Multi-pitch estimation based on mathematical models

A pitch estimation framework and harmonic overtone sieve algorithm

Luc de Jonckheere

April 8, 2021

Abstract

1 Introduction

Automatic Music Transcription (AMT) is a field of study which tries to convert acoustic recordings into some form of (digital) music notation [13]. This task can be divided into two subproblems, pitch estimation and onset detection [14]. Instead of trying to convert a song into music notation, this paper will focus on translating a guitar signal into a set of played notes in real-time. This problem is very similar to AMT, as both require pitch estimation and onset/offset detection, however, in our case we also have take latency into account.

Single pitch estimation can be accurately performed using simple mathematical models [12]. Multi pitch estimation is much harder as harmonic overtones are difficult to differentiate from the fundamental notes [17]. Because of this, most recent methods use machine learning or artificial intelligence. These methods are not well suited for real-time transcription. Also, no actual (mathematical) understanding of the problem is gained. Because of this, this paper will focus on mathematically solving this problem. Even if the methods researched in this paper might not outperform ML/AI solutions by themselves, they could be used to filter the input data for the ML/AI approaches. This might in turn improve the results of the AI/ML methods significantly.

Pitch estimation can be divided into a few subproblems. If a paper focusses on one of these subproblems, it also has to solve the other subproblems in order to test the system. Most papers do not publish their source code, which makes building upon other's work difficult. It also limits the research of the interaction between different solutions for the different subproblems. Because of this, we provide a framework in which can be built upon, as the implementation of a subproblem can easily be interchanged with another. The code can be found on "github.com/lucmans/dechord".

In short, the goal of this paper is developing a program which can translate a guitar signal to a set of observed notes in real-time. We also provide an algorithm which can estimate a set of played notes using the set of observed notes. The code should be publically available and easily extensible so others can optimize a subproblem without having to solve the other subproblems.

Note that this paper is a preliminary paper for a possible master thesis. The focus is on building a prototype which can perform monophonic AMT and presenting a first draft of a polyphonic version. Using this information, we will estimate if the thesis project is feasible.

2 Related work

Spectral peak picking based AMT methods are often deemed infeasible for real-time AMT [21], due to the relatively low resolution in the low frequencies when using the Fourier transform [5]. This problem cannot be alleviated by zero-padding the Fourier transform input, as it does not add any information, it merely increases the resolution by interpolation [10][4] and increases the computing time of the transform [18]. However, using quadratic interpolation, we can very accurately interpolate peaks within Fourier frequency bins [11]. Also, other transforms such as the constant Q transform [6] which do provide more resolution in lower frequencies are getting more popular, which will allow us to differentiate two notes on a very small interval.

A big problem in polyphonic pitch estimation is the occurrence of overtones [17]. As these overtones also have a periodic nature in the frequency domain, they can be detected using another Fourier transform [15]. This does however have a big impact on the latency, as performing a Fourier transform is computationally expensive. Also, notes which are at octave intervals have the same periodicity [20], which makes them difficult to distinguish.

Another problem is choosing the real-time constraint for processing the input. Some papers claim 140 ms is sufficient [9]. This is however ridiculously long, as it would be about a note difference when playing eight notes in 200 BPM. Also, as there is already a large latency due to how audio drivers work [16]. In practice, musicians often tweak their audio driver settings in order to gain 5 to 10 ms in latency, which makes the difference between being able to play faster pieces or not.

When limiting ourselves to transcribing a guitar, we can use physical limitations of the instrument to our advantage [8]. However, this forces the guitarist to use the guitar in a specific way. For instance, these methods fail when the strings are tuned to different notes.

3 Preliminaries

Note: Explain knowledge required for the methods described in the next section. For instance, latency we can control (speed of code and Fourier window size) and what we can't (audio driver settings). Fourier works on frames, which is a buffer filled with samples.

4 Pitch estimation

Automatic music transcription can be separated into two main problems; onset detection and pitch estimation [14]. In this paper, we will focus on pitch estimation. Because we always output the set of played notes for the last analysed frame, we do not require onset information. However, this information can improve the results of pitch estimation. For example, when an onset is detected, we can discard all samples from the transient.

As stated before, our goal is to perform the transcription in real-time. We choose a constraint of 6 ms. Most musicians work with a driver latency of 9 to 11 ms. On a good computer, this latency can be reduced to 3-5 ms. We will use these 6 ms of latency difference to perform our calculations in. Note that when playing more notes per second, latency becomes more of a problem. On slower pieces, we can get away with much larger latencies. In order to test this for yourself, we developed a tool which plays back the incoming audio signal with a user specifiable delay. This tool can be found under the "latency" branch in our GitHub repository.

Pitch detection can be broken down into a few subproblems. First, a frequency domain representation of the signal is obtained (e.g. Fourier transform). Then, significant peaks have to be selected from the frequency domain. Based on the found peaks, we have to determine what notes were actually played on the instrument.

4.1 Transform to frequency domain

To estimate the frequency components of a frame, transforms are used. Most commonly, the Fourier transform is used as it is the most researched transform and much is known about it [14]. Its main downside is that the frequency bins are constant is size, where the distance between notes increases exponentially. Because of this, low notes are hard to discern in the frequency domain where high notes have more resolution than needed. Because of this, other transforms such as the constant Q transform (CQT) are getting more popular.

When using these transforms, window functions have to by applied to the frames to prevent spectral leakage. Furthermore, the actual peak locations can be accurately interpolated using (Lagrangian) quadratic interpolation [11]. These concepts are further described in Section 5.1 and Section 5.2 respectively.

4.2 Peak picking

Peaks in the frequency domain correspond to predominant frequencies in the analysed frame. However, not every peak is of interest. Some peaks are generated by noise. Many of these peaks can be filtered by requiring a minimal signal level to be considered a peak. Furthermore, due to spectral leakage, peaks can be observed which are not in the original signal. This kind of noise can be minimized by choose a good window function [19]. In polyphonic signals, harmonic overtones may slightly interfere (TODO: Maybe make preliminary section about overtone dissonance), which will may lead to inconsistent and oscillating peaks.

On order to determine if a peak is significant enough, we first calculate a Gaussian average envelope [7]. For every point in a frame, the Gaussian average is a weighted average of all points in the frame. The weight is determined by the distance from the current point. The distance in number of bins goes through a Gaussian function to determine the actual weight. Here, higher values for σ make points close by weight more compared to distant points. How this envelope is precisely calculated and used for peak picking is described in Section 5.2.

4.3 Note sets

Given the spectral peaks from the previous step, we can try to calculate what notes they represent. Unfortunately, the found peaks do not only contain fundamental frequencies, but also harmonic overtones. This is only a small problem in monophonic transcription, as there is only one fundamental frequency and its overtones. However, in the polyphonic case, it is difficult to determine which peaks are fundamentals and by extension know how many notes are played at the same time.

To determine if notes are fundamentals or overtones, we use a ”harmonic overtone sieve”. Here, we look if any higher peak can be explained by the currently considered peak. The more other peaks a peak can explain, the more likely it is to be a fundamental. Moreover, if we cannot explain a certain peak, it is likely that there is a second note in the source signal.

5 Implementation

The program developed for this paper is written in C++ (11) and compiled with g++ (10.2.0). It uses SDL (2.0.14) for audio input/output and spectrum visualisation. FFTW (3.3.9) is used for efficient Fourier transforms.

There are many parameters which change the behaviour of the overall system. For this paper, we choose some parameters which seemed perform well, however, these parameters can likely be optimized. All parameters can be found in `config.h` along with comments explaining them.

By default, the program assumes an A_4 of 440 Hz. This can also be edited in `config.h`. Only 12 tone equal temperament (12-TET) tuning is currently supported. It could easily be changed to n-TET tuning, but non TET tuning is not easily supported.

5.1 Transform to frequency domain

Our program uses the Fourier transform to obtain the frequency domain representation. We choose this transform as FFTW provides a very fast implementation. This speed is necessary for the real-time constrains on our system.

The resolution of the Fourier transform is dependent on the number of samples in a frame. We sample the guitar signal at 192 kHz, which is the maximum sample rate for most audio interfaces. To allow guitarists to play mildly fast, we have to limit our frame length. If the frame size if too large, multiple separate notes may be included in one transform, which would make it seem like the notes were played at the same time. Moreover, frame sizes which are a power of two can be transformed more efficiently, which further restricts are frame size choice. We chose a frame size of $2^{14} = 16384$ samples, which at 192 kHz is 0.085 seconds (11.72 frames per second). This gives us a Fourier bin size of 11.72 Hz. At first glance, such a large bin size seems like a big problem, as the smallest interval that can be played on a guitar (G#$_2$-A$_2$) is about 6 Hz. However, the overtones of these notes will have more Hz between them, which makes to possible to differentiate the two notes.

Before applying a transform, a window function is often applied to minimize the artefacts produced by the transform. In this paper, we mainly use the Blackman-Nuttall window, but the following windows are also provided: Hamming window, Hann window, Blackman window, Nuttall window, Blackman-Harris window, Flat top window. Other window functions can easily be added.

5.2 Peak picking

Three different peak picking algorithms have been implemented. The first picks every local maximum (every `i` where `data[i - 1] < data[i] && data[i] > data[i + 1]`). The other two algorithms pick peaks based on a Gaussian average envelope. One algorithm picks all peaks which are above this average and the other only picks the highest peak for each spectral lobe above this average. Using this envelope, we effectively filter small local maxima close to large local maxima.

The Gaussian average envelope is calculated for every Fourier bin i by taking the weighted average of a arbitrary number of bins around i. The weights are determined by a Gaussian function ($e^{-\pi(\frac{x}{\sigma})^2}$). The value of sigma and number of bins around i can be changed in `config.h`. Higher values for σ makes points around the considered point have a higher weighting. We choose a value of 1.2, but this value could be optimized for better results.

Even though this envelope works well for parts of the spectrum with peaks, it will still find peaks in noisy areas without peaks. This can be prevented in two ways. The first is setting a threshold which a peak has to exceed. This will result in less sustain, but works well. Secondly, we know that peaks outside the frequency range of a guitar cannot be a fundamental frequency. We can discard peaks lower than the lowest note, but peaks higher than the highest note still provide us information as will be described in Section 5.3

It is possible to determine the actual peak location within a Fourier bin using quadratic interpolation [22].

Here, we calculate a value $p \in [-\frac{1}{2}, \frac{1}{2}]$, which is the location of the actual peak relative to the found peak in number of Fourier bins. In order to calculate p, we need the height of our found peak ($y(0)$) and the height of the two neighbouring bins ($y(-1)$ and $y(1)$). As quadratic interpolation is more accurate in the dB (logarithmic) scale, we define:

$$\alpha = \log y(-1)$$
$$\beta = \log y(0)$$
$$\gamma = \log y(1)$$

Then, we can calculate p as follows:

$$p = \frac{1}{2} \frac{\alpha - \gamma}{\alpha - 2\beta + \gamma}$$

Given the Fourier bin size, we can calculate the frequency of the peak i by multiplying the bin size by $i + p$.

5.3 Note sets

Using all the found peaks, we can determine which peaks are overtones and which are fundamentals. First, we determine which notes the frequencies represent and calculate the error on these notes. The error is calculated as the number of cents from the closest harmonic note. The errors are different for every overtone (the series of errors is constant for every harmonic overtone series), as overtones are dissonant compared to equal temperament tuning. See Table 1 for an example.

n	f_{overtone}	closest note	f_{note}	error
0	261.626	C_4	261.626	0
1	523.251	C_5	523.251	0
2	784.877	G_5	783.991	1.955
3	1046.502	C_6	1046.502	0
4	1308.128	E_6	1318.510	-13.686
5	1569.753	G_6	1567.982	1.955
6	1831.379	$A_6^{\#}$	1864.655	-31.174

Table 1: Example of overtone series and the errors compared to the closest note

If only harmonic notes were played on a guitar, we could filter all notes with an error higher than an arbitrary threshold. However, as most guitars are not perfectly intonated and strings detune by pressing them down on a fret, this information has to be used with care. Correctly using this information is out of the scope of this paper.

Creating a monophonic algorithm is relatively easy given the frequencies of the peaks. For instance, we can iterate over all found peaks and check if every higher peak can be a overtone of the considered peak. After counting the number of overtones for every peak, we can pick the peak with the most overtones.

Due to small errors, the overtones are not exactly integer multiples of the fundamental. As the distance between notes increases with frequency, we cannot set a constant threshold of error in Hz. Instead, we calculate the error in cents and choose a threshold for this error (configurable in `config.h`). This also allows for equal error tolerance above and below the note, which percentage based methods would not.

Polyphonic note detection can use a algorithm similar to the monophonic algorithm. As the number of played notes in unknown in the polyphonic case, we have to determine if a note is a fundamental or overtone. We cannot just set a threshold for the number of overtones a fundamental should have, because the number of detected overtones is very inconsistent. To solve this, we present a harmonic overtone sieve algorithm. Instead of directly looking for fundamentals, we try to sieve away all harmonics until we are left with fundamentals. We start at the lowest peak end mark all its possible overtones. If there are still unmarked peaks left, they are likely from another note and we sieve its overtones away. We can repeat this process until no more peaks can be sieved. If two peaks are not distinguishable in the root notes but are in the overtones, this algorithm will report the first distinguishable overtone as the root note. It is possible to calculate what the root note should have been, but that is outside the scope of this paper and will be further discussed in Section 8.

After this, we can remove all peaks which are outside of the frequency range of a guitar. The minimum and maximum can be changed in `config.h` to accommodate for different tunings of guitars with extra strings.

5.4 Latency

The latency of our system can be divided in two different latencies, the time to fill a frame with samples and frame processing time. Because our algorithm works on a ”per frame” basis instead of a ”per sample” basis, we have to wait until the audio driver has fetched enough samples for us to fill a frame. The time it takes to start analysing a frame after receiving the first sample of this frame can be calculated by dividing the number of samples per frame by the sample rate. This is equal to the length of a frame. In practical terms, it is the maximum latency on detecting a newly played note. The average latency is obtained by dividing this number by two.

We use a frame size of 16384 samples and a sample rate of 192000 samples per second, which equates to a frame length of 85.3 ms. This is well over our realtime constraint. Unfortunately, we cannot change this value too much. The sample rate is already at the maximum of most audio interfaces. We also cannot lower the number of samples per frame while using the Fourier transform, as it would further reduce the frequency resolution. However, as will be explained in Section 8.1, this might not be a problem, as we can reduce this latency by employing multi-threading.

We can control the second kind of latency by optimizing our algorithms. In Section 6, we will measure the processing time of the different algorithms.

6 Experiments

We will perform a few experiments to test our system. The used audio equipment has a large effect on the results and in turn effects what parameters work best for the system. During development, an electric guitar with active EMC humbucker pickups was used. To convert the analog signal to a sampled digital signal, a Behringer UMC404HD was used. This audio interface has a dynamic range of 100 dB. Like most audio interfaces, this is much less than the dynamic range of a sample (144 dB for 24 bit integer). We opted not to amplify the signal in software, as it will distort it; especially if not done carefully. The tests were performed on an AMD 3700X CPU.

Often used datasets for evaluating pitch estimation focus on different goals than this paper. For instance, they focus on transcribing multiple instruments at once or having instrument invariance. They also do not focus on real-time transcription, which makes the problem much harder. However, in order to present some comparative measure, we will use the Mirex dataset for "Multiple Fundamental Frequency Estimation & Tracking" [3]. TODO (when I receive it): Short description of dataset.

We also constructed our own dataset to evaluate the performance. This dataset will be a more technical dataset; consisting of scales and intervals played on a guitar. Scales allow us to measure the monophonic performance. As scales are easy to play, multiple recordings should have the same results. This allows us to isolate and measure other variables, such as the effect of tempo or playing the scale higher/lower. Recordings of different intervals allow us to measure the polyphonic performance. By measuring all two note intervals, we can identify any problems with specific intervals. As the distance in frequency in an interval increases when the same interval is played at higher notes, we record the intervals at multiple root notes. We will also measure the performance on the four basic triads: the major triad, minor triad, augmented triad and diminished triad.

Note that when repeating our experiments, your results may very slightly. This is because the results of the Fourier transform from FFTW are not consistent [1]. This is especially noticeable in inconsistent latencies.

6.1 Monophonic performance: scales

To test the monophonic performance of our system, we first measure the performance of scales at different pitch ranges. The scales are played at a speed of two notes per second. This equates to playing fourth notes on 120 BPM. We play the scales at a relatively slow speed, such that this cannot become a limiting factor. The slower speed also helps to see if mistakes are the result of transients, as these mistakes should vanish as the note persists. We only play the C major scale, but change the starting note (mode). We play the full vertical scale from this starting note [2]. Each position has 17 notes, except for F_2, which has 18 notes. In Table 2, we summarize the results of the experiment. We denote the different starting positions as n_{start}. For every starting position, we denote the accuracy (a), the accuracy without transient errors (a_{trans}), the accuracy without octave errors a_{oct} and the total accuracy without both of these errors (a_{tot}). The accuracy is calculated by dividing the number of correctly determined frames by the total number of frames with notes. The only frames without notes are positioned at the start and end of a file. We omitted these frames from the results, as they are always correctly determined to be empty frames. Erroneous notes due to transients usually have a lower amplitude where octave errors usually have a higher amplitude compared the amplitude of the base note. This allows is to distinguish between them. This information cannot be used to detect transients in real-time, as we compare the amplitude of the erroneous note to amplitude of the correct note in the following frames.

n_{start}	a	a_{trans}	a_{oct}	a_{tot}
E_2	91.4%	95.2%	96.2%	100%
F_2	92.3%	96.2%	96.2%	100%
G_2	92.0%	95.0%	97.0%	100%
A_2	89.1%	96.0%	93.0%	100%
B_2	90.6%	96.2%	94.3%	100%
C_3	82.7%	95.9%	86.7%	100%
D_3	76.6%	95.7%	80.9%	100%
E_3	91.6%	97.9%	93.7%	100%

Table 2: Accuracy of monophonic f_0 estimation on C major scale at two notes per second for different starting positions.

We can see that the transient errors are very consistent. In our data, we noticed that the first played note almost always had a transient error. This could be because we start playing from a silent signal instead of changing from a previous note. The transient errors show up more often between lower frequency notes. The octave errors become more of a problem the higher we play. In the scales starting from A_2 and lower, we only see octave error at note changes. However, from B_2 and higher, we start to see consecutive octave errors. Here, between 50% and 100% of the frames of a note. During these errors, there are often noisy peaks in much lower frequency bands which throw the estimation process off. Other than these problems, the system worked perfectly.

6.2 Polyphonic performance: intervals

To test the polyphonic performance of our system, we let it determine the played notes in an interval. We play every interval within an octave to test if there are any problems with separating the notes in specific intervals. For instance, some interference may cause oscillating peaks in the frequency domain. Again, we try all the intervals at different root notes, as especially for intervals the pitch of the root note may make a big difference. As we mentioned before, when two peaks are too close together, we cannot distinguish them. Often, the second note is distinguished an octave higher. In Table 3 is a summary of the results. Here, we list the number of semitones n between the two notes in an interval, if both notes were correctly found. If not, we denote if the second note was detected an octave higher. This often happens, because we cannot distinguish two notes in small intervals at low pitches. The last column shows if there were other notes reported which were not transients.

n	both	octave	other
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			

Table 3: Performance of distinguishing notes in an interval with root note A_2.

In the tables, we can see that... (not enough overtones for low pitched intervals...)

6.3 Latency

The processing time of our system is very important, as it will directly impact the latency. We will only measure the time between receiving a frame from the audio driver until the answer is computed, as the time spent waiting on enough samples is fully determined by the number of samples in a frame and the sampling rate. The total processing time and the processing time of the individual steps and the total system can be seen in Figure 1.

Scales

Intervals

Figure 1: Processing time of the different algorithms

TODO: The results are what they are...

7 Conclusions

adsf

8 Future work

How to make results more consistent (onset detection, transient filtering). How to make the project more real-time (threading rolling transforms). Also deliberately detuning strings for better results. Amplification-¿more overtones detected but also more noise. Remove spikes in latency by saving FFTW wisdom.

This algorithm has some problems in its current form. Solving these is outside of the scope of this paper and will be the topic of the follow-up thesis, but we will address the problems here. The first big problem is transients. They will appear as many random peaks in the frequency domain. Many of these peaks will be outside of the fundamental frequency range of a guitar, but not all of them, so discarding all high peaks does not solve the problem. Usually, during transients, there any significantly more peaks and even multiple peaks within one note. If this can pattern can consistently be detected, samples can be discarded until the transient is filtered. Another problem is the resolution in the lower frequencies.

8.1 Multi-threading

asdf

References

[1] FFTW page on non-determinism of the Fourier transform.
http://fftw.org/faq/section3.html#nondeterministic
Last accessed on 02-04-2021.

[2] Full vertical scale positions.
https://www.all-guitar-chords.com/scales
Last accessed on 05-04-2021.

[3] Mirex website about multiple fundamental frequency estimation & tracking.
https://www.music-ir.org/mirex/wiki/2020:Multiple_Fundamental_Frequency_Estimation_%26_Tracking
Last accessed on 01-04-2021.

[4] Webpage with shows effect of zero-padding interactively.
https://jackschaedler.github.io/circles-sines-signals/zeropadding.html
Last accessed on 01-04-2021.

[5] Eric J. Anderson. Limitations of short-time fourier transforms in polyphonic pitch recognition. 1997.

[6] Z. Ding and L. Dai. A study of constant q transform in music signal analysis. 24:259–263, 2005.

[7] Z. Duan, Y. Zhang, C. Zhang, and Z. Shi. Unsupervised single-channel music source separation by average harmonic structure modeling. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(4):766–778, 2008.

[8] X. Fiss and A. Kwasinski. Automatic real-time electric guitar audio transcription. 2011.

[9] T. A. Goodman and I. Batten. Real-time polyphonic pitch detection on acoustic musical signals. 2018.

[10] F. Guichard and F. Malgouyres. Total variation based interpolation. 1998.

[11] Julius Orion Smith III. Spectral audio signal processing. 2013.
https://www.dsprelated.com/freebooks/sasp/Quadratic_Interpolation_Spectral_Peaks.html
Last accessed on 26-03-2021.

[12] S.S. Limaye K.A. Akant, R. Pande. Accurate monophonic pitch tracking algorithm for qbh and microtone research. *The Pacific Journal of Science and Technology*, 11(2):342–352, 2010.

[13] Emmanouil Benetos, Simon Dixon, Dimitrios Giannoulis, Holger Kirchhoff, Anssi Klapuri. Automatic music transcription: challenges and future directions. *Journal of Intelligent Information Systems*, 41:407–434, 2013.

[14] Tiago Fernandes Tavares, Jayme Garcia Arnal Barbedo, Romis Attux, Amauri Lopes. Survey on automatic transcription of music. *Journal of the Brazilian Computer Society*, 19:589–604, 2013.

[15] Michael F. Zbyszyński† Matthew Wright, Ryan J. Cassidy. Audio and gesture latency measurements on linux and osx. 2004.

[16] Michael F. Zbyszyński† Matthew Wright, Ryan J. Cassidy. Audio and gesture latency measurements on linux and osx. 2004.

[17] James A. Moorer. On the transcription of musical sound by computer. *Computer Music Journal*, 1(4):32–38, 1977.

[18] Meinard Müller. *Fundamentals of Music Processing*. 2015.

[19] K. M. M. Prabhu. *Window Functions and Their Applications in Signal Processing*. 2014.

[20] A. Schutz and D. Slock. Periodic signal modeling for the octave problem in music transcription. 2009.

[21] Adrian von dem Knesebeck, Udo Zölzer. Comparison of pitch trackers for real-time guitar effects. 2010.

[22] Kurt James Werner. The xqifft: Increasing the accuracy of quadratic interpolation of spectral peaks via exponential magnitude spectrum weighting. 2015.