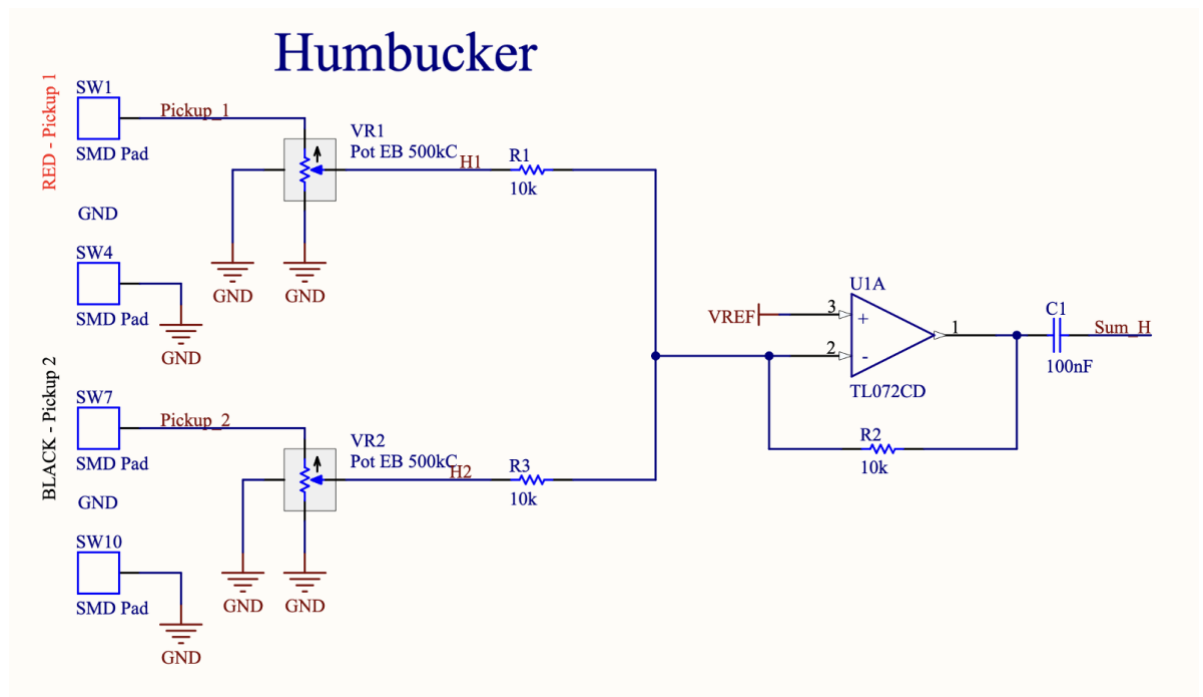


Figure 31. Humbucker Pickup Circuit Schematics.

The two coil leads containing signals were connected to the volume potentiometers. Ideally, the volume potentiometer should be a logarithmic taper.

But due to large lead times and costs, an anti-log taper potentiometer was connected in reverse to approximate a logarithmic taper. The potentiometer values were chosen based on the original electronics of the bass guitar.

R1 and R3 are the input resistors of the summing amplifier formed by U1A. The values of R1 and R3 determine the weights of the sum. The feedback resistor R2 determines the overall gain of the circuit. U1A is biased to VREF since the op amp is used in a single supply mode. For Alternating Current (AC) applications using single supply, the bias values determine the device operating voltage. In this application, the signal is DC shifted by 4.5 volts, thus initiating the signal to swing between 9 and 0 volts effectively. C1 blocks DC shifts to prevent the signal from distorting due to clipping if other additional shifts are present.

The piezo pickups did not require additional circuitry as the Fishman piezo pre-amp was adequate. SMD pads were placed for each individual output of the piezo and the summed output from the pre-amp. Figure 32 shows the piezo pickup schematics.

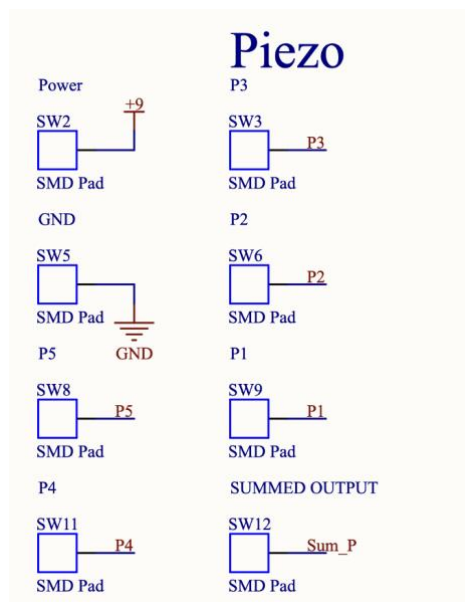


Figure 32. Piezo Pickup Schematics.

Finally, the outputs of the pre-amp and the summing amplifier were fed to a Single Pole-Double Throw switch, to select between the two pickup types. Furthermore,

the signals from the pickups were routed to debugging headers. Figure 33 contains the output circuit schematics.

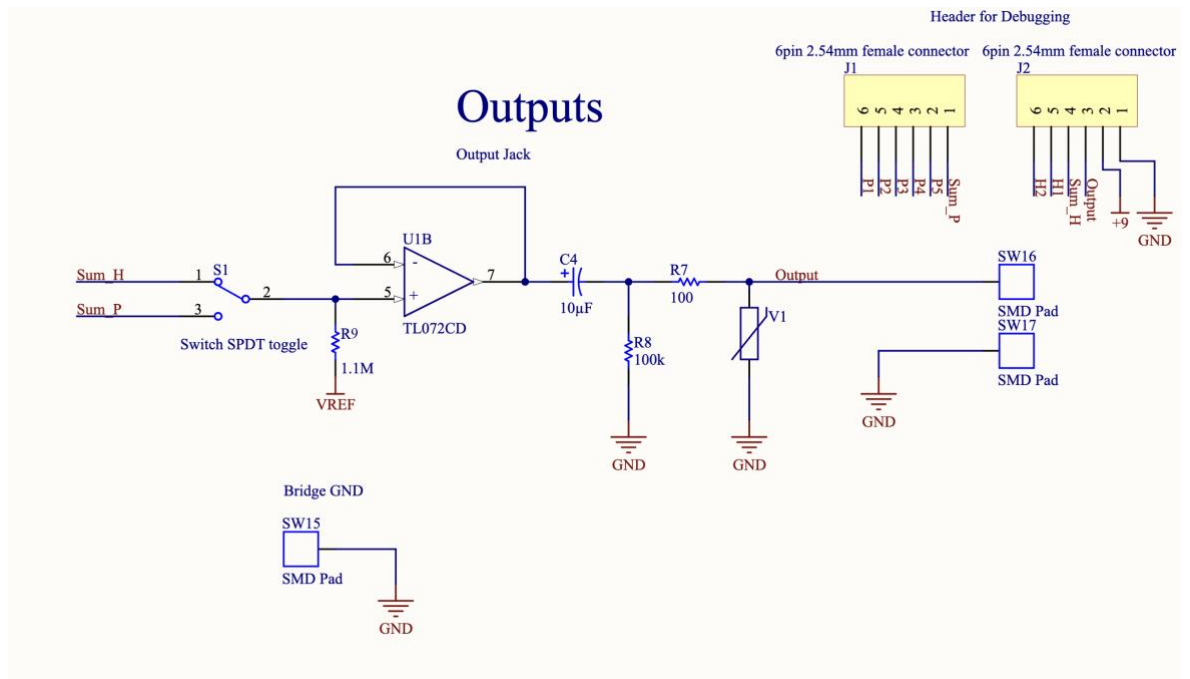


Figure 33. Output Circuit Schematics.

The output circuit contains an op amp buffer formed by U1B. The signal output from the selector switch is biased to VREF to prevent clipping. The output capacitor C4 forms a high pass filter with R8 to remove DC offsets at the output. The cut-off frequency of the filter is calculated using Equation (11) and the frequency response is depicted in Figure 34.

$$F_c = \frac{1}{2\pi * 10 * 10^{-6} * 100 * 10^3} \rightarrow F_c = 0.159 \text{ Hz}$$

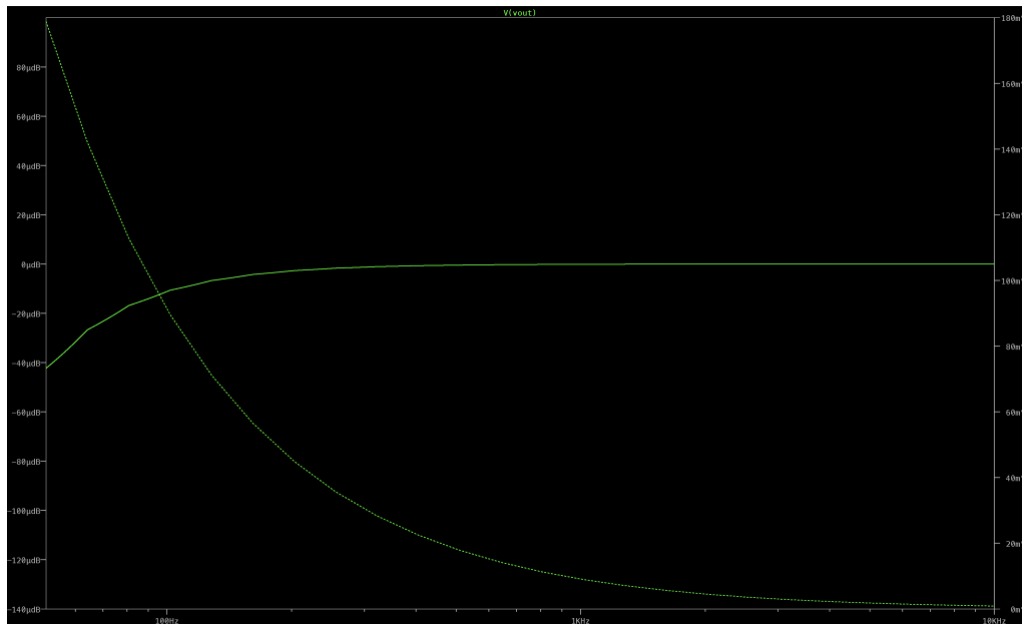


Figure 34. Output RC High Pass Filter Frequency Response.

R7 in the output circuit fixes the output impedance to 100 Ohms. A low output impedance aids in reducing the low pass filter effect due to excessive capacitive load in the input of the next device in the signal chain. Lastly, varistor V1 is implemented to protect the output circuit from transient voltage changes. Varistors are components that have varying resistances based on the voltage supplied across them [32].

The PCB was designed using a board shaped based on the cavity on the bass. A template was traced using graphic designing tools and imported to Altium. The PCB's primary goal was to have ease of accessibility for any necessary modifications. Since the total component count was low, a two-layer board was considered satisfactory. The SMD pads were placed and routed on the Bottom Layer of the PCB, whereas the components were routed on the Top Layer. Additionally, mounting holes were placed on the board for assembling the Fishman piezo pre-amp. In Figure 35 the Top and Bottom Layers are shown.

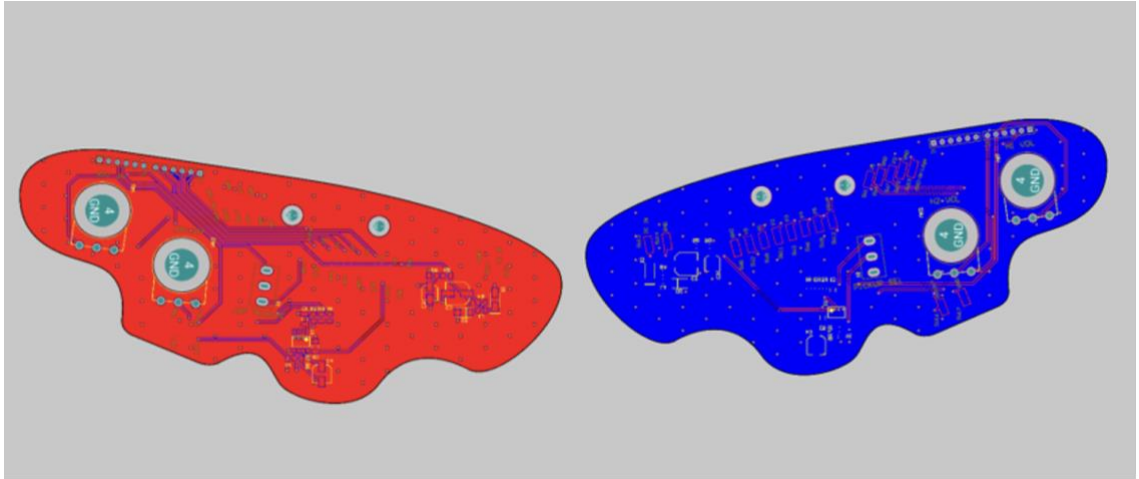


Figure 35. PCB Layout. Left: Top Layer. Right: Bottom Layer.

To hold all the cables and PCB in the cavity, a 3D printed panel was made. The 3D printed panel was insulated with copper tape to introduce shielding from RF noise. Figure 36 and Figure 37 below contain the fully assembled PCB and bass guitar.

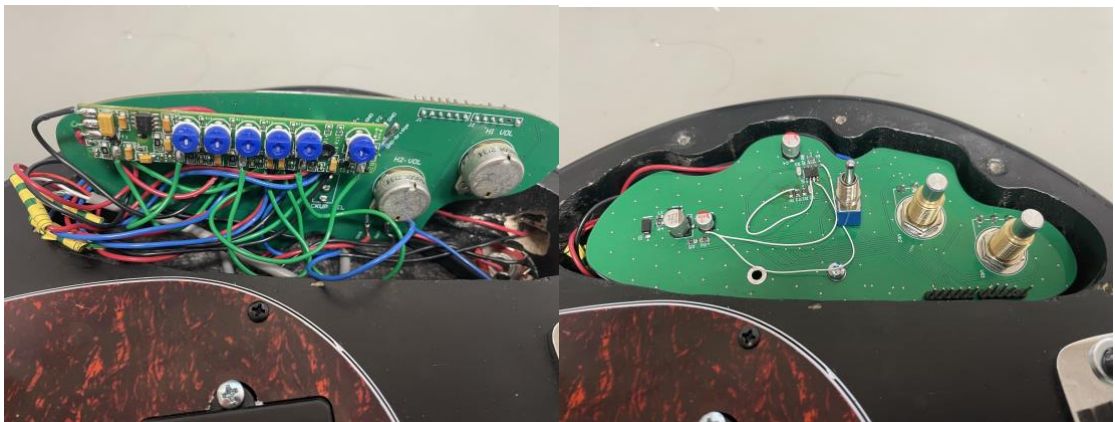


Figure 36. Assembled PCB. Left: Bottom Layer. Right: Top Layer.





Figure 37. Modified Bass and 3D Printed Panel.

### 3.2 Test Data and Considerations

The test data for the correlation process were audio clips recorded at 48 kHz sampling frequency using both pickups simultaneously. Using the audio recordings, the correlation data was gathered using the python script. The fundamental frequency of the clean and the processed signal are estimated using python, and the data is correlated. Three octaver versions were tested: floating-point processing, integer processing, and a release version. Due to the synthesizer being in development, an early test candidate of the virtual instrument was tested. The main points of consideration from the correlation are as follows:

1. Tracking Stability.
2. Pitch Deviations.
3. Settling Time.
4. Error Conditions.
5. Spectral Data.

The tracking stability is defined as the overall ability of the algorithm to maintain the F0 estimates throughout the duration of the audio. The F0 estimates are made using python signal processing tool kit (PYSPTK). Good algorithm behavior indicates no pitch errors in the algorithm functionality. Furthermore, pitch

6deviations may occur at the initial transient of the audio clip – these cases are ignored as various factors that are difficult to control may cause the deviation.

The settling time outlines the time the algorithm requires to return to a stable value in case a deviation occurs. Ideal behavior of the algorithm would result in instantaneous recovery. However, in practice, this may not be the case, therefore settling times of the less than 50 ms (milliseconds) or less than the total duration of the audio segment is deemed satisfactory. Moreover, the setting time may not be difficult to gauge since minor pitch deviations will occur. Therefore, only values out of bounds were considered.

Although most error conditions are known and flagged, new errors may arise due to the nature of the pickups. The errors are categorized and used as the main factor for the pickup selection. The least number of errors produced by a specific pickup would be the straightforward choice.

Lastly, the spectral data of each pickup is analyzed to assess the harmonic content differences between the two pickup types. Spectral analysis of the pickups may provide a broader understanding of the errors and additional measures can be implemented or developed to overcome them. It is quite vital to note that the errors may not be directly perceivable by the end-user, but it may cause disruptions in other aspects of the algorithm and usage. Therefore, all errors cases are considered.

### 3.3 Python Testing Script and Sonic Visualizer

Using python, the audio data from using the two pickups were correlated. The script intends to detect error cases and flag them appropriately. Furthermore, it provides a graphical representation of transient changes in the detection parameters. The data flow for testing is shown in Figure 38.

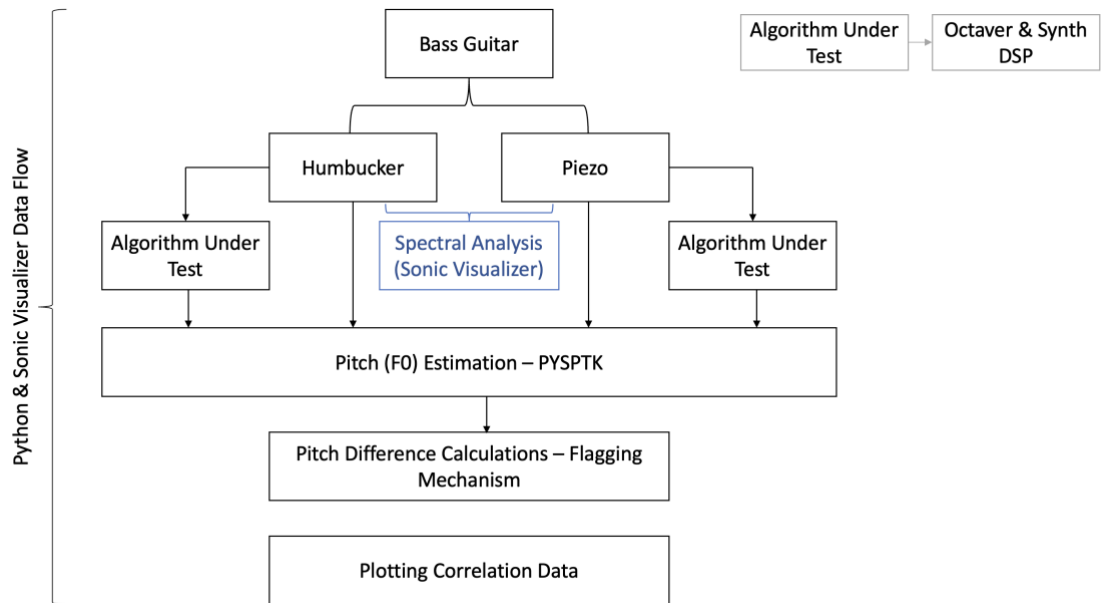


Figure 38. Testing Data Flow.

The python testing script utilized various libraries – primary operations were carried out using Python Signal Processing Tool Kit (PYSPTK), Librosa, Matplot and Numpy libraries. Using object-oriented programming, the correlation tools were written in the class Analyzer (Appendix 2). A diverse set of tools were chosen since the generic tools inherently cause difficulties in interpreting the data. The software classes in the script use a set of fixed parameters such as the following:

1. Window size: Averaging and Processing.
2. Threshold: For Gating Unvoiced Segments of the Signal.
3. Hop Size: F0 Detection and Processing.
4. Tolerance: Overall Tolerance of the Error Detection

The audio signal is loaded using Librosa. A sampling averaging function was implemented for large audio data for the correlation utilities. The averaging function calculates the average of the samples of window width  $W$ . Although this reduces the overall processing time, using large window sizes causes aliasing, thus disrupting the correlation. The averaging function was implemented as a redundancy step for preventing excessive processing time for development.

Various optimization steps were implemented in other functions. Subsequently, the audio data is batch processed using the octaver's DSP. The synthesizer uses a post-processed rendered sample.

The PYSPTK library's SWIPE fundamental frequency (F0) estimation algorithm was the core tool employed. The function calculates the F0 estimate over a fixed window length with hop size  $n$ . Clean and processed signal's F0 estimates generally showed an abundance of noise in the estimates at the silent segments of the signal. Furthermore, the estimates had large F0 jumps when a wide search range was utilized. To overcome this, the search range was limited from 10 Hz to 600 Hz, as the bass guitar's highest note is at 523.25 Hz (C5). Similarly, a noise gate was implemented on the F0 data using the Librosa library. A value from the estimate is only accepted when the signal crosses a fixed threshold. Using binary masks, the unvoiced segments are rendered to null and multiplied with the F0 data array. The accuracy of the F0 estimates were also checked for errors using Sonic Visualizer's P-YIN plugin. Listing 5 contains the binary mask calculation and noise gating of the F0 array.

```

if self.threshold is not None:
    #Compute the non-silent intervals (i.e., the intervals
    where the signal is above a certain threshold)
    non_silent_intervals = librosa.effects.split(data,
    top_db=self.threshold)

    #Create a binary mask to nullify the silent parts
    mask = np.zeros_like(data, dtype=bool)
    for interval in non_silent_intervals:
        start = interval[0]
        end = interval[1]
        mask[start:end] = True

    #Applying the mask
    f = f * mask
else: mask = None

```

Listing 5. Binary Mask Generation and Noise Gating.

The octaver models generate debugging data and the script implements functions to detect changes in square wave data. The main processing is implemented for the square waves generated from the set-reset cycle of the octave divider. Essentially, the fundamental frequency can also be estimated using this data because of the readily available wavelengths. Moreover, the F0 changes help detect octave and phase errors are easily viable from the correlation of the set-reset cycle. The data is processed using the function presented in Listing 6.

```

ctr = 0
f = []
store = 0
for j in range(1, len(data)):
    if (data[j] * data[j-1] < 0):
        if ctr > 1:
            store = (fs/ctr)/2
        else:
            store = 0
            ctr = 0
    else:
        ctr += 1
    f.append(store)
f.append(f[len(f)-1])

```

Listing 6. Set-Reset Cycle Frequency Estimation.

The process is performed by calculating the product of the sample at index  $j$  and the preceding sample at  $j-1$ . When the product returns a negative value, it denotes a change in the polarity of the square wave. Subsequently, the value is stored, and the frequency is calculated only when the number of samples counted exceeds 1; the estimation is performed by dividing the sampling frequency against the number of samples counted before the sign changes.

The final function performs the main correlation by calculating the deviation in F0 estimates with respect to the clean signal. Flags for excessive deviation and octave errors are also set. The pitch differences between the clean and processed signal are calculated using Equation (10). Tolerance for differences is

set between 5% and 10% bounds. The flags raise a value of 1 (HIGH) when a discrepancy beyond the bounds is detected, otherwise sets a 0 (LOW). In certain cases, a value of -1 is set for ignoring the values. Below Listing 7 shows the flagging mechanism. The last value of each flag array is set to LOW or Ignore.

```

        tol = self.tolerance / 10
        #Setting a flag for unstable values and detecting octave
differences
        for i in range(0, len(semi)-1):
            if semi[i] == float('nan'):
                setFlag.append(-1)
            elif semi[i] == 0:
                setFlag.append(0)
            elif semi[i+1] - semi[i] != semi[i]:
                if semi[i] >= semi[i+1] * (1 - tol) or semi[i] <=
semi[i+1] * (1 + tol):
                    setFlag.append(1)
                else:
                    setFlag.append(0)
            #Checking for octave differences within bounds and check
if there are values over an octave
            if (1 - tol) * 12 >= semi[i] or (1 + tol) * 12 <= semi[i]
or semi[i] > 12:
                isOctave.append(0)
            else:
                isOctave.append(1)
        setFlag.append(-1) #Ignoring last value
        isOctave.append(0) #Ignoring last value

```

#### Listing 7. Flagging Mechanism.

The data from the correlation functions are plotted against time along with the clean and processed signal using the Matplot library. The function Plot (Appendix 2) implements a method for plotting certain correlation values at a time, to improve the ease of analysis. Furthermore, the data values are scaled such that information does not overlap. Legends were also implemented to enhance visibility. Figure 39 shows a sample plotting function output.

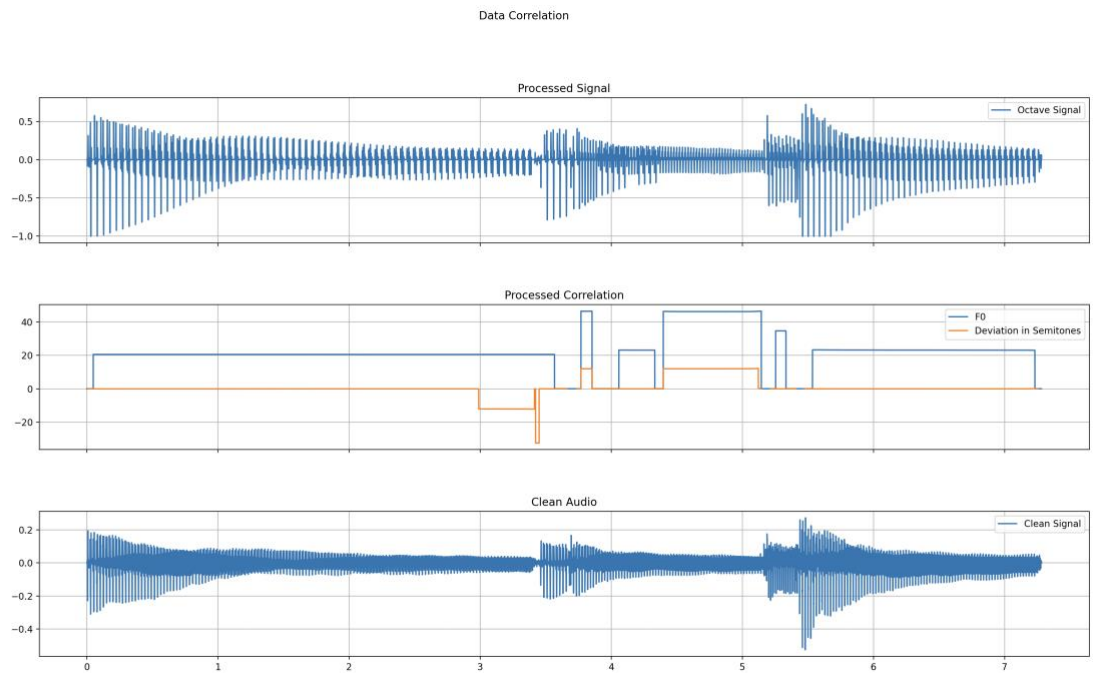


Figure 39. Sample Plotting Function Output.

Spectral analysis of the pickups was performed using Sonic Visualizer's spectrogram generator and spectrum analyzer. The clean signal was utilized for the analysis. Utilizing the spectral aids in understanding the conditions that led to errors in a pickup type. Moreover, it also helps introduce distinction between the two pickup types. The spectrogram and spectrum analyzer view of Sonic Visualizer is presented in Figure 40.

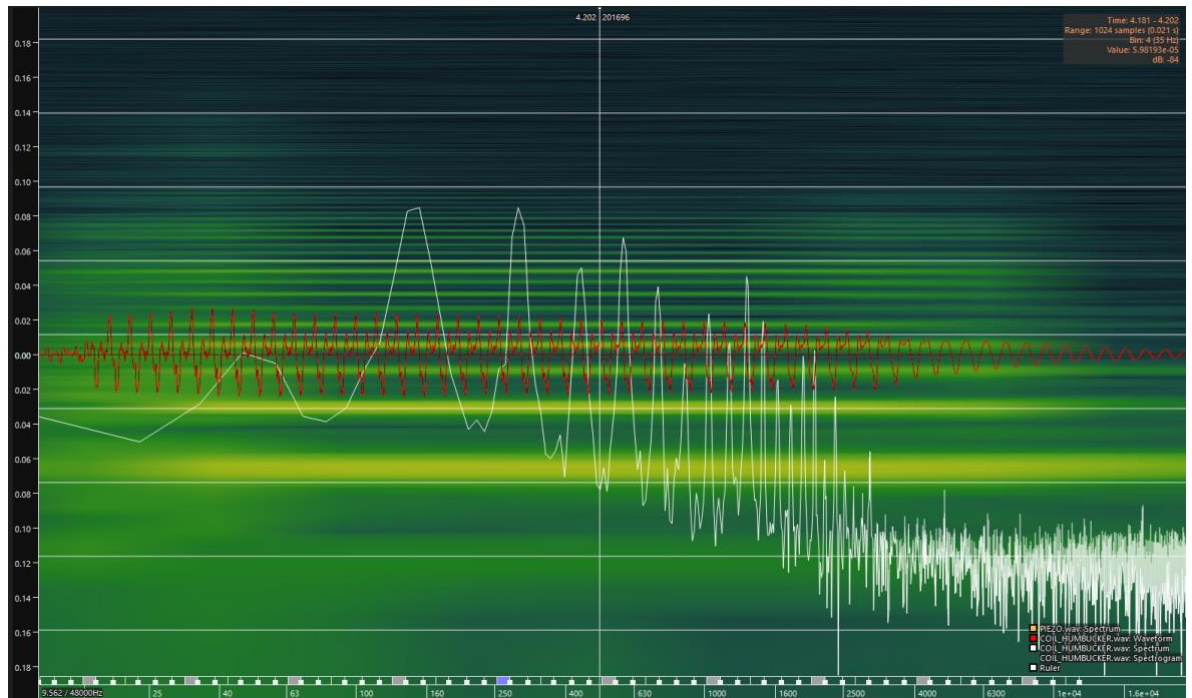


Figure 40. Spectrogram and Spectrum Analyzer View using Sonic Visualizer.

## 4 Results

The final testing results revealed that the piezo is more error prone with the Darkglass octaver and synthesizer algorithms. The test data spanned across 4 different types of audio recordings with both pickups, and the signals were subjected to the algorithms under test. Furthermore, the selection of the audio recording was based on commonly known error conditions. The correlation analysis was performed with no averaging and a hop size of 1 sample, for more resolution. A total of 108 errors were detected from both pickup and algorithm types, with a 10% tolerance on the error detection bounds. The percentage share of errors between the pickups is displayed in Table 1.

Table 1. Total Error Count and Percentages: Sorted by Pickup Type.

Pickup Type	Error Count	Percentage
Piezo	66	61%
Humbucker	42	39%
<b>Total Errors Detected</b>	<b>108</b>	-



With the octaver algorithm, the piezo pickup produced 65% of the errors and the humbucker contributed 35% of the errors. The figure worsens with the bass synthesizer, where the piezo generated close to 70% of the errors and the remainder were humbucker prone errors.

The leading cause for errors with the piezo pickup were low frequency oscillations. This produced infra-audio contents (sub 20 Hz) and worsened the peak tracking algorithm in the octaver. Similarly, the ACF in the YIN algorithm produces significant errors prior to the intermediate steps, leading to errors in the frequency estimate. The errors in the ACF are caused due to the DC shifts producing large ACF values and possible completely falling out of favor since the signal's correlation becomes improbable. Moreover, the humbucker also produced DC shifts in certain cases, but were much controlled and did not produce low frequency oscillations, or infra-audio contents. The low frequency oscillations are depicted in Figure 41.

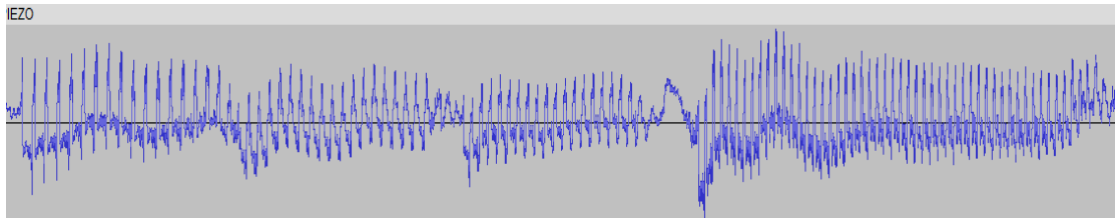


Figure 41. Low Frequency Oscillations.

Further harmonic analysis using a spectrogram reveals the significant differences between the humbucker and piezo pickup signal contents in the sub frequencies (black box), as shown in Figure 42. A window size of 4096 samples was used for the spectrogram.

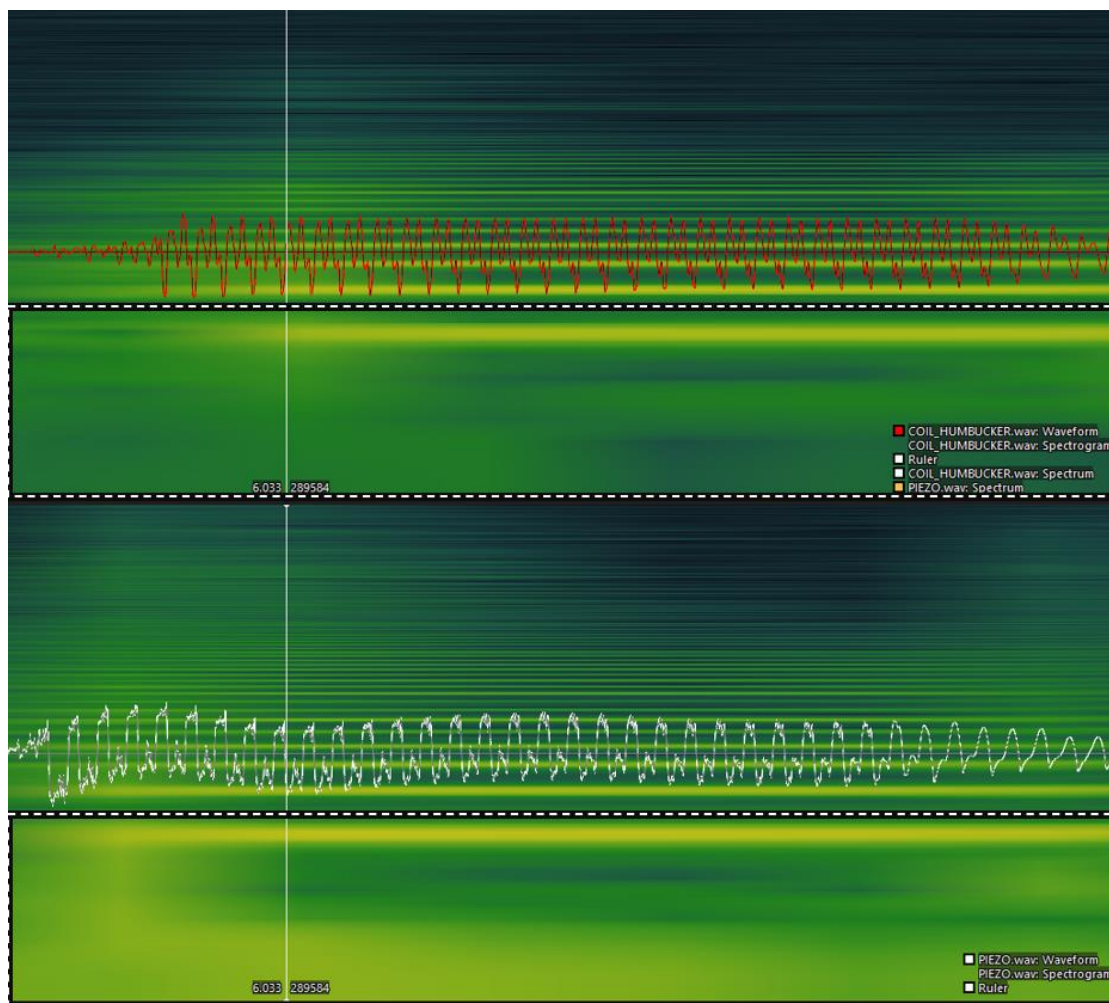


Figure 42. Spectrogram Contents of the Signals. Checkered Box Marks the Region of Interest.

The speculative causes for this effect may be the nature of the string motion and the relationship it has with the output signal. Although this signal characteristic can be inaudible in a clean context; the octaver processing produces low frequency distortion and perceivable audio artefacts, thus worsening the signal's quality. Furthermore, the piezo caused synthesizer errors that produced shifts larger than or equal to 12 semitones above the F0.

Another peculiar nature of the piezo pickup signal was the lack of negative peak amplitude. In most cases, this did not produce any effects, but in instances with significant DC shifts, the peak tracking and the ACF produced instabilities or outright errors. Additionally, the muting signal sequence in octaver was aperiodic in certain instances due to this effect. The irregular nature of the muting led to

significant deviation errors. In the humbuckers case, the positive and negative peak of the signal had adequate amplitude, therefore had higher stability. An example of the weak negative peak amplitude is shown in Figure 43.

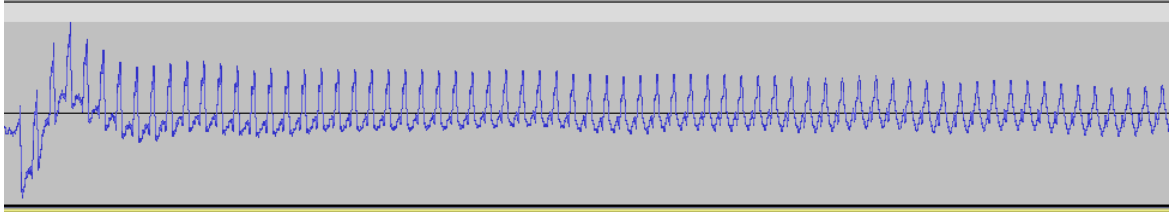


Figure 43. Weak Negative Peak Amplitude in the Piezo Pickup Signal.

In certain cases, the piezo pickup signal produced scenarios where the algorithms did not recover within the expected duration for stability. The instability typically lasted for over 50 ms and in certain cases the signal did not recover for the entire duration of the signal. An example state of instability is illustrated in Figure 44.

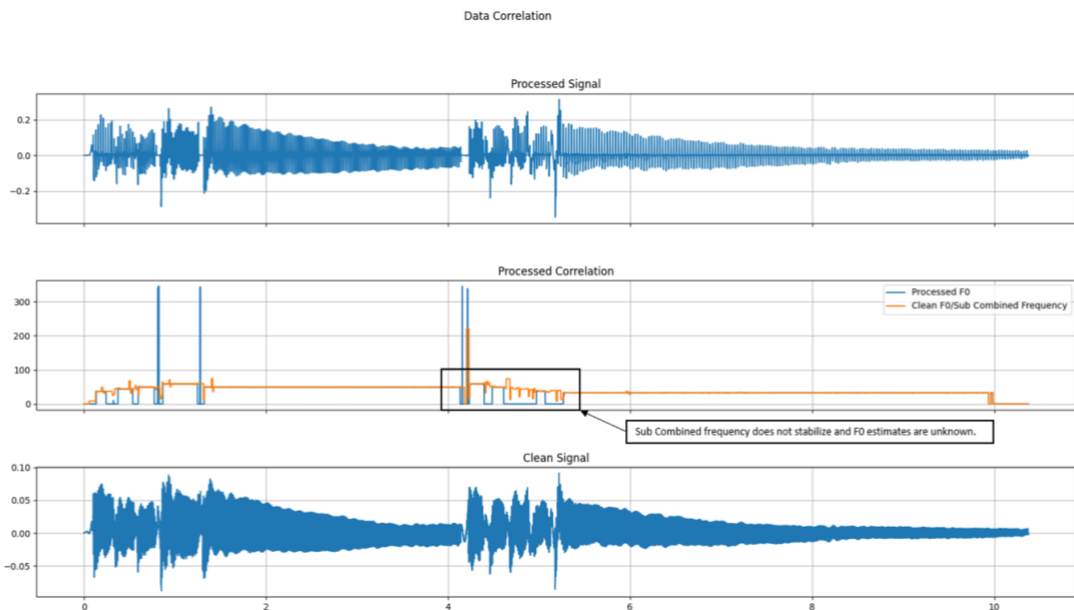


Figure 44. Unstable Sub Combined Frequency Regions.

Further harmonic analysis of the signal spectrum revealed a much stronger presence of higher order harmonics in the piezo pickup. Similarly, the audible perception of the piezo pickup correlates to the spectral data. Although the FFT of the signal reveals these contents, a significant limitation of the frequency

domain analysis is the lack of correlation with the time domain. In some instances, a direct correlation of the time and frequency domain was visible. Hence, the high frequency contents of the piezo pickup signal were assumed to be the cause for certain octave errors in both the algorithms. The spectrum analyzer of the humbucker and the piezo pickup are presented in Figure 45.

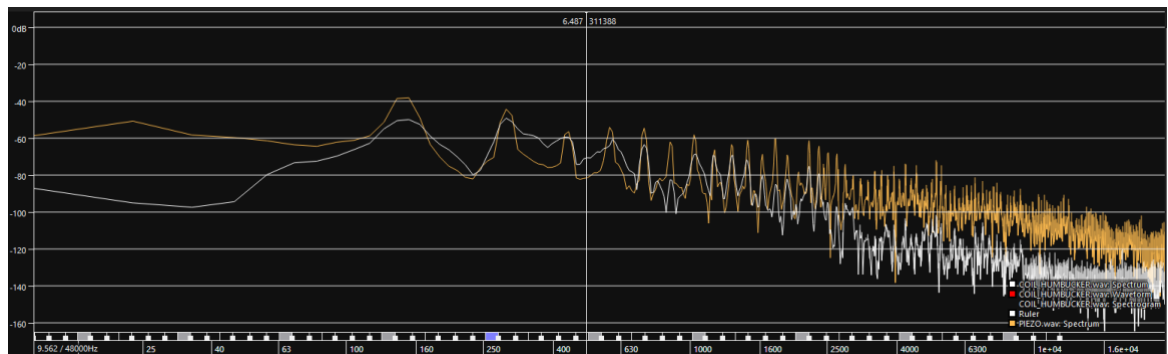


Figure 45. Spectrum Analysis of the Pickup Signals. Orange: Piezo. White: Humbucker.

Another observation from the spectrum analysis was the dominant presence of the second harmonic over the F0 in the humbucker signal, whereas the F0 is the highest peak in the piezo pickup signal. This effect does not directly pertain to the algorithms under test, but the variance in the spectral content is an important distinction between the pickup types. Although the humbucker pickup was wired in a split coil configuration, the noise performance of the pickup's individual coils was not viable for the testing methods. This limitation may mostly apply to the generic pickup used for testing.

Both pickup types required restriction in the overall usability of the bass guitar, but the piezo pickup imposed more constraints due to the sensitivity of additional string noise and other unwanted signal presence. Furthermore, the piezo pickup's non-linear output characteristics were much harder to control and produce consistent results. Though the piezo pickup's signal can be compressed to an extent to overcome this, it was deemed impractical and may have profound effects on the testing. Additionally, neither pickup produced any significant phase errors.

Between the three algorithms of the octaver, namely the fixed point, floating point, and release version, the errors were vastly produced with the fixed-point version. Furthermore, the floating point and release version produced the same number of errors, therefore, to streamline the testing procedure, the release version was primarily considered. The synthesizer on the other hand required a significant amount of tuning with the YIN algorithm parameters to acquire realistic results. Thus, the number of recordings used was limited to three. Lastly, it is quite important to note that these algorithms are naturally prone to errors since certain assumptions on the properties of a bass guitar's signals were made. This naturally, may cause poor performance with certain types of signals, in this case, the piezo pickup. The error rates per algorithm type and version are presented in Table 2.

Table 2. Total Error Counts and Rates: Sorted by Algorithm.

<b>Algorithm Type - Version</b>	<b>Errors</b>	<b>Percentage</b>
Octaver – Fixed Point	55	50.9%
Octaver – Release	46	42.5%
Octaver – Floating Point	<i>Same as Release</i>	-
Synthesizer	7	6.5%

Finally, the F0 estimation algorithm of the PYSPTK SWIPE function showed excellent correlation to the PYIN algorithm in Sonic Visualizer, thus validating the reliability of the overall testing procedure.

## 5 Discussion

The main goal of the correlation was to detect errors the pickup types produce when subjected to the algorithm, which was met with satisfactory results. Although the piezo pickup produced the most errors with the Darkglass DSP algorithms, it may not necessarily mean the pickup is inherently inappropriate for usage in other octaver and synthesizer algorithms styles. Moreover, the piezo pickup contains certain qualities the humbucker pickup does not possess. For

instance, the piezo has higher definition in the note onsets, whereas the humbucker severely lacks this quality. Humbuckers also largely limit the acoustic qualities of the strings on a bass guitar. Piezo, in this regard, captures these acoustic tendencies excellently.

The piezo also allows for multichannel and polyphonic processing using little-to-no modification to existing techniques in the algorithms, since each string contains an individual piezo element. This also offers better string isolation, hence reducing unnecessary noise from adjacent strings to interfere with the signal contents. Humbuckers require polyphonic DSP algorithms to achieve this, which is heavier for processing in small embedded systems. Another unique pickup type to evaluate would be hexaphonic pickups, which use individual magnetic pole pieces per string. Existing guitar synthesizers use hexaphonic pickups combined with MIDI (Musical Instrument Digital Interface) technology to attain polyphonic signal processing.

Additionally, filtering the higher order harmonic contents of the piezo signal did not improve the errors produced. As previously mentioned, though the high frequency contents were removed in the frequency domain, the signal did not necessarily undergo vast time domain changes to offer improvement. This was mostly because both algorithms are time domain dependent as opposed to frequency.

However, tracking the positive peaks for the piezo pickup would vastly reduce error rates in the octaver algorithm. Thus, negating the poor negative peak amplitude in the signal of the piezo pickup and conversely improving the effects of DC offset oscillation errors as well. Certain popular octavers utilize this method of peak tracking and auditory testing revealed that the detected effects and errors were either negated or on par with the humbucker pickup.

Lastly, from a historical point of view, the humbuckers have been foundational to the legacy of the electric basses and guitars, therefore most effects were designed in favor of them. The results do not come as a surprise when this factor

is considered. Utilizing humbuckers is also cost effective as they are much more traditional in this regard and have taken rather large strides in development.

## **6 Conclusion**

The testing yielded excellent results in correlating data from the piezo and humbucker pickups. Furthermore, a profound understanding in the fundamentals of signals and DSP was attained through the process. Although the piezo pickup produced the most errors, certain valuable features for various applications were also realized. Subsequent modifications to the existing algorithms would certainly reduce the overall errors and render the piezo as a viable option. Consequently, the harmonic analysis of the signals from both pickups helped determine the natural differences between them and evaluate the conditions for errors.

The python testing script was beneficial in detecting errors and representing them visually. Redundant checks with other open-source applications (Sonic Visualizer) for specific tasks were beneficial in validating the overall functionality of the script. Moreover, the modified test bass guitar performed reliably for the testing procedure. The noise performance and distortion (THD+N) of the debugging pre-amplifier was eventually negligible as the design carefully identified all sources. This was evident since versions preceding the final design introduced a significant amount of noise, thus rendering the data inappropriate for testing.

Understanding the fundamental theories and concepts behind the algorithms helped simplify the analysis process. Similarly, error conditions were predictable with the aid of pre-existing knowledge. The overall thesis work enabled discovery and understanding of similar technologies. Furthermore, the process of selecting the appropriate pickup for embedding Darkglass' DSP was greatly simplified. The results were vastly necessary and useful for Darkglass' future projects and implementation of embedded effects in bass guitars.

## References

1. Georgia State University. Sound Quality or Timbre [Internet]. United States: Georgia State University; 2001. Source: <http://hyperphysics.phy-astr.gsu.edu/hbase/Sound/timbre.html> [Cited: 14 April 2023]
2. Analog Devices. A Beginner's Guide to Digital Signal Processing [Internet]. United States; 2023. Source: <https://www.analog.com/en/design-center/landing-pages/001/beginners-guide-to-dsp.html> [Cited: 17 April 2023]
3. Analog Devices. Sampling Rate: What is Sampling Rate? [Internet]. United States; 2023. Source: <https://www.analog.com/en/design-center/glossary/sampling-rate.html> [Cited: 17 April 2023]
4. Tamara Smyth. Nyquist Sampling Theorem [Internet]. United States: UC San Diego; 2019. Source: [http://musicweb.ucsd.edu/~trsmyth/digitalAudio171/Nyquist\\_Sampling\\_Theorem.html](http://musicweb.ucsd.edu/~trsmyth/digitalAudio171/Nyquist_Sampling_Theorem.html) [Cited: 17 April 2023]
5. Harsh Maheshwari. Terms you need to know to start Speech Processing with Deep Learning [Internet]. Towards Data Science; May 10, 2021. Source: <https://towardsdatascience.com/all-you-need-to-know-to-start-speech-processing-with-deep-learning-102c916edf62#:~:text=Window%20length%20is%20the%20length,portio n%20of%20the%20window%20length.> [Cited: 17 April 2023]
6. Laurent Duval and Aravinda Murthy. What is the relation between windowing and hopping in audio DSP. Stack Exchange: Signal Processing; May 6, 2017. Source: <https://dsp.stackexchange.com/questions/40784/what-is-the-relation-between-windowing-and-hopping-in-audio-dsp> [Cited: 19 April 2023]
7. Karthik Chaudary. Understanding Audio Data, Fourier Transform, FFT and Spectrogram Features for a Speech Recognition System. Towards Data Science; January 19, 2020. Source: <https://towardsdatascience.com/understanding-audio-data-fourier-transform-fft-spectrogram-and-speech-recognition-a4072d228520> [Cited: 19 April 2023]
8. Georgia State University. Fundamental and Harmonic Resonances [Internet]. United States: Georgia State University; 2001. Source: <http://hyperphysics.phy-astr.gsu.edu/hbase/Waves/funhar.html> [Cited: 20 April 2023]
9. Learn Cigar Box Guitar, Patrick. Standing Waves and The Harmonic Series [Internet]. United States; 2020. Source: <https://learncigarboxguitar.com/content/standing-waves-and-harmonic-series> [Cited: 20 April 2023]
10. University of Zurich. Signal Processing and Analysis: Spectral Leakage [Internet]. Switzerland: University of Zurich; June 2012. Source: <https://www.physik.uzh.ch/local/teaching/SPI301/LV-2015->



- [Help/ivanlsconcepts.chm/Spectral\\_Leakage.html#:~:text=In%20spectral%20leakage%2C%20the%20energy,exactly%20repeats%20throughout%20all%20time.](https://www.tutorialspoint.com/autocorrelation-function-and-its-properties) [Cited: 21 April 2023]
11. Manish Kumar Saini. Autocorrelation Function and its Properties [Internet]. Tutorialspoint; January 7, 2022. Source: <https://www.tutorialspoint.com/autocorrelation-function-and-its-properties> [Cited: 21 April 2023]
  12. B. H. Suits, Physics Department, Michigan Technological University. Physics Notes: Autocorrelation (for sound signals) [Internet]. United States: Michigan Technological University; 1998. Source: <https://pages.mtu.edu/~suits/autocorrelation.html> [Cited: 21 April 2023]
  13. Alain de Cheveigné and Hideki Kawahara. YIN, a fundamental frequency estimator for speech and music. United States: Acoustical Society of America; October 10, 2001. [Cited: 22 April 2023]
  14. Richard Pryn. What is a semitone? [Internet]. United States: Richard Pryn; October 6, 2022. Source: <https://richardpryn.com/what-is-a-semitone/> [Cited: 22 April 2023]
  15. Swinburne University. Destructive Interference [Internet]. Australia: Swinburne University. Source: <https://astronomy.swin.edu.au/cosmos/d/Destructive+Interference#:~:text=Destructive%20interference%20occurs%20when%20the,the%20resulting%20wave%20is%20zero.> [Cited: 22 April 2023]
  16. Sobreira et al. Soundhole [Internet]. Wikipedia; March 20, 2023. Source: [https://en.wikipedia.org/wiki/Sound\\_hole](https://en.wikipedia.org/wiki/Sound_hole) [Cited: 23 April 2023]
  17. Monkbot et al. Pickup (Music Technology) [Internet]. Wikipedia; April 10, 2023. Source: [https://en.wikipedia.org/wiki/Pickup\\_\(music\\_technology\)](https://en.wikipedia.org/wiki/Pickup_(music_technology)) [Cited: 24 April 2023]
  18. J Donald Tillman. Response Effects of Guitar Pickup Position and Width [Internet]. United States; July 1, 2000. Source: <http://www.till.com/articles/PickupResponse/index.html> [Cited: 24 April 2023]
  19. Yamaha Corporation. The Structure of the Electric Guitar: What are Pickups? [Internet]. Yamaha Corporation. Source: [https://www.yamaha.com/en/musical\\_instrument\\_guide/electric\\_guitar/mechanism/mechanism002.html](https://www.yamaha.com/en/musical_instrument_guide/electric_guitar/mechanism/mechanism002.html) [Cited: 24 April 2023]
  20. Georgia State University. Faraday's Law [Internet]. United States: Georgia State University; 2001. Source: <http://hyperphysics.phy-astr.gsu.edu/hbase/electric/farlaw.html> [Cited: 24 April 2023]
  21. Helmuth E. W. Lemme. The Secrets of Electric Guitar Pickups [Internet]. Build Your Guitar: Austria; 1986. Source: <http://buildyourguitar.com/resources/lemme/> [Cited: 24 April 2023]
  22. P, Lotton, B, Lihoreau, E, Brasseur. Experimental Study of a Guitar Pickup. France: ISMA; 2014. [Cited: 24 April 2023]

23. Nick Stoubis. How to Pick your Pickup? [Internet]. United States: Fender Musical Instruments Corporation. Source: <https://www.fender.com/articles/instruments/electric-guitar-pickup-types-how-to-choose-your-pickup#:~:text=Single%2Dcoil%20pickups%20have%20been,while%20you%20are%20not%20playing>. [Cited: 25 April 2023]
24. Douglas Self. Small Signal Audio Design: Third Edition. London: Focal Press; 2019. [Cited: 25 April 2023]
25. Seymour Duncan. The Anatomy of a Single Coil Pickup [Internet]. United States: Seymour Duncan; October 5, 2022. Source: <https://www.seymourduncan.com/blog/latest-updates/the-anatomy-of-single-coil-pickups> [Cited: 25 April 2023]
26. Tommy Chung. The EMG H1A: my number 1 choice humbucker [Internet]. Hong Kong. Source: <https://48chicagoblues.com/EMG%20H1A/EMG%20details.htm> [Cited: 25 April 2023]
27. Chris Woodford. Piezoelectricity [Internet]. United States: Explain That Stuff; May 21, 2022. Source: <https://www.explainthatstuff.com/piezoelectricity.html> [Cited: 25 April 2023]
28. ESP, Takamine. Inside the Takamine Palathetic Pickup [Internet]. United States: Takamine; April 16, 2020. Source: <https://www.esptakamine.com/articles/2013621-inside-the-takamine-palathetic-pickup> [Cited: 25 April 2023]
29. Seymour Duncan. Piezo Vs. Magnetic Pickups. [Internet]. United States: Seymour Duncan; October 5, 2022. Source: <https://www.seymourduncan.com/blog/latest-updates/piezo-vs-magnetic-pickups> [Cited: 25 April 2023]
30. Rod Elliot. Piezo Pickup Preamplifiers [Internet]. Australia: Elliot Sound Products; March 2020. Source: <https://sound-au.com/project202.htm> [Cited: 26 April 2023]
31. Cadence PCB Solutions. How to Stop Radio Frequency Interference [Internet]. United States: Cadence. Source: <https://resources.pcb.cadence.com/blog/2022-how-to-stop-radio-frequency-interference> [Cited: 30 April 2023]
32. Electronics Tutorials. Varistor Tutorial [Internet]. Electronics Tutorials; 2014. Source: <https://www.electronics-tutorials.ws/resistor/varistor.html> [Cited: 1 May 2023]

## Datasheets

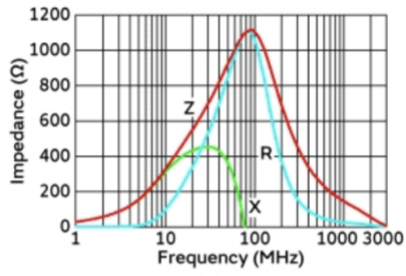
### Specifications

Shape	SMD
Size Code (in mm)	2012
Size Code (in inch)	0805
Length	2.0mm
Length Tolerance	±0.2mm
Width	1.25mm
Width Tolerance	±0.2mm
Thickness	0.85mm
Thickness Tolerance	±0.2mm
Impedance (at 100MHz)	1000Ω
Impedance (at 100MHz) Tolerance	±25%
Rated Current (at 85°C)	1.6A
Rated Current (at 125°C)	1.1A
DC Resistance(max.)	0.12Ω
Operating Temperature Range	-55°C to 125°C
Mass(typ.)	0.01g
Number of Circuit	1

**BLM21SP102SN1#**

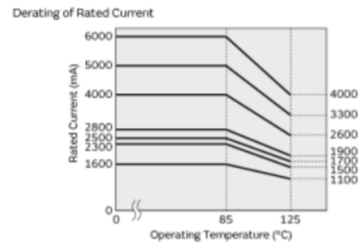
"#" indicates a package specification code.

**Product Data**

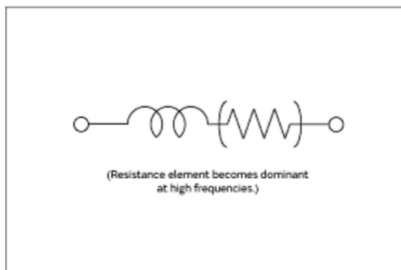


Impedance-Frequency Characteristics

In operating temperature exceeding +85°C, derating of current is necessary for BLM21SP series. Please apply the derating curve shown in chart according to the operating temperature.



Derating of Rated Current



Equivalent Circuit

## Source Code

```
class analyzer:

    def __init__(self, window=None, threshold=None, hopsize=None, tolerance=None):
        self.window = window
        self.threshold = threshold
        self.hopsize = hopsize
        self.tolerance = tolerance
        self.mask = 0

    def getData(self, path):
        data, sr = librosa.load(path)

        #Audio can be subsampled, by averaging it with a window width of W
        subsampled = []
        if self.window is None:
            return data, sr
        else:
            for i in range(0, len(data), self.window):
                subsampled.append(np.mean(data[i:i+self.window]))
            return np.array(subsampled), sr

    def getFreq(self, data, fs):
        #Getting frequency values
        f = sp.swipec(data, fs, self.hopsize, min=10, max=600, otype='f0')

        if self.threshold is None:
            pass
        else:
            #Compute the non-silent intervals (i.e., the intervals where the signal is above a certain threshold)
            non_silent_intervals = librosa.effects.split(data, top_db=self.threshold)

            #Create a binary mask to nullify the silent parts
            self.mask = np.zeros_like(data, dtype=bool)
            for interval in non_silent_intervals:
                start = interval[0]
                end = interval[1]
                self.mask[start:end] = True

        #Applying the mask
        f = f * self.mask
```

```

#Find total length by multiplying the window width with the size of the array and multiplying it with the sampling period
total_length = (len(f) * self.hopsize) * (1 / fs)

#Creating an array of numbers with fixed windowed sampling intervals
time = np.arange(0, total_length, self.hopsize * (1 / fs))

data = {'f0': f,
        'Time': time}

return data, f, time

def subprocess(self, subprocess_path):
    #Clean Signal
    #Subprocess method is the square waved signal output. Used to detect timbre changes and an alternative reference for pitch detection.
    #Shows up as an abrupt period change, which reflects in the fundamental frequency.
    data, fs = librosa.load(subprocess_path)
    ctr = 0
    f = []
    store = 0

    #When a product of a sample is negative then the signal must have a sign change,
    #which is when the counter (number of samples before sign change) is reset to 0 and a new count starts.
    #Dividing it with the sampling frequency gives the estimated fundamental frequency.
    for j in range(1, len(data)):
        if (data[j] * data[j-1] < 0):
            if ctr > 1:
                store = (fs/ctr)/2
            else:
                store = 0
                ctr = 0
            else:
                ctr += 1
                f.append(store)
    f.append(f[len(f)-1]) #Since the sub file is being read from the 2nd element, it has one less element.

    if self.threshold is None:
        pass
    else:
        f = self.mask * f #Applying gate on the sub process

    if self.window is None:
        return np.array(f)

```

```

    else:
        averaged = []
        for i in range(0, len(f), self.window):
            averaged.append(np.mean(f[i:i+self.window]))
        return np.array(averaged)

def processDiff(self, clean, dirt, time, mode):
    #Mode if true processes clean as ideal octave values. Mode if false doesn't do anything to the clean signal frequencies
    if mode:
        clean = clean/2
    elif not mode:
        clean = clean

    #Calculate pitch deviation from ideal values
    semi = (12 * np.log2(dirt/clean))

    #Remove values that are -inf/inf
    semi = np.where( semi != float('inf'), semi, 0)
    semi = np.where( semi != - float('inf'), semi, 0)
    cents = semi * 100

    setFlag = []
    isOctave = []
    tol = self.tolerance / 10
    #Setting a flag for unstable values and detecting octave differences
    for i in range(0, len(semi)-1):
        if semi[i] == float('nan'):
            setFlag.append(-1)
        elif semi[i] == 0:
            setFlag.append(0)
        elif semi[i+1] - semi[i] != semi[i]:
            if semi[i] >= semi[i+1] * (1 - tol) or semi[i] <= semi[i+1] * (1 + tol):
                setFlag.append(1)
            else:
                setFlag.append(0)
        #Checking for octave differences within bounds and check if there are values over an octave
        if (1 - tol) * 12 >= semi[i] or (1 + tol) * 12 <= semi[i] or semi[i] > 12:
            isOctave.append(0)
        else:
            isOctave.append(1)
    setFlag.append(-1) #Ignoring last value
    isOctave.append(0) #Ignoring last value

```

```

    data = {'Time': time,
            'Clean': clean,
            'Dirt': dirt,
            'Semitones': semi,
            'Cents': cents,
            'isStable': setFlag,
            'isOctave': isOctave}

    return data, semi, setFlag, isOctave

def plot(self, time, octave, clean=None, f=None, sub=None, dev=None, flags=None, isOctave=None):

    #Plotting frequency data and the audio clip for visualization
    fig, ax = plt.subplots(3, sharex=True)
    fig.suptitle("Data Correlation")
    ax[0].plot(time, octave)
    ax[0].set_title("Processed Signal")
    ax[0].legend(["Octave Signal"], loc = "upper right")
    ax[0].grid()

    legend = []
    if f is not None:
        ax[1].plot(time, f)
        legend.append("F0")
    else: pass

    if dev is not None:
        ax[1].plot(time, dev)
        legend.append("Deviation in Semitones")
    else: pass

    if flags is not None:
        for i in range(0, len(time)):
            flags[i] = 10 + 10 * flags[i]
        ax[1].plot(time, flags)
        legend.append("Flags")
    else: pass

    if isOctave is not None:
        for i in range(0, len(time)):
            isOctave[i] = 10 + 10 * isOctave[i]

    if sub is not None:
        ax[1].plot(time, sub)
        legend.append("Sub Combined Frequency")
    else: pass
    ax[1].legend(legend, loc = "upper right")
    ax[1].set_title("Processed Correlation")
    ax[1].grid()

    if clean is not None:
        ax[2].plot(time, clean)
        ax[2].legend(["Clean Signal"], loc="upper right")
        ax[2].set_title("Clean Audio")
        ax[2].grid()
    else: ax[2].set_visible(False)

    plt.tight_layout()
    plt.show()

def runOctaver(self, exe_path, folder, filename, args, dryrun):
    #Runs the batch processor executable with which runs the audio
    #processing and creates audio files of all debug streams.

    if not args:
        args = ['1']
    if '.wav' in filename:
        raise Exception('Filenames should be given without file extensions')
    # find executable based on our system
    if not os.path.isfile(exe_path):
        raise Exception(f'Executable not found: "{exe_path}"')
    args = [exe_path, f'sounds/clean/{folder}/', filename] + args
    print(' '.join(args))
    if dryrun:
        return
    output = subprocess.run(args, capture_output=True)
    if output.returncode != 0:
        print(' '.join(output.args))
        raise Exception(output.stderr)

```

```
##### INIT #####
file = 'COIL_HUMBUCKER'
mode = 'fixed'
folder = 'multi_notes'

#Args: Averaging Window Width, Threshold for Gating, Hopsize, Tolerance. If None: Averaging and Gating can be skipped.
#Init Object
analyzer = analyzer(None, None, 1, 5)

#Run Octaver exe and generate data
exe_path = f'exe/hybrid_octaver_batch_processor_{mode}.exe'
analyzer.runOctaver(exe_path, folder, file, False, False)

#Audio
clean_file = f'sounds/clean/{folder}/{file}.wav'
octaver_file = f'sounds/clean/{folder}/processed_{mode}/{file}/MAIN_OUT.wav'
sub_file = f'sounds/clean/{folder}/processed_{mode}/{file}/SUB_COMBINED.wav'

##### MAIN PROCESSING #####

#Args: File path
octave, sr = analyzer.getData(octaver_file)
clean, sr = analyzer.getData(clean_file)

#Args: Data, Fs.
#Clean
clean_data, clean_freq, time = analyzer.getFreq(clean, sr)
#Dirt
octave_data, octave_freq, time_octave = analyzer.getFreq(octave, sr)

#Sub Combined File Processing
#Args: File Path, Binary Mask
sub, sr = analyzer.getData(sub_file)
sub_freq = analyzer.subProcess(sub_file)

#Args: Clean Freq, Dirt Freq, Time
#True = OCTAVER, False = SYNTH
processor_data, dev, flags, isOctave = analyzer.processDiff(clean_freq, octave_freq, time, True)

#Args: Time, Processed Signal, Clean, Processed Frequency, Sub Process Freq, Deviation, Flags, Octave Errors.
#Use None for omitting data (cannot omit Processed Audio and Time).
analyzer.plot(time, octave, clean, None, None, dev, flags, isOctave)
```