Real-time monophonic guitar pitch estimation using 12 tuned concurrent Fourier transforms

Luc de Jonckheere

April 6, 2022

Abstract

Short summary.

Contents

1 Introduction

Pitch estimation, which is also referred to as f0 estimation, is an important subtask within the field of Automatic Music Transcription (AMT). The goal of pitch estimation is to estimate the pitch or fundamental frequency f_0 of a given signal. In the context of AMT, pitch estimation is used to determine what note is played in a given signal.

Real-time pitch estimation is a subproblem where we want to estimate the note associated with the measured pitch while the musician is playing it with minimal latency. This entails we have to use the latest received signal. In contrast to non-real-time methods, we have no knowledge of what may happen ahead of time (we cannot peak-ahead) and signal corresponding to previous notes is mostly irrelevant. This limits the methods we can use to solve this problem.

If pitch estimation can accurately be performed in real-time, it can be used to create a digital (MIDI) instrument from an acoustic instrument. This digital instrument can then be used as an input for audio synthesizers, allowing musicians to produce sounds from a wide variety of instruments. Furthermore, accurate real-time pitch estimation can be used to automatically correct detuned instruments by pitch shifting the original signal to the closest harmonious note.

The Fourier transform is often used for pitch estimation. The transform decomposes a signal into the frequencies that make up the signal. Predominant frequencies in the signal show up as spectral peaks in the frequency domain. These peaks are important to human perception of melody [11]. Other popular methods used for pitch estimation include non-negative matrix factorization, autocorrelation, statistical model based estimation and hidden Markov model based estimation.

Our research focuses on monophonic pitch estimation. Here, we assume the signal contains at most one note. It is much easier to perform monophonic pitch estimation compared to polyphonic pitch estimation [12], especially when using Fourier transform based methods, as fundamental limits of the Fourier transform inhibit our ability to discern two low pitched notes [6]. Furthermore, hexaphonic guitar pickups are becoming more widespread, which allows us to view the guitar as six monophonic instruments instead of one six-way polyphonic instrument. Commercial guitar synthesizer solutions from companies with big research departments such as Roland also use these hexaphonic pick-ups, which indicates the infeasibility of accurate and responsive polyphonic pitch estimation of a guitar.

This thesis builds upon a preliminary research project [9]. In our research project, we found that Fourier transform based pitch estimation methods might not be well suited for real-time use due to fundamental limitations of the Fourier transform [7]. In this work, we will further research if Fourier transform based methods are viable, as real-time transcription research often relies Fourier transform based methods.

TODO: Researching pitch estimation is not accessible, as multiple AMT subproblems (e.g. onset detection) have to be solved in order to produce a working prototype on which experiments can be performed. This hinders research of one specific subproblem. Furthermore, as every pitch estimation system is set up slightly different, the combinations of different solutions for the subproblems cannot easily be experimented on. Lastly, experimentation is difficult... To solve these problem, we set out to create a framework in which...

TODO: Rewrite this paragraph. The goal of this thesis is to research the limits of Fourier transform based real-time pitch estimation. To correctly assess the limits, we develop a pitch estimation framework. This framework will focus on extensibility and the ability to perform automated tests. This is important, as much work in this field does not provide its associated source code. This limits the ability to build on other's work and hinders direct comparisons between different methods. Our framework can provide a common ground for the different methods and algorithms to be implemented and compared in. The framework is available at `www.github.com/lucmans/digistring`.

2 Related work

Much research has been performed on Fourier transform based real-time pitch estimation. All research we found relies on obtaining a high resolution frequency domain in which spectral peaks can be isolated and notes can be associated with. These methods are deemed infeasible by some due to low frequency resolution [7].This is especially problematic when adhering to a real-time constraint, as extra short signal frames have to be used. Some papers circumvent this problem by choosing a very high real-time constraint [14, 13], however, this inhibits the use for real-world applications. Furthermore, conventional operating systems also have a latency when delivering audio samples to your program due to how audio drivers work [22].

A big problem with Fourier based pitch estimation is the occurrence of overtones [23]. Especially octaves are a problem, as the fundamental and all overtones of the higher note overlap with overtones of the lower note. This is referred to as the octave problem [25]. Overtones are periodic in nature, as they diminishingly repeat every multiple of the fundamental frequency. As a consequence, they could also be detected using a subsequent Fourier transform [19] on the frequency domain. However, this does not solve the octave problem.

Many different transform have been researched for pitch estimation, however, Fourier transform remains popular as it is broadly studied and its behavior is well known [21]. Lately, the CQT transform is gaining popularity as it may provide higher resolution in the frequency domain [29] at the cost of lower computational efficiency [26]. However, the CQT transform is efficiently implemented using Fourier transforms [8] and the main problem with Fourier is the fundamentally low frequency resolution on short frames, so we are left with the same problem. One big advantage is that the frequency bins can perfectly align with the notes of an instrument [10]. However, as described in Section 3.5, overtones are dissonant with respect to our notes and consequently, the CQT bins do not align with the overtones. If a note perfectly aligns with a Fourier bin, all overtones will also align. In order to cover every note, we could instead perform 12 Fourier transform in parallel. This difference is especially important when performing polyphonic transcription.

Problems with other research, such as no experiments, no available source code, downsampling

Vergelijk materiaal

3 Preliminaries

Jargon required to understand this paper.

3.1 Audio in computers

Audio in computers is represented through a series of equally spaced samples. A sample is the height of the audio wave at a specific point in time. The sample rate determines the number of samples per second used for representing the audio. The sample format determines how the height of the audio wave is represented in a sample. Often used formats are 16/24 bit integers and 32 bit IEEE-754 floating point numbers (which we will refer to as floats). The integer samples simply uniformly spread the range of the waveform over range of the integer [28]. Float samples typically take values in $[-1, 1]$; samples with values outside this range are considered to be clipping. Because float samples have 24 bits precision (23 mantissa bits and a sign bit [5]), 24 bit integer samples can converted lossless to float samples.

When working with audio input/output in computers, a small latency is always introduced [1]. One part of the latency comes from the used audio hardware and is not configurable. The other part comes from the audio driver's buffers. Audio drivers work on buffer of samples instead of single samples for efficiency. A full buffer of data has to be gathered from the audio in, or a full buffer has to be send to the audio out, so the first or last sample respectively is one buffer length behind. The buffer length is calculated by dividing the number of samples in the buffer by the sample rate. In order to minimize latency, the number of samples per buffer has to be minimized and the sample rate has to be maximized. Since these latencies are outside of Digistring's control and can be mostly circumvented by running Digistring's algorithms on specialized hardware, we won't take them into account in this thesis.

3.2 Fourier transform

The Fourier transform is a mathematical transform which transforms a function of time to a complex valued function of frequency and phase. Here, the magnitude represents the amplitude and the argument represents the phase of the corresponding sine wave. The Fourier transform works on continuous functions and assumes an infinite time interval. Concepts such as continuous and infinite cannot be represented by a computer. Consequently, the discrete Fourier transform (DFT) has to be used for Fourier analyses on computers. The DFT can efficiently be calculated using the fast Fourier transform (FFT) algorithm.

The DFT transforms a finite sequence of equally spaced samples, which we will refer to as a frame, into an equal number of complex values representing the amplitude and phase, which we refer to as bins. When working with audio, the samples are real valued, and the DFT output is symmetrical. Because of this, we can discard the second half of the output. In the rest of this thesis, we will only consider the first half of the output. Each bin corresponds to a specific frequency. All other frequencies are spread out over multiple bins due to spectral leakage, which will be discussed later. Given a frame F, the number of samples is the frame is $n_F = |F|$. Using n_F and sample rate f_{SR}, we can calculate the distance between bins in Hz:

$$\Delta f_{bin} = \frac{f_{\text{SR}}}{n_F}$$

This is also referred to as the frequency resolution. Closely related to the frequency resolution is the frame length, which is calculated as follows:

$$t_F = \frac{n_F}{f_{\text{SR}}}$$

Given a bin number $i \in [0, \lfloor \frac{n_F}{2} \rfloor]$, the frequency of a bin can be calculated as:

$$f_{bin} = \Delta f_{bin} * i$$

The 0 Hz bin corresponds to the so called DC offset. This is the average amplitude of the signal. The frequency of the last bin is also called the Nyquist frequency and is equal to half the sample rate. The Nyquist frequency is important for two reasons [31]. The first relates to the sampling theorem, which states that if a continuous function is sampled at a rate of f_{SR} and only contains frequencies f for which $f \leq \frac{f_{\text{SR}}}{2}$, the samples will completely determine the original function. In other words, the samples perfectly describe the original waveform. Secondly, frequencies f for which $f > \frac{f_{\text{SR}}}{2}$ are spuriously moved into $[0, \frac{f_{\text{SR}}}{2}]$, which causes aliasing.

The DFT assumes the frame to be periodic. In other words, the frame is regarded as infinitely repeating. This may lead to aliasing if the beginning and end of a frame do not align, see Figure 1. Here, we take a frame shown by the red lines. The frame is not aligned to a period of the sine wave and causes aliasing as seen in the second graph. This kind of aliasing is called spectral leakage and causes other frequencies which are not a multiple of Δf_{bin} to spread out over multiple bins. By applying a window function, the beginning and end of a frame are forced to align.

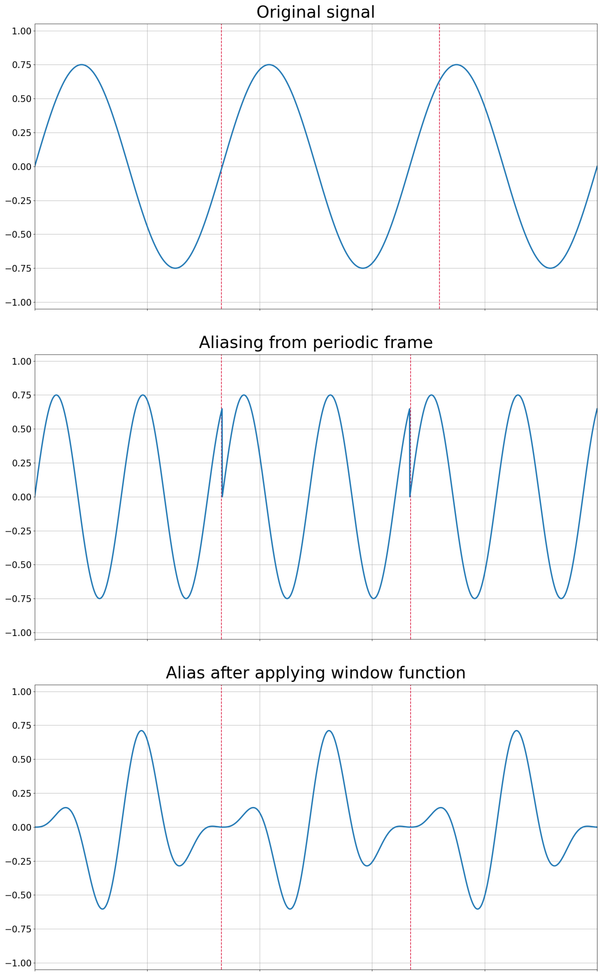

Figure 1: Aliasing which occurs when performing the discrete Fourier transform

Spectral leakage can be controlled using window functions. Given signal $s(n)$ and window function $w(n)$, we get the resulting signal $\text{res}(n)$ using:

$$\text{res}(n) = s(n) * w(n)$$

Figure 2 shows the working of a window function on a frame graphically.

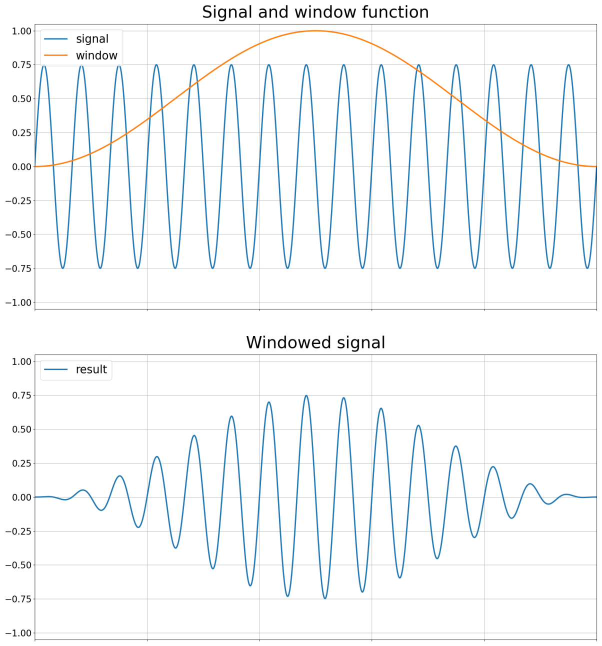

Figure 2: An example of using a window function on a signal

Spectral leakage falls into three categories: Scalloping loss, Main lobe width (of a sine wave) and Sidelobes max level/decrease. The different windows functions trade of between a narrow center lobe and little overall leakage [15], see Figure 3 for some examples. Using the rectangular window could be considered as using no window function. It simply only takes n_F samples from the wave without any alteration. The Hann window is an all round performing window and as a result is often used. The Welch window is a window with a very narrow center lobe. The Dolph–Chebyshev window has little and very evenly spread overall leakage.

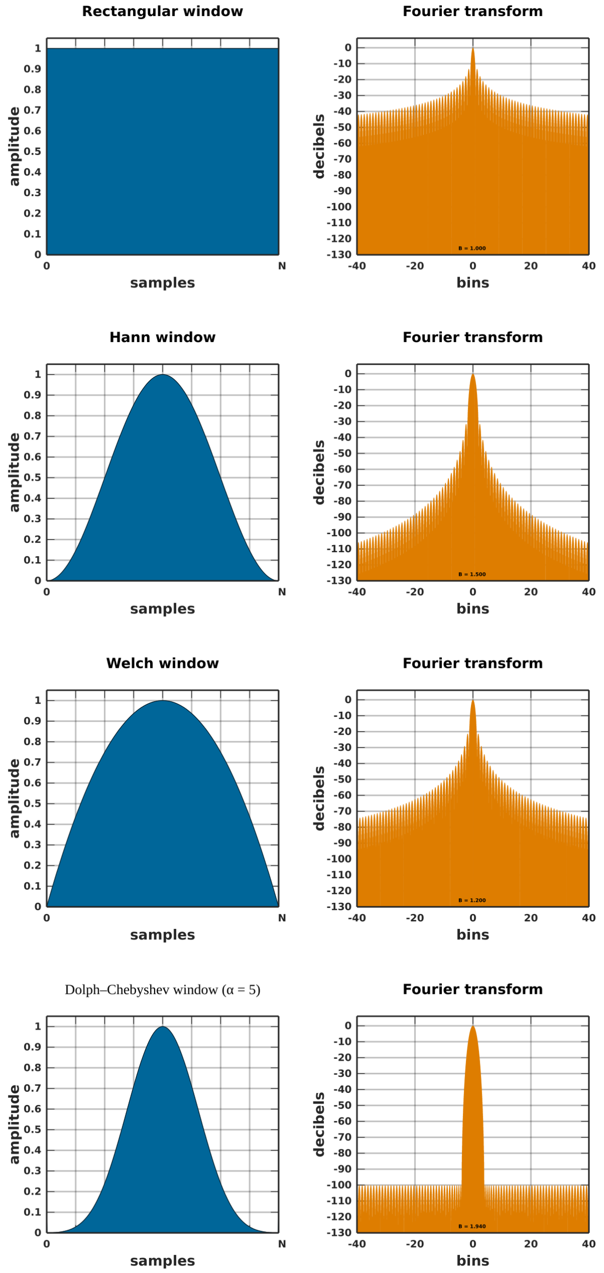

Figure 3: Examples of the spectral leakage of some window functions

Zero-padding can be used to increase the number of output bins of the DFT. It is important to note that it does not increase the resolution of the DFT, it merely interpolates the coarse spectrum to become more smooth [3]. Two frequencies closer than Δf_{bin} together will still fall in the same bin [2]. This form of interpolation is relatively compute intensive [30].

A less compute intensive method of interpolation is quadratic interpolation. It interpolates the actual location of a spectral peak between bins by fitting a quadratic function through the bins on a decibel scale [18, 16]. We calculate a value $p \in [-\frac{1}{2}, \frac{1}{2}]$, which is the offset in bins of the interpolated peak with respect to the peak bin. Using the magnitude of the peak $y(0)$ and the magnitude of the neighboring bins $y(-1)$ and $y(1)$, we define:

$$\begin{aligned} \alpha &= 20 * {}^{10}\log y(-1) \\ \beta &= 20 * {}^{10}\log y(0) \\ \gamma &= 20 * {}^{10}\log y(1) \end{aligned}$$

Then, we can calculate p as follows:

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

The amplitude a_i corresponding to the interpolated peak is:

$$a_i = \beta - \frac{(\alpha - \gamma) * p}{4}$$

Given the bin number b of the spectral peak location, the frequency f_i corresponding to the interpolated peak is:

$$f_i = \Delta f_{bin} * (b + p)$$

Quadratic interpolation can be combined with zero-padding for better results [30].

3.3 Real-time

Real-time is a difficult concept, as it has many definitions. Here are a few examples of relevant definitions:

1. Being synced with actual clock time (or wall time). This is for instance relevant when playing media such as audio and video. When such media is played at an incorrect speed, it could be considered distorted. The hardware which keeps track of the clock time is called a real-time clock.

2. A system must response within a specified time constraint, which is called the real-time constraint or deadline. This constraint is usually a relatively short time. This definition comes from real-time computing and is relevant when making car airbags or airplane control systems. Failing to response within the real-time constraint leads to failures of the system. Real-time systems are often classified into hard, firm and soft real-time based on the impact of missing the deadline [20]. Note that systems which have to respond seemingly instantaneous, such as graphical user interfaces or instant chatting/calling, are not real-time. There are no hard deadlines which the system has to respond within and the system does not fail if some delay does occur. Only user experience is slightly impacted.

3. A system which can process data faster than it acquires data. This is technically not real-time, however, it is often used as such in academic literature. It is important for real-time system to process data faster than it acquires data so it does not lag behind after some time, however, this is an implicit deadline. Not having this deadline explicit may lead to non-sensible expectations of the system.

Even though the first definition is very relevant when working with audio, it is not relevant for us. The audio drivers of operating systems handle all timing for us. We simply have to wait for samples to be recorded and made available to our program. We only have to keep the sampling rate in mind as shown in the previous section.

In order to allow guitarists to use their guitar as a MIDI instrument, our system has to respond within a small time frame. On top of that, if the system fails to respond quickly enough, the usefulness of the result degrades, as timing is very important when playing an instrument. These restrictions would classify our system as a soft real-time system. We choose a real-time of constraint of TODO milliseconds. We elaborate on this choice in section 4.1.

Other work in real-time pitch estimation often uses the third definition of real-time. This is a problem when using Fourier transform based methods, as many papers choose large frame sizes to get a high resolution in the frequency domain. For instance, in order to discern the two lowest notes on a guitar which are 4.9 Hz apart, we would need a frame length of 204 milliseconds. This is well over our real-time constraint and would be unplayable for any musician. Other papers which do explicitly set a real-time constraint, choose very high constraints from 140 ms [14] up to 360 ms [13]. We have found no papers which choose a sensible real-time constraint.

3.4 Music theory and notation

In modern western music, we use the twelve-tone equal temperament (12-TET) music system. This system divides an octave, which is the interval between a pitch and another pitch with double the frequency, into twelve equally spaced semitones on the logarithmic scale. The logarithmic scale is used such that the perceived interval between two adjacent notes is constant [24]. As a result, the ratio between two frequencies in an n-semitone interval is $\sqrt[12]{2}^n$ or $2^{\frac{n}{12}}$, invariant to pitch. A semitone can be further divided into 100 logarithmically scaled cents.

Using scientific pitch notation, every note can be uniquely identified by combining the traditional note names A to G (with accidentals such as $\sharp$ and $\flat$) with an octave number (e.g. E$^\flat_3$). An octave starts at C, which means the octave number increases between B and C. This counter intuitively implies A$_3$ is higher than C$_3$. Note that in 12-TET, C$^\sharp$ and D$^\flat$ are enharmonically equivalent. In this thesis, we will always refer to the sharp ($\sharp$) note instead of the enharmonically equivalent flat note ($\flat$). The range of a typical guitar in standard tuning is from E$_2$ up to E$_6$.

The 12-TET music system only describes the relation between two notes in an interval. In order to play with other musicians in harmony, an arbitrary note has to be tuned to a specific frequency. Per ISO 16, the standard tuning frequency of the A$_4$ is 440 Hz within an accuracy of 0.5 Hz [4].

Using the above information, we can translate frequencies into scientific note names and vice versa. In order to numerically work with note names, we assign a value to each note as shown in Table 1.

name	number	name	number
C	0	F$^\sharp$	6
C$^\sharp$	1	G	7
D	2	G$^\sharp$	8
D$^\sharp$	3	A	9
E	4	A$^\sharp$	10
F	5	B	11

Table 1: The number corresponding to each note name

In order to make calculations easier, we use C$_0$ as a tuning note instead of A$_4$. We can calculate the frequency of C$_0$ using the fact that C$_0$ is 57 semitones lower than A$_4$:

$$\begin{aligned} f_{C_0} = f_{A_4} * 2^{\frac{-57}{12}} &= 440 * 2^{\frac{-57}{12}} \\ &\approx 16.352 \text{ Hz} \end{aligned}$$

We can calculate the frequency f_{N_O}, where N is the note name which is represented by a numerical value given by Table 1 and O is the octave number using:

$$\begin{aligned} f_{N_O} &= f_{C_0} * 2^O * 2^{\frac{N}{12}} \\ &= f_{C_0} * 2^{O+\frac{N}{12}} \end{aligned}$$

To calculate the closest note N_O corresponding to a frequency f, we first calculate the number of semitones n_s between the tuning note f_{C_0} and f:

$$n_s = \left\lfloor 12 * {}^2\log \frac{f}{f_{C_0}} \right\rceil$$

Here, $\lfloor \ldots \rceil$ denotes rounding to the nearest integer. By rounding, we find the closest note to f. Now we can calculate N and O as follows:

$$N = n_s \mod 12$$
$$O = \left\lfloor \frac{n_s}{12} \right\rfloor$$

Note that we assume $a \mod b$ always return a number c for which $0 \leq c < b$. Some programming languages allow the modulo operator to return a value c for which $-b < c < b$, resulting in $-13 \mod 10 = -3$ instead of $-13 \mod 10 = 7$. Furthermore, when using a conversion to an integer instead of a floor, the octave number is rounded up when the note distance is negative.

In order to calculate the error e (in cents) between the given frequency to the closest tuned note, we first calculate the tuned frequency f_t of the closest note:

$$f_t = f_{C_0} * 2^{\frac{n_s}{12}}$$

Then the error e can be calculated using:

$$e = 1200 * {}^2\log \frac{f}{f_t}$$

In digital music processing, notes are often represented through MIDI note numbers. A MIDI note number can take a value from 0 to 127. The MIDI standard defines C_4 to be MIDI note number 60. This makes the standard tuning frequency A_4 number 69 and C_0 number 12.

The MIDI note number m corresponding to the note closest to frequency f can be calculated using the semitone distance from a frequency with a known MIDI note number. Let $m(N_O)$ denote the MIDI note number of N_O:

$$m = \left\lfloor 12 * {}^2\log \frac{f}{f_{N_O}} \right\rceil + m(N_O)$$

Conversely, the frequency f of the note corresponding to MIDI number m can be calculated as follows:

$$f = f_{N_O} * 2^{(m - m(N_O))/12}$$

3.5 Physical properties of sound

The perceived loudness of a note over time can be described using an ADSR envelope. The ADSR envelope of a played note is the convex hull of the waveform of the signal, see Figure 4 for an example. This convex hull can be divided into four parts: Attack, Decay, Sustain and Release. When a note is strummed on the guitar, a percussive sound is generated which causes a loud and sharp attack along with the note. This percussive sound quickly decays and only the actually fretted note will sustain. Finally, when the note is released, it dies out quickly.

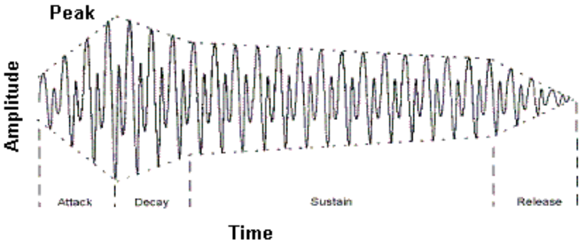

Figure 4: Example of an ADSR envelope (TODO: Better image)

The percussive sound generated when strumming a note is called a transient. Transients contain a high degree of non-periodic components. Because of this, transients appear very chaotically in the frequency domain and are often considered noise. As a transient is of high amplitude, it overshadows the note which will eventually sustain. Consequently, we cannot use the samples from a transient for pitch estimation. This in turn increases our minimum latency, as we have to wait for samples which do not contain the transient anymore.

When playing a note on an instrument, many sine waves are generated. The most notable frequency is called the fundamental frequency and determines what note is actually played. Integer multiples of the fundamental frequency can resonate and give rise to harmonic overtones [27]. In practice, these overtones are not exact integer multiples due to non-linear effect.

Many other frequencies are generated along with the fundamental and its overtones. The instrument specific pattern of these frequencies, along with its envelope, is called the timbre of the instrument [17]. The timbre is what differentiates the sound of the same note played on two different instruments [24]. Generally, the amplitude of the timbre frequencies is low compared to the fundamental frequency and can be disregarded as noise in the frequency domain.

In Section 2, we mentioned overtones are dissonant with respect to notes in 12-TET. This is true for all overtones, except for octaves, which are all overtones numbers equal to $2^n - 1$ for every n. In Table 2, we show an example for the overtones of C_4. Note that the series of errors is always the same, regardless of what the starting note is.

n	f_{overtone}	closest note	f_{note}	error
0	261.626	C_4	261.626	-
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 2: Example of overtone series from C_4 and the errors compared to the closest note

As mentioned in Section 2, when using the CQT transform, none of the non-octave overtones coincides with a CQT bin as the CQT bins are exponentially spaced like the notes in a scale. This causes the frequencies of the overtones to spread out over all other bins due to spectral leakage, resulting in more noise in the frequency domain. Furthermore, overtones are important for discerning fundamentals from noise generated by transients. By using a Fourier transform tuned to a specific note, all its overtones are also coincide with Fourier bins. By performing a Fourier tuned to every note in the 12 tone scale, we can measure every note and its overtones.

TODO: A big problem in monophonic pitch estimation is the octave problem. Octaves are difficult to discern as the fundamental frequency and overtones of the higher note coincide with the overtones of the lower note. A similar problem arises TODO: The main problem in polyphonic pitch estimation comes from the occurrence of overtones. As mentioned before, notes in octave tend overshadow each other. Furthermore, as shown in Table 3, the overtones of a note can coincide with the... TODO: Overtone overlap and polyphonic difficulty.

n	$f_0^{C_3} * n$	n	$f_0^{E_3} * n$	n	$f_0^{G_3} * n$
1	130.813	1	164.814	1	195.998
2	261.626	2	329.628	2	391.995
3	392.438	3	494.441	3	587.993
4	523.251	4	659.255	4	783.991
5	654.064	5	824.069	5	979.989

Table 3: Overtones of C, E and G

Note	f	Δf
$f_2^{C_3}$	392.438	0.443
$f_1^{G_3}$	391.995	
$f_4^{C_3}$	654.064	5.191
$f_3^{E_3}$	659.255	

Table 4: Colliding overtones

4 Thesis content

Actual research etc. Summary what was done in the research project which we build on. Notes on future work of research project.

As mentioned in Section 2 and shown in our research project, the resolution in the frequency domain is not sufficient for spectral peak selecting methods. Given the info in preliminaries, we can tune Fouriers to specific frequencies.

4.1 Real-time constraint

Start with the factors coming into play when choosing the real-time constraint for our system (latency, played notes per second etc). Note on measured latency in Appendix A. Empirically found bounds on latency using our latency program

4.2 Basic algorithm for pitch estimation

High level algorithm description, which consists of getting samples from a source, estimate note events from samples and output note events.

4.3 High resolution estimator

Most basic Fourier based estimator. Base algorithm (apply Fourier, calc norms, pick peaks, find notes in peaks, ...).
Different peak pickers, note finders (in peaks) and note set filtering

4.4 Tuned Fourier transforms

12 Fouriers for each note and it's overtones.

4.5 Overlapping frames

5 Implementation

Details about the program I've written. Usage instructions, code choices, code structure, screenshots, expandability

6 Experiments

Note that the parameters were empirically optimized with informal experiments. Datasets. Actual experiments.

7 Conclusions

What we did in this thesis. Reflection on the performance of the system. Final reference to the source code.

8 Future work

What could still be improved/further researched.

A Measuring the latency of the AXON AX 100 mkII

Lekker meten en weten.

B Effect of different window functions

Plots met effect van verschillende window functions.

References

[1] Digital audio latency explained. `https://www.presonus.com/learn/technical-articles/Digital-Audio-Latency-Explained` Last accessed on 06-04-2022.

[2] Webpage on the limits of zero-padding. `https://dspillustrations.com/pages/posts/misc/spectral-leakage-zero-padding-and-frequency-resolution.html#Frequency-Resolution` Last accessed on 20-10-2021.

[3] Webpage with interactive tool that shows the effect of zero-padding. `https://jackschaedler.github.io/circles-sines-signals/zeropadding.html` Last accessed on 20-10-2021.

[4] Iso 16:1975. acoustics — standard tuning frequency (standard musical pitch). 1975.

[5] IEEE 754-2008 - IEEE standard for floating-point arithmetics. 2008. `https://ieeexplore.ieee.org/document/4610935`.

[6] Eric J. Anderson. Limitations of short-time fourier transforms in polyphonic pitch recognition. Ph.d. qualifying project report, University of Washington, Department of Computer Science and Engineering, 1997.

[7] Eric J. Anderson. Limitations of short-time Fourier transforms in polyphonic pitch recognition. *Technical report, Department of Computer Science and Engineering, University of Washington*, 1997.

[8] Judith Brown and Miller Puckette. An efficient algorithm for the calculation of a constant Q transform. *Journal of the Acoustical Society of America*, 92:2698–2701, 11 1992.

[9] Luc de Jonckheere. Real-time guitar transcription using Fourier transform based methods; a pitch estimation framework and overtone sieve algorithm. 2021.

[10] Robert Dobre and Cristian Negrescu. Automatic music transcription software based on constant Q transform. *8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*, pages 1–4, 2016.

[11] Zhiyao Duan, Bryan Pardo, and Changshui Zhang. Multiple fundamental frequency estimation by modeling spectral peaks and non-peak regions. *IEEE Transactions on Audio, Speech, and Language Processing*, 18(8):2121–2133, 2010.

[12] Paolo Nesi Fabrizio Argenti and Gianni Pantaleo. *Automatic Music Transcription: From Monophonic to Polyphonic*, pages 27–46. Springer Berlin Heidelberg, 2011.

[13] Xander Fiss. Real-time software electric guitar audio transcription. Master's thesis, Rochester Institute of Technology, 2011.

[14] T. A. Goodman and I. Batten. Real-time polyphonic pitch detection on acoustic musical signals. *2018 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT)*, pages 1–6, 2018.

[15] F.J. Harris. On the use of windows for harmonic analysis with the discrete fourier transform. *Proceedings of the IEEE*, 66(1):51–83, 1978.

[16] Julius Orion Smith III. Spectral audio signal processing. 2013. `https://www.dsprelated.com/freebooks/sasp/Quadratic_Interpolation_Spectral_Peaks.html` Last accessed on 20-10-2021.

[17] Kristoffer Jensen. Timbre models of musical sounds. 2021. PhD thesis, University of Copenhagen.

[18] Xavier Serra Julius O. Smith III. PARSHL: An analysis/synthesis program for non-harmonic sounds based on a sinusoidal representation. 1987.

[19] 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.

[20] Hermann Kopetz. *Real-Time Systems: Design Principles for Distributed Embedded Applications.* Springer US, 1997.

[21] 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.

[22] Michael F. Zbyszyński Matthew Wright, Ryan J. Cassidy. Audio and gesture latency measurements on Linux and OSX. *In Proceedings of the ICMC*, pages 423–429, 2004.

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

[24] Michael J. O'Donnell. Digital sound modeling: Perceptual foundations of sound.
`http://people.cs.uchicago.edu/~odonnell/Scholar/Work_in_progress/Digital_Sound_Modelling/lectnotes/node4.html`
Last accessed on 04-10-2021.

[25] A. Schutz and D. Slock. Periodic signal modeling for the octave problem in music transcription. *in Proceedings of the 16th International Conference on Digital Signal Processing (DSP'09)*, pages 1–6, 2009.

[26] Christian Schörkhuber and Anssi Klapuri. Constant-q transform toolbox for music processing. *Proc. 7th Sound and Music Computing Conf.*, 2010.

[27] R. S. Shankland and J. W. Coltman. The departure of the overtones of a vibrating wire from a true harmonic series. *The Journal of the Acoustical Society of America*, 10(3):161 ff, 1939.

[28] Steven W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing.* California Technical Publishing, 1997.

[29] Gino Angelo Velasco, Nicki Holighaus, Monika Doerfler, and Thomas Grill. Constructing an invertible constant-q transform with nonstationary gabor frames. *Proceedings of the 14th International Conference on Digital Audio Effects, DAFx 2011*, 2011.

[30] Kurt James Werner. The XQIFFT: Increasing the accuracy of quadratic interpolation of spectral peaks via exponential magnitude spectrum weighting. *Proceedings of the International Computer Music Conference*, pages 326–333, 2015.

[31] William T. Vetterling William H. Press, Saul A. Teukolsky and Brian P. Flannery. *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press, 1986.