Guitar Tuner implementation on multi-platforms

Yang Lu, Anzhi Li and Zhaoyi Ming

University of Rochester

Abstract

As music industry continues to thrive, an increasing amount of aspiring and enthusiastic people start to devote themselves into this industry by learning the most popular instrument: guitar. However, tuning the guitar has always been a tricky issue for every guitarist since the conventional guitar tuner is not so convenient to carry and is often easy to forget when leaving in a hurry. Therefore, our goal was to create a guitar tuner on three different platforms: MATLAB, Arduino and IOS, which is able to provide a Ready-to- tune condition for guitarists wherever they are. This paper illustrates two different algorithms we used in order to realize the effect.

1. Introduction

This project is mainly designed and conducted in order to realize the function of guitar tuner on three different platforms: MATLAB, Arduino and IOS.
There are two approaches for the most important part, frequency detection. First one is frequency domain detection. We use this algorithm to implement the tuner for computer user in MATLAB and IPhone user in SWIFT.
The second one is pitch detection in time domain. The guitar tuner for traditional platform is based on this algorithm.

2. Methodology

There are two methods for frequency detection, which is core part of this project, are used in guitar tuner implementation, one is in frequency domain, another one is in time domain. For pitch detection in frequency domain, the input signal is transferred to frequency domain from time domain by Fourier transform first. Then through detect the frequency corresponding to the highest magnitude in the frequency domain to determine the fundamental frequency of the input signal. In time domain, the method to detect the pitch is finding the period of input signal directly. The basic idea is to find the repeat of the characteristic parts of the input signal like the peak or valley of input, while this method is not robust enough, the improved method used cross mid-point and time counter, which can work better for difficult input signal will be mentioned later.

2.1. MATLAB platform

Completed entirely in MATLAB, the guitar tuner mainly receives user's input from the computer 's built-in microphone. User's input is stored by first creating a recorder object using MATLAB built-in function *audiorecorder* and then returns the recorded audio data from object data to numeric array by using *getaudiodata* function. A bandpass filter which has cut-off frequencies set as 70Hz and 450Hz is arranged after the data is converted since the filter can efficiently extract the necessary and representative signal and at the same time filter the useless signal and the noise comes with it. The frequency response is calculated through applying MATLAB's built-in fft function (Fast Fourier Transform) on the data transformed from the recorder object. Since the fundamental frequency for instrument like guitar is the maximum frequency, therefore we simply utilize the *max* function to locate the maximum peak value from the entire frequency response and extract it as the user's note frequency. When the program is finished with the recording and analysis process, it will compare the user's note frequency with the target frequency listed previously in the program. If the absolute value between two frequencies is less than 2Hz, we are able to consider user's note frequency is on the correct tune. Otherwise the user will be informed that the string needs to be turn tight or lose.

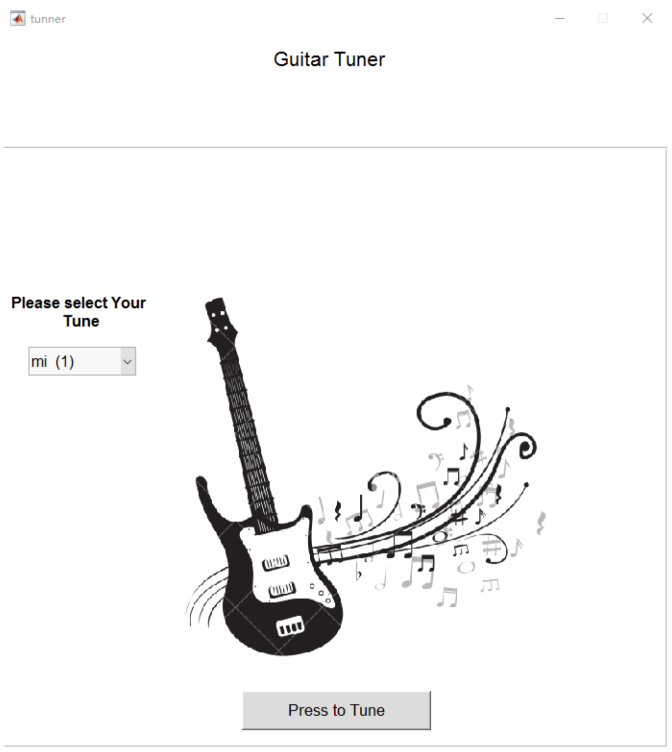


Figure 1: UI of MATLAB platform

2.2. iOS platform

Similar to the condition of MATLAB platform, iOS platform used iPhone's built-in microphone to catch input signals. We cannot avoid catching noise, so we choose same algorithm as MATLAB platform, which is the frequency detection on frequency domain. We used Pure Data to build the core functions of our App, and connected it to iOS platform by using Swift.

Our Pure Data patch is made up of two part. Part1 is an input signal processor. Function "adc~" is a Digital Analog Converter, which is used to convert analog signal to digital signal. Then, input signal is sent to the core function "fiddle~". "fiddle~" is a function based on FFT algorithm that performs an analysis of both volume and pitch. In this case, we only do pitch analysis function. Applying FFT (Fast Fourier Transform) on digital input signal, and output the estimated MIDI note of it. At last, we used "mtof" function to convert MIDI note to frequency in hertz.

Part2 is a sound generator. this program receive MIDI note from user, and convert it to frequency. Then, we used function "osc~" to generate output digital signal. Lastly, we used "dac~" function to convert digital signal to analog signal.

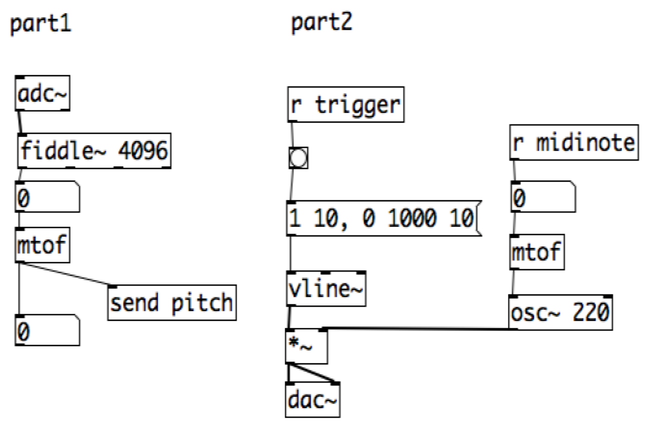
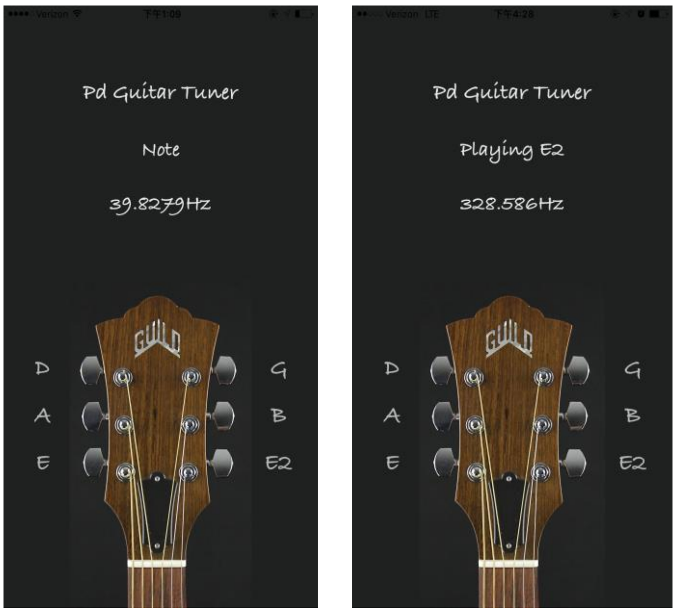


Figure 2: Pure Data Patch

After we finished Pure Data patch, we used Swift to apply this patch on iOS platform and design GUI. There are two labels and six buttons on our GUI. Six buttons are respect to the six notes of six strings. Users can input commands by press the six buttons, then, corresponding notes are coming out from speaker. Also, the upper Label will display the note which is playing. Those six reference notes can let users know whether their playing is accurate or not. Firstly, users press the button they want to tune, so App will detect the note playing by iPhone and displaying its frequency on the lower label. After remembering the frequency, users can

Figure 3&4: UI of IOS platform



keep playing and tuning their Guitar string. When users pluck string, the second Label displays the frequency of note from guitar. Therefore, users can directly know the frequency of the note they are playing and stop tuning their guitar when the displaying frequency is same as the one they remembered previously.

2.3. Traditional Platform

The guitar tuner on traditional platform has two main parts, one is hardware part concluding circuit analysis and connection with Arduino board, another one is software part which contains input frequency detection and display control.

2.3.1 Audio Input Circuit Analysis

To send the signal from electrical guitar to Arduino board, the circuit is used as shown in figure1. The output of guitar signal is around 200mv, while Arduino only can read the voltage from 0 to 5 volts. Thus, amplification is a necessary part for the circuit, which means increasing the amplitude of a signal. Amplification also buffers the audio source from any loads that is a good thing because it prevents distortion [4]. The amplitude of input signal after amplification could be increased to 2.5 volts which is our desired. However, the center voltage is at 0, which means the minimum voltage of the signal is -2.5 volts, which still cannot be read by Arduino board. Therefore, the second part DC offset is needed. Through DC offset, the center voltage could be moved from 0 volts to 2.5 volts, then the input signal oscillate around 2.5 volts with the amplitude 2.5 volts, which means the highest voltage is 5 volts and the minimum voltage is 0 volts, which satisfied with the requirement of Arduino board.

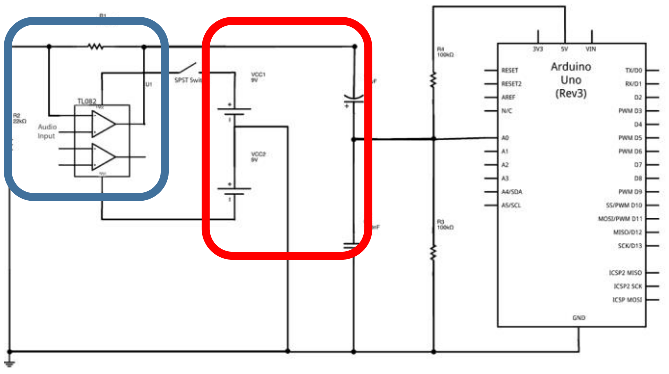


Figure 5: Main circuit used in the guitar tuner

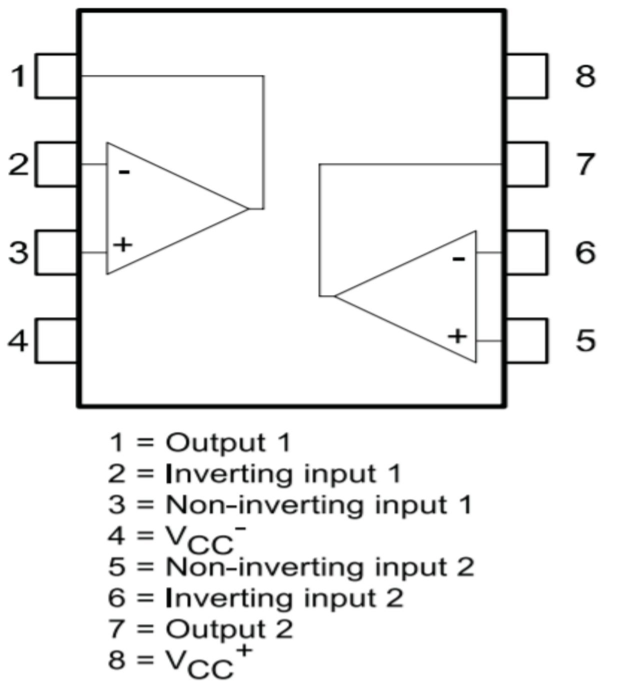


Figure 6: Pin Connections for TL082

2.3.2. Input Signal Frequency Detection

As mentioned above, the basic idea of frequency detection is to detect the peak and valley of input signal in time domain, while this method is not robust enough. To make this method adaptive more difficult input signal, there are several steps in the final method used in this improved method. First, choose a bond of the input signal. Since the input signal is amplified and moved from the circuit, it is easily to calculate the bound of the guitar input signal is 2.5 volts. Then, we define a new case called--threshold events, when the wave crosses this level with a positive slope. If this happened multiple times in one cycle, the threshold event with the largest slope is selected to be the beginning of the cycle. Like the last step, I used a variable called "time" to measure the time between threshold events and stored this is an array called timer. I also recorded the slope at each of the threshold events in an array called slope. The next step is comparing the elements of timer and slope to figure out where there was a match. Once a match was found in both arrays at the same time, the elements of timer are added up to determine the duration of the cycle and sent this value to a global variable called "period"[4], and then we can calculate the frequency of this input signal. After the frequency is determined by the guitar tuner, we can simply compare the frequency with the standard frequency of the pitch of this string, and determining whether the pitch is right or not and tell the user should tight or loose the strings according to the indicate LED lights.

2.3.3. Light Control

According to the figure: 7, there are thirteen LED lights are connected to the Arduino board. Six of them are yellow, which represent the six strings of guitar. And six red lights are set on the two side of the green lights, in order to indicate the accuracy of pitch. Once the green light is on, it means the pitch of this string is correct right now. The Character below the yellow light is the standard pitch of each strings, and the numbers below these characters are the corresponding standard frequency with unit hertz. The range in the bottom is the turn on range of each light. And the upper values are the error of the frequency.

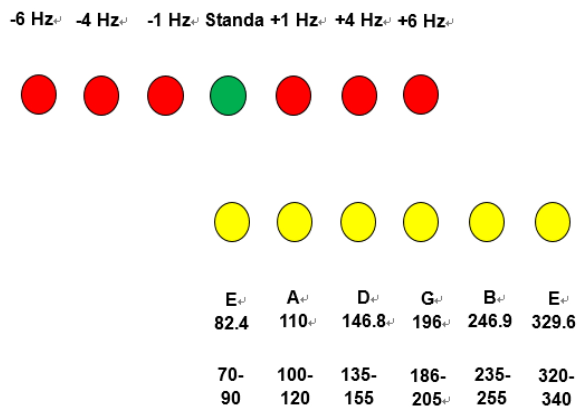


Figure 7: The control principle of LED lights of the guitar

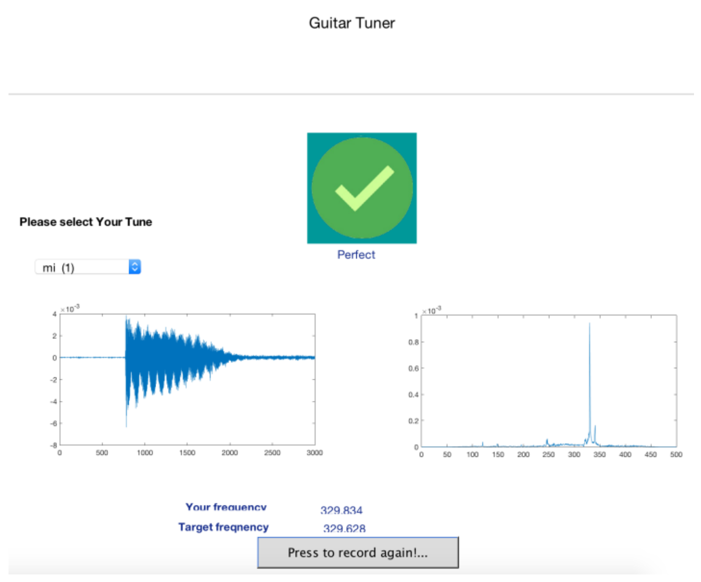
3. Result

3.1. IOS platform

We tried to tune the first string, which is E2. Firstly, we pressed E2 button, and the upper Label shows “playing E2”. Also, the lower Label told us reference frequency of E2 is 328.586Hz. Then, we plucked first string n guitar. The number on lower Label became 327.447Hz. Because error between guitar and reference frequency is smaller than 2Hz, we considered the first string of our guitar is accurate.

3.2. MATLAB platform

We also tuned the first string on this platform. We firstly set the slider to “mi (1)”, which means we want to tune first string. Then, we pressed the button to record input signal from guitar. However, output figure showed us that note from guitar is lower than reference note. Thus, we tuned this string little more tightly and recorded one more time. This time, the error is small than 2Hz, means this string is accurate.



Guitar Tuner

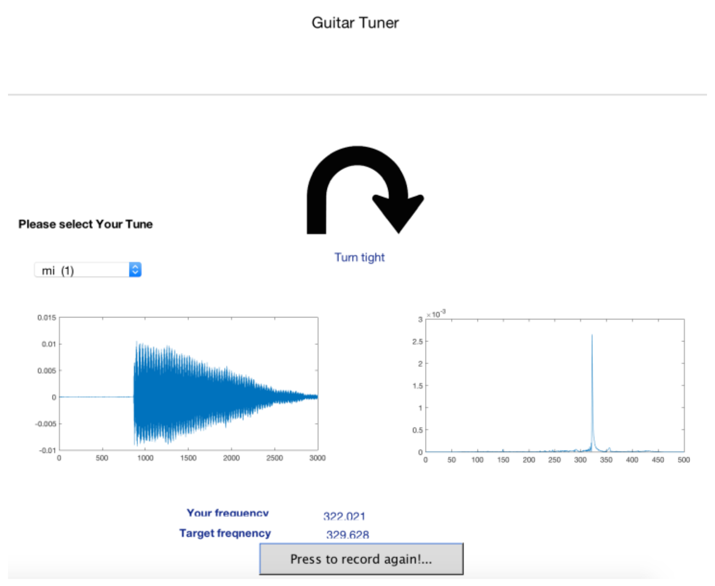
Please select Your Tune: mi (1)

Perfect

Your frequency 329.834

Target freqnency 329.628

Figure 7: The result of MATLAB tuning



Guitar Tuner

Please select Your Tune: mi (1)

Turn tight

Your frequency 322.021

Target freqnency 329.628

Figure 8: The result of MATLAB tuning

3.3. Hardware platform

This time, we tuned sixth string. Because we did not build a microphone in our circuit, we only can tune an electronic guitar. We turned on the circuit and plucked sixth string. The sixth yellow LED lights and one red light on right side were turned on. That means we are tuning the sixth string and input note is at least1Hz higher than reference frequency. Then, we kept plucking string and loose the string a little bit. Finally, the green light turned on, means input note is very close to reference frequency.

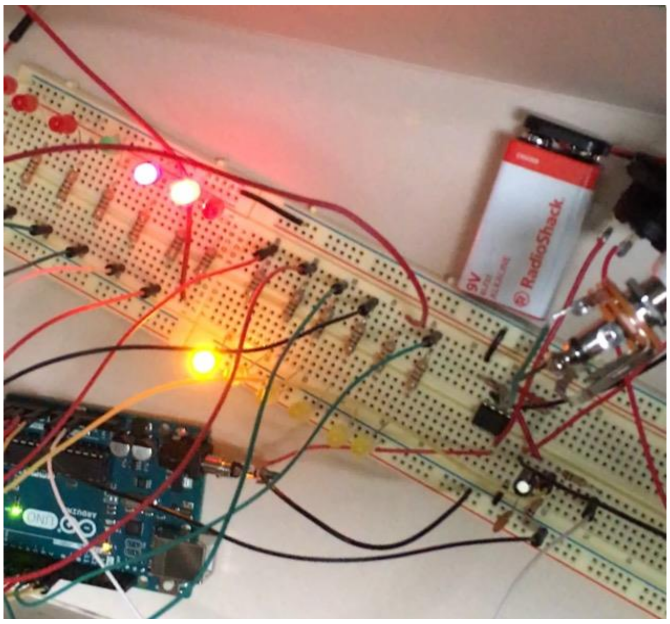


Figure 9: The experimental result of guitar tuner

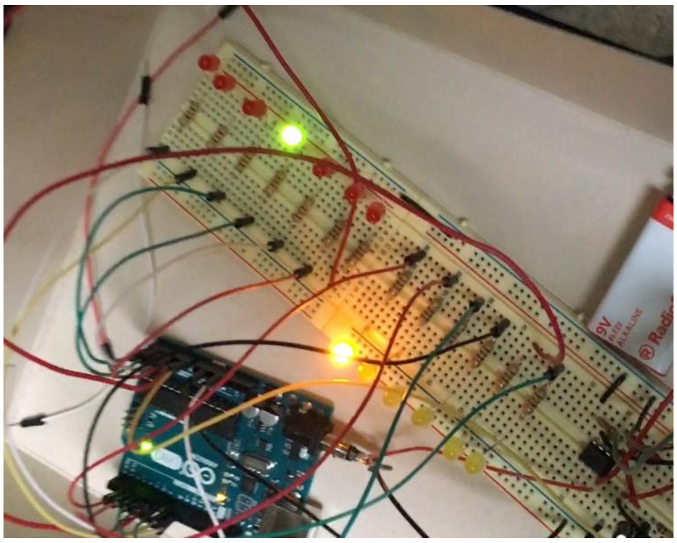


Figure 10: The experimental result of guitar tuner

5. Conclusion

5.1. Pros and cons

We tried to implement one algorithm on these three platforms. However, the results are far from satisfactory. For MATLAB and iOS platform, because we used built-in microphone to receive signal, noise came in together with signal. Those noises would change waveform of input signal on a vast scale so that period counter cannot get accurate number of period in one second. Therefore, result cannot be referred. On hardware platform, we can implement FFT method. However, although FFT algorithm is faster to detect frequency than period detection, a relatively huge power is required to supply the work. Thus, the requirement for power source is hard to satisfy. All in all, so far, we think FFT algorithm is better for MATLAB platform and period detecting algorithm is adapted to Hardware platform more.

5.2. Limitations

During the test of Hardware platform, we found that those LED lights could not be turned on if we plucked string a little bit softly. The reason we figured out is that input voltage is too small to reach the threshold of turning light on. Our planned solution is changing the threshold lower and increase the gain of our amplifier.

Besides, because we tested the guitar tuner app in a noisy environment, “fiddle~” function cannot estimate the accurate reference frequency. Therefore, users cannot get correct reference.

6. Future work

Currently, our project is working but have not reach our expectation. For refine our product, we firstly should make our hardware guitar tuner works better by adjust the circuit and program. Then, for our iOS App, we can add more Label to show users reference frequency directly instead of use “fiddle function” to estimate. This change can avoid the limitation of environment. Third, we can convert our output frequency to musical notes because players always prefer that way than exact frequency of note. Last but not least, for more convenient, we should figure out how to make guitar tuner work in real time on every platform. It is a long way to go.

7. Reference

[1]FFTGuitarTuner
http://www.codeproject.com/Articles/32172/FFT-Guitar-Tuner

[2]Discrete-Time Signal Processing, Alan V. Oppenheim BJORG, blog

[3]http://blog.bjornroche.com/2012/07/frequency-detection-using-fft-aka-pitch.html

[4]Amandaghassaei.ArduinoFrequencyDetection
http://www.instructables.com/id/Arduino-Frequency-Detection