AutoTab
Automatic Guitar Tablature Generation

Shaun Boley (shaun.boley@gmail.com)

Anthony Cisneros (a.cisneros7@gmail.com)

Eric Kauzlarich (Eric.Kauzlarich@gmail.com)

June 13, 2014

Abstract

Our project consists of two different parts:

The first part contains an analysis of various pitch estimation algorithms. A number of different open-source pitch estimation libraries were tested using guitar recordings in order to gauge how accurately and consistently they outputted the correct frequencies. The results showed that the algorithms did not perform up to expectations, especially when testing with guitars with wound strings and within certain frequency ranges.

The second part details the creation of a tablature generator. It reads in a note sequence as an input, then generates tablature in the form of ASCII text. A modified greedy algorithm was used to determine the most accurate and comfortable to play tab, basing its tabbing decisions on metrics such as note distance, among others.

Contents

List of Figures

List of Tables

1 Introduction

The guitar is an exemplification of the phrase "a labor of love". It is a difficult instrument to master in more ways than one. Not only is it intricate to learn how to play, reading music while playing a guitar is extremely tough. Thus, guitarists make use of "tablature", which substitutes strings and fret numbers in place of a musical staff and notes. This makes it so that anyone, no matter their skill in music theory, can learn to play their favorite songs.

Creating a tab for a song has always been a tedious process. Initially, tabs were written by hand, or typed out line by line in a simple text editing program. Thankfully, tablature software came along which facilitated the process, and even provided extra features such as playback and tempo changing. Even with these programs, however, creating a tab is still a tedious process.

When a user has mastered a song by ear (or a part of it), they typically arent too conscious of the fret numbers utilized. Sounds and "feel" are much more involved, and the numbers fade away. Thus, a user cannot simply sit at a computer and type out the song theyve just learned. They must sit with their guitar, play a part of it, memorize the fret numbers, then lean over their guitar and type the song into tablature piece by piece. This is a tiring process, and can discourage many players from sharing their favorite songs with the world. An "Automatic Guitar Tablature Generator" could transform tabbing into a much more enjoyable experience.

2 Part I - Pitch Detection

2.1 Pitch Detection Overview

Pitch is a perception of how "high" or "low" a sound is. This is directly related to the fundamental frequencies in that signal, with an increase in frequency causing an increase in the perceived pitch. When a guitar string is plucked, there is more than a simple sine-wave that resonates from the body of the guitar. Along with the fundamental frequency of the note, there are also harmonic frequencies that are produced, which are defined as integral multiples of the fundamental frequency. These harmonics are more specifically known as "overtones" in the realm of instruments such as the guitar. Harmonics are produced in string instruments when the strings are divided into two pieces at certain critical points. On a guitar, strings are divided by frets pressed by the player, normally resulting in the upper part of the string oscillating with the lower part remaining relatively stationary in most cases. The player can strike a note in such a way that the string oscillates on both sides of the fret, resulting in a 2nd harmonic octave, an octave plus a perfect fifth, and one octave respectively. Overtones can be inherent to certain played notes regardless of playstyle. They are largely what make pitch detection difficult, as the fundamental-frequency of the note needs to be separated from the generated overtones in order to determine the actual frequency of the note being played. There can also be a presence of lower harmonic partials, making a method of reading the lowest detected frequency inaccurate as well.

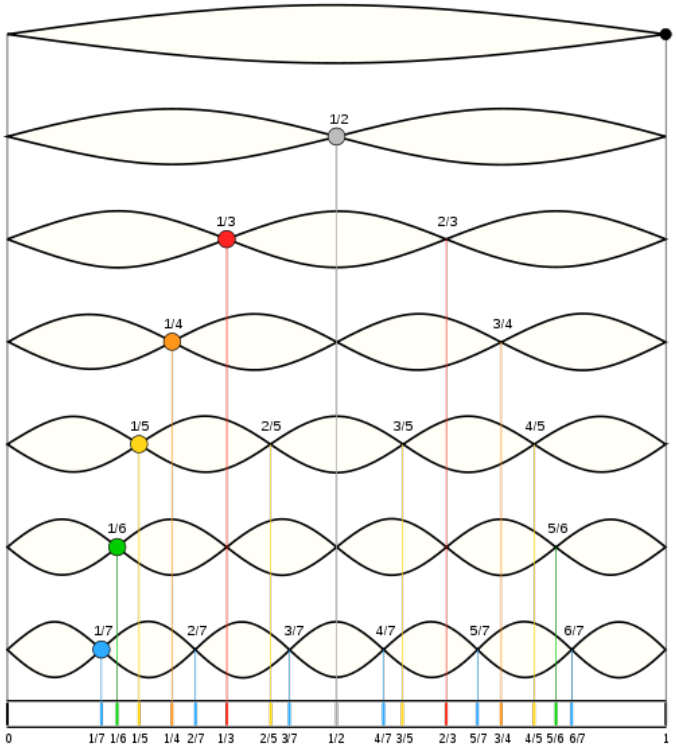

Figure 1: Visual Representation of Different Harmonics.

Pitch detection is a fundamental problem in several different areas of study, from speech recognition to musical pitch detection and score composition. Simply put, pitch detection uses a selected algorithm to determine the pitch and/or fundamental frequency of a periodic signal. Cleanly generated sounds can be analyzed as simple sine waves, while musical notes pose more of a challenge with the presence of noise, timbre, and overtones. Most pitch detection algorithms work by converting the sampled data into either time or frequency-domain for analysis.

2.2 Pitch Detection in Software vs. Hardware

At the beginning of the project, we had to choose a method of pitch detection either through software, hardware, or a combination of both. Many pitch estimation libraries exist for Java that could possibly be used to create a guitar tabber, and hardware options such as the circuits used in handheld guitar tuners could also be utilized. In the end, we chose the software route mainly for three reasons: Theres no special equipment needed other than a microphone, no complex setup is required to record the guitar, and it ideally should work with nearly every guitar, whether it is acoustic or electric. Because all of the pitch estimation is done through algorithms on a computer, no special circuits have to be created and tested thoroughly. This saves us significant time and money simply because we dont have to build multiple prototypes that may

or may not have what were looking for. It also enables anyone with a computer and a microphone to use our software, since they wont have to buy any special module to connect to their guitar and/or computer. Not having to bring an extra piece of equipment around when you want to make a tab away from your home computer means our software can be used almost anywhere without the need to set something up, which might deter some users from using it in the first place. No setup means less obstacles to creating a tab. And where hardware might not work with all types of guitars, software can be more easily fine-tuned to work with any guitar that a user might have. It is much easier to change parts of code than it is to physically change hardware.

2.3 Existing Pitch Detection Projects

Our project began with a search for a suitable audio processing library with pitch detection and onset detection. Over the course of two quarters, more than ten different libraries were tried and tested to see if they fulfilled our requirements. Initially, we were pointed towards a library called Aubio (`www.aubio.org`). This library seemed comprehensive, having pitch and onset detection examples, beat detection, and a host of other music analysis tools. Unfortunately, the library hadnt been updated for at least a couple years. The documentation was lacking, leading to quite a few roadblocks when it came to building the library itself. One of us was able to successfully compile and run on a Mac with the help of the Aubio creator himself, Paul Brossier. However, the remaining two members of the group were unable to get the library compiling on their respective Linux virtual machines, which would have made group collaboration much more difficult. Thus, we decided to forego the Aubio library in search of something else.

The next attempt made use of a project called 5k tuner. This project was intended to be a very simple, dependency-less method of tuning a guitar with pure Java code. The tuner used an autocorrelation method, which determines pitches by locating the recorded audios peaks. It then compares the peaks to other known waveforms until it finds something that is identical, thus finding the fundamental frequency. All members of our group were able to compile and run this code, and preliminary tests were promising. However, further testing revealed a puzzling issue: above a certain frequency, the tuner simply stopped detecting pitches. An audio recording of a simple locrian mode up and ionian scale down showed that at frequencies above 260 hertz, the notes were lost. The autocorrelation method read all other notes lower than 260 hertz on the way up, and on the way down. Another test with only the missed high notes revealed that the algorithm was reading those notes at half of their expected frequency. Since autocorrelation is known to be a less-than-ideal method for detecting pitch, we decided to forego it as well for yet another pitch detection method.

The next method we tried was the use of a library called TarsosDSP. This library, also entirely written in Java, provided several pitch detection algorithms: YIN, Macleod Pitch Method (MPM), Average Magnitude Difference (AMDF), and Dynamic Wavelet. Both YIN and MPM are iterations of autocorrelation with improvements. To our dismay, all of these methods produced the same cutoff result as the simple autocorrelation

method. This led to a frantic search for a library that overcame these issues, but one was not found in time. Thus, we decided to conduct a thorough test on a couple of the promising projects to see if the nature of the problem could be identified.

The projects tested, along with a brief description, are as follows:

- TarsosDSP (`https://github.com/JorenSix/TarsosDSP`)

An audio processing library built for Java. It contains a number of different pitch detection algorithms such as Yin, the Mcleod Pitch method, and a "Dynamic Wavelet Algorithm Pitch Tracking" algorithm. It was very easy to import into our program since we did not have to include any other external libraries along with it.

- ZetesWithApps (`https://github.com/decouto/ZetesWithApps`)

A pitch detection program built for Java. It uses FFT to try and guess the correct frequency for a given note.

- PolyphonicPitchDetection (`https://github.com/tjrantal/PolyphonicPitchDetection`)

An open source pitch detection library built for Java. It uses FFT to try and correctly guess the right frequency for a given note. Instead of giving it a file to analyze, the user plays notes through their microphone for the program to use.

- 5KTuner (`http://www.psychicorigami.com/2009/01/17/a-5k-java-guitar-tuner`)

A guitar tuner programmed in Java. It uses autocorrelation instead of FFTs in order to determine the pitch of a given note. A window displays the frequency of the note, a slider which tells you what note is being played (A, B, C, etc.), and a chart that shows the autocorrelation function of the sound.

2.4 Pitch Detection Testing and Results

A number of electric guitar recordings were made in order to accurately and consistently test the chosen pitch detection libraries. The same frets were recorded on all six strings of an electric guitar individually to test whether the octave problem was dependent on the string. These frets were 0, 2, 3, 5, 7, 8, 10, 12, 14, 15, 17, 19, 20, and 22. Care was taken in playing the notes for a decent interval, while separating them clearly with brief pauses in between. This would avoid any trouble the project might have in detecting note transitions. Each recording was given to every pitch detection library and the outputted frequencies were recorded and graphed against each other. For the Polyphonic Pitch Detection project, the notes were played live due to the greater difficulty in converting it to use a file. In this case, the note detected was recorded and simply compared to the expected value instead of graphing it. For all projects, fluctuations in note detection were not as crucial, since a filter could be easily written to remove any spikes or dips in frequency not related to the notes being tested. The graphs were then compared against the ideal results to determine the nature of the issue. Some graphs were split in order to accommodate their difference in number of samples per note.

The first projects tested were the two most viable options for our project: the autocorrelation method from the 5k tuner, and the TarsosDSP library. These were easily converted to accept an audio file, as our project intended. They both provided console outputs of the note at each sample, making them easy to compare.

The worst performing string, by far, was the low E string. When playing the sequence of fourteen notes all the way up the fretboard, the two projects were only able to capture at most the 7 lowest of those notes. Any notes above 147 hertz were simply missed. Unlike higher strings, however, the notes it did capture were of the correct frequency. The results can be seen in the graphs below.

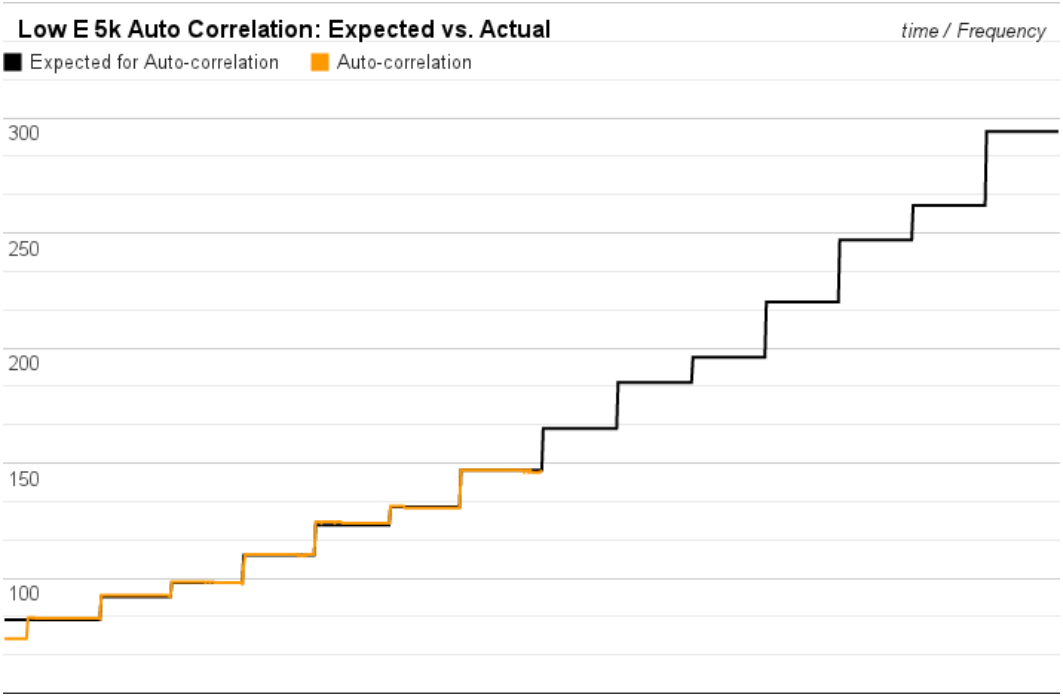

Figure 2: Expected vs Actual note detection for the low E string when using autocorrelation.

The results from the low E string established some patterns for each algorithm that were seen on other strings as well. For instance, all of the TarsosDSP algorithms were considerably jumpy at the beginning and end of the detected note range. A couple of the algorithms, like AMDF and MPM, exhibited severe spikes or drops during note transitions. As stated before, this could be easily filtered out by setting up a window that checks to see how many consecutive samples are the same before determining a note is being played. The fluctuations can be seen in the Figure 6 below.

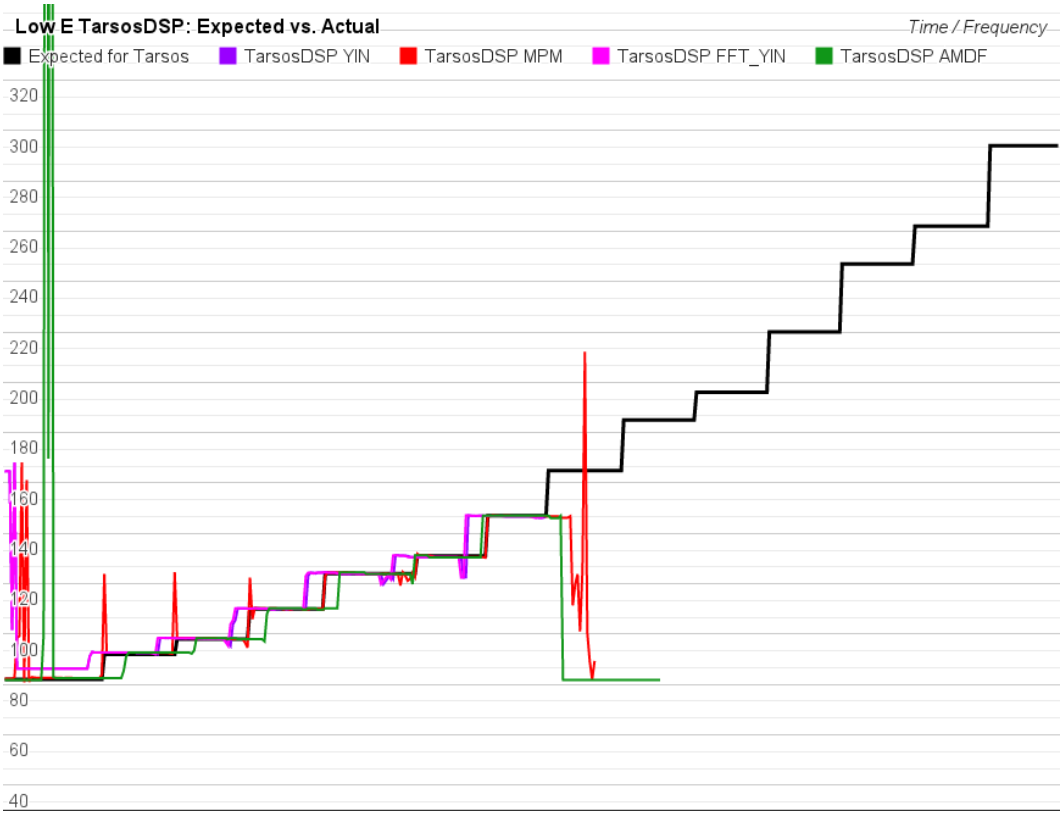

Figure 3: Expected vs Actual note detection for low E string with TarsosDSP algorithms.

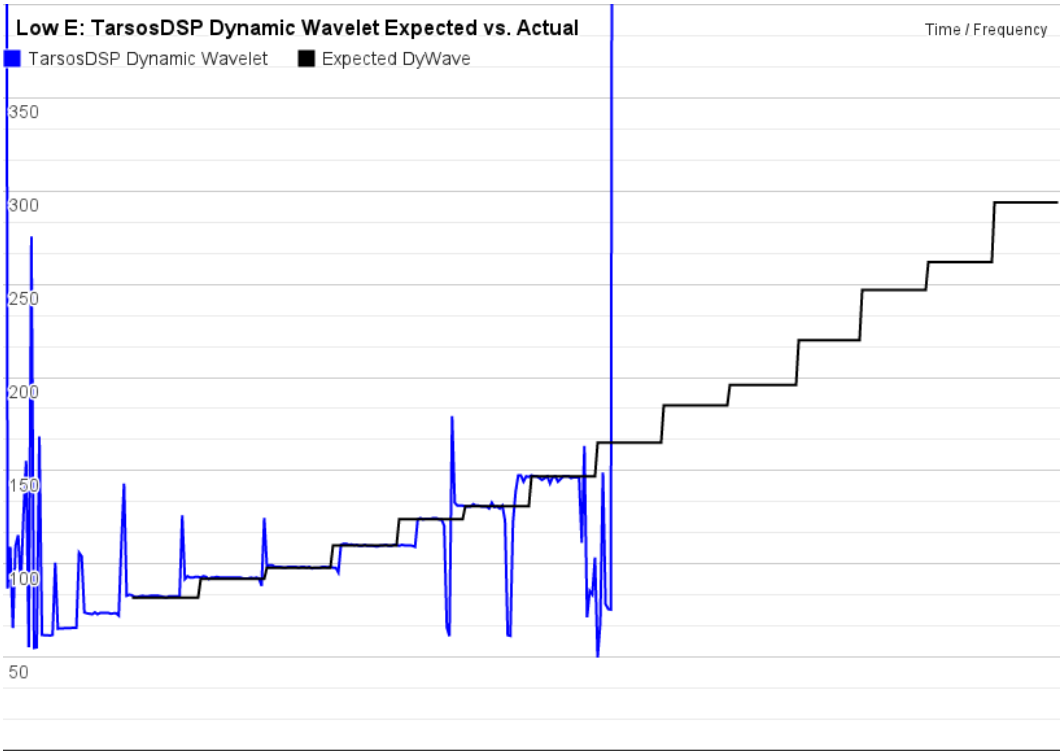

Figure 4: Expected vs. Actual note detection for low E string with TarsosDSP Dynamic Wavelet.

The frequency that marked "no man's land" held true on the A string as well, but in the opposite direction: any frequency below 147 hertz (110, 123, 131) was omitted. This is the 5th fret on the A string, or the open D note. On the D string, this note was omitted, and the next note (165 hz) was also omitted depending on the algorithm. For A and D, anything above these notes was detected, but consistently at half of the actual frequency of the note.

Conclusions about the cause of this mysterious behavior are hard to come by. Perhaps at frequencies above 147 hz on the low E string and below on the A string, the overtones have a certain quality that makes them very hard to detect? It could possibly be due to the thickness of the string, considering the low E string is much thicker than the A. This conclusion was supported by the test results from the G string. The G was the first string for which every single note was detected. Coincidentally, the G string is also the first string on most electric guitars (including the guitar used) which is not wound.

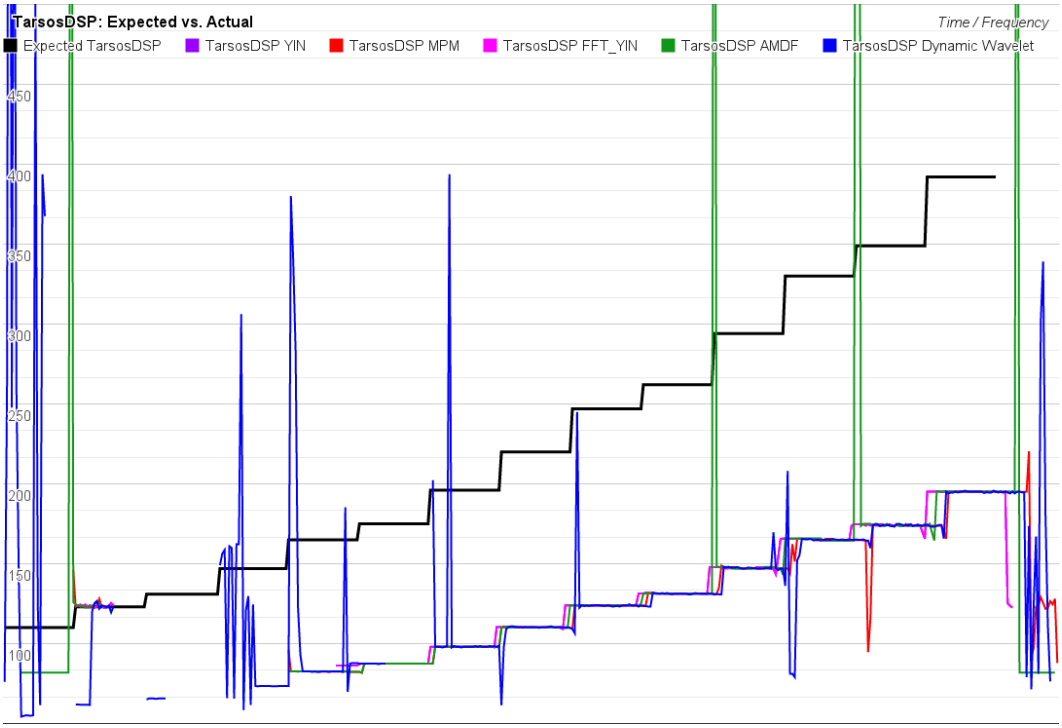

Figure 5: Expected vs. Actual note detection for the A string, showing that notes below 147 hz are missed, and that notes above 147 hz are detected as half of their actual frequency.

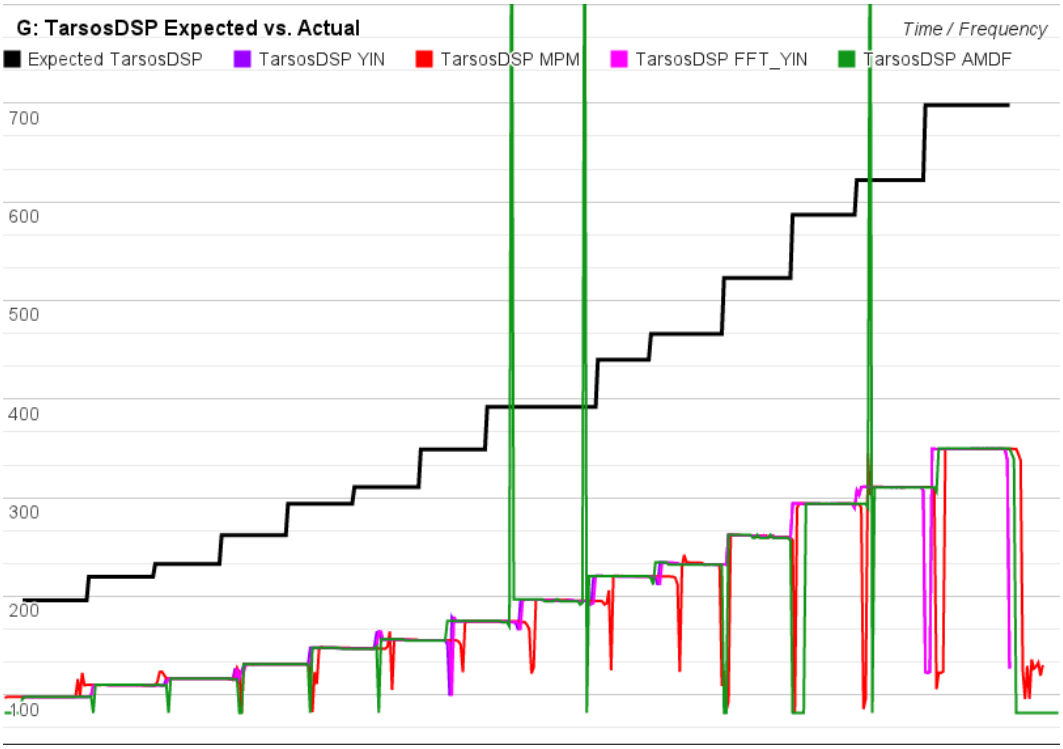

Figure 6: Expected vs. Actual note detection for G, which shows all notes are detected at half frequency.

To investigate the wound string problem, we tested with a heavier gauge set of strings which has a "wound 3rd" G. Playing the low E, wound G, and the high E through the autocorrelation and YIN methods revealed nearly identical results as the other set of strings. This suggests that the issue is not entirely with the string being wound, although the winding on the G string is considerably thinner than that of the other strings. The issue does not seem to depend on the age of the strings either, as shown by the identical results on the low and high E.

The last two unwound strings, B and high E, behaved the same as the G; all of their notes were detected. It was inconclusive to say that the detection got better with higher frequencies. The missing notes were unfortunately a deal-breaker for any sort of software-compensation for the notes detected at half-frequency. If all notes were detected at half frequency, it would have been an easy fix. The lack of a note at all makes it impossible to know whether or not you're missing notes when attempting to identify pitches in a played song. Short of some type of hardware, there is no way to fill the void.

Next, the Polyphonic Pitch Detection (PPD) project was tested. This project provided a GUI interface which displays a real-time reconstruction of the wave, and the corresponding FFT. The frequencies detected are displayed in the upper left corner of

the screen. Occasionally, multiple frequencies are listed (i.e. F0, F1, F2), but only the F0 frequency was used to determine its accuracy. Unlike the 5k Tuner and TarsosDSP projects, the PPD project was able to detect every note. However, the frequency displayed was off by either a factor of 2 or 4, depending on the note. The results for the lowest and highest string can be seen in table 1 below.

Fret	Low E			High E		
	Expected Freq (hz)	Actual Freq (hz)	Difference Factor	Expected Freq (hz)	Actual Freq (hz)	Difference Factor
0	82	333.8	4.1	330	656.8	2.0
2	92	366.1	4.0	370	742.9	2.0
3	98	387.6	4.0	392	786	2.0
5	110	441.4	4.0	440	872.1	2.0
7	123	495.3	4.0	494	979.8	2.0
8	131	516.8	3.9	523	1044.4	2.0
10	147	581.4	4.0	587	1173.6	2.0
12	165	323	2.0	659	1313.5	2.0
14	185	366	2.0	740	1475	2.0
15	196	387.6	2.0	784	786	1.0
17	220	441.4	2.0	880	882.9	1.0
19	247	495.3	2.0	988	990.5	1.0
20	262	527.6	2.0	1047	1044.4	1.0
22	294	592.2	2.0	1175	1173.6	1.0

Table 1: Results from the PPD test with the low E and high E string.

The low E and high E results show a considerable octave error, which varies depending on the actual note played. On the low E string, both the open note and the 12th fret–octaves of each other–are read as the same note, around 323-333 hertz. As with the previous two projects, this would make it impossible for compensation in software to correct for the difference factor. There is no way to discern between which fret is played when the frequency read is exactly the same. The same goes for the 3rd and 15th frets of the high E string, which are also octaves of each other. While this seems better than the missing notes from the previous projects, it does nothing to solve the initial octave error we discovered.

The next library to be tested was ZetesWithApps. However, getting the supplied code to work in the first place was a monumental task. Many hours were spent debugging and rearranging code in order to get a plausible output from the pitch detection algorithm. Unfortunately, even with a lot of effort expended to get everything to compile and fit together, the outputted frequencies made no sense in the context of what was played (frequencies of 1 reported when a scale was being played). Instead of spending more time trying to fix the library code, we decided to stop working on it and move onto the next library.

The next task was to determine which one would be most suitable for our program, despite its flaws. The graphs point to autocorrelation as the most steady and reliable method, a filter notwithstanding. All of the tested projects had the same octave shortcomings, so it came down to which were the easiest to implement with our project. This limited our choices to TarsosDSP and the 5k Tuner. Within the TarsosDSP project, the best option appeared to be the YIN algorithm, which presented the most steady output. However, these tests were only conducted on one guitar. One of the goals of this project is for everyone to use it, no matter their equipment. Thus, we ran some very small tests with an acoustic guitar to see which of these algorithms worked best.

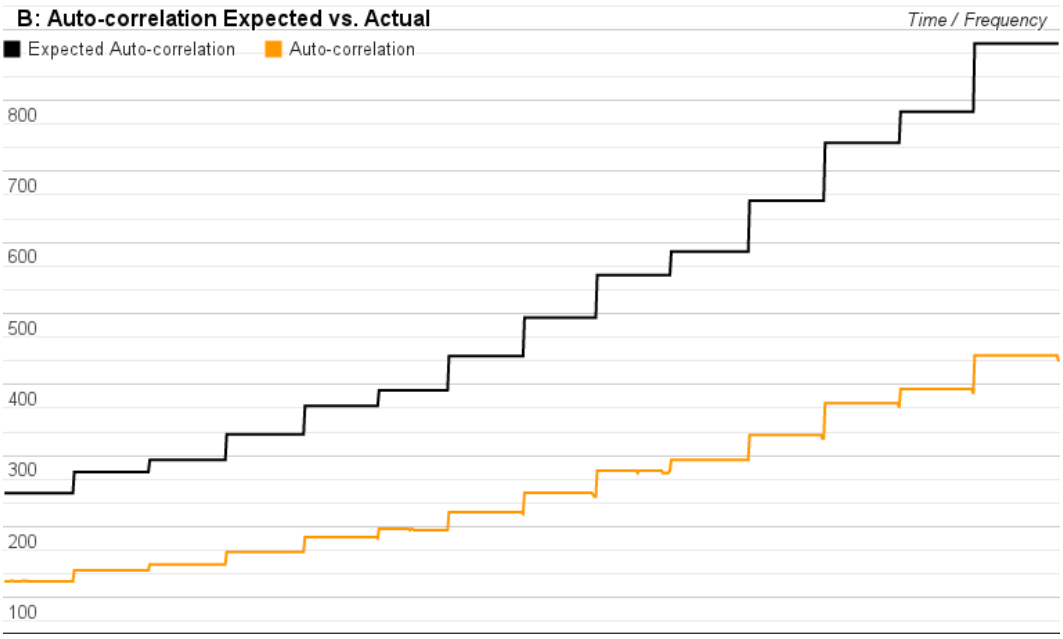

Figure 7: Expected vs. Actual note detection for B string with Autocorrelation, demonstrating its steady note detection and reliability.

The tests revealed that the TarsosDSP YIN algorithm was the most adept at covering more notes. Thus, to establish a working model for our program, we decided to use the YIN algorithm. Of course, there is more research to be done to determine a better software solution for detecting pitches on a guitar.

In summary, there does not seem to be an open-source Java library that is well supported and documented, which solves the octave error issues we encountered. Several of the libraries were effective at note detection, but it generally was the wrong octave. Perhaps a viable combination of guitar, amp, strings, and software could be found which works, but this would severely limit the broad applicability desired with our project. With more time and research, a better solution could be found.

3 Part II - Tablature Generation

3.1 Tablature Generation Overview

Tablature (or tab for short) is a method of notating a musical piece using provided finger positions rather than actual musical notes. This method is commonly seen for fretted string instruments, such as the guitar. When guitarists first learn how to play their new instrument, unless it is part of a guided class, they generally start with a guitar tab rather than standard musical notation. Guitar tabs offer a notation that is much easier for the player to pick up, allowing them to place their fingers on the fret and string number indicated, as opposed to learning all the musical pitches on the fretboard and associating those with fret positions.

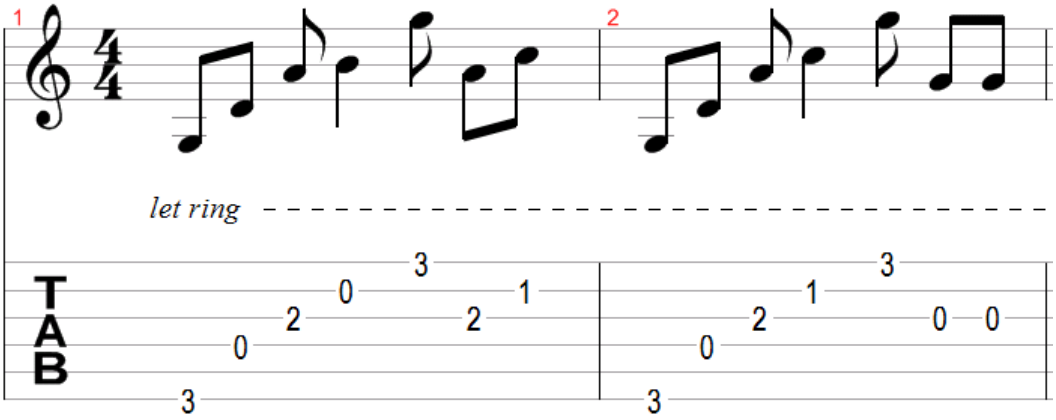

Figure 8: Guitar Tablature Notation shown on the lower row, with Standard Musical Notation shown above (from Sungha Jung's *"Always Be Here"* instrumental guitar piece as an example of "good" guitar tablature).[11]

3.1.1 The Difference Between "Good" and "Bad" Tablature

The relationship between two musical notes, known as an "interval", can be described in a sense of how many whole and half-steps it takes to get to one note from another. This can be easily represented with a fretboard.

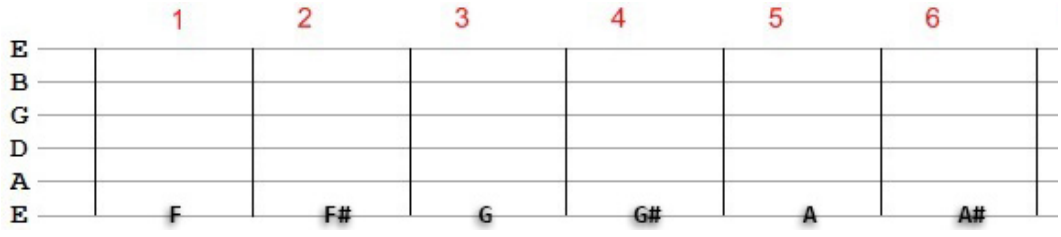

Figure 9: Guitar Fretboard shown with Standard Tuning and the low-E string labeled.

As seen above, on a guitar fretboard each movement of one fret up the neck translates into a half-step higher in musical pitch. For example, moving from Fret 1 to Fret 3 on the low-E string results in moving from the note F2, up two half-steps (or a whole-step) to the note G2. With standard guitar tuning, each open string is tuned to five half-steps above the previous one (e.g. The D string is five half-steps higher than the A string), with the only exception being the G to B string transition, having only four half-steps between them. Since these strings are relatively close together in pitch, there are multiple ways to play a certain note. For example, as seen in the figure, an A2 note can be played on the fifth fret of the low-E string, which is the same A2 that can be played by the open A string. The same holds true for other notes throughout the fretboard, with some notes having three or four different possible positions. This presents an interesting problem in that since a note can be played in multiple positions, there are many different ways to play a chord or progression in a song. Naturally, some choices for fret positioning are better than others, depending on the physical size of the player's hand and the position of their hand since the last note was played. An automatic tablature generation program would have to take this into account in order to generate useful tablature for the player, and one that is humanly playable.

Figure 10: How the above original/good tab for *"Always Be Here"* would look like when fretted on a guitar. This positioning is well-placed and easily playable, resembling a basic chord shape. The fingers are in line with each other and the frets are close together to make it comfortable.

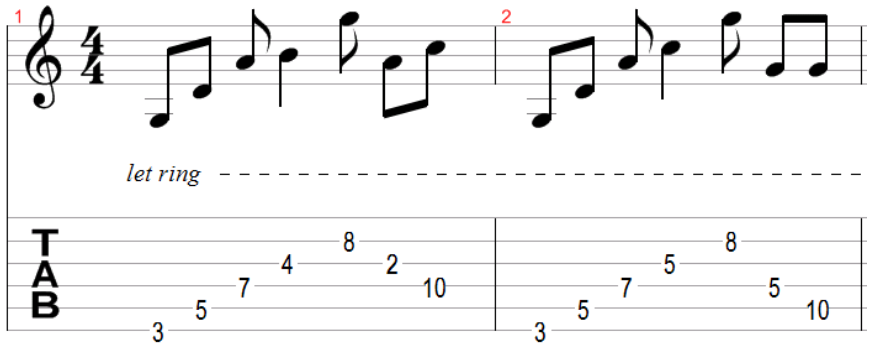

Figure 11: An example of "bad tablature" for the same section of *"Always Be Here."* Notice how far spaced out the frets are, and the complications it adds to the piece. The frets on the guitar have changed, but the notes have remained the same, resulting in the same sound.

Figure 12: How the above "bad tablature" for *"Always Be Here"* would look like when fretted on a guitar. As noted earlier, this is not a very comfortable finger position for playing this song. The fretting hand and fingers are stretched out across the fretboard, which results in an arrangement that is both harder to play and more uncomfortable.

3.1.2 Benefits of Automatic Tablature Generation

Currently, tablature is generated by manually inputting fret positions into a guitar tabbing software, such as the popular Guitar Pro package, or typing it out in ASCII format, which can take a lot of time and effort depending on the length and intricacies of the piece. As shown with the other examples in this section, the process of creating tab is rather meticulous because of the ability to play the same note in multiple places. The composer must take this into account when tabbing the song to create something that is as easy and comfortable to play as possible. This is a process that could greatly be sped up by auto tabbing, providing tablature that is identical to the intentions of the creator, or at least a very reasonable and playable starting point.

3.2 Tablature Generator Research and Design

3.2.1 Existing Research and Designs

There have been previous attempts at tablature generation through machine learning methods like Genetic Algorithms or Neural Networks, as well as other clever approaches through path finding. A Genetic Algorithm searching heuristic is used to generate solutions to various search optimization problems through mimicking natural selection to shrink the reasonable search space. One of the methods for generating tablature was described by Samir Sayegh with his "optimum path paradigm."[10] This algorithm tries to apply different hand positions as being different sequences of fretboard positions, and then attempts to find the shortest path through all of the positions. Due to the amount of possible combinations of hand positioning and fretting, the algorithm has a rather lengthy runtime. Another project from a conference on interdisciplinary musicology in Austria, 2004 attempted to use a path finding method that weighted the difficulty of each individual finger to generate tablature[6]. The results were generally close to an actual performer's method of playing a song, but were limited in certain cases where there was a large amount of neck movement.

3.2.2 Our Tablature Generation Method and Design Decisions

Part of our original goal for this project was to try and create an all-inclusive tool that handles both aspects of pitch detection and tablature generation. Many aspects of the program are intended to be modular to allow for other algorithms to be tested and compared with each other. In the tablature generation aspect of the project, we opted for a more simplistic, greedy approach with a fast runtime as another goal. Initially, CLOSEST_FRET and CLOSEST_STRING methods of cost were used to establish a baseline for the algorithm. As the names imply, they determine the next finger position by whichever note has the closest fret to the previous note, or the closest string appropriately.

These methods are certainly not ideal for creating decent tablature, but were able to demonstrate the simplicity of the algorithm and its runtime. A derivative of the closest fret/string methods was created as CLOSEST_BEST, which attempts to apply different weights to the amount of frets or strings that need to be traversed to play the

next note. In most cases, it is easier for the player to move up or down a fret than it is to change strings, so a 2x weight was added to string changes. This simple change gave us better results, but still did not make the best decision in some cases due to the tab requiring the player to continuously move up or down the neck, leading to awkward positioning.

Our final method of tablature generation has a running geometric average of the positions of the last played notes to give a basic idea of where the player's hand might be. Since the average person will fret the guitar with four fingers (thumb holding the neck, four fingers for fretting), a distancing average of the last 4 notes played is used to determine the next best finger position. This method proved to be rather effective and generated tablature that was either identical to the manually tabbed version, or a modified version that was still easily playable. The main issue with this method is that since it is a greedy algorithm, there is no form of pattern recognition, so a certain repeated passage in a song might be tabbed differently at different points in the song.

3.2.3 Tablature Output

Four different types of tabbing files were considered for this project: Guitar Pro's .gpx, Powertab's .ptb, TuxGuitar's .tg, and regular ASCII text. The .tg format was eliminated early on due to the relatively small user base of the TuxGuitar software and its incompatibility with most other popular tabbing software (mainly Guitar Pro). The .gpx format was also eliminated early due to its proprietary nature. Users can only create such files from the Guitar Pro software itself, and no documentation exists on how to create a file outside of the GP software. The .ptb file format has some documentation regarding how to create a file in code such as information regarding the header, but a complete description was not found. Further research yielded an old project on converting powertab files to other different file formats, but it did not contain any information on how to correctly create a .ptb file. In the end, we decided on ASCII text format due to the simplicity in creating such a file, its widespread acceptance on many guitar tab websites, and its non-reliance on software in order to be read correctly.

3.3 Tab Generation Testing and Results

3.3.1 Simple C-Major Scale

For initial testing, a standard C-Major scale midi file over two octaves was run through the different greedy algorithms to obtain results.

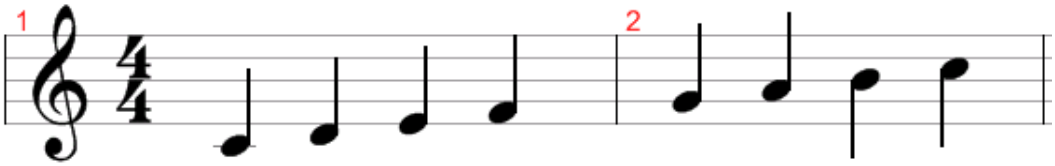

Figure 13: A one-octave standard C-Major scale from middle C (C4) up to C5.

```
Using CLOSEST_FRET method...
------------------------------------------------------------------
------------------------------------------------------------------
------------------------------------------------------------------
-------------------------------------------------------22--24--
-----------------------------------------------22--24--26----------
-8--10--12--13--15--17--19--20--22--24--25---------------------
```

Figure 14: Results of the CLOSEST_FRET method on the two-octave C-Major scale.

```
Using CLOSEST_STRING method...
-----------------------------------0---1---3---5---7---8--10--
-------------------------0---1---3-------------------------------
-----------------0---2-----------------------------------------
---------2---3---------------------------------------------------
-----5-----------------------------------------------------------
-8----------------------------------------------------------------
```

Figure 15: Results of the CLOSEST_STRING method on the two-octave C-Major scale.

```
Using CLOSEST_BEST method...
------------------------------------------------------------
------------------------------------------------------------
------------------------------------------------------------
-------------------------------------------------------22--24--
-----------------------------------------22--24--26----------
-8--10--12--13--15--17--19--20--22--24--25--------------------
```

Figure 16: Results of the CLOSEST_BEST method on the two-octave C-Major scale.

```
Using GEO_AVG method...
-----------------------------------------------------7---8--10--
-------------------------------------------8--10--------------
---------------------------------7---9--10----------------------
---------------------7---9--10-----------------------------------
---------7---8--10-------------------------------------------------
-8--10------------------------------------------------------------
```

Figure 17: Results of the GEO_AVG method on the two-octave C-Major scale.

The GEO_AVG function creates a much better arrangement than what was obtained from the other cost methods, and results in a tablature that would likely be created by a player tabbing it by hand. With this arrangement, the player's hands will stay in the same general Fret 8 - Fret 10 position on the fretboard and will simply need to move their hand across the strings to play this scale. The other generated arrangements are technically correct, but are generally very awkward to play.

3.3.2 Complex Arrangement

After the simple test with the C-Major scale, a much more complicated piece was used to further test the algorithm's abilities. Seen below is the original tablature as given by the composer. As with the C-Major scale test, a midi file containing the notes was used to run through the algorithm to produce results.

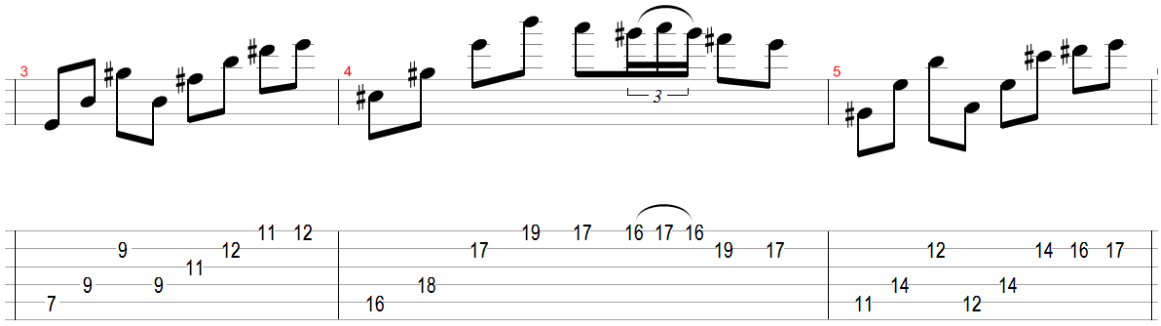

Figure 18: The introduction section from Neil Zaza's *"I'm Alright"* instrumental guitar piece.[12]

```
Using CLOSEST_FRET method...
---------4-------2---7--11--12----------12--19--17--16--17--16--14--12-----------7-----------9--11--12--
-----0-------------------------------9-------------------------------------------------5--------------
-------------4-------------------------------------------------------------------9--------------------
-2-----------------------------11-------------------------------------------------------7--------------
-------------------------------------------------------------------11-------------------------------
-------------------------------------------------------------------------------------------------------
```

Figure 19: Results of the CLOSEST_FRET method on *"I'm Alright"*.

```
Using CLOSEST_STRING method...
-----------------2---7--11--12-------4--12--19--17--16--17--16--14--12-----------7-----------9--11--12--
---------9---0-------------------2-----------------------------------------5-----------5--------------
-----4-------------------------------------------------------------------1-----------2----------------
-2-------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------
-------------------------------------------------------------------------------------------------------
```

Figure 20: Results of the CLOSEST_STRING method on *"I'm Alright"*.

```
Using CLOSEST_BEST method...
-----------------------------------------19--17--16--17--16--14--12---------------------------------
------------------------16--17----------17-----------------------------------------------------------
--------13----------16-------------------------------------------------16----------18--20--21--
----------------16------------------18-----------------------------14--------------------------
----14------14------------------16-----------------------------11--------------19--------------
12-------------------------------------------------------------------------17------------------
```

Figure 21: Results of the CLOSEST_BEST method on *"I'm Alright"*.

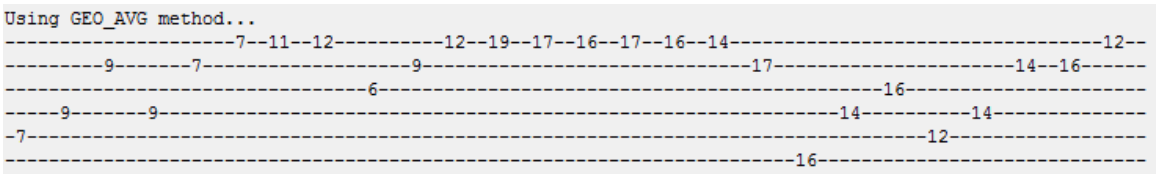

Figure 22: Results of the GEO_AVG method on *"I'm Alright"*.

As shown with the C-Major scale example, the GEO_AVG function produces a rather accurate arrangement when compared to how it is played when tabbed out by hand. Many of the sections are similar to the original tabbed piece, and while it is not identical to the original, it is still a playable and comfortable arrangement. The other cost methods produced awkward arrangements as with the previous test, with the CLOSEST_BEST arrangement at least having some semblance of balance and playability.

4 Conclusion

Although we were not able to completely finish the Automatic Guitar Tabber, we did learn a lot about the problems of software-based pitch estimation and generated a considerable amount data and information regarding it. We discovered that there is no currently available pitch estimation library that completely and accurately outputs the frequencies of any note it is given due in part to overtones and trouble with identifying the correct octave. Both autocorrelation and fast fourier transform algorithms ran into this problem, with no clear way around it. Graphs were created to illustrate this in hopes of highlighting the problem to any future developers working on pitch estimation programs. Producing the correct tablature for a given song also did not have a clear-cut solution, and made for the creation of a more complicated algorithm than expected in order to give more natural and easier to play tabs. All in all, working on this project made it clear that unforeseen setbacks can spring up at the most inopportune moments, and that getting an early start and following a set schedule can help alleviate those problems. Just with like learning how to play the guitar, persistence and great patience are key to getting the results you want.

References

[1] de Cheveingn, Alain, and Hideki Kawahara. "YIN, a fundamental frequency estimator for speech and music." Acoustical Society of America, 9 Jan. 2002. Web. <http://www.ircam.fr/pcm/cheveign/pss/2002_JASA_YIN.pdf>.

[2] Galembo, Alexander , and Lola Cuddy. "Acoustical Society of America; 134th Meeting Lay Language Papers." Acoustical Society of America. N.p., 2 Dec. 1997. Web. <http://www.acoustics.org/press/134th/galembo.htm>.

[3] Gerhard, David. "Pitch Extraction and Fundamental Frequency: History and Current Techniques." . University of Regina, Canada, 1 Nov. 2003. Web. <http://www.cs.uregina.ca/Research/Techreports/2003-06.pdf>.

[4] Hokin, Samuel. "The Physics of Everyday Stuff - The Guitar." The Physics of Everyday Stuff - The Guitar. N.p., 2014. Web. <http://www.bsharp.org/physics/guitar>.

[5] Marino, Jessie. "Cello Multiphonics." Cello Multiphonics. N.p., 2010. Web. <http://cellomultiphonics.blogspot.com>.

[6] Radicioni, Daniele, Luca Anselma, and Vincenzo Lombardo. "A Segmentation-based Prototype to Compute String Instruments Fingering." . N.p., 15 Apr. 2004. Web. <http://www.di.unito.it/ radicion/papers/radicioni04segmentation.pdf>.

[7] Six, Joren. " YIN Pitch Tracker in JAVA." YIN Pitch Tracker in JAVA. N.p., 9 Apr. 2010. Web. <http://0110.be/posts/YIN_Pitch_Tracker_in_JAVA>.

[8] Tuohy, Daniel, and W.D Potter. "GA-based Music Arranging for Guitar." . N.p., n.d. Web. <http://cobweb.cs.uga.edu/ potter/CompIntell/cec_arranging_paper.pdf>.

[9] Weisstein, Eric. "Half Step – from Eric Weisstein's Treasure Trove of Music." Half Step – from Eric Weisstein's Treasure Trove of Music. N.p., n.d. Web. <http://www.ericweisstein.com/encyclopedias/music/HalfStep.html>.

[10] S. Sayegh, "Fingering for String Instruments with the Optimum Path Paradigm", Computer Music Journal, vol. 13(6), pp.7684, 1989.

[11] Guitar Tabulature for "Always Be Here", by Sungha Jung. <http://tabs.ultimate-guitar.com/s/sungha_jung/always_be_here_guitar_pro.htm>.

[12] Guitar Tabulature for "I'm Alright", by Neil Zaza. <http://tabs.ultimate-guitar.com/n/neil_zaza/im_alright_guitar_pro.htm>.